An Introduction to Discrete Mathematics and Algorithms

Charles A. Cusack
cusack@hope.edu

David A. Santos

Version 1.0 (August 12, 2013)

History

- An Introduction to Discrete Mathematics and Algorithms, 2013, Charles A. Cusack. This document draws some content from each of the following.
    - Discrete Mathematics Notes, 2008, David A. Santos.
    - More Discrete Mathematics, 2007, David A. Santos.
    - Number Theory for Mathematical Contests, 2007, David A. Santos.
    - Linear Algebra Notes, 2008, David A. Santos.
    - Precalculus, An Honours Course, 2008, David Santos.

  These documents were all available from http://www.opensourcemath.org/books/santos/, but it appears that the site no longer exists.

Contents

Preface

This book is an attempt to present some of the most important discrete mathematics concepts to computer science students in the context of algorithms. I wrote it for use as a textbook for half of a course on discrete mathematics and algorithms.

Much of the material is drawn from several open-source books by David Santos. Other material is from handouts I have written and used over the years. I have extensively edited the material from both sources, both for clarity and to emphasize the connections between the material and algorithms where possible.

I should mention that I never met David Santos, who apparently died in 2011. I stumbled upon his books this summer (2013) when I was searching for a discrete mathematics book to use in a new course. When I discovered that I could adapt his material for my own use, I decided to do so. Since clearly he has no knowledge of this book, he bears no responsibility for any of the edited content. Any errors or omissions are therefore my fault.

This is still a work in progress, so I appreciate any feedback you have. Please send it to cusack@hope.edu.

Charles A. Cusack
August, 2013

Chapter 1

Proof Methods

In this chapter we will introduce you to the basics of mathematical proofs. Along the way we will review some mathematical concepts/definitions you have probably already seen, and introduce you to some new ones that we will find useful as we proceed.

1.1 Direct Proofs

A *direct proof* is one that follows from the definitions. Facts previously learned help many a time when making a direct proof.

1 Definition Recall that

- an *even integer* is one of the form $2k$, where k is an integer.
- an *odd integer* is one of the form $2l+1$ where l is an integer.

2 Example Prove that

❶ the sum of two even integers is even,

❷ the sum of two odd integers is even,

❸ the sum of an even integer and an odd integer is odd,

❹ the product of two odd integers is odd,

❺ the product of an even integer and an odd integer is even.

Solution: We argue from the definitions. We assume as known that the sum of two integers is an integer.

❶ If x and y are even integers, then $x = 2a$ and $y = 2b$ for some integers a and b. Then $x+y = 2a+2b = 2(a+b)$. Since $a+b$ is an integer, $2(a+b)$ is even.

❷ If x and y are odd, then $x = 2c+1$ and $y = 2d+1$ for some integers c and d. Then $x+y = 2c+1+2d+1 = 2(c+d+1)$. Now $c+d+1$ is an integer, so $2(c+d+1)$ is an even integer.

❸ Let $2f$ be an even integer and $2g+1$ be an odd integer. Then $2f+2g+1=2(f+g)+1$. Since $f+g$ is an integer, $2(f+g)+1$ is an odd integer.

❹ Let $2l+1$ and $2m+1$ be odd integers. Then

$$(2l+1)(2m+1)=4ml+2l+2m+1=2(2ml+l+m)+1.$$

Since $2ml+l+n$ is an integer, $2(2ml+m+l)+1$ is an odd integer.

❺ Let $2n$ be an even integer and let $2o+1$ be an odd integer. Then

$$(2n)(2o+1)=4no+2n=2(2no+1).$$

Since $2no+1$ is an integer, $2(2no+1)$ is an even integer.

3 Definition Let b and a be integers with $a \neq 0$. We say that b is *divisible by* a if there exists an integer c such that $b=ac$. If b is divisible by a, we also say that b is a *multiple* of a, a is a *factor* or *divisor* of b, and that a *divides* b, written as $a|b$. If a does not divide b, we write $a \nmid b$.

4 Example Since $6=2\cdot 3$, $2|6$, and $3|6$. But $4 \nmid 6$ since we cannot write $6=4\cdot c$ for any integer c.

5 Example Prove that the product of two even integers is divisible by 4.

Solution: Let $2h$ and $2k$ be even integers. Then $(2h)(2k)=4(hk)$. Since hk is an integer, $4(hk)$ is divisible by 4.

☞ *A common mistake when writing proofs is to make one or more invalid assumptions without realizing it. The problem with this is that it generally means you are not proving what you set out to prove, but since the proof seems to "work", the mistake isn't obvious. The next examples should illustrate what can go wrong if you aren't careful.*

6 Example What is wrong with this proof that the sum of two even integers is even?

Proof: Let x and y be even integers. Then $x=2a$ for some integer a and $y=2a$ for some integer a. So $x+y=2a+2a=2(a+a)$. Since $a+a$ is an integer, $2(a+a)$ is even, so the sum of two even integers is even. □

Solution: The problem is that this is actually a proof that $x+x$ is even if x is even since $x=2a=y$ was assumed.

Although this may not seem like a big deal since the statement is true, consider the next example.

7 Example What is wrong with the following proof that the sum of two even integers is divisible by 4?

Proof: Let x and y be to even integers. Then $x=2a$ for some integer a and $y=2a$ for some integer a. So $x+y=2a+2a=4a$. Since a is an integer, $4a$ is divisible by 4, so the sum of two even integers is divisible by 4. □

Solution: Notice that 4 and 6 are even, but $4+6=10$ is not divisible by 4. So clearly the statement is incorrect. Therefore, there must be something wrong with the proof. The problem is the same as it was above–the proof assumed $x=y$, even if that was not the intent of the writer. So what was proven was that if x is even, then $x+x$ is divisible by 4.

Let's continue with some more examples of proper proofs.

8 Example Prove that if n is an integer, then $n^3 - n$ is divisible by 6.

> **Proof:** We have $n^3 - n = (n-1)n(n+1)$, the product of three consecutive integers. Among three consecutive integers there is at least an even one, and exactly one of them which is divisible by 3. Since 2 and 3 do not have common factors, 6 divides the quantity $(n-1)n(n+1)$, and so $n^3 - n$ is divisible by 6. □

9 Definition The symbol $\forall$ is the *universal quantifier*, and it is read as "for all", "for each", "for every", etc. For instance, $\forall x$ means "for all x". When it precedes a statement, it means that the statement is true *for all values of* x.

As the name suggests, the "all" refers to everything in the *universe of discourse* (or *domain of discourse*, or simply *domain*), which is simply the set of objects to which the current discussion relates.

10 Example When you see the notation $\forall x \geq 0$, it means "for all x, x is greater than or equal to 0." However, what is the domain? In this case, the most logical possibilities are the integers or real numbers. Generally speaking, the context of its use should make it clear what the universe is.

As long as we are introducing quantifiers, I suppose we should introduce the other one that is often used.

11 Definition The symbol $\exists$ is the *existential quantifier*, and it is read as "there exists", "there is", "for some", etc. For instance, $\exists x$ means "For some x". When it precedes a statement, it means that the statement is true for *at least one value of* x in the universe.

Notice that $\neg\forall = \exists$ and $\neg\exists = \forall$.

12 Example Use the fact that the square of any real number is non-negative in order to prove the *Arithmetic Mean-Geometric Mean Inequality:* $\forall x \geq 0, \forall y \geq 0$

$$\sqrt{xy} \leq \frac{x+y}{2}.$$

> **Proof:** Since x and y are non-negative, $\sqrt{x}$ and $\sqrt{y}$ are real numbers, so $\sqrt{x} - \sqrt{y}$ is a real number. Since the square of any real number is greater than or equal to 0 we have
>
> $$(\sqrt{x} - \sqrt{y})^2 \geq 0.$$
>
> Expanding (recall the FOIL method?) we get
>
> $$x - 2\sqrt{xy} + y \geq 0.$$
>
> Subtracting $2\sqrt{xy}$ from both sides and dividing by 2, we get
>
> $$\frac{x+y}{2} \geq \sqrt{xy},$$
>
> yielding the result. □

The previous example illustrates the creative part of writing proofs. The proof started out considering $\sqrt{x}-\sqrt{y}$, which doesn't seem to be related to what we wanted to prove. But hopefully after you read the entire proof you see why it makes sense. If you are saying to yourself "I would never have thought of starting with $\sqrt{x}-\sqrt{y}$?," or "How do you know where to start?," I am afraid there are no easy answers. Writing proofs is as much of an art as it is a science. There are three things that can help, though. First, don't be afraid to *experiment*. If you aren't sure where to begin, try starting at the end. Think about the end goal and work backwards until you see a connection. Sometimes working both backward and forward can help. Try some algebra and see where it gets you. But in the end, make sure your proof goes from beginning to end. In other words, the order that you figured things out should not necessarily dictate the order they appear in your proof.

The second thing you can do is to *read example proofs*. Although there is some creativity necessary in proof writing, it is important to follow the proper proof writing techniques. Although there are often many ways to prove the same statement, there is often one technique that works best for a given type of problem. As you read more proofs, you will begin to have a better understanding of the various techniques used, know when a particular technique might be the best choice, and become better at writing your own proofs. If you see several proofs of similar problems, and the proofs look very similar, then when you prove a similar problem, your proof should probably resemble those proofs. This is one area where some students struggle—they submit proofs that look nothing like any of the examples they have seen, and they are often incorrect. Perhaps it is because they are afraid that they are plagiarizing if they mimic another proof too closely. However, mimicking a proof is not the same as plagiarizing a sentence. To be clear, by 'mimic', I don't mean just copy exactly what you see. I mean that you should read and understand several examples. Once you understand the technique used in those examples, you should be able to see how to use the same technique in your proof. For instance, in many of the examples related to even numbers, you may have noticed that they start with statement like *"Assume x is even. Then $x = 2a$ for some integer a."* So if you need to write a proof related to even numbers, what sort of statement might make sense to begin your proof?

The third thing that can help is *practice*. This applies not only to writing proofs, but to learning many topics. An analogy might help here. Learning is often like sports—you don't learn how to play basketball (or insert your favorite sport, video game, or other hobby that takes some skill) by reading books and/or watching people play it. Those things can be helpful (and in some cases necessary), but you will never become a proficient basketball player unless you practice. Practicing a sport involves running many drills to work on the fundamentals and then applying the skills you learned to new situations. Learning many topics is exactly the same. First you need to do lots of exercises to practice the fundamental skills. Then you can apply those skills to new situations. When you can do that well, you know you have a good understanding of the topic. So if you want to become better at writing proofs, you need to write more proofs.

Let's get back to some examples. But first another definition.

13 Definition The mod operator is defined as follows: for $a \geq 0$, $n > 0$, $a \bmod n$ is the integral non-negative remainder when a is divided by n. Observe that this remainder is one of the n numbers

$$0, \quad 1, \quad 2, \quad \ldots, \quad n-1.$$

Java, C, and C++ all use `%` for mod (e.g. `int x = a % n` instead of `int x = a mod n`).

14 Example Here are some example computations:

$$\begin{array}{lll}
234 \bmod 100 = 34 & 1961 \bmod 37 = 0 & 6 \bmod 5 = 1 \\
38 \bmod 15 = 8 & 1966 \bmod 37 = 5 & 11 \bmod 5 = 1 \\
15 \bmod 38 = 15 & 1 \bmod 5 = 1 & 16 \bmod 5 = 1
\end{array}$$

☞ *Our definition of* mod *required that $n > 0$ and $a \geq 0$. However, it is possible to define a* mod *n when a is negative. Unfortunately, there are two possible ways of doing so based on how you define the remainder when the dividend is negative. One possible answer is negative and the other is positive. However, they always differ by n, so computing one from the other is easy.*

15 Example Since $-13 = (-2) * 5 - 3$ and $-13 = (-3) * 5 + 2$, we might consider the remainder of $13/5$ as either -3 or 2. Thus, $-13 \bmod 5 = -3$ and $-13 \bmod 5 = 2$ both seem like reasonable answers. Fortunately, the two possible answers differ by 5. In fact, you can always obtain the positive possibility by adding n to the negative possibility.

☞ *When using the* mod *operator in computer programs in situations where the dividend might be negative, it is important to know which definition your programming language/compiler uses.* Java *returns a negative number when the dividend is negative. In* C *, the answer depends on the compiler.*

16 Example Show that for every integer n, $n^2 \bmod 4$ is either 0 or 1.

> **Proof:** Since every integer is either even (of the form $2k$) or odd (of the form $2k+1$) we have two possibilities:
>
> $$\begin{array}{lclcll}
(2k)^2 & = & 4k^2 & \equiv & 0 & \pmod n, \text{or} \\
(2k+1)^2 & = & 4(k^2+k)+1 & \equiv & 1 & \pmod n.
\end{array}$$
>
> Thus, n^2 has remainder 0 or 1 when divided by 4. □

17 Example Prove that the sum of two squares of integers leaves remainder 0, 1 or 2 when divided by 4.

> **Proof:** According to Example 16, the squares of integers have remainder 0 or 1 when divided by 4. Thus, when we add two squares, the possible remainders when divided by 4 are 0 $(0+0)$, 1 $(0+1$ or $1+0)$, and 2 $(1+1)$. □

1.2 Proof by Contradiction

In this section we will see examples of *proof by contradiction*. For this technique, when trying to prove a premise, we assume that its negation is true and deduce incompatible statements from this. This implies that the original statement must be true.

18 Definition Given a statement p, the *negation* of p, written $\neg p$, is the statement "not p" or "it is not the case that p."

19 Example If p is the statement "$x \leq y$" then $\neg p$ is the statement "it is not the case that $x \leq y$," or "$x > y$".

20 Example Prove that 2003 is not the sum of two squares.

> **Proof:** Assume that 2003 is the sum of two squares. In Example 17 we showed that the sum of two squares leaves a remainder of 0, 1, or 2 when divided by 4. But 2003 leaves a remainder of 3. This is a contradiction. So it must be the case that 2003 is not the sum of two squares. □

☞ *Here is the basic concept of contradiction proofs: You want to prove that a statement p is true. You "test the waters" by seeing what happens if p is* not *true. So you assume p is false and use proper proof techniques to arrive at a contradiction. By "contradiction" I mean something that cannot be possible true. Since you proved something that is not true, and you used proper proof techniques, then it must be that your assumption was incorrect. Therefore the negation of your assumption—which is the original statement you wanted to prove—must be true.*

For some students, the trickiest part of contradiction proofs is what to contradict. Sometimes the contradiction is the fact that p is true. At other times you arrive at a statement that is clearly false (e.g. $0 > 1$). Generally speaking, you should just try a few things (e.g. do some algebra) and see where it leads. With practice, this gets easier. In fact, with enough practice, this will become one of your favorite techniques.

21 Example Show, without using a calculator, that $6-\sqrt{35} < \dfrac{1}{10}$.

> **Proof:** Assume that $6-\sqrt{35} \geq \dfrac{1}{10}$. Then $6-\dfrac{1}{10} \geq \sqrt{35}$. Multiplying both sides by 10 and doing a little arithmetic, we get $59 \geq 10\sqrt{35}$. Squaring both sides we obtain $3481 \geq 3500$, which is clearly nonsense. Thus it must be the case that $6-\sqrt{35} < \dfrac{1}{10}$. □

22 Definition A *permutation* is a function from a finite set to itself that reorders the elements of the set.[1]

23 Example Let S be the set $\{a,b,c\}$. Then (a,b,c), (b,c,a) and (a,c,b) are permutations of S. (a,a,c) is not a permutation of S because it repeats a and does not contain b. (b,d,a) is not permutations of S because d is not in S, and c is missing.

☞ *In many contexts, when a list of objects occurs in* curly braces, *the order they are listed does not matter (e.g. $\{a,b,c\}$ and $\{b,c,a\}$ mean the same thing). On the other hand, when a list occurs in parentheses, the order* does *matter (e.g. (a,b,c) and (b,c,a)* do not *mean the same thing).*

24 Example Let $(a_1,a_2,\ldots,a_n)$ be an arbitrary permutation of the numbers $1,2,\ldots,n$, where n is an odd number. Prove that the product $(a_1-1)(a_2-2)\cdots(a_n-n)$ is even.

> **Proof:** Assume that the product is odd. Then all of the differences a_k-k must be odd. Now consider the sum $S=(a_1-1)+(a_2-2)+\cdots+(a_n-n)$. Since the a_k's are a just a reordering of $1,2,\ldots,n$, $S=0$ (think about it for a minute if you need to, but convince yourself of this fact). But S is the sum of an odd number of odd integers, so it must be odd. Since 0 is not odd, we have a contradiction. Thus our initial assumption that all of the a_k-k are odd is wrong, so one of them is even and hence the product is even. □

We will use facts about rational/irrational numbers to demonstrate some of the proof techniques. In case, you have forgotten, here are the definitions.

25 Definition Recall that

- A *rational number* is one that can be written as p/q, where p and q are integers, with $q \neq 0$.

[1] We will discuss sets more formally later. For now, just think of a set as a collection of objects of some sort.

- An *irrational number* is a real number that is not rational.

26 Example Prove that $\sqrt{2}$ is irrational.

We present two slightly different proofs. In both, we will use the fact that any positive integer greater than 1 can be factored uniquely as the product of primes (up to the order of the factors).

> **Proof:** (#1) Assume that $\sqrt{2} = \frac{a}{b}$, where a and b are positive integers with $b \neq 0$. We can assume a and b have no factors in common (since if they did, we could cancel them and use the resulting numerator and denominator as a and b). Multiplying by b and squaring both sides yields $2b^2 = a^2$. Clearly 2 must be a factor of a^2. Since 2 is prime, a must have 2 as a factor, and therefore a^2 has 2^2 as a factor. Then $2b^2$ must also have 2^2 as a factor. But this implies that 2 is a factor of b^2, and therefore a factor of b. This contradicts the fact that a and b have no factors in common. Therefore $\sqrt{2}$ must be irrational. □

> **Proof:** (#2) Assume that $\sqrt{2} = \frac{a}{b}$, where a and b are positive integers with $b \neq 0$. This yields $2b^2 = a^2$. Now both a^2 and b^2 have an even number of prime factors (think about why this is). So $2b^2$ has an odd numbers of primes in its factorization and a^2 has an even number of primes in its factorization. This is a contradiction since they are the same number. Therefore $\sqrt{2}$ must be irrational. □

Now that you have seen a few more examples, let's discuss how/why contradiction proofs work. It may not have occurred to you, but it turns out that if you start with a false assumption, then you can prove that *anything* is true. It may not be obvious how (e.g. how would you prove that all elephants are less than 1 foot tall given that $1+1=1$?), but in theory it is possible. This is because statements of the form "p implies q" are true if p is false, regardless of whether or not q is true or false. More on this in the chapter on logic.

On the other hand, if p is true, and "p implies q" is true, then q also has to be true (a rule called *modus ponens*). We won't prove this, but if you think about it for a few minutes, hopefully you'll see why it is correct. Contradiction proofs exploit this rule.

In a contradiction proof, we want to prove that p is true. We begin by assuming it is false—that is, we assume $\neg a$ is true. We use this fact to prove that q—some false statement—is true. In other words, we prove that the statement "$\neg p$ implies q" is true, where q is some false statement. But if $\neg a$ is true, and "$\neg p$ implies q" is true, modus ponens tells us that q has to be true. So what's wrong? We only have two choices: either $\neg p$ is false or "$\neg p$ implies q" is false. If we used proper proof techniques to establish that "$\neg p$ implies q" is true, then that isn't the problem. Therefore, the only other possibility is that $\neg p$ is false, implying that p must be true. And that is how/why contradiction proofs work.

Let's analyze the last proof we saw in light of this discussion. The *only* assumption we made was that $\sqrt{2}$ is rational ($\neg p$="$\sqrt{2}$ is rational"). From this (and only this), we were able to show that a^2 has both an even and an odd number of factors (q="a^2 has an even and an odd number of factors", and we showed that "$\neg p$ implies q" is true). Thus, we know for certain that if $\sqrt{2}$ is rational, then a^2 has an even and an odd number of factors.[2] This fact is indisputable since we proved it. If it is also true that $\sqrt{2}$ is rational, modus ponens implies that a^2 has an even and an odd number of factors. This is also indisputable. But we know that a^2 cannot have both an even and odd number of factors. In other words, we have a contradiction. Therefore, something that has been said somewhere is wrong. Everything we said is indisputable except

[2] We did not prove that a^2 has an even and an odd number of factors. We proved that *if $\sqrt{2}$ is rational*, then a^2 has an even and an odd number of factors. It is very important that you understand the difference between these two statements.

for one thing–that $\sqrt{2}$ is rational. That was never something we proved—we just assumed it. So it has to be the case that this is false, which means that $\sqrt{2}$ must be irrational.

To summarize, if you want to prove something is true using a contradiction proof, assume it is false, get a contradiction (i.e. prove a false statement), and declare that it must therefore be true.

Notice that what q is doesn't matter. In other words, given the assumption $\neg p$, the goal in a contradiction proof is to establish that *any* false statement is true. This is both a blessing and a curse. The blessing is that any contradiction will do. The curse is that we don't have a clear goal in mind, so it can sometimes be difficult to know what to do. As mentioned previously, this becomes easier as you read and write more proofs.

If this discussion has been a bit confusing, try re-reading it. The better you understand the theory behind contradiction proofs, the better you will be at writing them. We will revisit some of these concepts in the chapter on logic, so the more you understand from here, the better off you will be when you get there.

O.K., enough theory. Let's eat some ice cream and see some more examples![3]

27 Example Let a,b be real numbers and assume that for all numbers $\varepsilon > 0$ the following inequality holds:

$$a < b+\varepsilon.$$

Prove that $a \leq b$.

Solution: Assume that $a > b$. Subtracting b from both sides and dividing by 2, we obtain $\frac{a-b}{2} > 0$. Since the inequality $a < b+\varepsilon$ holds for every $\varepsilon > 0$ in particular it holds for $\varepsilon = \frac{a-b}{2}$. This implies that

$$a < b+\frac{a-b}{2} \quad \text{or} \quad a < b,$$

the last step requiring a little algebra. Thus starting with the assumption that $a > b$ we reach the incompatible conclusion that $a < b$. The original assumption must be wrong. We therefore conclude that $a \leq b$.

28 Example (Euclid) Show that there are infinitely many prime numbers.

Proof: We need to assume for this proof that any integer greater than 1 is either a prime or a product of primes. The following beautiful proof goes back to Euclid.

Assume that $\{p_1, p_2, \ldots, p_n\}$ is a list that exhausts all the primes. Consider the number

$$N = p_1 p_2 \cdots p_n + 1.$$

This is a positive integer, clearly greater than 1. Observe that none of the primes on the list $\{p_1, p_2, \ldots, p_n\}$ divides N, since division by any of these primes leaves a remainder of 1. Since N is larger than any of the primes on this list, it is either a prime or divisible by a prime outside this list. Thus we have shown that the assumption that any finite list of primes leads to the existence of a prime outside this list. This implies that the number of primes is infinite. □

29 Example If a,b,c are odd integers, prove that $ax^2+bx+c=0$ does not have a rational number solution.

[3] Ice cream not provided.

Proof: Suppose $\frac{p}{q}$ is a rational solution to the equation. We may assume that p and q have no prime factors in common, so either p and q are both odd, or one is odd and the other even. Then

$$a\left(\frac{p}{q}\right)^2+b\left(\frac{p}{q}\right)+c=0 \implies ap^2+bpq+cq^2=0.$$

If both p and p were odd, then $ap^2+bpq+cq^2$ is also odd and hence $\neq 0$. Similarly if one of them is even and the other odd then either ap^2+bpq or $bpq+cq^2$ is even and $ap^2+bpq+cq^2$ is odd. This contradiction proves that the equation cannot have a rational root. □

1.3 Proof by contraposition

Consider the statement "If it rains, then the ground will get wet." It should be pretty easy to convince yourself that this is essentially equivalent to the statement "If the ground is not wet, then it didn't rain." By this I simply mean that either both statements are true or both statements are false. This is the idea behind the proof technique in this section.

30 Definition The *contrapositive* of a statement of the form "if p, then q" is the statement "if p is not true, then q is not true" or "if not p, then not q"

31 Theorem A statement is true if and only if its contrapositive is true.

☞ *We will take a closer look at the relationship between statements of the form "if p then q", including the contrapositive, in the chapter on logic. For now it suffices to convince yourself that the previous theorem is true–or at least seems to be true.*

32 Definition A *proof by contraposition* is a proof of a statement of the form "if p, then q" that proves the equivalent statement "if p is not true, then q is not true."

33 Example Prove that if $5n+2$ is odd, then n is odd.

Proof: We will instead prove that if n is even (not odd), then $5n+2$ is even (not odd). Since this is the contrapositive of the original statement, a proof of this will prove that that the original statement is true.
Assume n is even. The $n=2a$ for some integer a. Then $5n+2=5(2a)+2=2(5a+1)$. Since $5a+1$ is an integer, $2(5a+1)$ is even. □

1.4 Other Proof Techniques

There are many other proof techniques. We conclude this chapter with a small sampling of the more common and/or interesting ones. We will see a few other important techniques later in the book.

34 Definition A *trivial proof* is a proof of a statement of the form "if p, then q" that doesn't use p in the proof.

35 Example Prove that if $x>0$, then $(x+1)^2-2x>x^2$.

Proof: It is easy to see that

$$\begin{aligned}(x+1)^2-2x &= (x^2+2x+1)-2x\\ &= x^2+1\\ &> x^2.\end{aligned}$$

□

Notice that we never used the fact that $x > 0$ in the proof.

36 Definition A *proof by counterexample* is used to disprove a statement by giving an example of it being false.

37 Example Prove or disprove that the product of two irrational number is irrational.

Proof: We showed in Example 26 that $\sqrt{2}$ is irrational. But $\sqrt{2}*\sqrt{2}=2$, which is an integer so it is clearly rational. Thus the product of 2 irrational number is not always irrational. □

38 Example Prove or disprove that "Everybody Loves Raymond" (or that "Everybody Hates Chris").

Proof: Since I don't really love Raymond (and I don't hate Chris), the statement is clearly false. □

39 Definition A *proof by cases* breaks up a statement into multiple cases and proves each one separately.

40 Example Prove that if $x \neq 0$ is a real number, then $x^2 > 0$.

Proof: If $x \neq 0$, then either $x > 0$ or $x < 0$. If $x > 0$ (case 1), then we can multiply both sides of $x > 0$ by x, giving $x^2 > 0$.
If $x < 0$ (case 2), then we can write y=-x, where $y > 0$. Then $x^2 = (-y)^2 = (-1)^2y^2 = y^2 > 0$ by case 1 (since $y > 0$). Thus $x^2 > 0$. In either case, we have shown that $x^2 > 0$. □

41 Definition Recall that for a non-negative integer n the quantity $n!$ (read "n factorial") is defined as follows. $0! = 1$ and if $n > 0$ then $n!$ is the product of all the integers from 1 to n inclusive:

$$n! = 1 \cdot 2 \cdots n.$$

42 Example $3! = 1 \cdot 2 \cdot 3 = 6$, and $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$.

Exercises

43 Problem Prove that if $n > 4$ is composite, then n divides $(n-1)!$.

44 Problem Prove that p, $p+2$, and $p+4$ are not all prime unless $p = 3$.

45 Problem If x is an integer and 7 divides $3x+2$ prove that 7 also divides $15x^2 - 11x - 14$.

46 Problem Let s be a positive integer. Prove that the closed interval $[s, 2s]$ contains a power of 2.

47 Problem Let $p < q$ be two *consecutive* odd primes. Prove that $p+q$ is a composite number, having at least three, not necessarily distinct, prime factors.

48 Problem Prove, by arguing by contradiction, that there are no integers a, b, c, d such that

$$x^4 + 2x^2 + 2x + 2 = (x^2 + ax + b)(x^2 + cx + d).$$

49 Problem Use the fact that any odd number is of the form $8k \pm 1$ or $8k \pm 3$ in order to give a direct proof that the square of any odd number leaves remainder 1 upon division by 8. Use this to prove that 2001 is not the sum of three odd squares.

Answers

43 Either n is a perfect square, $n = a^2$ in which case $2 < a < 2a \leq n-1$ and hence a and $2a$ are among the numbers $\{3, 4, \ldots, n-1\}$ or n is not a perfect square, but still composite, with $n = ab$, $1 < a < b < n-1$.

44 The statement is clearly false for $p = 2$. If $p > 3$ and prime, p is odd. But then one of the three consecutive odd numbers p, $p+2$, $p+4$, must be divisible by 3 and is different from 3 and hence not a prime.

45 We have $3x+2 = 7a$, with a an integer. Furthermore, $15x^2 - 11x - 14 = (3x+2)(5x-7) = 7a(5x-7)$, whence 7 divides $15x^2 - 11x - 14$.

46 If s is itself a power of 2 then we are done. Assume that s is strictly between two powers of 2: $2^{r-1} < s < 2^r$. Then $2^r = 2 \cdot 2^{r-1} < 2s < 2 \cdot 2^r = 2^{r+1}$, so $s < 2^r < 2s < 2^{r+1}$, and so the interval $[s, 2s]$ contains 2^r, a power of 2.

47 Since p and q are odd, we know that $p+q$ is even, and so $\dfrac{p+q}{2}$ is an integer. But $p < q$ gives $2p < p+q < 2q$ and so $p < \dfrac{p+q}{2} < q$, that is, the average of p and q lies between them. Since p and q are consecutive primes, any number between them is composite, and so divisible by at least two primes. So $p+q = 2\left(\dfrac{p+q}{2}\right)$ is divisible by the prime 2 and by at least two other primes dividing $\dfrac{p+q}{2}$.

48 We have

$$\begin{aligned} x^4 + 2x^2 + 2x + 2 &= (x^2 + ax + b)(x^2 + cx + d) \\ &= x^4 + (a+c)x^3 + (d+b+ac)x^2 + (ad+bc)x + bd. \end{aligned}$$

Thus

$$bd = 2, \quad ad + bc = 2, \quad d + b + bc = 2, \quad a + c = 2.$$

Assume a, b, c, d are integers. Since $bd = 2$, bd must be of opposite parity (one odd, the other even). But then $d + b$ must be odd, and since $d + b + bc = 2$, bc must be odd, meaning that both b and c are odd, whence d is even. Therefore ad is even, and so $ad + bc = 2$ is even plus odd, that is, odd: a contradiction since 2 is not odd.

49 We have

$$(8k \pm 1)^2 = 64k^2 \pm 16k + 1 = 8(8k^2 \pm 2) + 1,$$

$$(8k \pm 3)^2 = 64k^2 \pm 48k + 9 = 8(8k^2 \pm 6 + 1) + 1,$$

proving that in all cases the remainder is 1 upon division by 8.

Now, a sum of three odd squares must leave remainder 3 upon division by 8. Thus if 2001 were a sum of three squares, it would leave remainder $3 = 1 + 1 + 1$ upon division by 8. But 2001 leaves remainder 1 upon division by 8, a contradiction to the assumption that it is a sum of three squares.

Homework

50 Problem Prove that the product of two odd integers is odd.

51 Problem Prove or disprove that if x is irrational, then $1/x$ is irrational.

52 Problem Prove that if a and b are integers and ab is even, then at least one of a or b is even.

53 Problem Prove or disprove that there are 100 consecutive positive integers that are not perfect squares. (Recall: a number is a perfect square if it can be written as a^2 for some integer a.)

54 Problem Prove or disprove that if x and y are rational, then x^y is rational. (Don't over think it. This one should be easy.)

55 Problem Mersenne primes are primes that are of the form $2^p - 1$, where p is prime. Are all numbers of this form prime? Give a proof/counterexample.

56 Problem Consider the equation $x^4 + y^4 = 625$. Are there any integers x and y that satisfy this equation? Prove it.

57 Problem Let n be a positive integer. We can more formally define congruence modulo n by saying that $a \equiv b \pmod{n}$ if n divides $a - b$. Use this formal definition to prove each of the following:

1. $a \equiv a \pmod{n}$. (Reflexive property)
2. If $a \equiv b \pmod{n}$, then $b \equiv a \pmod{n}$. (Symmetric property)
3. Prove that if $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$. (Transitive property)

58 Problem Prove or disprove that $P = NP$.[4]

[4] A successful solution to this will earn you an A in the course.

Chapter 2

Programming Fundamentals and Algorithms

This chapter serves as a review of some of the programming concepts you should have picked up in previous classes. It also includes several mathematical definitions that you may or may not be familiar with. The emphasis is on presenting some basic algorithms as a way to refresh your memory on basic programming concepts. We will also practice our skills at proving things in this chapter by sometimes proving that an algorithm does as specified. Finally, we include a section on *recursion*, which you may not be as familiar with as the other topics.

Algorithms are presented in one of two forms: *pseudocode* (the syntax of which is described as deemed necessary) or *code* resembling Java and C++.

2.1 Algorithms

An *algorithm* is a set of instructions that accomplishes a task in a finite amount of time.

59 Example (Area of a Trapezoid) Write an algorithm that gives the area of a trapezoid with height h and bases a and b.

Solution: One possible solution is

```
Algorithm 2.1.1: AREATRAPEZOID($a,b,h$)

return $(h*(a+b)/2)$
```

Notice that we use the *return* keyword to indicate what value should be passed to whoever calls an algorithm. For instance, if someone calls x=AREATRAPAZOID(a,b,h), then x will be assigned the value $h*(a+b)/2$ since this is what was returned by the algorithm. Those who know *Java*, *C*, *C++*, or just about any other programming language should already be familiar with this concept.

60 Definition The symbol $\leftarrow$ is read "gets" and it is used to denote *assignment* of value.

61 Example The statement $x \leftarrow a+b$ means "assign to x the value of a plus the value of b."

☞ *Most modern programming languages use* $=$ *for assignment. Other symbols used include* $:=$, $=:$, $<<$, *and dozens more.*

As it turns out, the most common symbol for assignment (=) is perhaps the most confusing for someone who is first learning to program. One of the most common assignment statements is $x = x+1$;. What this means *is “assign to the x its current value plus one.” However, what it* looks like *is the mathematical statement “x is equal to $x+1$”, which is false for every value of x. If this has tripped you up in the recent past or still does, fear not. Eventually you will instinctively interpret it correctly, probably forgetting you were ever confused by it.*

62 Example (Swapping variables) Write an algorithm that will interchange the values of two variables, x and y. That is, the contents of x becomes that of y and vice-versa.

Solution: We introduce a temporary variable t in order to store the contents of x in y without erasing the contents of y:

Algorithm 2.1.2: SWAP(x,y)

```
{ $t \leftarrow x$      comment: First store $x$ in a temporary place
{ $x \leftarrow y$      comment: $x$ now has $y$'s original value
{ $y \leftarrow t$      comment: $y$ now has $x$'s original value
```

63 Example Prove that the algorithm in Example 62 works correctly.

Proof: Assume the values a and b are passed into Swap. Then at the beginning of the algorithm, $x = a$ and $y = b$. We need to prove that after the algorithm is finished, $x = b$ and $y = a$. After the first line, x and y are unchanged and $t = a$ since it was assigned the value stored in x, which is a. After the second line, $x = b$ since is is assigned the value stored in y, which is b. Currently $x = b$, $y = b$, and $t = a$. Finally, after the third line, $y = a$ since it is assigned the value stored in t, which is a. Since x is still b, and $y = a$, the algorithm works correctly. □

☞ *The correctness of this algorithm, as well as some of the other algorithms in this chapter, is based on the assumption that the variables are* passed by reference *rather than* passed by value.

In C and C++, it is possible to pass by value or by reference. The * *or* & *you sometimes see in argument lists is related to this. In Java, everything is passed by value and it is impossible to pass by reference. However, because non-primitive variables in Java are essentially reference variables (that is, they store a reference to an object, not the object itself), in some ways they act as if they were passed by reference. This is where things start to get complicated. I don't want to get into the subtleties here, especially since there are arguments about whether or not these are the best term to use, etc. Let me give an analogy instead.*[1]

If I share a Google Doc with you, I am passing by reference. We both have a reference to the same document. If you change the document, I will see the changes. If I change the document, you will see the changes. On the other hand, if I e-mail you a Word document, I am passing by value. You have a copy of the document I have. Essentially, I copied the current value *(or contents) of the document and gave that*

[1] Inspired by a response on http://stackoverflow.com/questions/373419/whats-the-difference-between-passing-by-reference-vs-passing-by-value

to you. If you change the document, my document will remain unchanged. If I change my document, your document will remain unchanged. This sounds pretty simple. However, it gets more complicated. In Java, you can create a "primitive" Word document, but in a sense you can't create an "object" Word document. Instead, a Google Doc is created and you are given access (i.e. a reference) to it. This is why in some ways primitive and object variables seem to act differently in Java.

O.K., I've already said too much. The bottom line is this: The assumption being made in the various swap algorithms is that when a variable is passed in, the algorithm has direct access to that variable *and not just a copy of it. Thus if changes are made to that variable in the algorithm, it is changing the variable that was passed in. As it turns out, this subtlety does not matter for most of the algorithms here.*

64 Example (Incorrect swap) Why is following approach to implement swap incorrect?

```
Algorithm 2.1.3: SwapWrong($x$,$y$)

  $x \leftarrow y$
  $y \leftarrow x$
```

Solution: To see why this doesn't work, notice that if we pass in a and b, then $x = a$ and $y = b$ at the beginning. After the first line, $x = b$ and $y = b$. After the second line $x = b$ and $y = b$. The problem is that the first line overwrites the value stored in x, and we can't recover it.

65 Example (Swapping variables 2) Write an algorithm that will interchange the values of two variables x and y *without introducing a third variable.*

Solution: The idea is to use sums and differences to store the values. Assume that initially $x = a$ and $y = b$.

```
Algorithm 2.1.4: Swap2($x$,$y$)

  $x \leftarrow x+y$      comment: $x = a+b$ and $y = b$.
  $y \leftarrow x-y$      comment: $y = a+b-b = a$ and $x = a+b$.
  $x \leftarrow x-y$      comment: $y = a$ and $x = a+b-a = b$.
```

Notice that *this will only work for numeric variables*, and that it won't always work for real numbers. We leave the details to the reader, but consider what happens if $x = 10,000,000,000$ and $y = .00000000001$, for instance. Also notice that the comments in the code essentially provide a proof that the algorithm is correct.

66 Example It was mentioned that the comments in the algorithm from Example 65 provide a proof of its correctness. What assumption is being made that might be incorrect?

Solution: It is assumed that the equations are exactly correct. For instance, after the first line we are told that $x = a+b$. However, depending on the data type and exact value, it may be the case that x is not *exactly* $a+b$, in which case the algorithm will fail.

2.2 If-then-else Statements

67 Definition The **If-then-else** control statement has the following syntax:

$$\begin{aligned}&\textbf{if } \mathit{expression}\\ &\quad\textbf{then } \begin{cases} \mathit{statementA}_1\\ \vdots\\ \mathit{statementA}_n \end{cases}\\ &\quad\textbf{else } \begin{cases} \mathit{statementB}_1\\ \vdots\\ \mathit{statementB}_m \end{cases}\end{aligned}$$

and evaluates as follows. If expression is true then all statementA's are executed. Otherwise all statementB's are executed.

68 Example (Maximum of 2 Numbers) Write an algorithm that will determine the maximum of two numbers. Prove your algorithm is correct.

Solution: Here is a possible approach.

```
Algorithm 2.2.2: MAX($x, y$)

if $x \geq y$
  then return ($x$)
  else return ($y$)
```

If x is the maximum or $x = y$, then $x \geq y$, so the algorithms returns x, the correct answer. If y is the maximum and $y \neq x$, then $y > x$ and the algorithm returns y, which is clearly also correct.

69 Example (Maximum of 3 Numbers) Write an algorithm that will determine the maximum of three numbers. Prove that your algorithm is correct.

Solution: Here is a possible approach using the preceding function.

```
Algorithm 2.2.3: MAX3($x, y, z$)

$w = Max(x, y)$
return ($Max(w, z)$)
```

We will use a proof by cases. If x is the maximum, then $w =$MAX$(x, y) = x$. so it returns MAX$(w, z) = x$, which is correct. If y is the maximum, the argument is the same. If z is the maximum, then w is either x or y, but in either case $w \leq z$, so it returns MAX$(w, z) = z$.

70 Example (Compound Test) Write an algorithm that prints "Hello" if one enters a number between 4 and 6 (inclusive) and "Goodbye" otherwise. You are not allowed to use any boolean operators like **and**, **or**, etc.

Solution: Here is a possible answer.

```
Algorithm 2.2.4: HELLOGOODBYE($x$)

if $x >= 4$
  then { if $x <= 6$
           then output (Hello.)
           else output (Goodbye.)
  else output (Goodbye.)
```

2.3 The `for` loop

71 Definition The **for** loop has either of the following syntaxes:[2]

```
for indexVariable ← lowerValue to upperValue
  do statements
```

or

```
for indexVariable ← upperValue downto lowerValue
  do statements
```

Here *lowerValue* and *upperValue* must be non-negative integers with $lowerValue \leq upperValue$. In both cases, the code in the loop is executed for every value from *lowerValue* to *upperValue*, in the first case in that order, and in the second case in the reverse order.

72 Example (Factorial Integers) Write an algorithm that given an arbitrary non-negative integer n outputs $n!$.

Solution: Here is a possible answer.

```
Algorithm 2.3.3: FACTORIAL($n$)

comment: Must input an integer $n \geq 0$.

$f \leftarrow 1$
if $n = 0$
  then return ($f$)
  else { for $i \leftarrow 1$ to $n$
           do $f \leftarrow f * i$
return ($f$)
```

73 Example (Positive Integral Powers 1) Write an algorithm that will compute x^n, where x is a given real number and n is a given positive integer.

Solution: We can approach this problem as we did the factorial function in example 72. Thus one possible answer would be

[2] The syntax in C, C++, and Java is slightly different and makes the **for** loop much more powerful than the one presented here.

```
Algorithm 2.3.4: POWER1($(x,n)$)

power $\leftarrow 1$
for $i \leftarrow 1$ to $n$
  do power $\leftarrow x *$ power
return (power)
```

2.4 Arrays

74 Definition An **array** is an aggregate of homogeneous types. The *length of the array* is the number of entries it has.

A 1-dimensional array is akin to a mathematical vector. Thus if X is 1-dimensional array of length n then

$$X = (X[0], X[1], \ldots, X[n-1]).$$

We will follow the convention of common languages like Java, C, and C++ by indexing the arrays from 0. We will always declare the length of the array at the beginning of a code fragment by means of a comment.

A 2-dimensional array is akin to a mathematical matrix. Thus if Y is a 2-dimensional array with 2 rows and 3 columns then

$$Y = \begin{bmatrix} Y[0][0] & Y[0][1] & Y[0][2] \\ Y[1][0] & Y[1][1] & Y[1][2] \end{bmatrix}.$$

75 Example (Maximum of n Numbers) Write an algorithm that determines the maximum element of a 1-dimensional array of n elements.

> **Solution:** We declare the first value of the array (the 0-th entry) to be the maximum (a *sentinel value*). Then we successively compare it to other $n-1$ entries. If an entry is found to be larger than it, that entry is declared the maximum.

```
Algorithm 2.4.1: MAXENTRYINARRAY($n, X$)

comment: $X$ is an array of length $n$.

max $\leftarrow X[0]$
for $i \leftarrow 1$ to $n-1$
  do { if $X[i] >$ max
     {   then max $= X[i]$
return (max)
```

76 Definition (Floor and Ceiling Functions) The *floor* of a real number x, written $\lfloor x \rfloor$, is the largest integer that is less than or equal to x. The *ceiling* of a real number x, written $\lceil x \rceil$, is the smallest integer that is greater than or equal to x.

77 Example $\lfloor 4.5 \rfloor = 4$, $\lceil 4.5 \rceil = 5$, $\lfloor 7 \rfloor = \lceil 7 \rceil = 7$. In general, if n is an integer, then $\lfloor n \rfloor = \lceil n \rceil = n$.

78 Theorem Let a be an integer and x be a real number. Then $a \leq x$ if and only if $a \leq \lfloor x \rfloor$.

Proof: If $a \leq \lfloor x \rfloor$, then $a \leq \lfloor x \rfloor \leq x$ is clear. On the other hand, assume $a \leq x$. Then a is an integer that is less than or equal to x. Since $\lfloor x \rfloor$ is the largest integer that is less than or equal to x, $a \leq \lfloor x \rfloor$. □

The floor function is important because in many programming languages, including Java, C, and C++, integer division *truncates*. That is, when you compute n/k for integer n and k, the result is rounded down to $\lfloor n/k \rfloor$. In light of this, the following Corollary is sometimes important to remember.

79 Corollary Let a, b, and c be integers. Then $a \leq b/c$ if and only if $a \leq \lfloor b/c \rfloor$.

Proof: Since b/c is a real number, this is just a special case of Theorem 78. □

80 Example (Reversing an Array) An array $(X[0],\ldots X[n-1])$ is given. Without introducing another array, put its entries in reverse order. Prove that your algorithm is correct.

Solution: Observe that we want to exchange the first and last element, the second and second-to-last element, etc. That is, we want to exchange $X[0] \leftrightarrow X[n-1]$, $X[1] \leftrightarrow X[n-2]$, $\ldots, X[k] \leftrightarrow X[n-k-1]$. But what value of k is correct? If we go all the way to $n-1$, the result will be that every element is swapped and then swapped back, so we will accomplish nothing. Hopefully you can see that if we swap elements when $k < n-k-1$, we will swap each element at most once. The "at most once" is because if the array has an odd number of elements, the middle element occurs when $k = n-k-1$, but we can skip it since it doesn't need to be swapped with anything. Notice that $k < n-k-1$ if and only if $2k < n-1$. Since k and n are integers, this is equivalent to $2k \leq n-2$. This is equivalent to $k \leq \lfloor (n-2)/2 \rfloor$ by Corollary 79. Thus, we need to swap the elements $0, 1, \ldots, \lfloor (n-2)/2 \rfloor$ with elements $n-1, n-2, \ldots, n-1-\lfloor (n-2)/2 \rfloor = n - \lfloor n/2 \rfloor$, respectively. The following algorithm implements this idea, using the swapping algorithm from example 62.

```
Algorithm 2.4.2: REVERSEARRAY($n$,$X$)

  comment: $X$ is an array of length $n$.

  for $i \leftarrow 0$ to $\lfloor (n-2)/2 \rfloor$
    do Swap($X[i]$,$X[n-i-1]$)
```

Hopefully the previous example helps you realize that you need to be careful when working with arrays. Formulas related to array indices change depending on whether arrays are indexed starting at 0 or 1. In addition, formulas involving the number of elements in an array can be tricky, especially when the formulas relate to partitioning the array into pieces (e.g. into two halves). These can both lead to the so-called "off by one" error that is common in computer science. The next example illustrates these problems, and one way to deal with it.

81 Example Give a formula for the index of the middle element of an array of size n.

Solution: Clearly the answer should be somewhere close to $n/2$. Unfortunately, if n is odd, $n/2$ isn't an integer. And clearly the answer won't be the same when indexing starting at both 0 and 1. Maybe we should try a few concrete examples.

Let's first assume indexing starts at 1. If $n = 9$, the middle element is the 5th element, which has index $5 = \lceil 9/2 \rceil$. If $n = 10$, the middle element is the 5th or 6th element. Let's go with the 5th element. Then the index is $5 = 10/2 = \lceil 10/2 \rceil$. Thus the formula $\lceil n/2 \rceil$ should work. You should plug in a few more values to convince yourself that this is correct.

Now let's assume indexing starts at 0. There are a a few equivalent formulas we can come up with. For starters, $\lceil n/2 \rceil - 1$ should work since this is just 1 less than the answer above, and the indices are all shifted by one. But let's come up with a formula from scratch. If $n = 9$, the index of the middle element is $4 = \lfloor 9/2 \rfloor$. If $n = 10$, the index is $4 \neq \lfloor 10/2 \rfloor$. So $\lfloor n/2 \rfloor$ works when n is odd, but not when n is even. This one is not as obvious as it was when we started indexing at 1. With a little thought, you may realize that $\lfloor (n-1)/2 \rfloor$ works.

Now you should ask yourself: Is $\lceil n/2 \rceil - 1 = \lfloor (n-1)/2 \rfloor$ for all values of n? If not, one of our formulas is incorrect. You should convince yourself that these are indeed equal.

82 Definition (Boolean Variable) A *boolean variable* is a variable that only accepts one of two possible values: **true** or **false**.

83 Definition (Not Operator) The **not** unary operator changes the status of a boolean variable from **true** to **false** and vice-versa.

The *not* operator is essentially the same thing as the *negation* we discussed earlier. The difference is context—we are applying *not* to a boolean variable, whereas we applied *negation* to a statement.

84 Example (The Locker-room Problem) A locker room contains n lockers, numbered 1 through n. Initially all doors are open. Person number 1 enters and closes all the doors. Person number 2 enters and opens all the doors whose numbers are multiples of 2. Person number 3 enters and toggles all doors that are multiples of 3. That is, he closes them if they are open and opens them if they are closed. This process continues, with person i toggling each door that is a multiple of i. Write an algorithm to determine which lockers are closed when all n people are done.

Solution: Here is one possible approach. We use a boolean array `Locker` of size $n+1$ to denote the lockers (we will ignore `Locker[0]`). The value **true** will denote an open locker and the value **false** will denote a closed locker.[3]

```
Algorithm 2.4.3: LOCKERROOMPROBLEM($n$, $Locker$)

comment: $Locker$ is an array of size $n+1$.

comment: Closing all lockers in the first for loop.

for $i \leftarrow 1$ to $n$
  do $Locker[i] \leftarrow$ false
comment: From open to closed and vice-versa in the second loop.

for $j \leftarrow 2$ to $n$
  do { for $k \leftarrow j$ to $n$
         do if $k \bmod j = 0$
         then $Locker[k] =$ not $Locker[k]$
for $l \leftarrow 1$ to $n$
  do { if $Locker[l] =$ false
       then output (Locker $l$ is closed.)
```

[3] We will later see that those locker doors whose numbers are squares are the ones which are closed.

2.5 The `while` loop

85 Definition The **while** loop has syntax:

```
while test
  do {body of loop
```

The commands in the body of the loop will be executed as long as `test` evaluates to true.

86 Example (Different Elements in an Array) An array X satisfies $X[0] \leq X[1] \leq \cdots \leq X[n-1]$. Write an algorithm that finds the number of entries which are different.

Solution: Here is one possible approach.

```
Algorithm 2.5.2: DIFFERENT($n, X$)

comment: $X$ is an array of length $n$.
$i \leftarrow 0$
different $\leftarrow 1$
while $i \neq n-1$
    do { $i \leftarrow i+1$
         if $x[i] \neq x[i-1]$
           then different $\leftarrow$ different $+ 1$
return (different)
```

87 Definition Recall that a positive integer $p > 1$ is a *prime* if its only positive factors are 1 and p. A positive integer $c > 1$ which is not prime is said to be *composite*.[4]

88 Theorem Let $n > 1$ be a positive integer. Either n is prime or n has a prime factor $\leq \sqrt{n}$.

Proof: If n is prime there is nothing to prove. Assume that n is composite. Then n can be written as the product $n = ab$ with $1 < a \leq b$, where a and b are integers. If every prime factor of n were $> \sqrt{n}$, then $a > \sqrt{n}$ and $b > \sqrt{n}$. But then $n = ab > \sqrt{n}\sqrt{n} = n$, which is a contradiction. Thus n must have a prime factor $\leq \sqrt{n}$. □

89 Example To determine whether 103 is prime we proceed as follows. Observe that $\lfloor\sqrt{103}\rfloor = 10$. We now divide 103 by every prime ≤ 10. If one of these primes divides 103, then 103 is not a prime. Otherwise, 103 is a prime. Notice that 103 mod $2 = 1$, 103 mod $3 = 1$, 103 mod $5 = 3$, and 103 mod $7 = 5$. Since none of these remainders is 0, 103 is prime.

90 Example (Eratosthenes' Primality Testing) Give an algorithm to determine whether a given positive integer n is prime.

Solution: We first deal with a few base cases. If $n = 1$, it is not prime, and if $n = 2$ or $n = 3$ it is prime. Then we determine if n is even, in which case it is not prime. Finally, we loop through all of the odd values, starting with 3 and going to $\sqrt{n}$, determining whether or not n is a multiple of any of them. If so, it is not prime. If we get through all of this, then n has no factors less than or equal to $\sqrt{n}$ which means it must be prime. Here is the algorithm based on this description.

[4] Thus 1 is neither prime nor composite.

```
Algorithm 2.5.3: IsPrime($n$)

if $n = 1$ output ($n$ is a unit.)
if $n = 2$ output ($n$ is prime.)
if $n = 3$ output ($n$ is prime.)
if $n > 3$
  then
    if $n \bmod 2 = 0$
      then output ($n$ is even. Its smallest factor is 2.)
      else
        flag ← true
        $i \leftarrow 1$
        while $i \leq \lfloor\sqrt{n}\rfloor$ and flag = true
          do
            $i \leftarrow i + 2$
            if $n \bmod i = 0$
              then flag ← false
        if flag = true
          then output ($n$ is prime.)
          else output (Not prime. Smallest factor is $i$.)
```

It should be noted that although this algorithm works, it is not very practical for large values of n. In fact, there is no known algorithm that can factor numbers efficiently on a "classical" computer. The most commonly used public-key cryptosystems rely on the assumption that there is no efficient algorithm to factor a number. However, if you have a quantum computer, you are in luck. Shor's algorithm actually *can* factor numbers efficiently.

Exercises

91 Problem What will the following algorithm return for $n = 5$? You must trace the algorithm carefully, outlining all your steps.

```
Algorithm 2.5.4: Mystery($n$)

$x \leftarrow 0$
$i \leftarrow 1$
while $n > 1$
  do
    if $n * i > 4$
      then $x \leftarrow x + 2n$
      else $x \leftarrow x + n$
    $n \leftarrow n - 2$
    $i \leftarrow i + 1$
return ($x$)
```

92 Problem Assume that the division operator $/$ acts as follows on the integers: if the division is not even, a/b truncates the decimal part of the quotient (This is how it works in Java, C, C++, and many other languages). For example $5/2 = 2$, $5/3 = 1$. Write an algorithm that reverses the digits of a given integer that exploits this fact. For example, if 123476 is the input, the output should be 674321. Use only one `while` loop, one mod operation, one multiplication by 10 and one division by 10.

93 Problem Write an algorithm that reads an array of n integers and finds the second smallest entry.

Answers

91 In the first turn around the loop, $n = 5, i = 1$, $n * i > 4$ and thus $x = 10$. Now $n = 3$, $i = 2$, and we go a second turn around the loop. Since $n * i > 4$, $x = 10 + 2 * 3 = 16$. Finally, $n = 1, i = 3$, and the loop stops. Hence $x = 16$ is returned.

92 Here is a possible approach.

```
Algorithm 2.5.5: REVERSE($n$)

comment: $n$ is a positive integer.
$x \leftarrow 0$
while $n \neq 0$
    do {
        comment: $x$ accumulates truncated digit.
        $x \leftarrow x * 10 + n \bmod 10$
        comment: We now truncate a digit of the input.
        $n \leftarrow n/10$
    }
return ($x$)
```

93 Here is one possible approach.

```
Algorithm 2.5.6: SECONDSMALLEST($n, X$)

comment: $X$ is an array of length $n$.
second $\leftarrow x[0]$
minimum $\leftarrow$ second
for $i \leftarrow 0$ to $n - 1$
    do {
        if minimum $=$ second
            then {
                if $X[i] <$ minimum
                    then minimum $\leftarrow X[i]$
                    else second $\leftarrow X[i]$
            }
            else {
                if $X[i] <$ minimum
                    then {
                        second $\leftarrow$ minimum
                        minimum $\leftarrow X[i]$
                    }
                    else {
                        if $X[i] >$ minimum and $X[i] <$ second
                            then second $\leftarrow X[i]$
                    }
            }
    }
```

Homework

☞ *When a problem asks for an algorithm, always assume it is asking for the most efficient algorithm you can find.*

94 Problem Implement the swap operation for integers without using an additional variable and without using addition or subtraction. (Hint: bit operations)

95 Problem Prove or disprove that the following method correctly computes the maximum of two integers `x` and `y`, assuming that the `minimum` method correctly computes the minimum of `x` and `y`.

```
int maximum(int x, int y) {
    int min = minimum(x,y);
    int max = x + y - min;
    return max;
}
```

96 Problem Give a recursive algorithm that computes $n!$. You can assume $n \geq 0$.

97 Problem Although different programming languages and compilers might return different answers to the computation $a \bmod b$ when $a < 0$, they always return a value between $-(b-1)$ and $b-1$. Given that fact, give an algorithm that will always return an answer between 0 and $b-1$, regardless of whether or not a is negative. (Note: Generally speaking, $a \bmod b \neq -a \bmod b$. In other words, multiplying by -1 will *not* work.)

98 Problem Repeat the previous problem, but do not use any conditional statements.

99 Problem What will the following algorithm return for $n = 3$?

```
Algorithm 2.5.7: MYSTERY($n$)

$x \leftarrow 0$
while $n > 0$
    do {
        for $i \leftarrow 1$ to $n$
            do {
                for $j \leftarrow i$ to $n$
                    do {$x \leftarrow ij + x$
            }
        $n \leftarrow n - 1$
    }
return ($x$)
```

100 Problem Give an algorithm that will round a real number `x` to the closest integer (round up at .5). Here's the trick, though: You can *only* use `floor(y)`, `ceiling(y)`, basic arithmetic (+, -, *, /) and/or numbers. You *cannot* use if statements or anything else!

101 Problem Assuming integer division truncates, write an algorithm that will compute n/m, but will *round* the result instead of truncating it. For instance, $5/4$ should return 1, but $7/4$ should return 2 instead of 1.

102 Problem Repeat the previous problem, but do not use any conditional statements, double, floats, or the mod operator. In other words, do it using only integer arithmetic.

103 Problem Assume you have a function $random(n)$ that returns a random integer between 0 and $n-1$. Write an algorithm that returns a random number between a and b, where a and b are integers. You may only call $random(n)$ once and you may not use conditional statements.

104 Problem Assume you have a function *random*() that returns a positive random number. Write an algorithm that returns a random number between *a* and *b*, where *a* and *b* are integers. You may only call *random*() once and you may not use conditional statements.

105 Problem The following method is a simplified version of a method that might be used to implement a hash table or in a cryptographic system. Assume that for one particular use the number returned by this function has to have the opposite parity (even/odd) of the parameter. For instance, `hash_it(4)` returns 49 which has the opposite parity of 4, so it works for 4. Prove or disprove that this function always returns a value of opposite parity of the parameter.

```
int hash_it(int x) {
    return x*x+6*x+9;
}
```

106 Problem Give an algorithm that computes all of the primes that are less than or equal to *n*. For simplicity, you can just print all of the prime numbers up to *n*. Your algorithm should be as efficient as possible. One approach is to incorporate an array into the algorithm from Example 90.

107 Problem Prove or disprove that the following method computes the absolute value of `x`. For simplicity, assume that all of the calculations are performed with perfect precision. Also, `sqrt(x)` computes $\sqrt{x}$. Finally, you may use the fact that $\sqrt{x^2} = x$ when $x \geq 0$ if it will help.

```
double absoluteValue(double x) {
    double square = x*x;
    double answer = sqrt(square);
    return answer;
}
```

108 Problem Prove or disprove that the following method computes the absolute value of `x`. For simplicity, assume that all of the calculations are performed with perfect precision. Also, `sqrt(x)` computes $\sqrt{x}$. Finally, you may use the fact that $(\sqrt{x})^2 = x$ when $x \geq 0$ if it will help.

```
double absoluteValue(double x) {
    double root = sqrt(x);
    double answer = root*root;
    return answer;
}
```

109 Problem Problems 107 and 108 both assumed that "all of the calculations are performed with perfect precision". Is that a realistic assumption? Give an example of an input for which the each algorithm will work properly. Then give an example of an input for which each algorithm will *not* work properly. You can implement and run the algorithms to do some testing if you wish.

110 Problem The following method is supposed to do some computations on a positive number that result in getting the original number back. Prove or disprove that this method always returns the *exact* value that was passed in. (Note: Unlike in the previous problems, here we will assume that although a `double` stores a real number as accurately as possible, it uses only a fixed amount of space. Thus a `double` is unable to store the exact value of any irrational number–it instead stores an approximation. Also, `sqrt(x)` computes $\sqrt{x}$. You may assume that $\sqrt{2}$ is irrational if you find this fact helpful.)

```
double returnTheParameterUnmodified(double x) {
    double a = sqrt(x);
    double b = a*a;
    return b;
}
```

Chapter 3

Propositional Logic, Sets, and Boolean Algebra

In this chapter we take a look at propositional logic and sets. On the surface they seem quite different, so placing them in the same chapter may seem odd. However, the section on Boolean algebra will make it clear that sets and logic actually have a lot more in common than you might think. When discussing sets we will take a brief look at relations and equivalence relations, with particular emphasis on an important equivalence relation for many computer science applications.

3.1 Propositional Logic

111 Definition A *boolean proposition* (or simply *proposition*) is a statement which is either **true** or **false** . We call this the *truth value* of the proposition.

Whether the statement is *obviously* true or false does not enter in the definition. One only needs to know that its certainty can be established.

112 Example The following are boolean propositions and their values, if known:

❶ $7^2 = 49$. (**true**)

❷ $5 > 6$. (**false**)

❸ If p is a prime then p is odd. (**false**)

❹ There exists infinitely many primes which are the sum of a square and 1. (unknown)

❺ There is a God. (unknown)

❻ There is a dog. (**true**)

❼ I am the Pope. (**false**)

❽ Every prime that leaves remainder 1 when divided by 4 is the sum of two squares. (**true**)

❾ Every even integer greater than 6 is the sum of two distinct primes. (unknown)

113 Example The following are not boolean propositions, since it is impossible to assign a **true** or **false** value to them.

❶ Whenever I shampoo my camel.

❷ Sit on a potato pan, Otis!

❸ $y \leftarrow x$.

❹ This sentence is false.

114 Definition A *boolean operator* is used to combine one or more boolean propositions to form a new one. A proposition formed in this way is called a *compound proposition*. For convenience, we call the propositions used to form a compound proposition *variables* for reasons that should become evident shortly. We will consider the following boolean operators in these notes. The evaluation rules for each are given in Table 3.1. For each, assume p and q are propositions.

❶ The *negation* (or *NOT*) of p, denoted by $\neg p$ is the proposition "it is not the case that p". $\neg p$ is true when p is false, and vice-versa. Other notations include $\overline{p}$, $\sim p$, and $!p$ (many programming languages use this one).

❷ The *conjunction* (or *AND*) of p and q, denoted by $p \wedge q$, is the proposition "p and q". The conjunction of p and q is true when p and q are both true and false otherwise.

❸ The *disjunction* (or *OR*) of p and q, denoted by $p \vee q$, is the proposition "p or q". The disjunction of p and q is false when both p and q are false and true otherwise. Put another way, if p is true, q is true, or both are true, the disjunction is true.

❹ The *exclusive or* (or *XOR*) of p and q, denoted by $p \oplus q$, is the proposition "p is true or q is true, but not both". The exclusive or of p and q is true when exactly one of p or q is true.

❺ The *conditional statement* (or *implies*) involving p and q, denoted by $p \rightarrow q$, is the proposition "if p, then q". It is false when p is true and q is false, and true otherwise. In the statement $p \rightarrow q$, we call p the *premise* (or *hypothesis* or *antecedent*) and q the *conclusion* (or *consequence*).

❻ The *biconditional statement* involving p and q, denoted by $p \leftrightarrow q$, is the proposition "p if and only if q". It is true when p and q have the same truth value, and false otherwise.

p	q	$(\neg p)$	$(p \wedge q)$	$(p \vee q)$	$p \oplus q$	$(p \rightarrow q)$	$(p \leftrightarrow q)$
T	T	F	T	T	F	T	T
T	F	F	F	T	T	F	F
F	T	T	F	T	T	T	F
F	F	T	F	F	F	T	T

Table 3.1: Evaluation Rules

Some of these definitions should be familiar to you. When you learned about Boolean expressions in your programming courses, you probably saw at least *AND*, *OR* and *NOT*. The notation is probably different, though. In Java, C, and C++, `!` is used for NOT, && is used for AND, and `||` is used for OR. Although propositional logic and Boolean expressions have a lot in common (more on that in Section 3.3), there are some subtle differences between them, especially as implemented in many programming languages. We will see more on this later.

☞ *Notice that* $\vee$ *is an* inclusive or, *meaning that it is true if both are true, whereas* $\oplus$ *is an* exclusive or, *meaning it is false if both are true. The difference between* $\vee$ *and* $\oplus$ *is complicated by the fact that in English, the word "or" to can mean either of these depending on context. For instance, if your mother tells you "you can have cake or ice cream" she is likely using the exclusive or, whereas a prerequisite of "Math 110 or demonstrated competency with algebra" clearly has the inclusive or in mind.*

☞ *The* conditional *operator is by far the one that is the most difficult to get a handle on for at least two reasons. First, the conditional statement* $p \rightarrow q$ *is not saying anything about* p *or* q *by themselves. It is only saying that if* p *is true, then* q *has to also be true. It doesn't say anything about the case that* p *is not true. This brings us to the second reason. Should* $F \rightarrow T$ *be true or false? Although it seems counterintuitive to some, it should be true. Again,* $p \rightarrow q$ *is telling us about the value of* q *when* p *is true (i.e., if* p *is true, the* q must be *true). What does it tell us if* p *is false? Nothing. As strange as it might seem, when* p *is false, the whole statement is true regardless of the truth value of* q*.*

If in the end you are still confused, you can simply fall back on the formal definition: $p \rightarrow q$ *is false when* p *is true and* q *is false, and is true otherwise. In other words, if interpreting* $p \rightarrow q$ *as the English sentence "*p *implies* q*" is more harmful than helpful in understanding the concept, don't worry about why it doesn't make sense and just remember the definition.*[1]

115 Example Consider the propositions:

- a : I will eat my socks.
- b : It is snowing.
- c : I will go jogging.

The sentences below are represented by means of logical operators.

❶ $(b \vee \neg b) \rightarrow c$: Whether or not it is snowing, I will go jogging.

❷ $b \rightarrow \neg c$: If it is snowing, I will not go jogging.

❸ $b \rightarrow (a \wedge \neg c)$: If it is snowing, I will eat my socks, but I will not go jogging.

❹ $a \leftrightarrow c$: When I eat my socks I go jogging, and when I go jogging I eat my socks.

The operators were listed in order of precedence, with the exception that $\vee$ and $\oplus$ are swapped.[2] Also, $\neg$ has right-to-left associativity, all other operators listed have left-to-right associativity. It is important to know the precedence rules for the boolean operators (or at least be able to look it up) so you can properly interpret complex boolean expressions. However, I generally prefer to always use enough parentheses to make it immediately clear, especially when I am writing code.

116 Example According to the precedence rules, $\neg a \rightarrow a \vee b$ should be interpreted as $(\neg a) \rightarrow (a \vee b)$.

117 Example According to the precedence rules, $a \wedge \neg b \rightarrow c$ should be interpreted as $(a \wedge \neg b) \rightarrow c$.

[1] In mathematics, one tries to define things so they make sense immediately. Sometimes this is not possible (if the concept is very complicated and/or it just doesn't relate to something that is familiar). Sometimes a term or concept is defined poorly but because of prior use the definition sticks. Sometimes it makes perfect sense to some people and not to others, probably based on each person's background. I think this last possibility may be to blame in this case.

[2] Why aren't they presented in the other order? Because it makes more sense to define *OR* before defining *XOR*.

118 Example According to the precedence rules, $a \wedge b \vee c$ should be interpreted as $(a \wedge b) \vee c$, which is not the same thing as $a \wedge (b \vee c)$. To convince yourself of this, consider the case when $a = F$, $b = F$, and $c = T$.

119 Example According to the associativity rules, $a \to b \to c$ should be interpreted as $(a \to b) \to c$. It is important to note that $(a \to b) \to c$ and $a \to (b \to c)$ are *not* equivalent. It is probably worth your effort to convince yourself of this by finding an assignment of truth values for a, b, and c such that the two proposition have different truth values.

120 Example Write a code fragment that determines whether or not three numbers can be the lengths of the sides of a triangle.

Solution: Let a, b, and c be the numbers. First we must have $a > 0$, $b > 0$, and $c > 0$. Also, the sum of any two of them must be larger than the third in order to form a triangle. More specifically, we need $a + b > c$, $b + c > a$, and $c + a > b$. This leads to the following algorithm.

```
Algorithm 3.1.1: IsItATriangle($a, b, c$)

if $a > 0$ and $b > 0$ and $c > 0$ and $a + b > c$ and $b + c > a$ and $c + a > b$
  then return ( true )
  else return ( false )
```

121 Definition A *truth table* is a table that shows the truth value of a compound proposition for all possible combinations of truth assignments to the variables in the proposition. If there are n variables, the truth table will have 2^n rows.

122 Example Construct the truth table of the proposition $a \vee \neg b \wedge c$.

Solution: Since there are three variables, the truth table will have $2^3 = 8$ rows. Notice that by the precedence rules, the given proposition is equivalent to $a \vee (\neg b \wedge c)$, since $\wedge$ has higher precedence than $\vee$. The truth table is in Table 3.2.

a	b	c	$\neg b$	$\neg b \wedge c$	$a \vee (\neg b \wedge c)$
T	T	T	F	F	T
T	T	F	F	F	T
T	F	T	T	T	T
T	F	F	T	F	T
F	T	T	F	F	F
F	T	F	F	F	F
F	F	T	T	T	T
F	F	F	T	F	F

Table 3.2: Example 122.

☞ *Notice that there are several columns in the truth table besides the columns for the variables and the column for the proposition we are concerned with. These are "helper" or "intermediate" columns (those are not official definitions). Their purpose is simply to help us compute the final column more easily and without (hopefully) making any mistakes.*

☞ *As long as all possible values of the variables are included, the order of the rows of a truth table does not matter. However, by convention one of two orderings is usually used. Since there is an interesting connection to the binary representation of numbers, we will discuss it further.*

123 Example (Ordering the rows of a Truth Table) Notice that the values of the variables can be thought of as the index of the row. So if a proposition involves two variables, the values in the first two columns are used as a sort of index. We can order the rows by assigning a number to each row based on the values in these columns. The order used here essentially computes an index as follows: For the "index" columns, think of each T as a 0 and each F as a 1. Now treat the numbers in these columns as binary numbers and order the rows appropriately. For instance, if there are three variables, we can think of it as shown in the Table 3.3.

a	b	c				index
T	T	T	0	0	0	0
T	T	F	0	0	1	1
T	F	T	0	1	0	2
T	F	F	0	1	1	3
F	T	T	1	0	0	4
F	T	F	1	0	1	5
F	F	T	1	1	0	6
F	F	F	1	1	1	7

Table 3.3: Ordering the rows (Example 123)

The other common ordering does the same thing, but maps T to 1 and F to 0.

There is also a way of thinking about this recursively. That is, given an ordering for a table with n variables, we can compute an ordering for a table with $n+1$ variables. It works as follows: Make two copies of the columns corresponding to the variables, appending a T to the beginning of the first copy, and an F to the beginning of the second copy.

124 Definition A compound proposition that is always true is called a *tautology*. One that is always false is a *contradiction*. Finally, one that is neither of these is called a *contingency*.

3.1.1 Propositional Equivalence

125 Definition We want to define what it means for two propositions to be *equivalent* (or *logically equivalent*). Here are three equivalent definitions:

❶ Propositions p and q are equivalent if they have the same truth table.

❷ Propositions p and q are equivalent if the proposition $p \leftrightarrow q$ is a tautology.

❸ Propositions p and q are equivalent if they have the same truth value for all assignments of truth values to the variables.

When p and q are equivalent, we write $p = q$. One alternative notation for this is $p \equiv q$.

☞ *$p = q$ is* not *a compound proposition. Rather it is a statement about the relationship between two propositions.*

There are many logical equivalences (or *identities*) that come in handy when working with compound propositions. Many of them (e.g. commutative laws, associative laws, distributive laws) will resemble the arithmetic laws you learned in grade school. Others are very different. We will give just a few examples here. We will see many more in Section 3.3 when we discuss the relationship between propositional logic and Boolean algebras.

126 Theorem (Double Negation) $\neg(\neg a) = a$.

Proof: Table 3.4 shows the truth table for a and $\neg(\neg a)$. Since the entries for both a and $\neg(\neg a)$ are the same for every row, $\neg(\neg a) = a$. □

a	$\neg a$	$\neg(\neg a)$
T	F	T
F	T	F

Table 3.4: Theorem 126.

127 Theorem (De Morgan's Laws) $\neg(a \vee b) = \neg a \wedge \neg b$ and $\neg(a \wedge b) = \neg a \vee \neg b$.

Proof: We can prove both of these by showing that in each case, both expression have the same truth table. Table 3.5 proves that $\neg(a \vee b) = \neg a \wedge \neg b$, and Table 3.6 proves that $\neg(a \wedge b) = \neg a \vee \neg b$. (Notice that the gray columns, which correspond to the expressions of interest, are the same in each case.) □

a	b	$a \vee b$	$\neg(a \vee b)$	$\neg a$	$\neg b$	$\neg a \wedge \neg b$
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

Table 3.5: $\neg(a \vee b) = \neg a \wedge \neg b$.

a	b	$a \wedge b$	$\neg(a \wedge b)$	$\neg a$	$\neg b$	$\neg a \vee \neg b$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

Table 3.6: $\neg(a \wedge b) = \neg a \vee \neg b$.

Although truth tables can be used to prove that two propositions are equivalent, it is not enough to just give a truth table. You should also include a statement like "since p and q have the same truth table, $p = q$."

128 Example Simplify $\neg(A \vee \neg B)$.

Solution: Using DeMorgan's Law and double negation: $\neg(A \vee \neg B) = \neg A \wedge \neg(\neg B) = \neg A \wedge B$.

129 Example Simplify the following code as much as possible.

```
if ( !(a==null || a.size()<=0 ) ) {
    a.clear();
}
```

Solution: First, notice that by DeMorgan's Law, `!(a==null || a.size()<=0 )` is equivalent to `!(a==null) && !(a.size()<=0)`. Simplifying a bit more, we get `a!=null && a.size()>0`. Thus, the code becomes:

```
if (a!=null && a.size()>0) {
    a.clear();
}
```

This may not look much simpler, but it is much easier to understand.

Identities involving $\to$ and $\oplus$ are important in the context of programming since these operators are not always present in a programming language. In these cases, there is a need to express a compound proposition in terms of the operators that are present.

130 Example Express $p \oplus q$ using only $\vee$, $\wedge$, and/or $\neg$.

Solution: Notice that if p is true then q must be false, which we represent as $p \wedge \neg q$. Similarly if q is true, p must be false and we must have $\neg p \wedge q$. In either of these cases, and only these cases, the expression is true. Thus, we have that

$$p \oplus q = (p \wedge \neg q) \vee (\neg p \wedge q).$$

☞ *There is an important difference between the logical operators as discussed here and how they are implemented in programming languages such as Java, C, and C++. It is something that is sometimes called* short circuiting. *You are probably familiar with the concept even if you haven't heard it called that before. It exploits the* domination laws:

$$F \wedge q = F$$

$$T \vee q = T$$

Let's see an example.

131 Example Consider the statement `if(x>=0 && a[x]!=0)`. The first domination law implies that when $x < 0$, the expression in the if statement will evaluate to false regardless of the truth value of `a[x]!=0`. Therefore, many languages will simply not evaluate the second part of the expression—they will *short circuit*. The same thing happens for statements like `if(x<0 || x>a.length)` when $x >= 0$, for instance.

There are at least two benefits of this. First, it is more efficient since sometimes less code needs to be executed. Second, it allows the checking of one condition before checking a second condition that might cause a crash. You have probably used it in statement like the above to make sure you don't index outside the bounds of an array. Another use is to avoid attempting to access methods or fields when a variable refers to null (e.g. `if(a!=null && a.size()>0)`).

But this has at least two consequences that can cause subtle problems if you aren't careful. First, although the AND and OR operators are *commutative* (e.g. $p \vee q$ and $q \vee p$ are equivalent), that is not always the case for Boolean expressions in these languages. For instance, the statement `if(x>=0 && a[x]!=0)` is not equivalent to `if(a[x]!=0 && x>=0)` since the second one will cause a crash if $x < 0$. Second, if the second part of the expression is code that you expect will always be executed, you may spend a long time tracking down the bug that this creates.

3.1.2 Predicates and Quantifiers

132 Definition A *predicate* or *propositional function* is a statement containing one or more variables, whose truth or falsity depends on the value(s) assigned to the variable(s).

Recall that the symbol $\forall$ is the *universal quantifier*, and it is read as “for all”, “for each”, or “for every”; and the symbol $\exists$ is the *existential quantifier*, and it is read as “there exists”, “there is”, or “for some”.

133 Example Let $P(x)$=“$x < 0$”. Then $P(x)$ is a propositional function, and $\forall xP(x)$ means “all values of x are negative.” If the domain is $\mathbb{Z}$, $\forall xP(x)$ is false. However, if the domain is negative integers, it is true.

134 Example Let $P(x)$=“$x < 0$”. Then $\neg\forall xP(x)$ means “it is not the case that all values of x are negative.” Put more simply, it means “some value of x is positive”, which we can write as $\exists x\neg P(x)$.

What we saw in the last example actually holds for any propositional function.

135 Theorem (DeMorgan’s Laws for quantifiers) If $P(x)$ is a propositional function, then

$$\neg\forall xP(x) = \exists x\neg P(x), \text{ and}$$

$$\neg\exists xP(x) = \forall x\neg P(x).$$

Proof: We will prove the first statement. The proof of the other is very similar. Notice that $\neg\forall xP(x)$ is true if and only if $\forall xP(x)$ is false. $\forall xP(x)$ is false if and only if there is at least one x for which $P(x)$ is false. This is true if and only if $\neg P(x)$ is true for some x. But this is exactly the same thing as $\exists x\neg P(x)$, proving the result. □

136 Example If you want to determine whether or not something (e.g. $P(x)$) is true for all values in a domain (e.g., you want to determine the truth value of $\forall xP(x)$), one method is to simply loop through all of the values and test whether or not $P(x)$ is true. If it is false for any value, you know the answer is false. If you test them all and none of them were false, you know it is true. Here is how you might determine if $\forall xP(x)$ is true or false for the domain $\{0,1,2,\ldots,99\}$:

```
boolean isTrueForAll() {
    for(int i=0;i<100;i++) {
        if( !P(i) ) {
            return false;
        }
    }
    return true;
}
```

Notice the negation in the code—this can trip you up if you aren’t careful. The following two methods implement this idea for two predicates, $P(x)$ and $Q(x)$, again for the domain $\{0,1,2,\ldots,99\}$.

```
boolean isTrueForAll2() {
    for(int i=0;i<100;i++) {
        if( !P(i) && !Q(i) )
            return false;
    }
    return true;
}
```

```
boolean isTrueForAll3() {
    boolean result = true;
    for(int i=0;i<100;i++) {
        if(!P(i)))
            result = false;
    }
    if(result==true)
        return true;
    for(int i=0;i<100;i++) {
        if(!Q(i)))
            return false;
    }
    return true;
}
```

❶ What is `isTrueForAll2` determining? Notice that if both $P(i)$ and $Q(i)$ are false for the same value of i, it returns false, and otherwise it returns true. Put another way, it returns true if for every value of i, either $P(i)$ or $Q(i)$ is true. Thus, `isTrueForAll2` is determining the truth value of $\forall i(P(i) \vee Q(i))$.

❷ What is `isTrueForAll3` determining? First notice that if $P(i)$ is true for every value of i, `result` will be true at the end of the first loop, so `isTrueForAll3` will return true without even considering Q. However, if $P(i)$ is false for any value of i, then it will go onto the second loop. The second loop will return false if $Q(i)$ is false for any value of i. But if $Q(i)$ is true for all values of i, the method returns true. So, how do we put this all together into a simple answer? Notice that the only time it returns true is if either $P(i)$ is always true or if $Q(i)$ is always true. In other words, `isTrueForAll3` is determining the truth value of $\forall iP(i) \vee \forall iQ(i)$.

❸ Now the million dollar question:[3] Are `isTrueForAll2` and `isTrueForAll3` determining the same thing? At first glance, it looks like they might be. But we need to dig deeper, and we need to prove one way or the other. To prove it, we would need to show that these expressions evaluate to the same truth value, regardless of what P and Q are. To disprove it, we just need to find a P and a Q for which these expressions have different truth values. But let's first talk it through to see if we can figure out which way we should go with it.

$\forall i(P(i) \vee Q(i))$ is saying that for every value of i, either $P(i)$ or $Q(i)$ has to be true. $\forall iP(i) \vee \forall iQ(i)$ is saying that either $P(i)$ has to be true for every i, or that $Q(i)$ has to be true for every i. These sound similar, but not exactly the same, so we cannot be sure yet. In particular, we cannot jump to the conclusion that they are not equivalent because we described each with different words. There are many ways of wording the same concept.

At this point we either need to try to tweak the wording so that we can see that they are really saying the same thing, or we need to try to convince ourselves they aren't. Let's try the latter. What if $P(i)$ is always true and $Q(i)$ is always false? Then both statements are true. But that doesn't necessarily mean it is always true, so that doesn't help. Let's consider this: What if we can find a P and a Q such that for any given value of i, we can ensure that either $P(i)$ or $Q(i)$ is true, but also that there is some value j such that $P(j)$ is false and some value k such that $Q(k)$ is false? Then $\forall i(P(i) \vee Q(i))$ would be true, but $\forall iP(i) \vee \forall iQ(i)$ false, so this would work. But in order to be certain, we have to

[3] There is no million dollars for answering this question. It's just an expression.

know that such a P and Q exist.[4] What if we let $P(i) =$"i is even", $Q(i) =$"i is odd", and the universe be $\mathbb{N}$. Then $\forall i P(i) = \forall i Q(i) =$ **false** , so $\forall i P(i) \vee \forall i Q(i) =$ **false** , but $\forall i (P(i) \vee Q(i)) =$ **true** .

Exercises

137 Problem Construct the truth table for $(p \to q) \wedge q$.

138 Problem By means of a truth table, decide whether $(p \wedge q) \vee (\neg p) = p \vee (\neg p)$. That is, you want to compare the outputs of $(p \wedge q) \vee (\neg p)$ and $p \vee (\neg p)$.

139 Problem Explain whether the following assertion is true and negate it without using the negation symbol $\neg$:

$$\forall n \in \mathbb{N}\ \exists m \in \mathbb{N}\ \left(n > 3 \to (n+7)^2 > 49 + m\right)$$

Answers

137

p	q	$p \to q$	$(p \to q) \wedge q$
F	F	T	F
F	T	T	T
T	F	F	F
T	T	T	T

138 The desired truth table is

p	q	$p \wedge q$	$\neg p$	$p \vee \neg p$	$(p \wedge q) \vee (\neg p)$
F	F	F	T	T	T
F	T	F	T	T	T
T	F	F	F	T	F
T	T	T	F	T	T

139 The assertion is true. We have

$$(n+7)^2 > 49 + m \leftrightarrow n^2 + 14n > m.$$

Hence, taking $m = n^2 + 14n - 1$ for instance (or any smaller number), will make the assertion true.

3.2 Sets

140 Definition By a *set* we will understand any well-defined collection of objects. These objects are called the *elements* of the set. If a belongs to the set A, then we write $a \in A$, read "a is an element of A." If a does not belong to the set A, we write $a \not\in A$, read "a is not an element of A." Generally speaking, repeated elements in a set or ignored.

141 Definition The number of elements in a set A, also known as the the *cardinality* of A, will be denoted by $\operatorname{card}(A)$ or $|A|$. If the set A has infinitely many elements, we write $|A| = \infty$.

[4] Consider this: If I can find an even number that is prime but is not 2, then there would be at least 2 even primes. That's great. Unfortunately, I can't find such a number.

142 Example Let $D = \{0,1,2,3,4,5,6,7,8,9\}$ be the set of the ten decimal digits. Then $4 \in D$ but $11 \notin D$. Also, $|D| = 10$.

Notice that the elements in a set are listed between curly braces. You should do the same when you specify the elements of a set.

143 Example The prime numbers less than 10 are 2, 3, 5, and 7. If you are asked to list the prime numbers less than 10, an appropriate answer would be $2,3,5,7$. However, if you are asked for the set of prime numbers less than 10, the answer is $\{2,3,5,7\}$.

144 Example The sets $\{1,2,3\}$ and $\{1,1,1,2,2,3\}$ actually represent the same set since repeated values are essentially ignored. The cardinality of both sets is 3.

☞ *We will normally denote sets by capital letters, say $A, B, S, \mathbb{N}$, etc. Elements will be denoted by lowercase letters, say a, b, ω, r, etc.*

145 Definition The following notation is pretty standard, and we will follow it in this book.

$\mathbb{N} = \{0,1,2,3,\ldots\}$	the set of *natural numbers*.
$\mathbb{Z} = \{\ldots -2,-1,0,1,2,\ldots\}$	the set of *integers*.
$\mathbb{Z}^+ = \{1,2,3,\ldots\}$	the set of *positive integers*.
$\mathbb{Z}^- = \{-1,-2,-3,\ldots\}$	the set of *negative integers*.
$\mathbb{R}$	the *real numbers*.
$\mathbb{C}$	the *complex numbers*.
$\varnothing = \{\}$	the *empty set* or *null set*.

☞ *There is no universal agreement of the definition of $\mathbb{N}$. Although here it is defined as $\{0,1,2,3,\ldots\}$, it is sometimes defined as $\mathbb{N} = \mathbb{Z}^+$. I prefer the definition given here because then we have a notation for the positive integers ($\mathbb{Z}^+$) as well as the non-negative integers ($\mathbb{N}$).*

146 Example Notice that $|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{R}| = \infty$. But this may be a bit misleading. Do all of these sets have the same number of elements? Believe it or not, it turns out that $\mathbb{N}$ and $\mathbb{Z}$ do, but that $\mathbb{R}$ has many more elements than both of these. If it seems strange to talk about whether or not two infinite sets have the same number of elements, don't worry too much about it. We probably won't bring it up again.

147 Example Let S be the set of the squares of integers. We can express this set in what we call *set builder notation*. In this case we can write it as $S = \{n^2 | n \in \mathbb{Z}\}$ (or $S = \{n^2 : n \in \mathbb{Z}\}$). We read the : or | as "such that". Thus, S is the set containing "numbers of the form n^2 such that n is an integer".

148 Definition If every element in A is also in B, we say that A is a *subset* of B and we write this as $A \subseteq B$. If $A \subseteq B$ and there is some $x \in B$ such that $x \notin A$, then we say A is a *proper subset* of B, denoting it by $A \subset B$.

If there is some $x \in A$ such that $x \notin B$, then A is not a subset of B, which we write as $A \nsubseteq B$.

☞ *Some authors use $\subset$ to mean subset without necessarily implying it is a proper subset.*

149 Example Let $S = \{1,2,\ldots,20\}$, that is, the set of integers between 1 and 20, inclusive. Let $E = \{2,4,6,\ldots,20\}$, the set of all even integers between 2 and 20, inclusive. Notice that $E \subseteq S$. Let $P = \{2,3,5,7,11,13,17,19\}$, the set of primes less than 20. Then $P \subseteq S$.

The following theorem can be used to prove that two sets are the same.

150 Theorem Two sets A and B are equal if and only if $A \subseteq B$ and $B \subseteq A$.

151 Example Let $S = \{n^2 | n \in \mathbb{Z}\}$. Then $A = \{1, 4, 9, 16\} \subseteq S$. We can also write that $A \subset S$ in this case since 25, for instance, is in S but not in A. Also notice that although $S \subseteq S$, $S \not\subset S$.

152 Example The set

$$S = \{\text{Roxan}, \text{Jacquelin}, \text{Sean}, \text{Fatimah}, \text{Wakeelah}, \text{Ashley}, \text{Ruben}, \text{Leslie}, \text{Madeline}, \text{Karina}\}$$

is the set of students in a particular section of Maths 016. This set can be split into two subsets: the set

$$F = \{\text{Roxan}, \text{Jacquelin}, \text{Fatimah}, \text{Wakeelah}, \text{Ashley}, \text{Madeline}, \text{Karina}\}$$

of females in the class, and the set

$$M = \{\text{Sean}, \text{Ruben}, \text{Leslie}\}$$

of males in the class. Thus we have $F \subseteq S$ and $M \subseteq S$. Notice that it is *not true* that $F \subseteq M$ or that $M \subseteq F$.

153 Example Find all the subsets of $\{a, b, c\}$.

Solution: They are

$$\begin{array}{rcl} S_1 & = & \varnothing \\ S_2 & = & \{a\} \\ S_3 & = & \{b\} \\ S_4 & = & \{c\} \\ S_5 & = & \{a, b\} \\ S_6 & = & \{b, c\} \\ S_7 & = & \{a, c\} \\ S_8 & = & \{a, b, c\} \end{array}$$

Notice that there are 8 subsets. Also notice that $8 = 2^3$. As we will see shortly, that is not a coincidence.

Also notice that we wrote $S_1 = \varnothing$, and not $S_1 = \{\varnothing\}$. It turns out that $\varnothing \neq \{\varnothing\}$. $\varnothing$ is the empty set–that is, the set that has no elements. $\{\varnothing\}$ is the set containing the empty set. Thus, $\{\varnothing\}$ is a set containing one element, $\varnothing$.

154 Example Find all the subsets of $\{a, b, c, d\}$.

Solution: We will use the result of example 153. A subset of $\{a, b, c, d\}$ either contains d or it does not. Since the subsets of $\{a, b, c\}$ do not contain d, we simply list all the subsets of $\{a, b, c\}$ and then to each one of them we add d. This gives

$$\begin{array}{rclrcl} S_1 & = & \varnothing & S_9 & = & \{d\} \\ S_2 & = & \{a\} & S_{10} & = & \{a, d\} \\ S_3 & = & \{b\} & S_{11} & = & \{b, d\} \\ S_4 & = & \{c\} & S_{12} & = & \{c, d\} \\ S_5 & = & \{a, b\} & S_{13} & = & \{a, b, d\} \\ S_6 & = & \{b, c\} & S_{14} & = & \{b, c, d\} \\ S_7 & = & \{a, c\} & S_{15} & = & \{a, c, d\} \\ S_8 & = & \{a, b, c\} & S_{16} & = & \{a, b, c, d\} \end{array}$$

155 Definition The *power set* of a set is the set of all subsets of a set. The power set of a set A is denoted by $P(A)$.

156 Example If $A = \{a,b,c\}$, example 153 implies that $P(A) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a,b\}, \{b,c\}, \{a,c\}, \{a,b,c\}\}$. Notice that the solution is a set, the elements of which are also sets.

An *incorrect answer* would be $\{\varnothing, a, b, c, \{a,b\}, \{b,c\}, \{a,c\}, \{a,b,c\}\}$. This is incorrect because a is not the same thing as $\{a\}$ (the set containing a).

157 Theorem Let A be a set with n elements. Then $|P(A)| = 2^n$.

Proof: We use induction[5] and the idea of example 154. Clearly if $|A| = 1$, A has $2^1 = 2$ subsets: $\varnothing$ and A itself.
Assume every set with $n-1$ elements has 2^{n-1} subsets. Let A be a set with n elements. Choose some $x \in A$. Every subset of A either contains x or it doesn't. Those that do not contain x are subsets of $A \setminus \{x\}$. Since $A \setminus \{x\}$ has $n-1$ elements, the induction hypothesis implies that it has 2^{n-1} subsets. Every subset that does contain x corresponds to one of the subsets of $A \setminus \{x\}$ with the element x added. That is, for each subset $S \subseteq A \setminus \{x\}$, $S \cup \{x\}$ is a subset of A containing x. Clearly there are 2^{n-1} such new subsets. Since this accounts for all subsets of A, A has a total of $2^{n-1} + 2^{n-1} = 2^n$ subsets. □

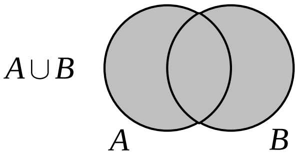
3.2.1 Set Operations

We can obtains new sets by performing various operations on other sets. In this section we discuss the most common set operations. When discussing sets, *Venn diagrams* are often used as a pictorial representation of the relationships between sets. We provide Venn diagrams to help visualize the various set operations.

158 Definition The *union* of two sets A and B is the set containing elements from either A or B . More formally,

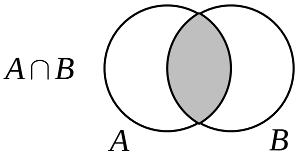
$$A \cup B = \{x : x \in A \text{ or } x \in B\}.$$

Notice that in this case the *or* is an *inclusive or*. That is, x can be in A, or it can be in B, or it can be in both.



159 Definition The *intersection* of two sets A and B is the set containing elements that are in both A and B. More formally,

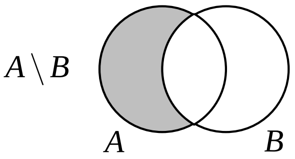
$$A \cap B = \{x : x \in A \text{ and } x \in B\}.$$



160 Definition The difference of sets, A *set-minus* B, is the set containing elements from A that are not in B. More formally,

$$A \setminus B = \{x : x \in A \text{ and } x \notin B\}.$$

The set difference of A and B is sometimes denoted by $A - B$.



[5] We will cover induction more fully and formally later. But since this use of induction is pretty intuitive, especially in light of Example 154, it serves as a useful foreshadowing of things to come.

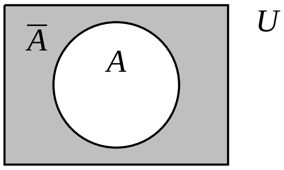
161 Example Let $A = \{1,2,3,4,5,6\}$, and $B = \{1,3,5,7,9\}$. Then

$$A\cup B = \{1,2,3,4,5,6,7,9\}, \qquad A\cap B = \{1,3,5\}, \qquad A\setminus B = \{2,4,6\}, \qquad B\setminus A = \{7,9\}.$$

162 Definition Two sets A and B are *disjoint* or *mutually exclusive* if $A\cap B = \varnothing$. That is, they have no elements in common.

163 Definition Let $A \subseteq U$. The *complement* of A with respect to U is just the set difference $U\setminus A$. More formally,

$$\overline{A} = \{x\in U : x \not\in A\} = U\setminus A.$$



Other common notations for set complement include A^c and A'.

Often the set U, which is called the universe *or* universal set, *is implied and we just use $\overline{A}$ to denote the complement. Generally speaking, we will follow this convention here. Futher, when talking about several sets, we will assume they have the same universal set unless otherwise specified.*

164 Example Let $U = \{0,1,2,3,4,5,6,7,8,9\}$ be the universal set of the decimal digits and let $A = \{0,2,4,6,8\} \subset U$ be the set of even digits. Then $\overline{A} = \{1,3,5,7,9\}$ is the set of odd digits.

Observe that

$$\begin{aligned} \overline{A}\cap A &= \varnothing, \text{ and} \\ \overline{A}\cup A &= U. \end{aligned}$$

The various intersecting regions for two and three sets can be seen in figures 3.1 and 3.2.

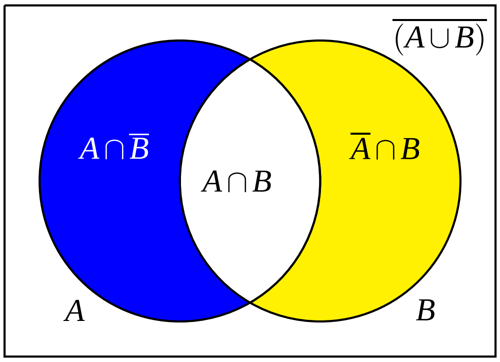


Figure 3.1: Venn diagram for two sets.

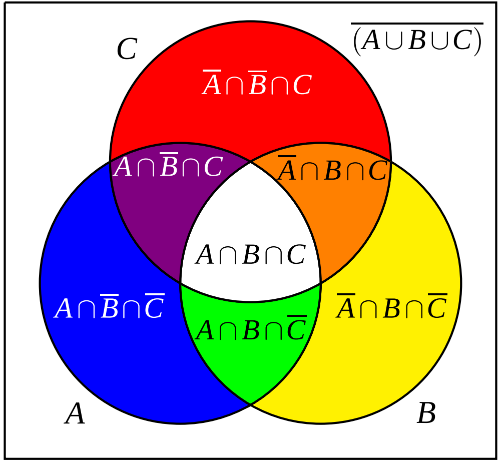

C
$\overline{(A\cup B\cup C)}$
$\overline{A}\cap\overline{B}\cap C$
$A\cap\overline{B}\cap C$
$\overline{A}\cap B\cap C$
$A\cap B\cap C$
$A\cap\overline{B}\cap\overline{C}$
$A\cap B\cap\overline{C}$
$\overline{A}\cap B\cap\overline{C}$
A
B

Figure 3.2: Venn diagram for three sets.

165 Example It should not be too difficult to convince yourself that $A\setminus B = A\cap\overline{B}$. This is an example of what we call a *set identity*.

There are many common set identities. Instead of boring you with a long list of them, for now we just present *De Morgan's Laws*.[6] We will see more of them in Section 3.3.

[6] That name sounds familiar. Haven't we seen this before?

166 Theorem De Morgan’s Laws state that for sets A and B,

$$\overline{(A\cup B)} = \overline{A}\cap\overline{B}, \text{ and} \tag{3.1}$$

$$\overline{(A\cap B)} = \overline{A}\cup\overline{B}. \tag{3.2}$$

Proof: We will prove the first one. As you might imagine, the proof of the other is very similar. We will use Theorem 150.
Let $x \in \overline{(A\cup B)}$. Then $x \notin A\cup B$ (by definition of complement). Thus $x \notin A \wedge x \notin B$ (by definition of union), that is, $x \in \overline{A} \wedge x \in \overline{B}$ (by definition of complement). This is the same as $x \in \overline{A}\cap\overline{B}$ (by definition of union). Notice that x was an arbitrary element from $\overline{(A\cup B)}$, and we showed that $x \in \overline{A}\cap\overline{B}$. Therefore, every element in $\overline{(A\cup B)}$ is also in $\overline{A}\cap\overline{B}$. In other words, $\overline{(A\cup B)} \subseteq \overline{A}\cap\overline{B}$.
Now, let $x \in \overline{A}\cap\overline{B}$. Then $x \in \overline{A} \wedge x \in \overline{B}$. This means that $x \notin A \wedge x \notin B$ or what is the same $x \notin A\cup B$. But this last statement asserts that $x \in \overline{(A\cup B)}$. Hence $\overline{A}\cap\overline{B} \subseteq \overline{(A\cup B)}$.
Since we have shown that the two sets contain each other, they are equal by Theorem 150. □

This proof is what is called a *set containment proof* since we showed set containment both ways. The technique is pretty straightforward: Theorem 150 tells us that if $X \subseteq Y$ and $Y \subseteq X$, then $X = Y$. Thus, to prove $X = Y$, we just need to show that $X \subseteq Y$ and $Y \subseteq X$. But how do we show that one set is a subset of another? This is easy: To show that $X \subseteq Y$, we show that every element from X is also in Y. In other words, we assume that $x \in X$ and use definitions and logic to show that $x \in Y$. Assuming we do not use any special properties of x other than the fact that $x \in X$, then x is an arbitrary element from X, so this shows that $X \subseteq Y$.

☞ *Be careful. To prove that $X = Y$, you generally need to prove two things: $X \subseteq Y$ and $Y \subseteq X$. Do not forget to do both. On the other hand, if you are asked to prove that $X \subseteq Y$, you do not need to (and should not) show that $Y \subseteq X$.*

Sometimes we can do a set containment proof in one step instead of two. This only works if every step of the proof is reversible. We illustrate this idea next. (Here, the ↔ means “if and only if”. Although it looks a lot like it, it is not the logical biconditional operator.

167 Example Prove that $A\setminus(B\cup C) = (A\setminus B)\cap(A\setminus C)$.

Proof: We have

$$\begin{array}{rcl}
x \in A\setminus(B\cup C) & \leftrightarrow & x \in A \wedge x \notin (B\vee C) \\
& \leftrightarrow & (x\in A) \quad\wedge\quad ((x\notin B) \quad\wedge\quad (x\notin C)) \\
& \leftrightarrow & (x\in A \quad\wedge\quad x\notin B) \quad\wedge\quad (x\in A \quad\wedge\quad x\notin C) \\
& \leftrightarrow & (x\in A\setminus B) \quad\wedge\quad (x\in A\setminus C) \\
& \leftrightarrow & x \in (A\setminus B)\cap(A\setminus C)
\end{array}$$

□

168 Example In Java, the `TreeSet` class is one implementation of a *set* that has several methods with perhaps unfamiliar names, but they do what should be familiar things. Let’s discuss a few of them.[7] Let `A` and `B` be `TreeSet`s.

[7] The method signatures and documentation have been modified from the official definition so we can focus on the point at hand.

1. The method `retainAll(TreeSet other)` "*retains only the elements in this TreeSet that are contained in the* `other` *TreeSet. In other words, removes from this TreeSet all of its elements that are not contained in* `other`." It is not too difficult to see that `A.retainAll(B)` is computing $A \cap B$.[8]

2. The method `boolean containsAll(TreeSet other)` "*returns true if this set contains all of the elements of* `other` *(and false otherwise).*" It should be evident that `A.containsAll(B)` returns true iff $B \subseteq A$.

3. Even without documentation, it seems likely that `A.size()` is determining $|A|$.

4. It is also seems likely that `A.isEmpty()` is determining if $A = \emptyset$.

Sometimes you need to find the cardinality of the union of several sets. This is easy of the sets do not intersect. If they do intersect, more care is needed to make sure no elements are missed or counted more than once. In the following examples we will use Venn diagrams to help us do this correctly.[9] Later, we will learn about a more powerful tool to do this—*inclusion-exclusion*.

169 Example Of 40 people, 28 smoke and 16 chew tobacco. It is also known that 10 both smoke and chew. How many among the 40 neither smoke nor chew?

> **Solution:** We fill up the Venn diagram in figure 3.3 as follows. Since $\text{card}(Smoke \cap Chew) = 10$, we put a 10 in the intersection. Then we put a $28 - 10 = 18$ in the part that *Smoke* does not overlap *Chew* and a $16 - 10 = 6$ in the part of *Chew* that does not overlap *Smoke*. We have accounted for $10 + 18 + 6 = 34$ people that are in at least one of the sets. The remaining $40 - 34 = 6$ are outside these sets.

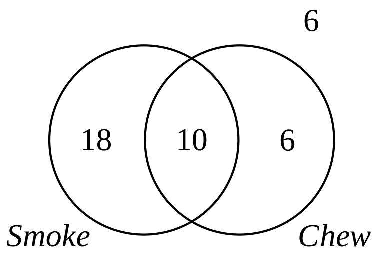


Figure 3.3: Example 482.

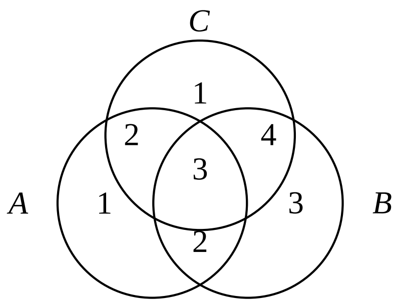


Figure 3.4: Example 170.

170 Example In a group of 30 people, 8 speak English, 12 speak Spanish and 10 speak French. It is known that 5 speak English and Spanish, 5 Spanish and French, and 7 English and French. The number of people speaking all three languages is 3. How many do not speak any of these languages?

[8] Technically it is doing more than that. It is storing the result in A. So it is like it is doing $A = A \cap B$, where $=$ here means assignment, not equals.

[9] Actually, both of the examples count the number of elements not in the union. But since this is just the number of elements in the universe minus the number in the union, the technique is the same.

> **Solution:** Let A be the set of all English speakers, B the set of Spanish speakers and C the set of French speakers in our group. We fill-up the Venn diagram in figure 3.4 successively. In the intersection of all three we put 3. In the region common to A and B which is not filled up we put $5-2=3$. In the region common to A and C which is not already filled up we put $5-3=2$. In the region common to B and C which is not already filled up, we put $7-3=4$. In the remaining part of A we put $8-2-3-2=1$, in the remaining part of B we put $12-4-3-2=3$, and in the remaining part of C we put $10-2-3-4=1$. Each of the mutually disjoint regions comprise a total of $1+2+3+4+1+2+3=16$ persons. Those outside these three sets are then $30-16=14$.

171 Definition The *Cartesian product* of sets A and B is the set $A \times B = \{(a,b) | a \in A \wedge b \in B\}$. In other words, it is the set of all ordered pairs of elements from A and B.

172 Definition If A is a set, then $A^2 = A \times A$, and $A^n = A \times A^{n-1}$.

173 Example If $A=\{1,2,3\}$ and $B=\{a,b\}$, then $A\times B=\{(1,a),(1,b),(2,a),(2,b),(3,a),(3,b)\}$. Also, $B^2=\{(a,a),(a,b),(b,a),(b,b)\}$.

3.2.2 Partitions and Equivalence Relations

174 Definition Let $S \neq \varnothing$ be a set. A *partition* of S is a collection of non-empty, pairwise disjoint subsets of S whose union is S.

175 Example Define $\mathbb{E}=\{2k : k\in\mathbb{Z}\}$ and $\mathbb{O}=\{2k+1 : k\in\mathbb{Z}\}$. Clearly $\mathbb{E}$ is the set of even integers and $\mathbb{O}$ is the set of odd integers. Since $\mathbb{E}\cap\mathbb{O}=\varnothing$ and $\mathbb{E}\cup\mathbb{O}=\mathbb{Z}$, $\{\mathbb{E},\mathbb{O}\}$ is a partition of $\mathbb{Z}$.

176 Example Let $3\mathbb{Z}=\{3k : k\in\mathbb{Z}\}$, $3\mathbb{Z}+1=\{3k+1 : k\in\mathbb{Z}\}$, and $3\mathbb{Z}+2=\{3k+2 : k\in\mathbb{Z}\}$.[10] Since

$$(3\mathbb{Z})\cup(3\mathbb{Z}+1)\cup(3\mathbb{Z}+2)=\mathbb{Z} \text{ and}$$

$$(3\mathbb{Z})\cap(3\mathbb{Z}+1)=\varnothing,\ (3\mathbb{Z})\cap(3\mathbb{Z}+2)=\varnothing,(3\mathbb{Z}+1)\cap(3\mathbb{Z}+2)=\varnothing,$$

$\{3\mathbb{Z},3\mathbb{Z}+1,3\mathbb{Z}+2\}$ is a partition of $\mathbb{Z}$.

177 Example Let $\mathbb{I}=\mathbb{R}\setminus\mathbb{Q}$ (the set of irrational numbers). Observe that $\mathbb{R}=\mathbb{Q}\cup\mathbb{I}$ and $\mathbb{Q}\cap\mathbb{I}=\varnothing$. Thus, the real numbers can be partitioned into the rational and irrational numbers, which shouldn't really be a surprise.

Recall that when a list of number is given between parentheses (e.g. $(1,2,3)$), it typically denotes an ordered list. That is, the order that the element are listed matters. So, for instance, $(1,2)$ and $(2,1)$ are not the same thing.

178 Definition Let A,B be sets. A *relation* R is a subset of the Cartesian product $A\times B$. If $(x,y)\in R$, we say that x is *related to* y, and write is as xRy. An alternative notation is $a\sim b$.

[10] The notation in this example may seem a bid odd at first. How are you supposed to interpret "$3\mathbb{Z}+1$"? Is this 3 times the set $\mathbb{Z}$ plus 1? What does it mean to do algebra with sets and numbers? I won't get into all of the technical details, but here is a short answer. You can think of "$3\mathbb{Z}+1$" as just a name. Sure, it may seem like an odd name, but why can't we name a set whatever we want? Some people name their kids *Jon Blake Cusack 2.0* and get away with it. You can also think of "$3\mathbb{Z}+1$" as describing how to create the set—by taking every element from $\mathbb{Z}$, multiplying it by 3, and then adding 1. Thus, you can think of "$3\mathbb{Z}+1$" as being both an algebraic expression and a name.

179 Definition Let A be a set and R be a relation on $A \times A$. Then R is said to be

- **reflexive** if $\forall x \in A, xRx$
  (or $(x,x) \in R$).
- **symmetric** if $\forall x,y \in A, xRy \to yRx$
  (or $(x,y) \in R \to (y,x) \in R$).
- **anti-symmetric** if $\forall x,y \in A, (xRy)$ **and** $(yRx) \to x = y$
  (or $(x,y) \in R$ **and** $(y,x) \in R \to x = y$).
- **transitive** if $\forall x,y,z \in A, (xRy)$ **and** $(yRz) \to (xRz)$
  (or $(x,y) \in R$ **and** $(y,z) \in R \to (x,z) \in R$).

A relation R which is reflexive, symmetric and transitive is called an *equivalence relation* on A. A relation R which is reflexive, anti-symmetric and transitive is called a *partial order* on A.

180 Example Let $S = \{\text{All Human Beings}\}$, and define the the relation M by $(a,b) \in M$ if a has the same (biological) mother[11] as b. Show that M is an equivalence relation.

> **Proof:** Since a has the same mother as a, $(a,a) \in M$, so M is reflexive. If a has the same mother as b, then b clearly has the same mother as a. Thus, $(a,b) \in M$ implies $(b,a) \in M$, so M is symmetric. Finally, if a has the same mother as b, and b has the same mother as c, then clearly a has the same mother as c. In other words, $(a,b) \in M$ and $(b,c) \in M$ implies that $(a,c) \in M$, so M is transitive. Therefore, M is reflexive, symmetric, and transitive, so it is an equivalence relation. □

181 Example Let X be a collection of sets. Let R to be the relation such that A is related to B if $A \subseteq B$. Then R is a partial order on X. We leave it to the reader to prove this. You need to show that R is reflexive, anti-symmetric, and transitive.

182 Definition For integers a, b, and n, where $n > 0$, we say that *a is congruent to b modulo n* if and only if $a - b = kn$ for some integer k (that is, n divides $a - b$). We write it as $a \equiv b \pmod{n}$. If $a - b \neq kn$ for any integer k, then a is not congruent to b modulo n, written as $a \not\equiv b \pmod{n}$.

The proof of the following is left as an exercise.

183 Theorem $a \equiv b \pmod{n}$ iff[12] $a \bmod n = b \bmod n$.

184 Example Notice that $21 - 6 = 15 = 3 \cdot 5$, so $21 \equiv 6 \pmod{5}$.

185 Example Since, $1961 \pmod{37} = 0 \neq 4 = 1966 \pmod{37}$, we know that $1961 \not\equiv 1966 \pmod{37}$.

186 Example Let n be a positive integer. Then $R = \{(a,b) : a \equiv b \pmod{n}\}$ is a relation on the set of integers. Show that R is an equivalence relation.

[11] The important assumption we are making is that each person has exactly one mother.

[12] **iff** is shorthand for **if and only if**.

> **Proof:** We need to show that R is reflexive, symmetric, and transitive.
> *Reflexive:* Since $a-a=0n$, $a\equiv a \pmod n$, so R is reflexive.
> *Symmetric:* If $a\equiv b \pmod n$, then $a-b=kn$ for some integer k. So $b-a=(-k)n$, and since $-k$ is an integer, $b\equiv a \pmod n$. Thus, R is symmetric.
> *Transitive:* If $a\equiv b \pmod n$ and $b\equiv c \pmod n$, then $a-b=kn$ for some integer k and $b-c=ln$ for some integer l. Thus
>
> $$a-c=(a-b)+(b-c)=kn+ln=(k+l)n,$$
>
> and since $k+l$ is an integer, $a\equiv c \pmod n$, and R is therefore transitive. □

187 Example Let R be the relation on the set of ordered pairs of positive integers such that $((a,b),(c,d))\in R$ if and only if $ad=bc$. Show that R is an equivalence relation.

> **Proof:** We need to show that R is reflexive, symmetric, and transitive.
> *Reflexive:* Since $ab=ba$ for all positive integers, $((a,b),(a,b))\in R$ for all (a,b). Thus R is reflexive.
> *Symmetric:* Notice that if $ad=bc$, then $cb=da$ for all positive integers a, b, c, and d. Thus $((a,b),(c,d))\in R$ implies that $((c,d),(a,b))\in R$, so R is symmetric.
> *Transitive:* Assume that $((a,b),(c,d):1)\in R$ and $((c,d),(e,f))\in R$. Then $ad=bc$ and $cf=de$. Solving the second for c, we get $c=de/f$, and plugging it into the first we get $ad=b(de/f)$. Multiplying both sides by f, and canceling the d on both sides yields $af=be$. Thus $((a,b),(e,f))\in R$. Thus R is transitive. □

188 Definition Let R be an equivalence relation on a set S. Then the *equivalence class of a* is the subset of S containing all of the elements that are related to a. More formally,

$$[a]=\{x\in S : xRa\}.$$

189 Example The equivalence class of 3 modulo 8 is $[3]=\{8k+3:k\in\mathbb{Z}\}$. Notice that $[11]=\{8k+11:k\in\mathbb{Z}\}=\{8k+3:k\in\mathbb{Z}\}=[3]$. In fact, $[3]=[8l+3]$ for all integers l.

190 Example Notice that if our relation is congruence modulo 4, then

$$\begin{array}{rcl}[0] & = & \{4k:k\in\mathbb{Z}\},\\ [1] & = & \{4k+1:k\in\mathbb{Z}\},\\ [2] & = & \{4k+2:k\in\mathbb{Z}\},\text{ and}\\ [3] & = & \{4k+3:k\in\mathbb{Z}\}.\end{array}$$

Thus, it isn't too difficult to notice that $\mathbb{Z}=[1]\cup[2]\cup[3]\cup[4]$. In other words, the equivalence classes $\{[1],[2],[3],[4]\}$ form a partition of $\mathbb{Z}$. As we will see next, this is not a coincidence.

191 Lemma Let R be an equivalence relation on a set S. Then two equivalence classes are either identical or disjoint.

Proof: Let $a, b \in S$, and assume $[a] \cap [b] \neq \varnothing$. We need to show that $[a] = [b]$. First, let $x \in [a] \cap [b]$ (which exists since $[a] \cap [b] \neq \varnothing$). Then xRa and xRb, so by symmetry aRx and by transitivity aRb.

Now let $y \in [a]$. Then yRa. Since we just showed that aRb, then yRb by transitivity. Thus $y \in [b]$. Therefore $[a] \subseteq [b]$.

A symmetric argument proves that $[b] \subseteq [a]$. Therefore, $[a] = [b]$. □

192 Theorem Let $S \neq \varnothing$ be a set. Every equivalence relation on S induces a partition of S and vice-verse.

Proof: By Lemma 191, if R is an equivalence relation on S then

$$S = \bigcup_{a \in S} [a],$$

and $[a] \cap [b] = \varnothing$ if a is not related to b. This proves the first half of the theorem.

Conversely, let

$$S = \bigcup_{\alpha} S_\alpha, \quad S_\alpha \cap S_\beta = \varnothing \text{ if } \alpha \neq \beta,$$

be a partition of S. We define the relation R on S by letting aRb if and only if they belong to the same S_α. Since the S_α are mutually disjoint, it is clear that R is an equivalence relation on S and that for $a \in S_\alpha$, we have $[a] = S_\alpha$. □

Exercises

193 Problem Prove by means of set inclusion (or set containment) that $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$.

194 Problem A survey of a group's viewing habits over the last year revealed the following information:

❶ 28% watched gymnastics

❷ 29% watched baseball

❸ 19% watched soccer

❹ 14% watched gymnastics and baseball

❺ 12% watched baseball and soccer

❻ 10% watched gymnastics and soccer

❼ 8% watched all three sports.

Calculate the percentage of the group that watched none of the three sports during the last year.

195 Problem In a group of 100 camels, 46 eat wheat, 57 eat barley, and 10 eat neither. How many camels eat both wheat and barley?

196 Problem At *Medieval High* there are forty students. Amongst them, fourteen like Mathematics, sixteen like theology, and eleven like alchemy. It is also known that seven like Mathematics and theology, eight like theology and alchemy and five like Mathematics and alchemy. All three subjects are favoured by four students. How many students like neither Mathematics, nor theology, nor alchemy?

197 Problem How many integers in the set $\{1, 2, \ldots, 200\}$ are neither divisible by 3 nor 7 but are divisible by 11.

Answers

193 We have,

$$\begin{aligned} x\in(A\cup B)\cap C & \leftrightarrow & x\in(A\cup B)\wedge x\in C \\ & \leftrightarrow & (x\in A\vee x\in B)\wedge x\in C \\ & \leftrightarrow & (x\in A\wedge x\in C)\vee(x\in B\wedge x\in C) \\ & \leftrightarrow & (x\in A\cap C)\vee(x\in B\cap C) \\ & \leftrightarrow & x\in(A\cap C)\cup(B\cap C), \end{aligned}$$

which establishes the equality.

194 52%

195 Let A be the set of camels eating wheat, and $|A|$ its number, and B be the set of camels eating barley, and $|B|$ its number. Then

$$90=100-10=|A\cup B|=|A|+|B|-|A\cap B|=46+57-|A\cap B|=103-|A\cap B|,$$

whence $|A\cap B|=13$.

196 15

197 10

3.3 Boolean Algebras

Now we address a topic that seems quite different from the two topics we have just covered. As we shall see shortly, however, they are actually very much related.

198 Definition A *Boolean algebra* consists of a set X with at least two different elements 0 (the additive identity) and 1 (the multiplicative identity), two binary operations $+$ (addition) and $\cdot$ (multiplication), and a unary operation^{-} (called*complementation*) satisfying the following axioms for all $A, B, C \in X$.

1. $A + B = B + A$ (commutativity of addition)

2. $A \cdot B = B \cdot A$ (commutativity of multiplication)

3. $A + (B + C) = (A + B) + C$ (associativity of addition)

4. $A \cdot (B \cdot C) = (A \cdot B) \cdot C$ (associativity of multiplication)

5. $A \cdot (B + C) = A \cdot B + A \cdot C$ (distributive law)

6. $A + (B \cdot C) = (A + B) \cdot (A + C)$ (distributive law)

7. $A + 0 = A$ (additive identity)

8. $A \cdot 1 = A$ (multiplicative identity)

9. $A + \overline{A} = 1$ (unit property)

10. $A \cdot \overline{A} = 0$ (zero property)

☞ *Sometimes the product $A \cdot B$ is written as AB, with the operator $\cdot$ omitted. In this case, it is understood that the operator between A and B is multiplication. This is nothing new–you omit $\times$ when multiplying number all of the time.*

Notice that the definition of a Boolean algebra requires that the elements 0 and 1 be in the set X, but it says nothing about other possible elements. This means that X may or may not have additional elements. The following properties of the 0 and 1 elements of a Boolean algebra are immediate.

199 Theorem $\overline{0} = 1$ and $\overline{1} = 0$.

> **Proof:** Since 0 is the additive identity, $\overline{0} = \overline{0} + 0$. But by axiom 9, $\overline{0} + 0 = 1$ and thus $\overline{0} = \overline{0} + 0 = 1$. Similarly, since 1 is the multiplicative identity, $\overline{1} = 1 \cdot \overline{1}$. But by axiom 10, $1 \cdot \overline{1} = 0$ and thus $\overline{1} = 1 \cdot \overline{1} = 0$. □

The operations of complementation, addition and multiplication act on 0 and 1 as shown in table 3.7. You might notice that this table resembles the truth table we saw earlier. We will see why in the next example.

200 Example If we regard $0 = F$, $1 = T$, $+ = \vee$, $\cdot = \wedge$, and $^{-} = \neg$, then the logic operations over $\{F, T\}$ constitute a boolean algebra.

201 Example If we regard $0 = \varnothing$, $1 = U$ (the universal set), $+ = \cup$, $\cdot = \cap$, and $^{-} = {}^{-}$, then the set operations over the subsets of U constitute a boolean algebra.

Why do we care about the connection between logic, sets, and boolean algebras? Because any property that we know about a Boolean algebra can be applied to logic and sets. Table 3.8 gives some of the important laws and identities of Boolean algebras, including a translation of some of them into logic and sets. Filling in the remainder of the table is left as an exercise.

The first 10 laws are the axioms from the definition of a Boolean algebra. The remaining laws can be proven using the axioms. The next several examples give proofs of several of these, as well as a few other laws.

202 Theorem (Idempotent Laws) $A+A=A$ and $AA=A$

Proof: We have

$$A=A+0=A+A\cdot\overline{A}=(A+A)(A+\overline{A})=(A+A)(1)=A+A.$$

Similarly

$$A=A1=A(A+\overline{A})=AA+A\cdot\overline{A}=AA+0=AA.$$

□

203 Theorem (Domination Laws) $A+1=1$ and $A\cdot 0=0$.

Proof: We have

$$A+1=A+(A+\overline{A})=(A+A)+\overline{A}=A+\overline{A}=1.$$

Also,

$$A\cdot 0=A(A\cdot\overline{A})=(AA)\overline{A}=A\overline{A}=0.$$

□

204 Theorem (Uniqueness of the Complement) If $AB=0$ and $A+B=1$ then $B=\overline{A}$.

Proof: We have

$$B=B1=B(A+\overline{A})=BA+B\overline{A}=0+B\overline{A}=B\overline{A}.$$

Also,

$$\overline{A}=\overline{A}1=\overline{A}(A+B)=\overline{A}\cdot A+\overline{A}B=\overline{A}B.$$

Thus

$$B=B\overline{A}=\overline{A}B=\overline{A}.$$

□

A	B	$\overline{A}$	$A+B$	AB
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

Table 3.7: Evaluation Rules

205 Theorem (Involution Law) $\overline{\overline{A}} = A$

Proof: By axioms 9 and 10, we have the identities

$$1 = \overline{A} + \overline{\overline{A}} \text{ and } \overline{\overline{A}} \cdot \overline{A} = 0.$$

By uniqueness of the complement we must have $A = \overline{\overline{A}}$. □

206 Theorem (De Morgan's Laws) $\overline{A+B} = \overline{A} \cdot \overline{B}$ and $\overline{A \cdot B} = \overline{A} + \overline{B}$.

Proof: Observe that

$$(A+B) + \overline{A} \cdot \overline{B} = (A+B+\overline{A})(A+B+\overline{B}) = (B+1)(A+1) = 1,$$

and

$$(A+B)\overline{A} \cdot \overline{B} = A\overline{A} \cdot \overline{B} + B\overline{A} \cdot \overline{B} = 0+0 = 0.$$

Thus $\overline{A} \cdot \overline{B}$ is the complement of $A+B$ and so we must have $\overline{A} \cdot \overline{B} = \overline{A+B}$.

To obtain the other De Morgan Law put $\overline{A}$ instead of A and $\overline{B}$ instead of B in the law just derived and use the involution law:

$$\overline{\overline{A}+\overline{B}} = \overline{\overline{A}} \cdot \overline{\overline{B}} = AB.$$

Taking complements once again we have

$$\overline{\overline{\overline{A}+\overline{B}}} = AB \to \overline{A} + \overline{B} = \overline{AB}.$$

□

Law	*Boolean Algebra*	*Logic*	*Sets*
commutativity	$A+B = B+A$ $AB = BA$	$p \wedge q = q \wedge p$	
associativity	$A+(B+C) = (A+B)+C$ $A(BC) = (AB)C$	$p \vee (q \vee r) = (p \vee q) \vee r$	
distributive	$A(B+C) = AB+AC$ $A+(BC) = (A+B)(A+C)$		$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
identity	$A+0 = A$ $A1 = A$		
unit property	$A+\overline{A} = 1$	$p \vee \neg p = T$	
zero property	$A\overline{A} = 0$		$A \cap \overline{A} = \varnothing$
domination	$A+1 = 1$ $A0 = 0$	$p \vee T \ \ = T$	
idempotent	$A+A = A$ $AA = A$		$A \cap A = A$
double negation	$\overline{\overline{A}} = A$		
DeMorgan's	$\overline{A+B} = \overline{A}\overline{B}$ $\overline{AB} = \overline{A}+\overline{B}$		

Table 3.8: Some of the most important laws of Boolean algebras. A translation into the language of logic and sets is given for a few of them. The rest are left as an exercise.

207 Theorem $AB + A\overline{B} = A$.

Proof: Factoring

$$AB + A\overline{B} = A(B + \overline{B}) = A(1) = A.$$

□

208 Example Simplify the following code as much as possible.

```
if ((x>0 && x<y) || (x>0 && x>=y)) {
    x=y;
}
```

Solution: Let $p =$"$x > 0$" and $q =$"$x < y$". Then the conditional above can be expressed as $(p \wedge q) \vee (p \wedge \neg q)$. Applying Theorem 207, this is just p. Therefore the code simplifies to:

```
    if (x>0) {
        x=y;
    }
```

209 Theorem $A(\overline{A} + B) = AB$ and $A + \overline{A}B = A + B$.

Proof: Multiplying

$$A(\overline{A} + B) = A\overline{A} + AB = 0 + AB = AB.$$

Using the distributive law,

$$A + \overline{A}B = (A + \overline{A})(A + B) = 1(A + B) = A + B.$$

□

210 Theorem (Absorption Laws) $A + AB = A$ and $A(A + B) = A$.

Proof: Factoring and using the domination laws:

$$A + AB = A(1 + B) = A1 = A.$$

Expanding and using the identity just derived:

$$A(A + B) = AA + AB = A + AB = A.$$

□

3.3.1 Sum of Products and Products of Sums

Given a truth table in some boolean variables, we would like to find a function whose output is that of the table. This can be done by either finding a *sum of products* (SOP) or a *product of sums* (POS) for the table.

To find a sum of products from a truth table:

❶ identify the rows having output 1.

❷ for each such row, write the variable if the variable input is 1 or write the complement of the variable if the variable input is 0, then multiply the variables forming a term.

❸ add all such terms.

To find a product of sums from a truth table:

❶ identify the rows having output 0.

❷ for each such row, write the variable if the variable input is 0 or write the complement of the variable if the variable input is 1, then add the variables forming a sum

❸ multiply all such sums.

211 Example Find a SOP and a POS for Z.

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Solution: The output (Z) is 1 on the rows with (i) $A = 0, B = 0, C = 0$, so we form the term $(\overline{A})(\overline{B})(\overline{C})$, (ii) $A = 0, B = 1, C = 0$, so we form the term $\overline{A}B\overline{C}$, (iii) $A = 1, B = 1, C = 0$, so we form the term $AB\overline{C}$, and (iv) $A = B = C = 1$, giving the term ABC. The required SOP is

$$Z = (\overline{A})(\overline{B})(\overline{C}) + \overline{A}B\overline{C} + AB\overline{C} + ABC.$$

The output (Z) is 0 on the rows with (i) $A = 0, B = 0, C = 1$, so we form the term $A + B + \overline{C}$, (ii) $A = 0, B = 1, C = 1$, so we form the term $A + \overline{B} + \overline{C}$, (iii) $A = 1, B = 0, C = 0$, so we form the term $\overline{A} + B + C$, and (iv) $A = 1, B = 0, C = 1$, giving the term $\overline{A} + B + \overline{C}$. The required POS is

$$Z = (A + B + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + B + C)(\overline{A} + B + \overline{C}).$$

Using the axioms of a boolean algebra and the aforementioned theorems we may simplify a given boolean expression, or transform a SOP into a POS or vice-versa.

212 Example Convert the following POS to a SOP:

$$(A + \overline{B}C)(A + \overline{B}D).$$

Solution:

$$\begin{aligned}(A+\overline{B}C)(A+\overline{B}D) &= AA+A\overline{B}D+A\overline{B}C+\overline{B}C\overline{B}D\\ &= A+A\overline{B}D+A\overline{B}C+\overline{B}CD\\ &= A+\overline{B}CD.\end{aligned}$$

213 Example Convert the following SOP to a POS:

$$A\overline{B}+\overline{C}D.$$

Solution:

$$\begin{aligned}A\overline{B}+\overline{C}D &= (A\overline{B}+\overline{C})(A\overline{B}+D)\\ &= (A+\overline{C})(\overline{B}+\overline{C})(A+D)(\overline{B}+D).\end{aligned}$$

214 Example Write $\overline{W}XY+\overline{W}XZ+\overline{Y+Z}$ as a sum of two products.

Solution: We have

$$\begin{aligned}\overline{W}XY+\overline{W}XZ+\overline{Y+Z} &= \overline{W}X(Y+Z)+\overline{Y+Z}\\ &= \overline{W}X+\overline{Y+Z}\\ &= \overline{W}X+\overline{Y}\cdot\overline{Z},\end{aligned}$$

where we have used the fact that $AB+\overline{B}=A+\overline{B}$ and the De Morgan laws.

Although we will not do much else with sum of products or products of sums, they are important in several areas of computer science, ranging from practical problems like circuit minimization to theoretical problems like the theory of NP-completeness.

3.3.2 Logic Puzzles

The boolean algebra identities from the preceding section may help to solve some logic puzzles.

215 Example Brown, Johns and Landau are charged with bank robbery. The thieves escaped in a car that was waiting for them. At the inquest Brown stated that the criminals had escaped in a blue Buick; Johns stated that it had been a black Chevrolet, and Landau said that it had been a Ford Granada and by no means blue. It turned out that wishing to confuse the Court, each one of them only indicated correctly either the make of the car or only its colour. What colour was the car and of what make?

Solution: Consider the sentences

$$\begin{aligned}A &= \text{the car is blue}\\ B &= \text{the car is a Buick}\\ C &= \text{the car is black}\\ D &= \text{the car is a Chevrolet}\\ E &= \text{the car is a Ford Granada}\end{aligned}$$

Since each of the criminals gave one correct statement, Brown's statement implies that $A+B$ is true. Similarly, Johns's statement implies $C+D$ is true, and Landau's statement implies that $\overline{A}+E$ is true. It now follows that

$$(A+B)\cdot(C+D)\cdot(\overline{A}+E)$$

is true. Upon multiplying this out, we obtain

$$(A\cdot C\cdot \overline{A})+(A\cdot C\cdot E)+(A\cdot D\cdot \overline{A})+(A\cdot D\cdot E)+(B\cdot C\cdot \overline{A})+(B\cdot C\cdot E)+(B\cdot D\cdot \overline{A})+(B\cdot D\cdot E).$$

Notice that $(A\cdot C\cdot \overline{A})$ is clearly false since A and $\overline{A}$ cannot both be true. Similarly, $(A\cdot C\cdot E)$ if false since the car can't be both blue and black. We can argue similarly that all of the terms are false except the fifth. Thus $B\cdot C\cdot \overline{A}$ is true, and so the criminals escaped in a black Buick.

216 Example Margie, Mimi, April, and Rachel ran a race. Asked how they made out, they replied:
Margie: "April won; Mimi was second."
Mimi: "April was second and Rachel was third."
April: "Rachel was last; Margie was second."

If each of the girls made one and only one true statement, who won the race?

Solution: Consider the sentences

$$\begin{array}{lcl} A & = & \text{April was first} \\ B & = & \text{April was second} \\ C & = & \text{Mimi was second} \\ D & = & \text{Margie was second} \\ E & = & \text{Rachel was third} \\ F & = & \text{Rachel was last} \end{array}$$

Since each of the girls gave one true statement we have that

$$(A+C)(B+E)(F+D)=1.$$

Multiplying this out

$$ABF+ABD+AEF+AED+CBF+CBD+CEF+CED=1.$$

Now, $AB=EF=BC=CD=0$ so the only surviving term is AED and so April was first, Margie was second, Rachel was third, and Mimi was last.

217 Example Having returned home, Maigret rang his office on quai des Orfèvres.

"Maigret here . Any news?"

"Yes Chief. The inspectors have reported. Torrence thinks that if François was drunk, then either Etienne is the murderer or François is lying. Justin is of the opinion that either Etienne is the murderer or François was not drunk and the murder occurred after midnight. Inspector Lucas asked me to tell you that if the murder had occurred after midnight, then either Etienne is the murderer or François is lying. Then there was a ring from ...."

"That's all, thanks. That's enough!" The commissar replaced the receiver. He knew that when François was sober he never lied. Now everything was clear to him. Find, with proof, the murderer.

Solution: Represent the following sentences as:

$$\begin{array}{lcl} A & = & \text{François was drunk,} \\ B & = & \text{Etienne is the murderer,} \\ C & = & \text{François is telling a lie,} \\ D & = & \text{the murder took place after midnight.} \end{array}$$

We then have

$$A \to (B+C),\ \ B+\overline{A}D,\ \ D \to (B+C).$$

Using the identity

$$X \to Y = \overline{X}+Y,$$

we see that the output of the product of the following sentences must be 1:

$$(\overline{A}+B+C)(B+\overline{A}D)(\overline{D}+B+C).$$

After multiplying the above product and simplifying, we obtain

$$B+C\overline{A}D.$$

So, either Etienne is the murderer, or the following events occurred simultaneously: François lied, François was not drunk and the murder took place after midnight. But Maigret knows that $\overline{A}C = 0$, thus it follows that $B = 1$, i.e., Etienne is the murderer.

Exercises

218 Problem Obtain a sum of products for the truth table

A	B	C	Z
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Answers

218

$$\overline{A}\cdot\overline{B}\cdot\overline{C}+\overline{A}\cdot\overline{B}\cdot C+\overline{A}\cdot B\cdot\overline{C}+A\cdot\overline{B}\cdot\overline{C}$$

Homework

219 Problem Make a copy of Table 3.8 and fill in the missing entries.

220 Problem Give 2 different proofs that $[(p\vee q)\wedge\neg p] \to q$ is a tautology.

221 Problem Give 2 different proofs that $[p\wedge(p\to q)] \to q$ is a tautology.

222 Problem Prove that $p \leftrightarrow q$ and $(p \wedge q) \vee (\neg p \wedge \neg q)$ are logically equivalent without using truth tables.

223 Problem Use a set containment proof to prove that if A and B are sets, then $A - B = A \cup \overline{B}$.

224 Problem Prove that if A, B and C are sets, then $(A \cap B \cap C) \subseteq (A \cap B)$ using a set containment proof.

225 Problem Draw a Venn diagram showing $A \cap (B \cup C)$, where A, B, and C are sets.

226 Problem Rusty has 20 marbles of different colours: black, blue, green, and yellow. Seventeen of the marbles are not green, five are black, and 12 are not yellow. How many blue marbles does he have?

227 Problem Let `A` and `B` be `TreeSets` (See Example 168).

1. The method `addAll(TreeSet other)` *adds all of the elements in* `other` *to this set if they're not already present.* What is the result of `A.addAll(B)` (in terms of `A` and `B` and set operators)?

2. The method `removeAll(TreeSet other)` *removes from this set all of its elements that are contained in* `other`. What is the result of `A.removeAll(B)` (in terms of `A` and `B` and set operators)?

3. Write `A.contains(x)` using set notation.

228 Problem

Consider the relation $R = \{(1,2),(1,3),(3,5),(2,2),(5,5),(5,3),(2,1),(3,1)\}$ on the set $\{1,2,3,4,5\}$. Is R reflexive? symmetric? antisymmetric? transitive? an equivalence relation?

229 Problem Let X be the set of all people, and define the following.
$R_1 = \{(a,b) \in X^2 | a \text{ is taller than } b\}$
$R_2 = \{(a,b) \in X^2 | a \text{ is at least as tall as } b\}$
$R_3 = \{(a,b) \in X^2 | a \text{ and } b \text{ are the same height }\}$
$R_4 = \{(a,b) \in X^2 | a \text{ and } b \text{ have the same last name }\}$
$R_5 = \{(a,b) \in X^2 | a \text{ and } b \text{ have last names that start with the same letter }\}$
Which of these is are equivalence relations? Prove it.

230 Problem Define three different equivalence relations on the set of all TV shows. For each, give examples of the equivalence classes, including one representative from each. Prove that each is an equivalence relation.

231 Problem Let $A = \{1,2,\ldots,n\}$. Let R be the relation on $P(A)$ (the power set of A) such that $a,b \in P(A)$ are related iff $|a| = |b|$. Prove that R is an equivalence relation. What are the equivalence classes of R?

232 Problem The class `Relation` is a partial implementation of a relation on a set A. It has a list of `Element` objects.

- An `Element` stores an ordered pair from A. `Element` has methods `getFrom()` and `getTo()` (using the language of the directed graph representation). So if an `Element` is storing (a,b), `getFrom()` returns a and `getTo()` returns b. The constructor `Element(Object a, Object b)` creates an element (a,b).

- The `Relation` class has methods like `areRelated(Object a,Object b)`, `getElements( )`, and `getUniverse( )`.
- Methods in the `Relation` class can use `for(Element e : getElements())` to iterate over elements of the relation.
- Similarly, the loop `for(Object a : getUniverse())` iterates over the elements of A.

Given all of this, implement the following methods in the `Relation` class:

1. `isReflexive()`
2. `isSymmetric()`
3. `isAntiSymmetric()`

233 Problem Draw a table to represent the following Boolean expressions

a. $\overline{x}+y$

b. $xy+\overline{(xy)}+z$

c. $(x+\overline{z})y$

234 Problem Find the sum-of-products expansion for each of the Boolean expressions from Problem 233

235 Problem Find the product-of-sums expansion for each of the Boolean expressions from Problem 233

236 Problem Express $x+y+z$ using only the Boolean operators $\cdot$ and $\overline{\ }$.

237 Problem The *NAND* of p and q, denoted by $p|q$, is the proposition "not both p and q". The NAND of p and q is false when p and q are both true and true otherwise.

a. Draw a truth table for *NAND*

b. Express $x|y$ using $\vee$, $\wedge$, and/or $\neg$ (you may not need all of them).

c. Express xy using only NAND.

d. Express $\overline{x}+y$ using only NAND.

238 Problem The *NOR* of p and q, denoted by $p\downarrow q$, is the proposition "neither p nor q". The NOR of p and q is true when p and q are both false and false otherwise. Express each of the following using only the NOR operator.

a. Draw a truth table for *NOR*

b. Express $x\downarrow y$ using $\vee$, $\wedge$, and/or $\neg$ (you may not need all of them).

c. Express xy using only NOR.

d. Express $\overline{x}+y$ using only NOR.

239 Problem Show that the NAND operator ($|$) is functionally complete given the fact that $\{\cdot,\overline{\ }\}$ (that is, the AND and NEGATION operators) is universally complete. (Hint: All you need to do is show how to implement each of AND and NEGATION using NAND.)

240 Problem A set of logical operators is *functionally complete* if any possible operator can be implemented using only operators from that set. It turns out that $\{\neg,\wedge\}$ is functionally complete. So is $\{\neg,\vee\}$. To show that a set if functionally complete, all one needs to do is show how to implement all of the operators from another functionally complete set. Given this,

a. Show that $\{\downarrow\}$ is functionally complete.

b. Show that $\{|\}$ is functionally complete.

241 Problem You need to settle an argument between your boss (who can fire you) and your professor (who can fail you). They are trying to decide who to invite to the Young Accountants Volleyball League. They want to invite freshmen who are studying accounting and are at least 6 feet tall. They have a list of all students.

a. Your boss says they should make a list of all freshmen, a list of all accounting majors, and a list of everyone at least 6 feet tall. They should then combine the lists (removing duplicates) and invite those on the combined list. Is he correct? Explain. If he is not correct, describe in the simplest possible terms who ends up on his guest list.

b. Your professor says they should make a list of everyone who is not a freshman, a list of everyone who does not do accounting, and a list of everyone who is under 6 feet tall. They should make a fourth list that contains everyone who is on all three of the prior lists. Finally, they should remove from the original list everyone on this fourth list, and invite the remaining students. Is he correct? Explain. If he is not correct, describe in the simplest possible terms who ends up on his guest list.

c. Give a simple description of how the guest list should be created.

242 Problem Explain whether the following assertion is true and negate it without using the negation symbol $\neg$:

$$\forall n \in \mathbb{N}\ \exists m \in \mathbb{N}\ \left(n^2 > 4n \to 2^n > 2^m + 10\right)$$

243 Problem Prove Theorem 183. (Note: This is an if and only if proof, so you need to prove both ways.)

244 Problem You are helping a friend debug the code below. He tells you "The code in the if statement never executes. I have tried it for `x=2`, `x=4`, and even `x=-1`, and it never gets to the code inside the if statement."

```
if((x%2==0 && x<0) || !(x%2==0 || x<0)) {
    // Do something.
}
```

1. Is he correct that the code inside the if statement does not execute for his chosen values? Justify your answer.

2. Under what conditions, if any, will the code in the if statement execute? Be specific and complete.

245 Problem Simplify the following code as much as possible.

```
    if (x>0) {
        if(x<y || x>0) {
            x=y;
        }
    }
```

246 Problem Simplify the following code as much as possible:

```
if(x<=0 && x>0) {
    doSomething();
} else {
    doAnotherThing();
}
```

247 Problem Consider the following code.

```
boolean notBothZero(int x, int y) {
    if(!(x==0 && y==0)) {
        return true;
    } else {
        return false;
    }
}
boolean unknown1(int x, int y) {
    if(x!=0 && y!=0) {
        return true;
    } else {
        return false;
    }
}
boolean unknown2(int x, int y) {
    if(x!=0 || y!=0) {
        return true;
    } else {
        return false;
    }
}
```

Is either `unknown1` or `unknown2` (or both) equivalent to `notBothZero`? Prove it.

248 Problem

Simplify the following code as much as possible. (It can be simplified into a single if statement that is about as complex as the original outer if statement).

```
    if ( (!x.size()<=0  && x.get(0)!=11) || x.size()>0 ) {
       if ( !(x.get(0)==11 && (x.size()>13 || x.size()<13) )
          && (x.size()>0 || x.size()==13) ) {
                // Do a few things.
       }
    }
```

249 Problem The following method returns true if and only if none of the entries of the array are 0:

```
boolean noZeroElements(int[] a, int n) {
    for(int i=0;i<n;i++) {
        if(a[i] == 0 )
            return false;
    }
    return true;
}
```

The two methods below implement this idea for two arrays. Assume `list1` and `list2` have the same size for both of these methods.

```
boolean unknown1(int[] list1, int[] list2, int n) {
    for(int i=0;i<n);i++) {
        if( list1[i]==0 && list2[i]==0 )
            return false;
    }
    return true;
}

boolean unknown2(int[] list1, int[] list2, int n) {
    if(noZeroElements(list1, n)) {
        return true;
    } else if(noZeroElements(list2, n) {
        return true;
    } else {
        return false;
    }
}
```

1. What is `unknown1` determining? (Give answer in terms of `list1` and `list2`.)
2. What is `unknown2` determining? (Give answer in terms of `list1` and `list2`.)
3. Prove or disprove that `unknown1` and `unknown2` are determining the same thing.

Chapter 4

Sequences and Summations

4.1 Sequences

250 Definition A *sequence* of real numbers is a function whose domain is the set of natural numbers and whose output is a subset of the real numbers. We usually denote a sequence by one of the notations

$$a_0, a_1, a_2, \ldots$$

or

$$\{a_n\}_{n=0}^{+\infty}$$

or

$$\{a_n\}.$$

The last notation is just a shorthand for the second notation.

☞ *Sometimes we may not start at $n = 0$. In that case we may write*

$$a_m, a_{m+1}, a_{m+2}, \ldots,$$

or

$$\{a_n\}_{n=m}^{+\infty},$$

where m is a non-negative integer. Most sequences we will deal with will start with $m = 0$ or $m = 1$.

We will be mostly interested in two types of sequences. The first type are sequences that have an explicit formula for their n-th term. They are said to be *closed form*.

251 Example Let $a_n = 1 - \frac{1}{2^n}, n = 0, 1, \ldots$. Then $\{a_n\}_{n=0}^{+\infty}$ is a sequence for which we have an explicit formula for the n-th term. The first five terms are

$$\begin{array}{lllll} a_0 & = & 1 - \frac{1}{2^0} & = & 0, \\ a_1 & = & 1 - \frac{1}{2^1} & = & \frac{1}{2}, \\ a_2 & = & 1 - \frac{1}{2^2} & = & \frac{3}{4}, \\ a_3 & = & 1 - \frac{1}{2^3} & = & \frac{7}{8}, \\ a_4 & = & 1 - \frac{1}{2^4} & = & \frac{15}{16}. \end{array}$$

If you can't work out the last step of each of these, you need to brush up on your algebra skills.

The second type of sequence are defined recursively. That is, each term is based on previous term(s). We call these *recurrence relations*.

252 Example Let

$$x_0=1,\ \ x_n=\left(1+\frac{1}{n}\right)x_{n-1},\quad n=1,2,\ldots.$$

Then $\{x_n\}_{n=0}^{+\infty}$ is a sequence recursively defined. The terms $x_1,x_2,\ldots,x_5$ are

$$\begin{array}{lllll} x_1 & = & \left(1+\frac{1}{1}\right)x_0 & = & 2, \\ x_2 & = & \left(1+\frac{1}{2}\right)x_1 & = & 3, \\ x_3 & = & \left(1+\frac{1}{3}\right)x_2 & = & 4, \\ x_4 & = & \left(1+\frac{1}{4}\right)x_3 & = & 5, \\ x_5 & = & \left(1+\frac{1}{5}\right)x_4 & = & 6. \end{array}$$

Notice that in the previous example, we gave an explicit definition of x_0. This is called an *initial condition*. Every recurrence relation needs one or more initial conditions. Without them, we have an abstract definition of a sequence, but cannot compute any values since there is no "starting point."

When we find an explicit formula (or closed formula) for a recurrence relation, we say we have *solved* the recurrence relation.

253 Example It seems relatively clear that $x_n=n+1$ is a solution for x_n from Example 252.

It is important to be careful about jumping to conclusions too quickly when solving recurrence relations.[1] Although it turns out that in the previous example, $x_n=n+1$ is the correct closed form (we will prove it shortly), just because it works for the first 6 terms does not necessarily imply that the pattern continues.

254 Example Define $\{a_n\}$ by $a(0)=1$, $a(1)=2$, and

$$a(n)=\left\lfloor\frac{1+\sqrt{5}}{2}\times a(n-1)\right\rfloor+a(n-2)$$

for $n\geq 2$. Let's try to figure out a closed form for $a(n)$. Then we can see that

$$\begin{array}{lllllll} a_2 & = & \left\lfloor\frac{1+\sqrt{5}}{2}\times a(1)\right\rfloor+a(0) & = & \left\lfloor\frac{1+\sqrt{5}}{2}\times 2\right\rfloor+1 & = & 4 \\ a_3 & = & \left\lfloor\frac{1+\sqrt{5}}{2}\times a(2)\right\rfloor+a(1) & = & \left\lfloor\frac{1+\sqrt{5}}{2}\times 4\right\rfloor+2 & = & 8 \\ a_4 & = & \left\lfloor\frac{1+\sqrt{5}}{2}\times a(3)\right\rfloor+a(2) & = & \left\lfloor\frac{1+\sqrt{5}}{2}\times 8\right\rfloor+2 & = & 16 \end{array}$$

You should verify these with a calculator. At this point it seems relatively clear that $a_n=2^n$. However,

$$a_5=\left\lfloor\tfrac{1+\sqrt{5}}{2}\times a(4)\right\rfloor+a(3)=\left\lfloor\tfrac{1+\sqrt{5}}{2}\times 16\right\rfloor+8=33$$

so the solution that seems "obvious" turns out to not be correct. We won't give the actual solution since the point of this example is to demonstrate that just because a pattern holds for the first several terms of a sequence, it does not guarantee that it holds for the whole sequence.

[1] These comments apply to other problems that involve seeing a pattern and finding an explicit formula.

Generally speaking, you need to *prove* that the closed form is correct. One way to do this is to plug it back into the recursive definition. It usually works best to plug it into the right hand side and use your algebra prowess to simplify it to the left hand side (which is just x_n or whatever the sequence is called).

255 Example Prove that $x_n = n+1$ is a solution to the recurrence relation given by

$$x_0 = 1, \quad x_n = \left(1+\frac{1}{n}\right)x_{n-1}, \quad n = 1,2,\ldots.$$

Proof: If $x_n = n+1$ for $n \geq 0$, then

$$\begin{aligned}\left(1+\frac{1}{n}\right)x_{n-1} &= \left(1+\frac{1}{n}\right)n\\ &= \left(\frac{n+1}{n}\right)n\\ &= n+1\\ &= x_n\end{aligned}$$

so $x_n = n+1$ is a solution to the recurrence relation. □

A more complete discussion of solving recurrences appears in Chapter 6.

256 Example The *Fibonacci numbers* are a sequence of numbers that is of interest in various mathematical and computing applications. They are defined using the following recurrence relation:[2]

$$f_n = \begin{cases} 0 & \text{if } n\text{=}0\\ 1 & \text{if } n\text{=}1\\ f_{n-1}+f_{n-2} & \text{if } n>1\end{cases}$$

The first few are:

$$\begin{aligned}f_0 &= 0\\ f_1 &= 1\\ f_2 &= f_1+f_0 = 1+0 = 1\\ f_3 &= f_2+f_1 = 1+1 = 2\\ f_4 &= f_3+f_2 = 2+1 = 3\\ f_5 &= f_4+f_3 = 3+2 = 5\\ f_6 &= f_5+f_4 = 5+3 = 8\\ f_7 &= f_6+f_5 = 8+5 = 13\end{aligned}$$

257 Definition A sequence $\{a_n\}_{n=0}^{+\infty}$ is said to be [3]

- *increasing* if $a_n \leq a_{n+1}$ $\forall n \in \mathbb{N}$
- *strictly increasing* if $a_n < a_{n+1}$ $\forall n \in \mathbb{N}$

[2] In the remainder of the book, when you see f_k, you should assume it refers to the k-th Fibonacci number unless otherwise specified.

[3] Some people call these sequences *non-decreasing*, *increasing*, *non-increasing*, and *decreasing*, respectively.

- *decreasing* if $a_n \geq a_{n+1}\ \forall n \in \mathbb{N}$
- *strictly decreasing* if $a_n > a_{n+1}\ \forall n \in \mathbb{N}$

A sequence is called *monotonic* if is any of these.

258 Example Recall that $0! = 1$, $1! = 1$, $2! = 1 \cdot 2 = 2$, $3! = 1 \cdot 2 \cdot 3 = 6$, etc. Prove that the sequence $x_n = n!, n = 0, 1, 2, \ldots$ is strictly increasing for $n \geq 1$.

Proof: For $n > 1$ we have

$$x_n = n! = n(n-1)! = nx_{n-1} > x_{n-1},$$

since $n > 1$. This proves that the sequence is strictly increasing. □

259 Example Prove that the sequence $x_n = 2 + \frac{1}{2^n}, n = 0, 1, 2, \ldots$ is strictly decreasing.

Proof: We have

$$\begin{aligned} x_{n+1} - x_n &= \left(2 + \frac{1}{2^{n+1}}\right) - \left(2 + \frac{1}{2^n}\right) \\ &= \frac{1}{2^{n+1}} - \frac{1}{2^n} \\ &= -\frac{1}{2^{n+1}} \\ &< 0. \end{aligned}$$

Thus, $x_{n+1} - x_n < 0$, so $x_{n+1} < x_n$, i.e., the sequence is strictly decreasing. □

260 Example Prove that the sequence $x_n = \frac{n^2+1}{n}, n = 1, 2, \ldots$ is strictly increasing.

Proof: First notice that $\frac{n^2+1}{n} = n + \frac{1}{n}$. Now,

$$\begin{aligned} x_{n+1} - x_n &= \left(n + 1 + \frac{1}{n+1}\right) - \left(n + \frac{1}{n}\right) \\ &= 1 + \frac{1}{n+1} - \frac{1}{n} \\ &= 1 - \frac{1}{n(n+1)} \\ &> 0, \end{aligned}$$

the last step since $1/n(n+1) < 1$ when $n \geq 1$. Therefore, $x_{n+1} - x_n > 0$, so $x_{n+1} > x_n$, i.e., the sequence is strictly increasing. □

261 Definition A sequence $\{x_n\}_{n=0}^{+\infty}$ is said to be *bounded* if eventually the absolute value of every term is less than or equal to a certain positive constant. The sequence that is not bounded is called *unbounded*.

Proving that a sequence is unbounded involves showing that for any arbitrarily large positive real number, we can always find a term whose absolute value is greater than this real number.

262 Example Prove that the sequence $x_n = n!, n = 0, 1, 2, \ldots$ is unbounded.

Proof: Let $M > 2$ be a large real number. Then $(\lceil M \rceil - 1)! > 1$. Therefore $\{x_n\}$ is unbounded since

$$x_{\lceil M \rceil} = \lceil M \rceil ! = \lceil M \rceil (\lceil M \rceil - 1)! > \lceil M \rceil > M.$$

□

263 Example Prove that the sequence $a_n = \dfrac{n+1}{n}$, $n = 1, 2, \ldots$, is bounded.

Proof: Notice that if $n \geq 1$, then $1/n \leq 1$. Therefore

$$a_n = \frac{n+1}{n} = 1 + \frac{1}{n} \leq 2,$$

so a_n is bounded. □

264 Definition A *geometric progression* is a sequence of the form

$$a,\ ar, ar^2,\ ar^3,\ ar^4, \ldots,$$

where a (the *initial term*) and r (the *common ratio*) are real numbers. That is, a geometric progression is a sequences in which every term is produced from the preceding one by multiplying it by a fixed number.

Notice that the n-th term is ar^{n-1}. If $a = 0$ then every term is 0. If $ar \neq 0$, we can find r by dividing any term by the previous term.

265 Example Find the 35-th term of the geometric progression

$$\frac{1}{\sqrt{2}},\ -2,\ \frac{8}{\sqrt{2}}, \ldots.$$

Solution: The common ratio is $-2 \div \frac{1}{\sqrt{2}} = -2\sqrt{2}$. Hence the 35-th term is $\frac{1}{\sqrt{2}}(-2\sqrt{2})^{34} = \frac{2^{51}}{\sqrt{2}} = 1125899906842624\sqrt{2}$.

266 Example The fourth term of a geometric progression is 24 and its seventh term is 192. Find its second term.

Solution: We are given that $ar^3 = 24$ and $ar^6 = 192$, for some a and r. Clearly, $ar \neq 0$, and so we find

$$\frac{ar^6}{ar^3} = r^3 = \frac{192}{24} = 8,$$

whence $r = 2$. Now, $a(2)^3 = 24$, giving $a = 3$. The second term is thus $ar = 6$.

267 Definition An *arithmetic progression* is a sequence of the form

$$a,\ a+d, a+2d,\ a+3d,\ a+4d, \ldots,$$

where a (the *initial term*) and r (the *common difference*) are real numbers. That is, an arithmetic progression is a sequences in which every term is produced from the preceding one by adding a fixed number.

268 Example If $s_n = 3n - 7$, then $\{s_n\}$ is an arithmetic progression with $a = -7$ and $d = 3$ (assuming we begin with s_0).

☞ *Notice that geometric progressions are essentially a discrete version of an exponential function and arithmetic progressions are a discrete version of a linear function. One consequence of this is that a sequence cannot be both of these unless is is the sequence $a, a, a, \ldots$ for some a.*

269 Example Consider the sequence $4, 7, 10, 13, 16, 19, 22, \ldots$. Assuming the pattern continues, is this a geometric progression? Is it an arithmetic progression?

> **Solution:** It is easy to see that each term is 3 more than the previous term. Thus, this is an arithmetic progression with $a = 4$ and $d = 3$. Clearly it is therefore not geometric.

Exercises

270 Problem Find the first five terms of the following sequences.

1. $x_n = 1 + (-2)^n, n = 0, 1, 2, \ldots$
2. $x_n = 1 + (-\frac{1}{2})^n, n = 0, 1, 2, \ldots$
3. $x_n = n! + 1, n = 0, 1, 2, \ldots$
4. $x_n = \dfrac{1}{n! + (-1)^n}, n = 2, 3, 4, \ldots$
5. $x_n = \left(1 + \dfrac{1}{n}\right)^n, n = 1, 2, \ldots,$

271 Problem Decide whether the following sequences are eventually monotonic or non-monotonic. Determine whether they are bounded or unbounded.

1. $x_n = n$, $n = 0, 1, 2, \ldots$
2. $x_n = (-1)^n n$, $n = 0, 1, 2, \ldots$
3. $x_n = \dfrac{1}{n!}$, $n = 0, 1, 2, \ldots$
4. $x_n = \dfrac{n}{n+1}$, $n = 0, 1, 2, \ldots$
5. $x_n = n^2 - n$, $n = 0, 1, 2, \ldots$
6. $x_n = (-1)^n$, $n = 0, 1, 2, \ldots$
7. $x_n = 1 - \dfrac{1}{2^n}$, $n = 0, 1, 2, \ldots$
8. $x_n = 1 + \dfrac{1}{2^n}$, $n = 0, 1, 2, \ldots$

272 Problem Find the 17-th term of the geometric sequence

$$-\frac{2}{3^{17}}, \frac{2}{3^{16}}, -\frac{2}{3^{15}}, \cdots.$$

273 Problem The 6-th term of a geometric progression is 20 and the 10-th is 320. Find the absolute value of its third term.

Answers

270 (1) 2, -1, 5, -7, 17; (2) 2, $1/2$, $5/4$, $7/8$, $17/16$; (3) 2, 2, 3, 7, 25; (4) $1/3$, $1/5$, $1/25$, $1/119$, $1/721$; (5) 2, $9/4$, $64/27$, $625/256$, $7776/3125$

271 (1) Strictly increasing, unbounded (2) non-monotonic, unbounded (3) strictly decreasing, bounded (4) strictly increasing, bounded (5) strictly increasing, unbounded, (6) non-monotonic, bounded, (7) strictly increasing, bounded, (8) strictly decreasing, bounded

272 $-\dfrac{2}{3}$

273 One is given that $ar^5 = 20$ and $ar^9 = 320$. Hence $|ar^2| = \frac{5}{2}$

4.2 Sums and Products

There is often a need to add or multiply terms from a sequence. The following notation is very helpful.

274 Definition Let $\{a_n\}$ be a sequence. Then for $1 \leq m \leq n$, we define

$$\sum_{k=m}^{n} a_k = a_m + a_{m+1} + \cdots + a_n.$$

$$\prod_{k=m}^{n} a_k = a_m a_{m+1} \cdots a_n.$$

We will often write these as

$$\sum_{m \leq k \leq n} a_k \qquad \text{and} \qquad \prod_{m \leq k \leq n} a_k$$

As with sequences, we are often interested in obtaining *closed forms* for a sum or product. There are many techniques to do so. We will present just a few.

Perhaps the simplest cases are when we have a sum/product of the form

$$(a_2 - a_1) + (a_3 - a_2) + \cdots + (a_n - a_{n-1}) = a_n - a_1,$$

and

$$\frac{a_2}{a_1} \cdot \frac{a_3}{a_2} \cdots \frac{a_n}{a_{n-1}} = \frac{a_n}{a_1},$$

in which case we say that the sum or the product *telescopes*. The trick can be to recognize when a sum/product telescopes.

In the following example we develop a formula for a geometric series by doing a little algebra so we can use the telescoping idea.

275 Theorem (Finite Geometric Series) Let $x \neq 1$. Then $\displaystyle\sum_{0 \leq k \leq n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1 - x}$.

Proof: First, let $S = \sum_{0\leq k\leq n} x^k$. Then

$$xS = x \sum_{0\leq k\leq n} x^k = \sum_{0\leq k\leq n} x^{k+1} = \sum_{1\leq k\leq n+1} x^k.$$

So

$$\begin{aligned} xS - S &= \sum_{1\leq k\leq n+1} x^k - \sum_{0\leq k\leq n} x^k \\ &= (x_1 + x_2 + \ldots + x_n + x_{n+1}) - (x_0 + x_1 + \ldots + x_n) \\ &= x^{n+1} - x^0 = x^{n+1} - 1. \end{aligned}$$

So we have $S(x-1) = x^{n+1} - 1$, so $S = \frac{x^{n+1}-1}{x-1}$, since $x \neq 0$. □

Putting $N = n+1$ in the above formula, we are provided with the following factorization, which is useful in certain situations.

$$x^N - 1 = (x-1)(x^{N-1} + x^{N-2} + \cdots + x + 1). \tag{4.1}$$

For example,

$$x^2 - 1 = (x-1)(x+1), \quad x^3 - 1 = (x-1)(x^2+x+1), \text{ and } \quad x^4 - 1 = (x-1)(x^3+x^2+x+1).$$

☞ *More important than remembering the* formula *above is remembering the* method *of how this formula was obtained. As you work with sums more, you will start to see some of the tricks that come in handy often. There may be "one ring to rule them all", but there is not one technique that always works when finding closed forms for sums.*

We just saw a technique that multiplies a sum by x, subtracts, and then does some algebra. The next example is just a special case of this for $x = 2$.

276 Example Find the sum

$$2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \cdots + 2^n.$$

Solution: We could just use the formula we computed above, but that would be boring. Instead, let's let

$$S = 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^n.$$

Then $2S = 2^1 + 2^2 + 2^3 + \cdots + 2^{n+1}$. Notice these have most of the same terms, except S has 2^0 and $2S$ has 2^{n1}. Therefore,

$$\begin{array}{rcl} S = 2S - S &=& \qquad\quad (2^1 + 2^2 + 2^3 + \cdots + 2^n + 2^{n+1}) \\ && -(2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^n) \\ &=& 2^{n+1} - 2^0 \end{array}$$

We have already implicitly used the following fact.

277 Theorem If x_n is a sequence and a is a real number, then

$$\sum_{k=m}^{n} a \cdot x_k = a \sum_{k=m}^{n} x_k.$$

Proof: This follows immediately from the distributive law. □

This and Theorem 275 imply that when $r \neq 1$,

$$\sum_{k=0}^{n} ar^k = \frac{a - ar^{k+1}}{1-r}.$$

Nevertheless, we will prove it from scratch.

278 Theorem If $r \neq 1$, then $\sum_{k=0}^{n} ar^k = \frac{a - ar^{k+1}}{1-r}$.

Proof: Let $S = a + ar + ar^2 + \cdots + ar^n$. Then $rS = ar + ar^2 + \cdots + ar^{n+1}$. Hence

$$\begin{aligned} S - rS &= a + ar + ar^2 + \cdots + ar^n - ar - ar^2 - \cdots - ar^{n+1} \\ &= a - ar^{n+1}. \end{aligned}$$

From this we deduce that

$$S = \frac{a - ar^{n+1}}{1-r},$$

that is,

$$\sum_{k=0}^{n} ar^k = \frac{a - ar^{n+1}}{1-r}$$

□

Notice that if $|r| < 1$ then r^n gets closer to 0 the larger n gets. More formally, if $|r| < 1$, $\lim_{n\to\infty} r^n = 0$. This implies that if $|r| < 1$,

$$a + ar + ar^2 + \cdots = \frac{a}{1-r}.$$

In other words, the infinite sum of all of the terms in a geometric sequence is $a/(1-r)$ if $|r| < 1$.

279 Example A fly starts at the origin and goes 1 unit up, $1/2$ unit right, $1/4$ unit down, $1/8$ unit left, $1/16$ unit up, etc., *ad infinitum*. In what coordinates does it end up?

Solution: Its x coordinate is

$$\frac{1}{2} - \frac{1}{8} + \frac{1}{32} - \cdots = \frac{\frac{1}{2}}{1 - \frac{-1}{4}} = \frac{2}{5}.$$

Its y coordinate is

$$1 - \frac{1}{4} + \frac{1}{16} - \cdots = \frac{1}{1 - \frac{-1}{4}} = \frac{4}{5}.$$

Therefore, the fly ends up in $(\frac{2}{5}, \frac{4}{5})$.

Another trick to simplify sums involves adding the terms in a sum twice, typically in a different order, and then dividing the result by two. This is known as Gauss' trick.

280 Corollary $\displaystyle\sum_{1\leq k\leq n} k = \frac{n(n+1)}{2}$.

Proof: If

$$S = 1+2+3+\cdots+n$$

then

$$S = n+(n-1)+\cdots+1.$$

Adding these two quantities,

$$\begin{array}{rcccccccc} S & = & 1 & + & 2 & + & \cdots & + & n \\ S & = & n & + & (n-1) & + & \cdots & + & 1 \\ \hline 2S & = & (n+1) & + & (n+1) & + & \cdots & + & (n+1) \\ & = & n(n+1), & & & & & & \end{array}$$

since there are n summands. This gives $S = \dfrac{n(n+1)}{2}$, as was to be proved. □

Here are a few more useful summations that come up often. There are various ways of proving each of these. For now, we will provide the results without proof. We will return to some of them in the section on Induction since that is perhaps the easiest way to prove many of them.

$$\begin{aligned} \sum_{1\leq k\leq n} k^2 &= \frac{n(n+1)(2n+1)}{6} \\ \sum_{1\leq k\leq n} k^3 &= \frac{n^2(n+1)^2}{4} \\ \sum_{2\leq k\leq n} \frac{1}{(k-1)k} &= \frac{1}{1\cdot 2}+\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}+\cdots+\frac{1}{(n-1)\cdot n} = \frac{n-1}{n} \end{aligned}$$

The following infinite sums are sometimes useful.

281 Theorem The following expansions hold:

$$\begin{aligned} \frac{1}{1-x} &= \sum_{n=0}^{\infty} x^n = 1+x+x^2+x^3+\cdots, \quad |x|<1 \\ \frac{1}{(1-x)^2} &= \sum_{n=0}^{\infty} nx^{n-1} = 1+x+x^2+x^3+\cdots, \quad |x|<1 \\ \sin x &= \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\cdots+(-1)^n\frac{x^{2n+1}}{(2n+1)!}+\cdots, \quad x\in\mathbb{R} \\ \cos x &= \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1-\frac{x^2}{2!}+\frac{x^4}{4!}-\cdots+(-1)^n\frac{x^{2n}}{(2n)!}+\cdots, \quad x\in\mathbb{R} \\ e^x &= \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots+\frac{x^n}{n!}+\cdots, \quad x\in\mathbb{R} \\ \log(1+x) &= \sum_{n=1}^{\infty} \frac{(-1)^{n+1}x^n}{n} = x-\frac{x^2}{2}+\frac{x^3}{3}-\cdots+(-1)^{n+1}\frac{x^n}{n}+\cdots, \quad |x|<1 \\ (1+x)^\tau &= \sum_{n=0}^{\infty} \binom{\tau}{n} x^n = 1+\tau x+\frac{\tau(\tau-1)}{2!}x^2+\cdots+\frac{\tau(\tau-1)(\tau-2)\cdots(\tau-n+1)}{n!}x^n+\cdots, |x|<1. \end{aligned}$$

Exercises

282 Problem Find the sum of the following geometric series.

1. $$1+3+3^2+3^3+\cdots+3^{49}.$$

2. If $y\neq 1$, $$1+y+y^2+y^3+\cdots+y^{100}.$$

3. If $y\neq 1$, $$1-y+y^2-y^3+y^4-y^5+\cdots-y^{99}+y^{100}.$$

4. If $y\neq 1$, $$1+y^2+y^4+y^6+\cdots+y^{100}.$$

283 Problem Find the sum of all the integers from 1 to 1000 inclusive, which are not multiples of 3 or 5.

284 Problem Find the sum of all integers between 1 and 100 that leave remainder 2 upon division by 6.

285 Problem Find a closed formula for

$$D_n = 1-2+3-4+\cdots+(-1)^{n-1}n.$$

286 Problem Find a closed form for $\sum_{1\leq k\leq n} 3^k$.

287 Problem Let $n\geq 1$. Find a closed form for $\sum_{0\leq k\leq n}\binom{n}{k}(-1)^k$.

288 Problem Find a closed form for $\sum_{1\leq k\leq n}\binom{n}{k}3^k$.

289 Problem Evaluate the double sum $\sum_{1\leq i\leq n}\sum_{1\leq k\leq n} 1$.

290 Problem Evaluate the double sum $\sum_{1\leq i\leq n}\sum_{1\leq k\leq i} 1$.

291 Problem Evaluate the double sum $\sum_{1\leq i\leq n}\sum_{1\leq k\leq i} k$.

292 Problem Evaluate the double sum $\sum_{1\leq i\leq n}\sum_{1\leq k\leq n} ik$.

293 Problem Factor

$$1+x+x^2+\cdots+x^{80}$$

as a polynomial with integer coefficients.

294 Problem Obtain a closed formula for $\sum_{1\leq k\leq n} k\cdot k!$. Hint: $(k+1)! = (k+1)k!$.

295 Problem A colony of amoebas[4] is put in a glass at 2 : 00 PM. One second later each amoeba divides in two. The next second, the present generation divides in two again, etc.. After one minute, the glass is full. When was the glass half-full?

[4] Why are amoebas bad mathematicians? Because they divide to multiply!

Answers

282 (1) $\frac{3^{50}-1}{2} = 358948993845926294385124$, (2) $\frac{1-y^{101}}{1-y}$, (3) $\frac{1+y^{101}}{1+y}$, (4) $\frac{1-y^{102}}{1-y^2}$

283 We compute the sum of all integers from 1 to 1000 and weed out the sum of the multiples of 3 and the sum of the multiples of 5, but put back the multiples of 15, which we have counted twice. Put

$$A_n = 1+2+3+\cdots+n,$$

$$B = 3+6+9+\cdots+999 = 3A_{333},$$

$$C = 5+10+15+\cdots+1000 = 5A_{200},$$

$$D = 15+30+45+\cdots+990 = 15A_{66}.$$

The desired sum is

$$\begin{array}{rcl} A_{1000}-B-C+D & = & A_{1000}-3A_{333}-5A_{200}+15A_{66} \\ & = & 500500-3\cdot 55611-5\cdot 20100+15\cdot 2211 \\ & = & 266332. \end{array}$$

284 We want the sum of the integers of the form $6r+2, r=0,1,\ldots,16$. But this is

$$\sum_{r=0}^{16}(6r+2) = 6\sum_{r=0}^{16} r + \sum_{r=0}^{16} 2 = 6\frac{16(17)}{2} + 2(17) = 850.$$

285 $\dfrac{(2n+1)(-1)^{n+1}+1}{4}$.

286 Use the same method as in theorem 275: put

$$S = 3+3^2+\cdots+3^n.$$

Then

$$3S = 3^2+3^3+\cdots+3^n+3^{n+1}.$$

Subtracting,

$$3S-S = (3^2+3^3+\cdots+3^n+3^{n+1})-(3+3^2+\cdots+3^n) = 3^{n+1}-3.$$

The answer is $\dfrac{3^{n+1}-3}{2}$.

287 By the binomial theorem, $0 = (1-1)^n = \sum_{0\le k\le n}\binom{n}{k}(-1)^k$.

288 By the binomial theorem, $4^n = (1+3)^n = \sum_{0\le k\le n}\binom{n}{k}3^k$, and so $\sum_{1\le k\le n}\binom{n}{k}3^k = 4^n - 1$.

289 We have

$$\sum_{1\le i\le n}\sum_{1\le k\le n} 1 = \sum_{1\le i\le n} n = n^2.$$

290 We have

$$\sum_{1\le i\le n}\sum_{1\le k\le i} 1 = \sum_{1\le i\le n} i = \frac{n(n+1)}{2}.$$

291 We have

$$\sum_{1\le i\le n}\sum_{1\le k\le i} k = \sum_{1\le i\le n}\frac{i(i+1)}{2} = \frac{n(n+1)(n+2)}{6}.$$

292 We have

$$\sum_{1\leq i\leq n}\sum_{1\leq k\leq n} ik = \left(\sum_{1\leq i\leq n} i\right)\left(\sum_{1\leq k\leq n} k\right) = \frac{n^2(n+1)^2}{4}.$$

293 Put $S = 1+x+x^2+\cdots+x^{80}$. Then

$$S - xS = (1+x+x^2+\cdots+x^{80}) - (x+x^2+x^3+\cdots+x^{80}+x^{81}) = 1-x^{81},$$

or $S(1-x) = 1-x^{81}$. Hence

$$1+x+x^2+\cdots+x^{80} = \frac{x^{81}-1}{x-1}.$$

Therefore

$$\frac{x^{81}-1}{x-1} = \frac{x^{81}-1}{x^{27}-1}\cdot\frac{x^{27}-1}{x^{9}-1}\cdot\frac{x^{9}-1}{x^{3}-1}\cdot\frac{x^{3}-1}{x-1}.$$

Thus

$$1+x+x^2+\cdots+x^{80} = (x^{54}+x^{27}+1)(x^{18}+x^{9}+1)(x^{6}+x^{3}+1)(x^{2}+x+1).$$

294 From the hint: $k\cdot k! = (k+1)!-k!$ and we get the telescoping sum

$$\sum_{1\leq k\leq n} k\cdot k! = \sum_{1\leq k\leq n}(k+1)!-k! = (2!-1!)+(3!-2!)+(4!-3!)+\cdots((n+1)!-n!) = (n+1)!-1!.$$

295 At $2:00:59$ PM (the second just before $2:01$ PM.)

Homework

296 Problem Find at least three *different* sequences that begin with 1, 3, 7 whose terms are generated by a simple formula or rule. By different, I mean none of the sequences can have exactly the same terms. In other words, your answer cannot simply be three different ways to generate the same sequence.

297 Problem Let $q(n) = 2q(n-1)+2n+5$, and $q(0) = 0$. Compute $q(1)$, $q(2)$, $q(3)$ and $q(4)$.

298 Problem Compute each of the following:

1. $\sum_{k=5}^{40} k$
2. $\sum_{j=5}^{22}(2^{j+1}-2^{j})$
3. $\sum_{i=1}^{3}\sum_{j=1}^{4} j$

299 Problem Here is a standard interview question for prospective computer programmers: You are given a list of $1,000,001$ positive integers from the set $\{1,2,\ldots,1,000,000\}$. In your list, every member of $\{1,2,\ldots,1,000,000\}$ is listed once, except for x, which is listed twice. How do you find what x is without doing a $1,000,000$ step search?

300 Problem Find a closed formula for

$$T_n = 1^2 - 2^2 + 3^2 - 4^2 + \cdots + (-1)^{n-1}n^2.$$

301 Problem Show that

$$1 + 3 + 5 + \cdots + 2n - 1 = n^2.$$

302 Problem Show that

$$\sum_{k=1}^{n} \frac{k}{k^4 + k^2 + 1} = \frac{1}{2} \cdot \frac{n^2 + n}{n^2 + n + 1}.$$

303 Problem Legend says that the inventor of the game of chess, Sissa ben Dahir, asked the King Shirham of India to place a grain of wheat on the first square of the chessboard, 2 on the second square, 4 on the third square, 8 on the fourth square, etc..

1. How many grains of wheat are to be put on the last (64-th) square?

2. How many grains, total, are needed in order to satisfy the greedy inventor?

3. Given that 15 grains of wheat weigh approximately one gram, what is the approximate weight, in kg, of the wheat needed?

4. Given that the annual production of wheat is 350 million tonnes, how many years, approximately, are needed in order to satisfy the inventor (assume that production of wheat stays constant)?

304 Problem Consider the following function:

```
int ferzle(int n) {
    if(n<=0) {
        return 3;
    } else {
        return ferzle(n-1) + 2;
    }
}
```

1. Determine what `ferzle(n)` returns for $n = 0, 1, 2, 3, 4$.

2. Re-write `ferzle` without using recursion.

305 Problem It is easy to see that we can define $n!$ recursively by defining $0! = 1$, and if $n > 0$, $n! = n \cdot (n-1)!$. Does the following method correctly compute $n!$? If not, state what is wrong with it and fix it.

```
int factorial(int n) {
    return n * factorial(n-1);
    }
}
```

306 Problem A students turned in the code below (which does as its name suggests). I gave them a 'C' on the assignment because although it works, it is very inefficient. Write the 'A' version of the method (in other words, a more efficient version). You can safely assume that $n \geq 1$. Then compute `sumFromOneToN(30)`.

```
int sumFromOneToN(int n) {
    int sum = 0;
    for(int i=1;i<=n;i++) {
        sum = sum + i;
    }
return sum;
}
```

307 Problem A students turned in the code below (which does as its name suggests). I gave them a ‘C’ on the assignment because although it works, it is very inefficient. Write the ‘A’ version of the method (in other words, a more efficient version). You can safely assume that $n, m \geq 1$. Then compute `sumFromMToN(10,50)`.

```
int sumFromMToN(int m, int n) {
    int sum = 0;
    for(int i=1;i<=n;i++) {
        sum = sum + i;
    }
    for(int i=1;i<m;i++) {
        sum = sum - i;
    }
    return sum;
}
```

This page intentionally left blank.

Chapter 5

Algorithm Analysis

In this chapter we take a look at the analysis of algorithms. Before we dive into that topic, we discuss one of the most important tools used in analyzing algorithms–*asymptotic notation*. We end the chapter with a discussion of the growth rates of several common functions.

5.1 Asymptotic Notation

Asymptotic notation is used to express and compare the growth rate of functions. In our case, the functions will represent the running time of algorithms. Since the running time of an algorithms is always nonnegative, and since it simplifies the definitions somewhat, we will define the asymptotic notations in terms of nonnegative functions. We will focus on the most commonly used notations in the analysis of algorithms.

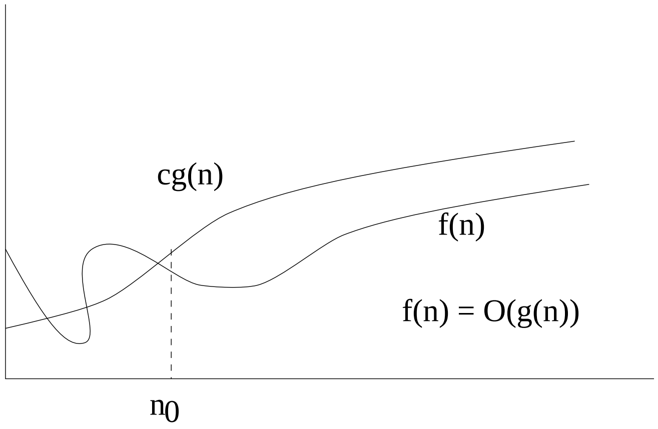
5.1.1 The Notations

308 Definition (Big-O) Let f be a nonnegative function.

We say that $f(n)$ is *Big-O* of $g(n)$, written as $f(n) = O(g(n))$, iff there are positive constants c and n_0 such that

$$f(n) \leq c\,g(n) \text{ for all } n \geq n_0.$$

If $f(n) = O(g(n))$, $f(n)$ grows no faster than $g(n)$. In other words, $g(n)$ is an *asymptotic upper bound* (or just *upper bound*) on $f(n)$.



☞ *The "=" in $f(n) = O(g(n))$ should be read and thought of as "is", not "equals". An alternative notation is to write $f(n) \in O(g(n))$ instead of $f(n) = O(g(n))$. Since $O(g(n))$ is actually the set of all functions that grow no faster than $g(n)$, the set notation is actually in some sense more correct. The "=" notation is used because it comes in handy when doing algebra. You can essentially think of these as being two different notations for the same thing. Similar statements are true for the other asymptotic notations.*

309 Example Prove that $n^2 + n = O(n^3)$.

Solution: Here, we have $f(n) = n^2+n$, and $g(n) = n^3$ Notice that if $n \geq 1$, $n \leq n^3$ and $n^2 \leq n^3$. Therefore,

$$n^2+n \leq n^3+n^3 = 2n^3$$

Thus,

$$n^2+n \leq 2n^3 \text{ for all } n \geq 1$$

Thus, we have shown that $n^2+n = O(n^3)$ by definition of Big-O, with $n_0 = 1$, and $c = 2$.

☞ *Notice that if a and b are real numbers with $a \leq b$, then $n^a \leq n^b$ whenever $n \geq 1$. This fact is used often in these types of proofs.*

Sometimes the easiest way to prove that $f(n) = O(g(n))$ is to take c to be the sum of the positive coefficients of $f(n)$, although this trick doesn't always work. We can usually ignore the negative coefficients, however.[1] We leave it to the reader to figure out why.

310 Example To prove $5n^2-3n+20 = O(n^2)$, we pick $c = 5+20 = 25$. Then if $n \geq n_0 = 1$,

$$5n^2-3n+20 \leq 5n^2+20 \leq 5n^2+20n^2 = 25n^2.$$

Therefore, $5n^2-3n+20 = O(n^2)$.

Things are not always so easy. How would you show that $(\sqrt{2})^{\log n}+\log^2 n+n^4$ is $O(2^n)$? Or that $n^2 = O(n^2-13n+23)$? In general, we simply (or in some cases with much effort) find values c and n_0 that work. This gets easier with practice.

☞ *The values of the constants used in the proofs do not need to be the best possible. So, for instance, if you can show that $f(n) \leq 345g(n)$ for all $n \geq 712$, then $f(n) = O(g(n))$. It doesn't matter whether or not it is actually true that $f(n) \leq 3g(n)$ for all $n \geq 5$.*

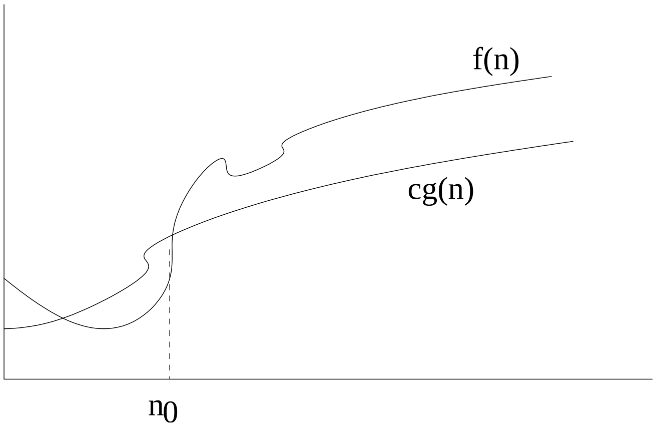
Let's move on to the other two notations.

311 Definition (Big-Omega) Let f and g be a nonnegative function.

We say that $f(n)$ is *Big-Omega* of $g(n)$, written as $f(n) = \Omega(g(n))$, iff there are positive constants c and n_0 such that

$$cg(n) \leq f(n) \text{ for all } n \geq n_0$$

If $f(n) = \Omega(g(n))$, $f(n)$ grows no slower than $g(n)$. In other words, $g(n)$ is an *asymptotic lower bound* (or just *lower bound*) on $f(n)$.



312 Example Prove that $n^3+4n^2 = \Omega(n^2)$.

Proof: Here, we have $f(n) = n^3+4n^2$, and $g(n) = n^2$. It is not too hard to see that if $n \geq 1$,

$$n^2 < n^3 \leq n^3+4n^2$$

Therefore,

$$n^2 \leq n^3+4n^2 \text{ for all } n \geq 1$$

so $n^3+4n^2 = \Omega(n^2)$ by definition of Big-Ω, with $n_0 = 1$, and $c = 1$. □

[1] By "ignore", I do not literally mean ignore. I mean you can easily deal with them in an inequality like in the next example.

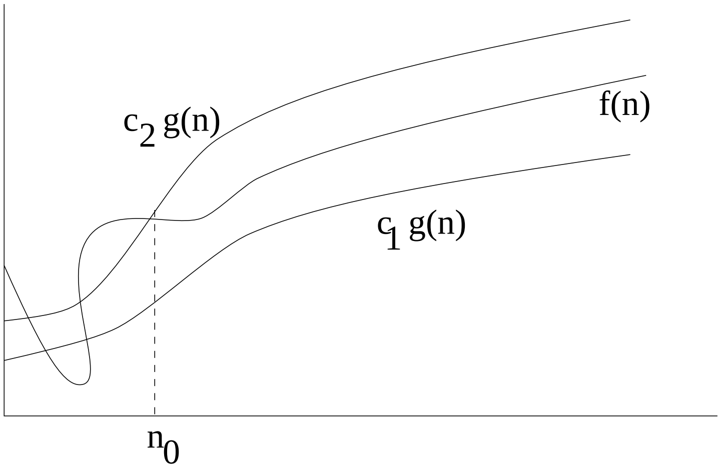
Proving that a $f(n) = \Omega(g(n))$ often requires more thought than proving that $f(n) = O(g(n))$. Quite often, we have to pick $c < 1$. A good strategy is to pick a value of c that you think will work, and determine which value of n_0 is needed. Being able to do some algebra helps. We can sometimes simplify by ignoring the slower growing terms of $f(n)$ with positive coefficients.[2]

313 Definition (Big-Theta) Let f and g be a nonnegative function.

We say that $f(n)$ is *Big-Theta* of $g(n)$, written as $f(n) = \Theta(g(n))$, iff there are positive constants c_1, c_2 and n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0$$

If $f(n) = \Theta(g(n))$, $f(n)$ grows at the same rate as $g(n)$. In other words, $g(n)$ is an *asymptotically tight bound* (or just *tight bound*) on $f(n)$.



314 Example Prove that $n^2 + 5n + 7 = \Theta(n^2)$

Proof: When $n \geq 1$,

$$n^2 + 5n + 7 \leq n^2 + 5n^2 + 7n^2 \leq 13n^2.$$

When $n \geq 0$,

$$n^2 \leq n^2 + 5n + 7$$

Thus,

$$n^2 \leq n^2 + 5n + 7 \leq 13n^2 \text{ for all } n \geq 1,$$

so $n^2 + 5n + 7 = \Theta(n^2)$ by definition of Big-Θ, with $n_0 = 1$, $c_1 = 1$, and $c_2 = 13$. □

Using the definition of Big-Theta can be inconvenient since it involves a double inequality. Luckily, the following theorem provides us with an easier approach.

315 Theorem If f and g are nonnegative function, then $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Proof: The result follows almost immediately from the definitions. We leave the details to the reader. □

This theorem implies that no new strategies are necessary for Big-Theta proofs since they can be split into two proofs—a Big-O proof and a Big-Omega proof.

The following is just a small sampling of the properties these notations have.

316 Theorem The following properties hold.

- Transitivity:

[2] Again, not literally ignore, but easily deal with with an inequality.

  - $f(n) \in O(g(n))$ and $g(n) \in O(h(n)) \to f(n) \in O(h(n))$
  - $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n)) \to f(n) \in \Theta(h(n))$
  - $f(n) \in \Omega(g(n))$ and $g(n) \in \Omega(h(n)) \to f(n) \in \Omega(h(n))$

- Scaling: If $f(n) \in O(g(n))$ then for any $k > 0$, $f(n) \in O(kg(n))$. (Also holds for the other two).

- Sums: If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$ then $(f_1 + f_2)(n) \in O(max(g_1(n), g_2(n)))$. (Also holds for the other two).

- Symmetry (sort of): $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$.

How do you use asymptotic notation to express the fact that $f(n)$ grows slower than $g(n)$? Saying $f(n) = O(g(n))$ doesn't work, because we only know that $f(n)$ doesn't grow faster than $g(n)$. It might grow slower, but it also might grow at the same rate. With the notation we have, the best way to express this idea is that $f(n) = O(g(n))$ and $f(n) \neq \Theta(g(n))$. But that is awkward. Let's define a notation for this instead.

317 Definition Let f and g be nonnegative functions, with g being eventually non-zero. We say that $f(n)$ is *little-o* of $g(n)$, written $f(n) = o(g(n))$ iff

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

If $f(n) = o(g(n))$, $f(n)$ grows asymptotically slower than $g(n)$.

Little-omega (ω) can be defined similarly, but where the limit is ∞.

5.1.2 Proofs using the definitions

The following example is annotated with comments about the technique that is used in many of these proofs. We use the following terminology in our explanation. By *lower order term* we mean a term that grows slower, and *higher order* means a term that grows faster. The *dominating term* is the term that grows the fastest. For instance, in $x^3 + 7x^2 - 4$, the x^2 term is a lower order term than x^3, and x^3 is the dominating term. We will discuss common growth rates, including how the relate to each other, in Section 5.3.

318 Example Find a tight bound on $f(x) = x^8 + 7x^7 - 10x^5 - 2x^4 + 3x^2 - 17$.

Solution: We will prove that $f(x) = \Theta(x^8)$. First, we will prove an upper bound for $f(x)$. It is clear that when $x > 0$,

$$x^8 + 7x^7 - 10x^5 - 2x^4 + 3x^2 - 17 \leq x^8 + 7x^7 + 3x^2.$$

- *We can upper bound any function by removing the lower order terms with negative coefficients, as long as $x > 0$.*

Next, it is not hard to see that when $x \geq 1$,

$$x^8 + 7x^7 + 3x^2 \leq x^8 + 7x^8 + 3x^8 = 11x^8.$$

- *We can upper bound any function by replacing lower order terms that have positive coefficients by the dominating term with the same coefficients. Here, we must make sure that the dominating term is larger than the given term for all values of x larger than some threshold x_0, and we must make note of the threshold value x_0.*

Thus, we have

$$f(x) = x^8 + 7x^7 - 10x^5 - 2x^4 + 3x^2 - 17 \le 11x^8 \text{ for all } x \ge 1,$$

and we have proved that $f(x) = O(x^8)$.

Now, we will get a lower bound for $f(x)$. It is not hard to see that when $x \ge 0$,

$$x^8 + 7x^7 - 10x^5 - 2x^4 + 3x^2 - 17 \ge x^8 - 10x^5 - 2x^4 - 17.$$

- *We can lower bound any function by removing the lower order terms with positive coefficients, as long as $x > 0$.*

Next, we can see that when $x \ge 1$,

$$x^8 - 10x^5 - 2x^4 - 17 \ge x^8 - 10x^7 - 2x^7 - 17x^7 = x^8 - 29x^7.$$

- *We can lower bound any function by replacing lower order terms with negative coefficients by a sub-dominating term with the same coefficients. (By sub-dominating, I mean one which dominates all but the dominating term.) Here, we must make sure that the sub-dominating term is larger than the given term for all values of x larger than some threshold x_0, and we must make note of the threshold value x_0. Making a wise choice for which sub-dominating term to use is crucial in finishing the proof.*

Next, we need to find a value $c > 0$ such that $x^8 - 29x^7 \ge cx^8$. Doing a little algebra, we see that this is equivalent to $(1-c)x^8 \ge 29x^7$. When $x \ge 1$, we can divide by x^7 and obtain $(1-c)x \ge 29$. Solving for c we obtain

$$c \le 1 - \frac{29}{x}.$$

If $x \ge 58$, then $c = 1/2$ suffices. We have just shown that if $x \ge 58$, then

$$f(x) = x^8 + 7x^7 - 10x^5 - 2x^4 + 3x^2 - 17 \ge \frac{1}{2}x^8.$$

Thus, $f(x) = \Omega(x^8)$. Since we have shown that $f(x) = \Omega(x^8)$ and that $f(x) = O(x^8)$, we have shown that $f(x) = \Theta(x^8)$.

319 Example Show that $\frac{1}{2}n^2 + 3n = \Theta(n^2)$

Proof: Notice that if $n \ge 1$,

$$\frac{1}{2}n^2 + 3n \le \frac{1}{2}n^2 + 3n^2 = \frac{7}{2}n^2,$$

so $\frac{1}{2}n^2 + 3n = O(n^2)$. Also, when $n \ge 0$,

$$\frac{1}{2}n^2 \le \frac{1}{2}n^2 + 3n,$$

so $\frac{1}{2}n^2 + 3n = \Omega(n^2)$. Since $\frac{1}{2}n^2 + 3n = O(n^2)$ and $\frac{1}{2}n^2 + 3n = \Omega(n^2)$, $\frac{1}{2}n^2 + 3n = \Theta(n^2)$ □

320 Example Show that $(n\log n - 2n + 13) = \Omega(n\log n)$

Proof: We need to show that there exist positive constants c and n_0 such that

$$0 \leq cn\log n \leq n\log n - 2n + 13 \text{ for all } n \geq n_0.$$

Since $n\log n - 2n \leq n\log n - 2n + 13$, we will instead show that

$$cn\log n \leq n\log n - 2n,$$

which is equivalent to

$$c \leq 1 - \frac{2}{\log n}, \text{ when } n > 1.$$

If $n \geq 8$, then $2/(\log n) \leq 2/3$, and picking $c = 1/3$ suffices. Thus if $c = 1/3$ and $n_0 = 8$, then for all $n \geq n_0$, we have

$$0 \leq cn\log n \leq n\log n - 2n \leq n\log n - 2n + 13.$$

Thus $(n\log n - 2n + 13) = \Omega(n\log n)$. □

321 Example Show that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Proof: We need to find positive constants c_1, c_2, and n_0 such that

$$0 \leq c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \text{ for all } n \geq n_0$$

Dividing by n^2, we get

$$0 \leq c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2.$$

Notice that $c_1 \leq \frac{1}{2} - \frac{3}{n}$ holds for $n \geq 10$ and $c_1 = 1/5$. Also, $\frac{1}{2} - \frac{3}{n} \leq c_2$ holds for $n \geq 10$ and $c_2 = 1$. Thus, if $c_1 = 1/5$, $c_2 = 1$, and $n_0 = 10$, then for all $n \geq n_0$,

$$0 \leq c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \text{ for all } n \geq n_0.$$

Thus we have shown that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$. □

322 Example Show that $n! = O(n^n)$

Proof: Notice that when $n \geq 1$, $0 \leq n! = 1 \cdot 2 \cdot 3 \cdots n \leq n \cdot n \cdots n = n^n$. Therefore $n! = O(n^n)$ (Here $n_0 = 1$, and $c = 1$.) □

323 Example Show that $(\sqrt{2})^{\log n} = O(\sqrt{n})$, where log means $\log_2$.

Proof: It is not too hard to see that

$$(\sqrt{2})^{\log n} = n^{\log\sqrt{2}} = n^{\log 2^{1/2}} = n^{\frac{1}{2}\log 2} = n^{\frac{1}{2}} = \sqrt{n}.$$

Thus it is clear that $(\sqrt{2})^{\log n} = O(\sqrt{n})$. □

Proving properties of the asymptotic notations is actually no more difficult than the rest of the proofs we have seen.

324 Example Prove that if $f(x) = O(g(x))$, and $g(x) = O(f(x))$, then $f(x) = \Theta(g(x))$.

Proof: If $f(x) = O(g(x))$, then there are positive constants c_2 and n_0' such that

$$0 \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0'$$

Similarly, if $g(x) = O(f(x))$, then there are positive constants c_1' and n_0'' such that

$$0 \leq g(n) \leq c_1' f(n) \text{ for all } n \geq n_0''.$$

We can divide this by c_1' to obtain

$$0 \leq \frac{1}{c_1'} g(n) \leq f(n) \text{ for all } n \geq n_0''.$$

Setting $c_1 = 1/c_1'$ and $n_0 = \max(n_0', n_0'')$, we have

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0.$$

Thus, $f(x) = \Theta(g(x))$. □

325 Example Let $f(x) = O(g(x))$ and $g(x) = O(h(x))$. Show that $f(x) = O(h(x))$.

Proof: If $f(x) = O(g(x))$, then there are positive constants c_1 and n_0' such that

$$0 \leq f(n) \leq c_1 g(n) \text{ for all } n \geq n_0',$$

and if $g(x) = O(h(x))$, then there are positive constants c_2 and n_0'' such that

$$0 \leq g(n) \leq c_2 h(n) \text{ for all } n \geq n_0''.$$

Set $n_0 = \max(n_0', n_0'')$ and $c_3 = c_1 c_2$. Then

$$0 \leq f(n) \leq c_1 g(n) \leq c_1 c_2 h(n) = c_3 h(n) \text{ for all } n \geq n_0.$$

Thus $f(x) = O(h(x))$. □

5.1.3 Proofs using limits

So far we have used the definitions in all of our proofs. The following theorem provides another technique that is often much easier, assuming you understand and are comfortable with limits.

326 Theorem Let $f(n)$ and $g(n)$ be functions such that

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = A.$$

Then

1. If $A = 0$, then $f(n) = O(g(n))$, and $f(n) \neq \Theta(g(n))$. That is, $f(n) = o(g(n))$.

2. If $A=\infty$, then $f(n)=\Omega(g(n))$, and $f(n)\neq\Theta(g(n))$. That is, $f(n)=\omega(g(n))$.
3. If $A\neq 0$ is finite, then $f(n)=\Theta(g(n))$.

Notice that if the above limit does not exist, then you probably need to resort to using the definitions. Luckily, in the analysis of algorithms the above approach works most of the time.

Now is probably a good time to recall a very useful theorem for computing limits, called **l'Hopital's Rule**.

327 Theorem (l'Hopital's Rule) Let $f(x)$ and $g(x)$ be differentiable functions. If $\lim\limits_{x\to\infty} f(x)=\lim\limits_{x\to\infty} g(x)=0$ or $\lim\limits_{x\to\infty} f(x)=\lim\limits_{x\to\infty} g(x)=\infty$, then

$$\lim_{x\to\infty}\frac{f(x)}{g(x)}=\lim_{x\to\infty}\frac{f'(x)}{g'(x)}$$

Now let's see a bunch of examples.

328 Example Find a tight bound on $f(x)=x^8+7x^7-10x^5-2x^4+3x^2-17$ using Theorem 326.

Solution: We guess (or know, if we read the solution to Example 318) that $f(x)=\Theta(x^8)$. To prove this, notice that

$$\begin{aligned}\lim_{x\to\infty}\frac{x^8+7x^7-10x^5-2x^4+3x^2-17}{x^8} &= \lim_{x\to\infty}\frac{x^8}{x^8}+\frac{7x^7}{x^8}-\frac{10x^5}{x^8}-\frac{2x^4}{x^8}+\frac{3x^2}{x^8}-\frac{17}{x^8}\\ &= \lim_{x\to\infty}1+\frac{7}{x}-\frac{10}{x^3}-\frac{2}{x^4}+\frac{3}{x^6}-\frac{17}{x^8}\\ &= \lim_{x\to\infty}1+0-0-0+0-0=1\end{aligned}$$

Thus, $f(x)=\Theta(x^8)$ by the Theorem.

329 Example Find a tight bound on $f(x)=x^4-23x^3+12x^2+15x-21$.

Solution: We will give solutions using both the definitions and Theorem 326 so you can compare the techniques.

Solution #1

It is clear that when $x\geq 1$,

$$x^4-23x^3+12x^2+15x-21\leq x^4+12x^2+15x\leq x^4+12x^4+15x^4=28x^4.$$

Also,

$$x^4-23x^3+12x^2+15x-21\geq x^4-23x^3-21\geq x^4-23x^3-21x^3=x^4-44x^3\geq\frac{1}{2}x^4,$$

whenever

$$\frac{1}{2}x^4\geq 44x^3\Leftrightarrow x\geq 88.$$

Thus

$$\frac{1}{2}x^4\leq x^4-23x^3+12x^2+15x-21\leq 28x^4,\text{ for all } x\geq 88.$$

We have shown that $f(x)=x^4-23x^3+12x^2+15x-21=\Theta(x^4)$.

Solution #2
From Solution #1 we already know that $f(x) = \Theta(x^4)$. We verify this by noticing that

$$\begin{aligned}\lim_{x\to\infty} x^4 - 23x^3 + 12x^2 + 15x - 21 &= \lim_{x\to\infty} \frac{x^4}{x^4} - \frac{23x^3}{x^4} + \frac{12x^2}{x^4} + \frac{15x}{x^4} - \frac{21}{x^4} \\ &= \lim_{x\to\infty} 1 - \frac{23}{x} + \frac{12}{x^2} + \frac{15}{x^3} - \frac{21}{x^4} \\ &= \lim_{x\to\infty} 1 - 0 + 0 + 0 - 0 = 1\end{aligned}$$

330 Example Show that $\log x = O(x)$.

Proof:

$$\lim_{x\to\infty} \frac{\log x}{x} = \lim_{x\to\infty} \frac{\frac{1}{x}}{1} = \lim_{x\to\infty} \frac{1}{x} = 0$$

Therefore, $\log x = O(x)$. □

331 Example Show that $\log n! = O(n \log n)$

Proof: In the previous problem, we showed that when $n \geq 1$, $n! \leq n^n$. Notice that $n! \geq 1$, so taking logs of both sides, we obtain $0 \leq \log n! \leq \log n^n = n \log n$ for all $n \geq 1$. Therefore $\log n! = O(n \log n)$. (Here, $n_0 = 1$, and $c = 1$.) □

332 Example Find a good upper bound on $n \log(n^2+1) + n^2 \log n$.

Solution: Another example we will do in two ways.
Solution #1:
If $n > 1$,

$$\log(n^2+1) \leq \log(n^2+n^2) = \log(2n^2) = (\log 2 + \log n^2) \leq (\log n + 2\log n) = 3\log n$$

Thus when $n > 1$,

$$0 \leq n\log(n^2+1) + n^2\log n \leq n3\log n + n^2\log n \leq 3n^2\log n + n^2\log n \leq 4n^2\log n.$$

Thus, $n\log(n^2+1) + n^2\log n = O(n^2\log n)$.

Solution #2:

$$\begin{aligned}\lim_{x\to\infty} \frac{n\log(n^2+1) + n^2\log n}{n^2\log n} &= \lim_{x\to\infty} \frac{n\log(n^2+1)}{n^2\log n} + 1 \\ &= 1 + \lim_{x\to\infty} \frac{\log(n^2+1)}{n\log n} \\ &= 1 + \lim_{x\to\infty} \frac{\frac{2n}{n^2+1}}{1\cdot\log n + n\cdot\frac{1}{n}} && \text{(l'Hopital)} \\ &= 1 + \lim_{x\to\infty} \frac{2n}{(n^2+1)(\log n + 1)} \\ &= 1 + \lim_{x\to\infty} \frac{2}{2n(\log n + 1) + (n^2+1)\cdot\frac{1}{n}} && \text{(l'Hopital)} \\ &= 1 + 0 = 1.\end{aligned}$$

Therefore, $n\log(n^2+1) + n^2\log n. = \Theta(n^2\log n)$.

The last example illustrates an important point: If asked to prove that $f(n) = O(g(n))$, Theorem 315 implies that proving $f(n) = \Theta(g(n))$ suffices.

333 Example Show that $2^x = O(3^x)$.

Proof: This is easy to see since $\lim\limits_{x\to\infty} \frac{2^x}{3^x} = \lim\limits_{x\to\infty} \left(\frac{2}{3}\right)^x = \lim\limits_{x\to\infty} 0$.

Proof #2:
If $x \geq 1$, then clearly $(3/2)^x \geq 1$, so

$$2^x \leq 2^x \left(\frac{3}{2}\right)^x = \left(\frac{2\times 3}{2}\right)^x = 3^x.$$

□

Summary
It is important to remember that a Big-O bound is only an *upper bound*, and that it may or may not be a *tight bound*. So if $f(n) = O(n^2)$, it is possible that $f(n) = 3n^2 + 4$, or $f(n) = \log n$, but $f(n) \neq n^3$. Conversely, a Big-Omega bound is only a *lower bound*. Thus, if $f(n) = \Omega(n\log n)$, it might be the case that $f(n) = 2^n$, but we know that $f(n) \neq 3n$. Unlike the other bounds, a Big-Theta bound is precise. So, if $f(n) = \Theta(n^2)$, then we know that f has a quadratic growth rate. It might be that $f(n) = 3n^2$, $2n^2 - 43n - 4$, or even $n^2 + n\log n$. But we are certain that the fastest growing term of f is cn^2 for some constant c.

334 Example If $f(n) = O(n^2)$ and $g(n) = O(n^3)$, can we say that $g(n)$ grows faster? No. Because the bounds given are not tight, it is possible that $f(n) = n^2$ and $g(n) = n$, so that $f(n)$ grows faster, or vice-versa.

5.2 Analyzing Algorithms

In this chapter we are dealing with a seemingly simple question: *Given an algorithm, how good is it?* I say “seemingly” simple because unless we define what we mean by “good”, we cannot answer the question. Do we mean how elegant it is? How easy it is to understand? How easy it is to update if/when necessary? Whether or not it can be generalized?

Although all of these may be important questions, in algorithm analysis we are usually more interested in the following two questions: *How long does the algorithm take to run, and how much space does the algorithm require.* In fact, we follow the tradition of most books and restrict our discussion to just the first question. This is usually reasonable since the amount of memory used by most algorithms is reasonable enough to not matter. There are times, however, when analyzing the space required by an algorithm is important. For instance, when the data is really large (e.g. the graph that represents friendship on Facebook) or when you are implementing a *space-time-tradeoff* algorithms.

Although we have simplified the question, we still need to be more specific. What do we mean by “time”? Do we mean how long it takes in real time (called *wall-clock time*)? Or the actual amount of time our processor used (called *CPU time*)? Or the *number of instructions* (or *number of operations*) executed?

Because the running time of an algorithm is greatly affected by the characteristics of the computer system (e.g. processor speed, number of processors, amount of memory, file-system type, etc.), it does not provide a comparable measure, regardless of whether you use CPU time or wall-clock time. For instance, if I have an algorithm that takes 1 second and you have an algorithm that takes 1 minute, is mine better?

It depends. If I ran mine on a supercomputer and you ran yours on a TRS-80 Model 4 (which came out in 1983), it is very possible your algorithm is better. Wall-clock time has the added problem that the other programs running on the computer can greatly influence it. So the same algorithm might take 5 seconds to run one time and 60 seconds to run another time.

This leaves us with the number of instructions. However, we still have a problem. What is meant by "instruction"? When you write a program in a language such as Java or C++, it is not executed exactly as you wrote it–it is compiled into some sort of machine language. The process of compiling does not generally involve a one-to-one mapping of instructions, so counting Java instructions versus C++ instructions wouldn't necessarily be fair. On the other hand, we certainly do not want to look at the machine code in order to count instructions—machine code is ugly. Further, when analyzing an algorithm, should we even take into account the exact implementation in a particular language, or should we analyze the algorithm apart from implementation?

O.K., that's enough of the complications. Let's get to the bottom line. When analyzing algorithms, we generally want to ignore what sort of machine it will run on and what language it will be implemented in. We also generally do not want to know *exactly* how many instructions it will take. Instead, we want to know the *rate of growth* of the number of instructions. This is sometimes called the *asymptotic running time* of an algorithm. In other words, as the size of the input increases, how does that affect the number of instructions executed? We will typically use the notation from the previous section to specify the running time of an algorithm. We will call this the *complexity* of the algorithm.

Given an algorithm, the *size of the input* is exactly what it sounds like—the amount of space required to specify the input. For instance, if an algorithm operates on an array of size n, we generally say the input is of size n. For a graph, it is usually the number of vertices or the number of vertices and edges. When the input is a single number, it surprisingly gets a lot more complicated for reasons I do not want to get into right now. We usually don't need to worry about this case.

Algorithm analysis involves determining the size of the input, n, and then finding a function based on n that tells us how long the algorithm will take if the input is of size n. By "how long", we of course mean how many operations.

335 Example (Sequential Search) Given an array of n elements, often one needs to determine if a given number val is in the array. One way to do this is with the *sequential search* algorithm that simply looks through all of the elements in the array until it finds it or reaches the end. The most common version of this algorithm returns the index of the element, or -1 if the element it not in the array. Here is one implementation.

```
int sequentialSearch(int a[],int val) {
    for(int i=0;i<a.size();i++) {
        if(a[i]==val) {
            return i;
        }
    }
    return -1;
}
```

How many operations does `sequentialSearch` take to search an array of size n?

Solution: As mentioned above, we consider n as the size of the input. Assigning $i = 0$ takes one instruction. Each iteration through the for look increments i, compares i with $a.size()$, and compares $a[i]$ with val. Don't forget that accessing $a[i]$ and calling `a.size()` each take (at least) one instruction. Finally, it takes an instruction to return the value. If the val is in the array at position k, the algorithm will take $2+5k = \Theta(k)$ operations, the 2 coming from the assignment `i=0` and the return statement. If *val* is not in the array, the algorithm takes $2+5n = \Theta(n)$ instructions.

This last example should bring up a few questions. Did we miss any instructions? Did we miss any possible outcomes that would give us a different answer? How exactly should we specify our analysis?

Let's deal with the possible outcomes question first. Generally speaking, when we analyze an algorithm we want to know what happens in one of three cases: The best case, the average case or the worst case.

As the name suggests, when performing a *best case* analysis, we are trying to determine the smallest possible number of instructions an algorithm will take. Typically, this is the least useful type of analysis. If you have experienced a situation when someone said something like "it will only take an hour (or a day) to fix your MP3 player," and it actually took 3 hours (or days), you will understand why.

Similarly, the *worst case* analysis considers what is the largest number of instructions that will execute. This is probably the most common analysis, and typically the most useful. When you pay Amazon for guaranteed 2-day delivery, you are paying for them to guarantee a worst-case delivery time. However, this analogy actually breaks down quickly. When you do a worst-case analysis, you know the algorithm will *never* take longer than what your analysis specified, but occasionally an Amazon delivery is lost or delayed.

The *average case* is a little more complicated, both to define and to compute. The first problem is determining what "average" means for a particular input and/or algorithm. For instance, what does an "average" array of values look like? The second problem is that even with a good definition, computing the average case complexity is usually much more difficult than the other two. It also must be used appropriately. If you know what the average number of instructions for an algorithm is, you need to remember that sometimes it might take less time and sometimes it might take more time.

336 Example Continuing the `sequentialSearch` example, notice that our analysis above reveals that the best-case performance is $7 = \Theta(1)$ operations (if the element sought is the first one in the array) and the worst-case performance is $2+5n = \theta(n)$ operations (if the element is not in the array). If we assume that the element we are searching for is equally likely to be anywhere in the array or not in the array, then the average-case performance should be around $2+5(n/2) = \Theta(n)$ operations.

Notice that the average and worst case complexities are the same. This actually makes sense. We estimate that the average case takes about half as long as the worst case. But no matter how large n gets, it is still just half as long. That is, the rate of growth of the running times is the same.

Now onto another important question: How do we know we counted all of the operations? As it turns out, we don't actually care. This is good because determining the exact number is very difficult, if not impossible. Recall that we said we wanted to know the rate of growth of an algorithm, not the exact number of instructions. As long as we count all of the "important" ones, we will get the correct rate of growth. But what are the "important" ones? The term *abstract operation* is sometimes used to describe the operations that we will count. Typically you choose one type of operation or a set of operations that you know will be performed the most often and consider those as the the abstract operation(s).

337 Example The analysis of `sequentialSearch` is much easier than I made it out to be earlier. Notice that the comparison (`a[i]==val`) is executed as often as any other instruction. In the best case it is executed once, so the best case is $\Theta(1)$. In the worst case it is executed $n = \Theta(n)$ times. With the same assumptions as above, we expect the average to be about $n/2 = \Theta(n)$. Notice that we obtained the same answers here as we did above.

It is important to make sure that you choose the operations you will count carefully so your analysis is not incorrect. In addition, you need to look at every instruction in the algorithm to determine whether or not it can be accomplished in constant time. If not, you need to count it correctly.

338 Example Analyze the following algorithm that find the maximum value in an array.

```
int max(int a[],int n) {
    int max = int.MIN_VAL;
    for (int i=0; i<n; i++)
        max = Maximum(max, a[i]);
    return max;
}
```

Solution: We focus on the assignment (=) inside the loop and ignore the other instructions. This should be fine since assignment occurs at least as often as any other instruction. We are assuming that finding the maximum of two numbers (`Maximum`) takes constant time, which is reasonable. It isn't too difficult to see that the assignment will occur n times for an array of size n since the code goes through a loop with $i = 0, \ldots, n-1$. Thus, the complexity of `max` is $\Theta(n)$. Notice that for this algorithm the best, worst, and average are all exactly the same.[3]

339 Example Find the complexity of Insertion Sort.

```
void insertion(int a[]) {
    for (int i=1;i<a.size();i++) {
        int v=a[i];
        int j=i;
        while (j > 0 && a[j] > v) {
            a[j+1] = a[j];
            j--;
        }
        a[j]=v;
    }
}
```

Solution: Let $n = a.size()$. We will use the comparison in the while loop as our abstract operation because it occurs at least as often as any other operation. In fact, we can just count the number of times the while loop executes since each time it executes, it only does a constant amount of work.[4] Notice that the while loop goes from $j = i$ down to $j = 1$ unless `a[j] > v` at some point.

In the worst case the while loop executes i times (because $a[i]$ is always less than or equal to v). The for loop changes the value of i from 0 to $n-1$. A first guess at the complexity would be $n \cdot i$. But this doesn't make sense. What is i? The complexity has to be expressed in terms of n.

[3] When an algorithm has no conditional statements, or at least none that can cause the algorithm to end earlier, the best, average, and worst case complexities will usually be the same. I say usually because there is always the possibility of a weird algorithm that I haven't thought of that could be an exception.

[4] Here I am analyzing the algorithm slightly differently. Instead of focusing on an abstract operation, I am counting all of the operations, but when there are 4 or 7 or 42, I will just think of it as being a constant number—like 1. But be careful! If something takes more than constant time, but you say it takes constant time, you will get the answer.

Instead we need to realize that since i is changing each time through the for loop, we need to sum the worst case of the while loop as i is changing. In other words, the complexity is

$$\sum_{i=1}^{n-1} i = (n-1)n/2 = \Theta(n^2).$$

This happens, by the way, if the elements in the array begin in reverse order.

In the best case, the while loop only executes once each time through the for loop. This happens if the array is already sorted. In this case, the complexity is $\Theta(n)$.

340 Example Analyze the following algorithm.

```
for (int i=1; i<=a.size(); i++) {
    for (int j=1; j<=a.size(); j++) {
        double V=A[i]*A[j];
    }
}
```

Solution: Let $n = a.size()$. Clearly the assignment (`V=A[i]*A[j]`) occurs the most often. The inner loop[5] always executes n times, each time doing one assignment. The outer loop executes n times, and each time it executes, it executes the inner loop. Therefore the total time is $n \cdot n = \Theta(n^2)$. This is the best, worst, and average case complexity since nothing about the input can change what the algorithm does.

Sometimes people mistakingly think the algorithm takes $n+n$ operations. It is very important to carefully think through the analysis so you multiply/add as appropriate for the given circumstance. In this case, you can think of it being $n \cdot n$ because each time the outer loop executes it takes n time, and that happens n times, so you get $n+n+\cdots+n$ operations (where there are n terms in the sum), which is just $n \cdot n$.

It is important to be careful not to jump to conclusions when analyzing algorithms. For instance, a double-nested for-loop should always take $\Theta(n^2)$ to execute, right? This is wrong for many reasons. For instance, consider the next example.

341 Example Analyze the following algorithm.

```
int k=50;
for (i = 0; i < n; i ++)
  for (j = 0; j < k; j ++)
    a[i][j] = b[i][j] * x;
```

Solution: The line in the inner for loop takes constant time (let's call it c). The inner loop executes $k = 50$ times, each time doing c operations. Thus the inner loop does $50 \cdot c$ operations, which is still just a constant. The outer loop executes n times, each time executing the inner loop, which takes $50 \cdot c$ operations. Thus, the whole algorithm takes $50 \cdot c \cdot n = \Theta(n)$ time.

[5] *Always* analyze from the inside out. The more practice you get, the more it will be obvious that this is the only way that will consistently work.

We end this section with a comment that perhaps too few people think about. *Theory* and *practice* don't always agree. Since asymptotic notation ignores the *constants*, two algorithms that have the same complexity are not, in practice, equally good. For instance, if one takes $4 \cdot n^2$ operations and the other $10,000 \cdot n^2$ operations, clearly the first will be preferred even though they are both $\Theta(n^2)$ algorithms.

Further, consider this (real-life) situation: You want to multiply two matrices. The standard algorithm to multiply matrices has complexity $\Theta(n^3)$. Strassen's algorithm for matrix multiplication has a complexity of about $\Theta(n^{2.8})$. Clearly, Strassen's algorithm is better asymptotically. In other words, if your matrices are large enough, Strassen's algorithm is certainly the better choice. However, it turns out that if $n = 50$, the standard algorithm performs better.[6]

Analyzing recursive algorithms can be a little more complex. We will consider such algorithms in Chapter 6, where we develop the necessary tools.

5.3 Common Growth Rates

In order for us to compare the efficiency of algorithms, we need to know the growth rates of some common functions and how they compare to one another. We will briefly discuss each of the following complexity classes. For each, k is a constant.

- Constant: $\Theta(k)$, for example $\Theta(1)$
- Linear: $\Theta(n)$
- Logarithmic: $\Theta(\log_k n)$
- $n \log n$: $\Theta(n \log_k n)$
- Quadratic: $\Theta(n^2)$
- Polynomial: $\Theta(n^k)$
- Exponential: $\Theta(k^n)$

342 Definition (Constant) An algorithm with running time $\Theta(k)$ for some constant k is said to have *constant* complexity. Note that this does not necessarily mean that the algorithm takes exactly the same amount of time for all inputs, but it *does* mean that there is some number K such that it always takes no more than K operations.

343 Example The following algorithms have constant complexity.

```
int Fifth_Element(int A[],int n) {
    return A[4];
}
```

```
int Partial_Sum(int A[],int n) {
    int sum=0;
    for(int i=0;i<42;i++)
       sum=sum+A[i];
    return sum;
}
```

[6] There is debate about the "crossover point." This is the point at which the more efficient algorithm is worth using. For smaller inputs, the overhead associated with the cleverness of the algorithm isn't worth the extra time it takes. For larger inputs, the extra overhead is far outweighed by the benefits of the algorithm. For Strassen's algorithm, this point may be somewhere between 75 and 100, but don't quote me on that.

344 Definition (Linear) Algorithms with running time of $\Theta(n)$ are said to have *linear* complexity. As n increases, the run time increases in proportion with n. Linear algorithms access each of their n inputs at most some constant number of times.

345 Example The following are linear algorithms.

```
void sum_first_n(int n) {
   int i,sum=0;
   for (i=1;i<=n;i++)
      sum = sum + i;
    }
```

```
void m_sum_first_n(int n) {
     int i,k,sum=0;
     for (i=1;i<=n;i++)
         for (k=1;k<7;k++)
                 sum = sum + i;
     }
```

346 Definition (Logarithmic) Algorithms with running time of $\Theta(\log n)$ are said to have *logarithmic* complexity. As the input size n increases, so does the running time, but very slowly. Logarithmic algorithms are typically found when the algorithm can systematically ignore fractions of the input.

Recall that a logarithmic function is the inverse of an exponential function. That is, $b^x = n$ is equivalent to $x = \log_b n$. The following identity is very relevant to our discussion of complexity classes involving logarithms:

$$(\log_a b)(\log_b n) = log_a n$$

This identity implies that $\log_a n = \Theta(\log_b n)$. In other words, changing the base of a logarithm just changes the value by a constant amount. Therefore all logarithms belong to the same complexity class. Because of this, the base will often be omitted from logarithms when they appear in asymptotic notation. In computer science, the base of logarithms is often 2. Finally, in case you have not seen this notation, you should know that $\log^a n = (\log n)^a$.

347 Example The binary search algorithm has logarithmic complexity. We will prove this fact later.

```
int binarysearch(int a[], int n, int val) {
   int l=1, r=n, m;
   while (r>=1) {
       m = (l+r)/2;
       if(a[m]==val)
           return m;
       if(a[m]>val)
           r=m-1;
       else
           l=m+1;
    }
   return -1;
}
```

348 Definition ($n \log n$) Many divide-and-conquer algorithms have complexity $\Theta(n \log n)$. These algorithms break the input into a constant number of subproblems of the same type, solve them independently, and then combine the solutions together. Not all divide-and-conquer algorithms have this complexity, however. Examples include *Quicksort* and *Mergesort*. We'll analyze Mergesort later.

349 Definition (Quadratic) Algorithms with running time of $\Theta(n^2)$ are said to have *quadratic* complexity. As n doubles, the running time quadruples.

350 Example The following algorithm is quadratic.

```
int compute_sums(int A[], int n) {
    int M[n][n];
    int i,j;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            M[i][j]=A[i]+A[j];
    return M;
    }
```

351 Definition (Polynomial) Algorithms with running time of $\Theta(n^k)$ for some constant k are said to have *polynomial* complexity. Notice that linear and quadratic are special cases of polynomial. When we say an *efficient* algorithm exists to solve a problem, we typically mean an algorithm with polynomial complexity.

352 Definition (Exponential) Algorithms with running time of $\Theta(k^n)$ for some constant k are said to have *exponential* complexity. Since exponential algorithms can only be run for small values of n, they are not considered to be efficient. Brute-force algorithms are often exponential.

☞ *Unlike logarithms, the the base of exponentials changes the complexity class. In other words,* $a^n \neq \Theta(b^n)$ *unless* $a = b$.

5.3.1 Comparing Growth Rates

Figure 5.1 shows the value of several functions for various values of n to give you an idea of their relative rates of growth. Notice that the bottom of the table is labeled so you can get a sense of how slow $\log n$ and $\log(\log n)$ grow. Figures 5.2 and 5.3 are attempting to demonstrate that as n increases, the constants and lower-order terms do not matter. For instance, notice that although $100n$ is much larger than 2^n for small values of n, as n increases, 2^n quickly gets much larger than 100. Similarly, in Figure 5.3, notice that when $n = 4$, n^3 and $n^3 + 234$ are virtually the same.

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
0.6931	2	1.39	4	8	4
1.099	3	3.30	9	27	8
1.386	4	5.55	16	64	16
1.609	5	8.05	25	125	32
1.792	6	10.75	36	216	64
1.946	7	13.62	49	343	128
2.079	8	16.64	64	512	256
2.197	9	19.78	81	729	512
2.303	10	23.03	100	1000	1024
2.398	11	26.38	121	1331	2048
2.485	12	29.82	144	1728	4096
2.565	13	33.34	169	2197	8192
2.639	14	36.95	196	2744	16384
2.708	15	40.62	225	3375	32768
2.773	16	44.36	256	4096	65536
2.833	17	48.16	289	4913	131072
2.890	18	52.03	324	5832	262144
$\log\log m$	$\log m$				m

Figure 5.1: A comparison of growth rates

n	$100n$	n^2	$11n^2$	n^3	2^n
1	100	1	11	1	2
2	200	4	44	8	4
3	300	9	99	27	8
4	400	16	176	64	16
5	500	25	275	125	32
6	600	36	396	216	64
7	700	49	539	343	128
8	800	64	704	512	256
9	900	81	891	729	512
10	1000	100	1100	1000	1024
11	1100	121	1331	1331	2048
12	1200	144	1584	1728	4096
13	1300	169	1859	2197	8192
14	1400	196	2156	2744	16384
15	1500	225	2475	3375	32768
16	1600	256	2816	4096	65536
17	1700	289	3179	4913	131072
18	1800	324	3564	5832	262144
19	1900	361	3971	6859	524288

Figure 5.2: Constants don’t matter

n	n^2	n^2-n	n^2+99	n^3	n^3+234
2	4	2	103	8	242
6	36	30	135	216	450
10	100	90	199	1000	1234
14	196	182	295	2744	2978
18	324	306	423	5832	6066
22	484	462	583	10648	10882
26	676	650	775	17576	17810
30	900	870	999	27000	27234
34	1156	1122	1255	39304	39538
38	1444	1406	1543	54872	55106
42	1764	1722	1863	74088	74322
46	2116	2070	2215	97336	97570
50	2500	2450	2599	125000	125234
54	2916	2862	3015	157464	157698
58	3364	3306	3463	195112	195346
62	3844	3782	3943	238328	238562
66	4356	4290	4455	287496	287730
70	4900	4830	4999	343000	343234
74	5476	5402	5575	405224	405458

Figure 5.3: Lower-order terms don't matter

Figures 5.4 through 5.8 give a more graphical representation of relative growth rates of several functions.

Here are some of the most important results about the relative growth rate of some common functions. If you want proofs, try them yourself. Theorems 326 and 327 will help.

353 Theorem Let $a < b$ be real numbers. Then

1. $n^a = o(n^b)$.

2. $a^n = o(b^n)$.

354 Theorem Let $a > 0$ and $b > 0$ be real numbers. Then $\log^a n = o(n^b)$. In other words, any power of a log grows slower than any polynomial.

355 Theorem Let $a > 0$ and $b > 1$ be real numbers. Then $n^a = o(b^n)$. In other words, any exponential with base larger than 1 grows faster than any polynomial.

An alternative notation of little-o is $\ll$. In other words, $f(n) = o(g(n))$ iff $f(n) \ll g(n)$. This notation is useful in certain contexts, including the following comparison of the growth rate of common functions.

$$c \ll \log n \ll log^2 n \ll \sqrt{n} \ll n \ll n\log n \ll n^{1.1} \ll n^2 \ll n^3 \ll n^4 \ll 2^n$$

You should convince yourself that these are correct.

Let me end on a very important note regarding analysis of algorithms and asymptotic growth of functions. If algorithm A is faster than algorithm B, then the running time of A is less than the running time of B. On the other hand, if A's running time is asymptotically faster than the running time of B, that means B is faster! In other words, the words fast/slow need to be reverse when discussing algorithm speeds versus the growth of the functions. Put simply: *A faster growing complexity means a slower algorithm, and vice-versa*.

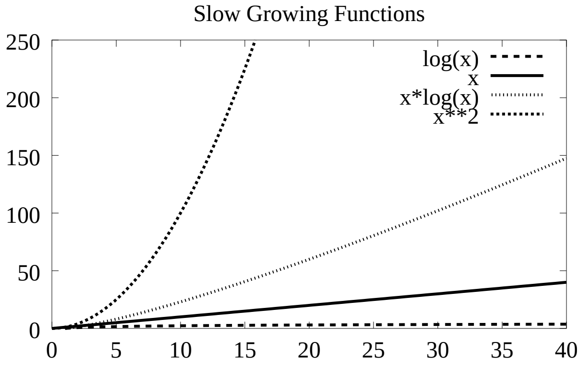


Figure 5.4: The growth rate of some slow growing functions.

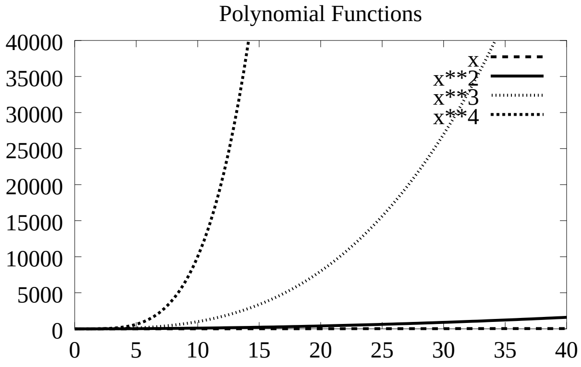


Figure 5.5: The growth rate of some polynomials.

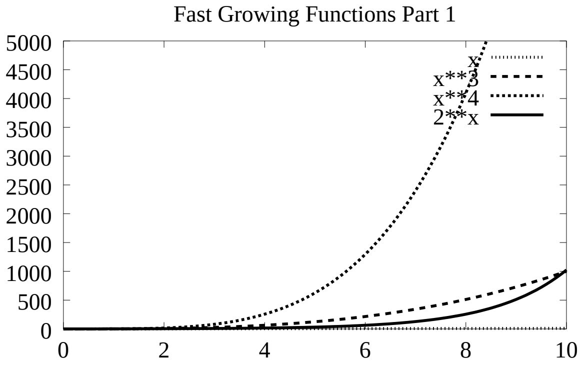


Figure 5.6: The growth rate of some polynomials and an exponential. This graph makes it look like x^4 is growing faster than 2^x. But see Figure 5.7.

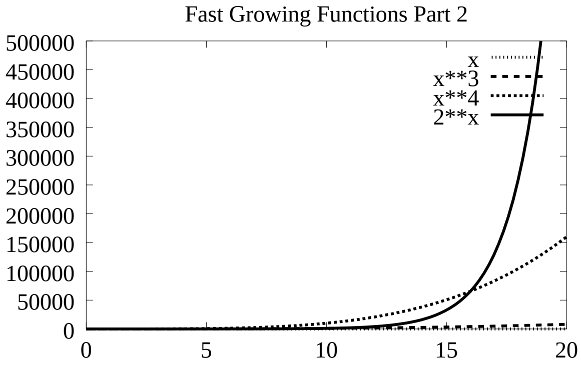


Figure 5.7: The growth rate of some polynomials and an exponential. If we make n large enough, it is more clear that 2^n grows faster than n^4.

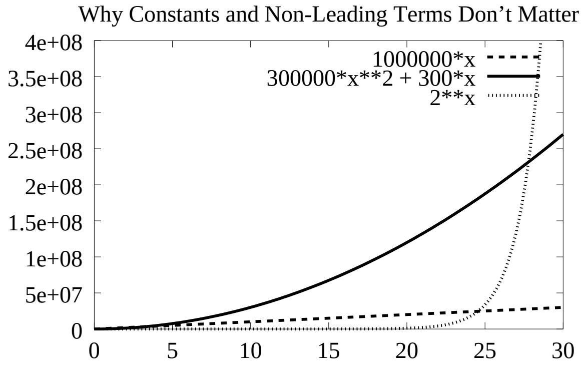


Figure 5.8: Notice that as n gets larger, the constants eventually matter less.

Homework

356 Problem Prove Theorem 315.

357 Problem Big-Theta can be thought of as a relation on the set of functions, where $(f,g) \in$ Big-Theta iff $f(n) = \Theta(g(n))$. Prove that Big-Theta is an equivalence relation.

This page intentionally left blank.

Chapter 6

Recursion, Recurrences, and Mathematical Induction

In this chapter we will explore a proof technique, an algorithmic technique, and a mathematical technique. Each topic is in some ways very different than the others, yet they have a whole lot in common. They are also often used in conjunction.

You have already seen *recurrence relations*. Recall that a recurrence relation is a way of defining a sequence of numbers with a formula that is based on previous numbers in the sequence. You may or may not be familiar with *recursion*, which is an algorithmic technique in which an algorithm calls itself (such an algorithm is called *recursive*), typically with "smaller" input. Finally, the *principle of mathematical induction* is a slick proof technique that works so well that sometimes it feels like you are cheating.

We will see that induction can be used to prove formulas, prove that algorithms—especially recursive ones—are correct, and help solve recurrence relations. Among other things, recurrence relations can be used to analyze recursive algorithm. Recursive algorithms can be used to compute the values defined by recurrence relations and to solve problems that can be broken into smaller versions of themselves.

As we will see, each of these has one or more *base cases* that can be proved/computed/determined directly and a *recursive* or *inductive* step that relies on previous steps. With each, the inductive/recursive steps must eventually lead to a base case.

Because induction can be used to prove things about the other two, we will begin there.

6.1 Mathematical Induction

Let $P(n)$ be a propositional function with domain $\mathbb{N}$, $\mathbb{Z}^+$, or sometimes $\{a, a+1, a+2, \ldots\}$. The *principle of mathematical induction* (*PMI*, or simply *induction*) is usually used to prove statements of the form

$$\text{for all } n \geq a, P(n) \text{ is true,}$$

where a is a constant, usually 0 or 1.

Induction is based on the following fairly intuitive observation. Suppose that we are to perform a task that involves a certain number of steps. Suppose that these steps must be followed in strict numerical order. Finally, suppose that we know how to perform the n-th task provided we have accomplished the $(n-1)$-th task. Thus if we are ever able to start the job (that is, if we have a base case), then we should be able to finish it (because starting with the base case we go to the next case, and then to the case following that, etc.).

Let's see an example. But first, recall *modus ponens*: if p is true and $p \to q$ is true, then q is true. In English, "If p is true, and whenever p is true q is true, then q is true."[1]

358 Example Assume that I know that $P(1)$ is true and that whenever $k \geq 1$, $P(k) \to P(k+1)$ is true. What can I conclude?

Solution: Let's start from the ground up.

$$\begin{array}{rlll}
\text{We know: } P(1) \text{ is true,} & \text{and if } k \geq 1, & P(k) \to P(k+1). & \\
\text{Since } P(1) \text{ is true,} & \text{and since } 1 \geq 1, & P(1) \to P(2), & \text{therefore } P(2) \text{ is true.} \\
\text{Since } P(2) \text{ is true,} & \text{and since } 2 \geq 1, & P(2) \to P(3), & \text{therefore } P(3) \text{ is true.} \\
\text{Since } P(3) \text{ is true,} & \text{and since } 3 \geq 1, & P(3) \to P(4), & \text{therefore } P(4) \text{ is true.} \\
\vdots & \vdots & \vdots & \vdots
\end{array}$$

and this continues, so $P(k)$ is true for all $k \geq 1$.

This example illustrates the idea behind induction. Induction is based on the fact that if $P(a)$ is true for some $a \geq 0$ (the *base case*), and for $k \geq a$, if $P(k)$ is true, then $P(k+1)$ is true (the *inductive case*), then $P(n)$ is true for all $n \geq a$. In other words, the principle of mathematical induction is based on the tautology

$$[P(a) \wedge \forall k(P(k) \to P(k+1))] \to (\forall n P(n)),$$

where the universe is $\{a, a+1, a+2, \ldots\}$. We won't prove that this is a tautology, but the previous example should help you convince yourself that it is indeed a tautology. *It is definitely worth your time to convince yourself that this is a tautology, so if you aren't convinced, reread the example, think about it some more, and/or ask someone to help you see it.*

To prove a statement using induction, you start by proving one or more *base cases*. Then you show that if $P(k)$ is true for any k which is at least as large as the base case(s), then $P(k+1)$ is true. Alternatively, you can show that if $P(k-1)$ is true for k larger than any of the base cases, then $P(k)$ is true. The assumption that $P(k)$ is true is called the *inductive hypothesis*, and proving that $P(k+1)$ is true based on the inductive hypothesis is called the *inductive step*. Let's see an example.

359 Example Prove that the sum of the first n odd integers is n^2. That is, show that $\sum_{i=1}^{n}(2i-1) = n^2$ for all $n \geq 1$.

Proof:

Let $P(n)$ be the statement "$\sum_{i=1}^{n}(2i-1) = n^2$".[2]
Since $\sum_{i=1}^{1}(2i-1) = 2 \cdot 1 - 1 = 1 = 1^2$, $P(1)$ is true (the base case).
Now let $k \geq 1$ and assume $P(k)$ is true. That is, $\sum_{i=1}^{k}(2i-1) = k^2$ (the inductive hypothesis).
Then (the inductive step)

$$\begin{aligned}
\sum_{i=1}^{k+1}(2i-1) &= \sum_{i=1}^{k}(2i-1) + (2(k+1)-1) \\
&= k^2 + (2k+2-1) \\
&= k^2 + 2k + 1 \\
&= (k+1)^2
\end{aligned}$$

[1] We can also write this as the tautology $[p \wedge (p \to q)] \to q$.

[2] Notice the quotes in the statement. It is important that you include these. This is particularly important if you use notation such as $P(n) =$"$\sum_{i=1}^{n}(2i-1) = n^2$", which is common. Without the quotes, this becomes $P(n) = \sum_{i=1}^{n}(2i-1) = n^2$, which is defining $P(n)$ to be $\sum_{i=1}^{n}(2i-1)$ and saying that it is also equal to n^2. These are *not* saying the same thing.

> Thus $P(k+1)$ is true. Since we proved that $P(1)$ is true, and that $P(k) \to P(k+1)$ whenever $k \geq 1$, $P(n)$ is true for all $n \geq 1$ by the principle of mathematical induction.
>
> □

Notice that the proof had four components (or three if you lump the second and third together), each of which is necessary:

1. We proved the statement for the *base case*.
2. We assumed it was true for k. That is, we made the *inductive hypothesis*.
3. We proved that it was true for $k+1$ based on the assumption that it is true for k. That is, we did the *inductive step*.
4. We appealed to the principle of mathematical induction in the *summary*.

It is important to note that explicitly defining $P(k)$ and using it throughout is not necessary. The whole proof could have been written without defining $P(k)$, although the proof would have been longer. In other words, we often use $P(k)$ for convenience and clarity.

The form of induction we have discussed up to this point only assumes the statement is true for one value of k. This is sometimes called *weak induction*. In *strong induction*, we assume that the statement is true for all values up to and including k. In other words, with strong induction, the inductive hypothesis involves proving that

$$[P(a) \wedge P(a+1) \wedge \cdots \wedge P(k)] \to P(k+1) \text{ if } k \geq a.$$

This may look more complicated, but practically speaking, there is really very little difference. Essentially, strong induction just allows us to assume *more* than weak induction. Let's see an example of why we might need strong induction.

360 Example Show that every integer $n \geq 2$ can be written as the product of primes.

> **Proof:** Let $P(n)$ be the statement "n can be written as the product of primes." We need to show that for all $n \geq 2$, $P(n)$ is true.
> **Base case:** Since 2 is clearly prime, it can be written as the product of one prime. Thus $P(2)$ is true.
> **Inductive Hypothesis:** Assume $[P(2) \wedge P(3) \wedge \cdots \wedge P(k-1)]$ is true for $k > 2$.
> **Inductive Step:** We need to show that $P(k)$ is true. If k is prime, clearly $P(k)$ is true. If k is not prime, then we can write $k = a \cdot b$, where $2 \leq a \leq b < k$. By hypothesis, $P(a)$ and $P(b)$ are true, so a and b can be written as the product of primes. Therefore, k can be written as the product of primes, namely the primes from the factorizations of a and b. Thus $P(k)$ is true.
> **Summary:** Since we proved that $P(2)$ is true, and that $[P(2) \wedge P(3) \wedge \cdots \wedge P(k-1)] \to P(k)$ if $k > 2$, by the principle of mathematical induction, $P(n)$ is true for all $n \geq 2$. That is, every integers $n \geq 2$ can be written as the product of primes. □

Notice that there is no way we could have used weak induction in the previous example. Also notice that we labeled the four parts of the proof. Although that is not required, I highly recommend labeling at least the *base case* and *inductive step* for a while.

You can split an induction proof into more than four steps. Here I outline one way to approach induction proofs that I think helps you work through the whole concept of induction. These steps and this approach are not required, but I think if you use it for your first several proofs, it will help you immensely.

1. **Define:** Define $P(n)$ based on the statement.
   $P(n)$ should be a statement about a single instance, not about a series of instances. For example, it should be statements like "$2n$ is even" or "A set with n elements has 2^n subsets," *NOT* of the form "$2n$ is even if $n > 1$," "$n^2 > 0$ if $n \neq 0$," or "For all $n > 1$, a set with n elements has 2^n subsets."

2. **Rephrase:** Rephrase the statement using $P(n)$. This step is mostly for your own clarity.
   In almost all cases, the rephrased statement should be "For all $n \geq a$, $P(n)$ is true," where a is some constant, often 0 or 1. If the statement cannot be phrased in this way, induction may not be appropriate.

3. **Base Case:** Prove the base case or cases.
   For most statements, this means showing that $P(a)$ is true, where a is the value from the rephrased statement. Sometimes one must prove multiple base cases, usually $P(a)$, $P(a+1)$, $\ldots$, $P(a+i)$ for some $i > 0$. For most statements, 1 or 2 base cases suffice.

4. **Hypothesis:** Write down what you are assuming. This is almost always one of

   $$P(k) \text{ is true},$$

   $$P(k-1) \text{ is true},$$

   or

   $$[P(a) \wedge P(a+1) \wedge \cdots \wedge P(k)] \text{ is true.}$$

   Sometimes it is helpful to write out the hypothesis explicitly (that is, write down the whole statement with k or $k-1$ plugged in).

5. **Goal:** Explicitly write down what your next step is. This is another step that is mostly for clarity.
   It is almost always "I need to show that $P(k+1)$ is true" (or "I need to show that $P(k)$ is true").

   Note: At this point in the proof you should *not* write out $P(k+1)$ unless you preface it with a statement like "I need to show that...". Since you are about to prove that $P(k+1)$ is true, you don't know that it is true yet, so writing it down as if it is a fact is incorrect and confusing.

6. **Inductive:** Given the **Hypothesis**, prove the **Goal** statement.
   This is the longest, and most varied, part of the proof. Once you get the hang of induction, you will typically only think about two parts of the proof—the base case and this step. The rest will become second nature.

   Note: The inductive step should *not* start with writing down $P(k+1)$. Especially when P involves a formula, some students want to write out $P(k+1)$ and work both sides until they get them to be the same. This is *not* a proper proof technique. You cannot start with something you do not know and then work it until you get to something you do know and then declare it is true. I will have more to say on this later.

7. **Summary:** Almost always either:
   "Since we proved that $P(a)$ is true, and that $P(k) \to P(k+1)$, by *PMI*, $P(n)$ is true for all $n \geq a$."
   or
   "Since we proved that $P(a)$ is true, and that $[P(a) \wedge P(a+1) \wedge \cdots \wedge P(k)] \to P(k+1)$, by *PMI*, $P(n)$ is true for all $n \geq a$."

Let's see this approach in action.

361 Example Prove that $n < 2^n$ for all integers $n \geq 1$.

Proof: For this proof, I will explicitly show the steps as suggested above.

Define: Let $P(n)$ be the statement "$n < 2^n$".
Rephrase: We want to prove that $P(n)$ is true for all $n \geq 1$.
Base Case: Since $1 < 2^1$, $P(1)$ is clearly true.
Hypothesis: We assume $P(k)$ is true if $k \geq 1$. That is, $k < 2^k$
Goal: We need to show that $P(k+1)$is true.
Inductive: By hypothesis (since $P(k)$ is true), we know that $k < 2^k$, thus

$$\begin{array}{rcll} k+1 & < & 2^k+1 & \text{adding 1 to both sides of previous} \\ & < & 2^k+2^k & \text{since } 1 < 2^k \text{ when } k \geq 1 \\ & = & 2(2^k) & \text{algebra} \\ & = & 2^{k+1} & \text{algebra} \end{array}$$

Since we have shown that $k+1 < 2^{k+1}$, $P(k+1)$ is true.
Summary: Since we proved that $P(1)$ is true, and that $P(k) \to P(k+1)$, by *PMI*, $P(n)$ is true for all $n \geq 1$.

□

Here is one final tip that is often relevant to the *inductive* step. Many statements $P(k)$ are of the form "$LHS(k) = RHS(k)$,"[3] where $=$ might be replaced with $\geq$, $\leq$, etc. For instance, if $P(k)$ is the statement "$k > 2^k$", $LHS(k) = k$, and $RHS(k) = 2^k$. In these cases, the goal of the induction step is to show that $LHS(k+1) = RHS(k+1)$ given that $LHS(k)$=$RHS(k)$. The way this is usually done is as follows:

$$\begin{array}{rcll} LHS(k+1) & = & LHS(k)+\text{stuff} & \text{applying algebra} \\ & = & RHS(k)+\text{stuff} & \text{by hypothesis (since } P(k) \text{ is true)} \\ & = & \cdots & \text{1 or most steps, usually involving algebra} \\ & = & RHS(k+1) & \text{more algebra, resulting in the goal} \end{array}$$

Several of the examples in this section follow this pattern, including the first two examples you saw. Notice that these example *do not* begin the inductive step by writing out $LHS(k+1) = RHS(k+1)$. They start by writing $LHS(k+1)$, and use algebra, etc. until they get to $RHS(k+1)$. You should do the same.

362 Example Prove the generalized form of DeMorgan's law. That is, show that for any $n \geq 2$, if p_1, p_2, …, p_n are propositions, then $\neg(p_1 \vee p_2 \vee \cdots \vee p_n) = (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_n)$.

Proof: To see that there are variations on how induction proofs can be written, we provide several proofs of this one. Each one is shorter than the previous one.
Proof 1: (Using the approach described above)

Define: Let $P(n)$ be the statement "$\neg(p_1 \vee p_2 \vee \cdots \vee p_n) = (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_n)$."
Rephrase: For all $n \geq 2$, $P(n)$ is true.
Base Case: $P(2)$ is DeMorgan's law, which is clearly true.
Hypothesis: We assume $P(k)$ is true.

[3] *LHS* stands for *left hand side* and *RHS* stands for *right hand side*

Goal: We need to show that $P(k+1)$is true.
Induction: Notice that

$$\begin{array}{rcll}\neg(p_1 \vee p_2 \vee \cdots \vee p_{k+1}) & = & \neg((p_1 \vee p_2 \vee \cdots \vee p_k) \vee p_{k+1}) & \text{associative law} \\ & = & \neg(p_1 \vee p_2 \vee \cdots \vee p_k) \wedge \neg p_{k+1} & \text{DeMorgan's law} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k) \wedge \neg p_{k+1} & \text{hypothesis} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k \wedge \neg p_{k+1}) & \text{associative law}\end{array}$$

Thus $P(k+1)$ is true.
Summary: Since we proved that $P(2)$ is true, and that $P(k) \to P(k+1)$ if $k \geq 2$, by the *PMI*, $P(n)$ is true for all $n \geq 2$.

Proof 2: (A typical proof)
Let $P(n)$ be the statement "$\neg(p_1 \vee p_2 \vee \cdots \vee p_n) = (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_n)$." We want to show that for all $n \geq 2$, $P(n)$ is true. $P(2)$ is DeMorgan's law, so the base case is true. Assume $P(k)$ is true. Then

$$\begin{array}{rcll}\neg(p_1 \vee p_2 \vee \cdots \vee p_{k+1}) & = & \neg((p_1 \vee p_2 \vee \cdots \vee p_k) \vee p_{k+1}) & \text{associative law} \\ & = & \neg(p_1 \vee p_2 \vee \cdots \vee p_k) \wedge \neg p_{k+1} & \text{DeMorgan's law} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k) \wedge \neg p_{k+1} & \text{hypothesis} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k \wedge \neg p_{k+1}) & \text{associative law}\end{array}$$

Thus $P(k+1)$ is true. Since we proved that $P(2)$ is true, and that $P(k) \to P(k+1)$ if $k \geq 2$, by the *PMI*, $P(n)$ is true for all $n \geq 2$.

Proof 3: (Not explicitly defining/using $P(n)$)
We know that $\neg(p_1 \vee p_2) = (\neg p_1 \wedge \neg p_2)$ since this is simply DeMorgan's law. Assume the statement is true for k. That is, $\neg(p_1 \vee p_2 \vee \cdots \vee p_k) = (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k)$. Then we can see that

$$\begin{array}{rcll}\neg(p_1 \vee p_2 \vee \cdots \vee p_{k+1}) & = & \neg((p_1 \vee p_2 \vee \cdots \vee p_k) \vee p_{k+1}) & \text{associative law} \\ & = & \neg(p_1 \vee p_2 \vee \cdots \vee p_k) \wedge \neg p_{k+1} & \text{DeMorgan's law} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k) \wedge \neg p_{k+1} & \text{hypothesis} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k \wedge \neg p_{k+1}) & \text{associative law}\end{array}$$

Thus the statement is true for $k+1$. Since we have shown that the statement is true for $n=2$, and that whenever it is true for k it is true for $k+1$, by the *PMI*, the statement is true for all $n \geq 2$.

Proof 4: (Shorter base case proof, no restatement of hypothesis)
The case $k=2$ is DeMorgan's law. Assume the statement is true for k. Then

$$\begin{array}{rcll}\neg(p_1 \vee p_2 \vee \cdots \vee p_{k+1}) & = & \neg((p_1 \vee p_2 \vee \cdots \vee p_k) \vee p_{k+1}) & \text{associative law} \\ & = & \neg(p_1 \vee p_2 \vee \cdots \vee p_k) \wedge \neg p_{k+1} & \text{DeMorgan's law} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k) \wedge \neg p_{k+1} & \text{hypothesis} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k \wedge \neg p_{k+1}) & \text{associative law}\end{array}$$

Thus the statement is true for $k+1$. Since we have shown that the statement is true for $n=2$, and that whenever it is true for k it is true for $k+1$, by the *PMI*, the statement is true for all

$n \geq 2$.

Proof 5: (Shorter summary)
The case $k = 2$ is DeMorgan's law. Assume the statement is true for k. Then

$$\begin{array}{rcll} \neg(p_1 \vee p_2 \vee \cdots \vee p_{k+1}) & = & \neg((p_1 \vee p_2 \vee \cdots \vee p_k) \vee p_{k+1}) & \text{associative law} \\ & = & \neg(p_1 \vee p_2 \vee \cdots \vee p_k) \wedge \neg p_{k+1} & \text{DeMorgan's law} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k) \wedge \neg p_{k+1} & \text{hypothesis} \\ & = & (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k \wedge \neg p_{k+1}) & \text{associative law} \end{array}$$

Thus the statement is true for $k+1$. By the *PMI*, the statement is true for all $n \geq 2$.

Proof 6: (Algebra steps not justified)
The case $k = 2$ is DeMorgan's law, and if the statement is true for k, then

$$\neg(p_1 \vee \cdots \vee p_{k+1}) = \neg(p_1 \vee \cdots \vee p_k) \wedge \neg p_{k+1} = (\neg p_1 \wedge \cdots \wedge \neg p_k) \wedge \neg p_{k+1} = (\neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_k \wedge \neg p_{k+1})$$

Thus the statement is true for $k+1$. By the *PMI*, the statement is true for all $n \geq 2$.

Proof 7: (An unacceptable proof for this class, but common in journal articles)
The result follows easily by induction. □

363 Example Prove that the expression

$$3^{3n+3} - 26n - 27$$

is a multiple of 169 for all natural numbers n.

Proof: Let $P(n)$ be the assertion "$\exists T \in \mathbb{N}$ with $3^{3n+3} - 26n - 27 = 169T$." We will prove that $P(1)$ is true and that $P(n-1) \to P(n)$. When $n = 1$ notice that $3^{3\cdot 1+3} - 26 \cdot 1 - 27 = 676 = 169 \cdot 4$, so $P(1)$ is true. Now, $P(n-1)$ means there is $N \in \mathbb{N}$ such that $3^{3(n-1)+3} - 26(n-1) - 27 = 169N$, i.e., for $n > 1$,

$$3^{3n} - 26n - 1 = 169N$$

for some integer N. Then

$$\begin{array}{rcl} 3^{3n+3} - 26n - 27 & = & 27 \cdot 3^{3n} - 26n - 27 \\ & = & 27(3^{3n} - 26n - 1) + 676n \\ & = & 27 \cdot 169N + 169 \cdot 4n \\ & = & 169(27 \cdot N + \cdot 4n) \end{array}$$

which is divisible by 169. The assertion is thus established by induction. □

364 Example Prove that if k is odd, then $2^{n+2} | k^{2^n} - 1$ for all natural numbers n.

Proof: The statement is evident for $n = 1$ since $k^{2^1} - 1 = k^2 - 1 = (k-1)(k+1)$ is divisible by $2^{1+2} = 8$ for any odd natural number k because both $(k-1)$ and $(k+1)$ are divisible by 2 and one of them is divisible by 4. Assume that $2^{n+2} | k^{2^n} - 1$, and let us prove that $2^{n+3} | k^{2^{n+1}} - 1$. Since $k^{2^{n+1}} - 1 = (k^{2^n} - 1)(k^{2^n} + 1)$, we see that 2^{n+2} divides $(k^{2n} - 1)$, so the problem reduces to proving that $2 | (k^{2n} + 1)$. This is obviously true since k^{2n} odd makes $k^{2n} + 1$ even. □

365 Example Let f_n be the n-th Fibonacci number. Prove that for integer $n \geq 1$,

$$f_{n-1}f_{n+1} = f_n^2 + (-1)^n.$$

Proof: For $n = 1$, we have

$$f_0 f_2 = 0 \cdot 1 = 1^2 + (-1)^1 = f_1^2 + (-1)^1,$$

and so the assertion is true for $n = 1$. Suppose $n > 1$, and that the assertion is true for n. That is,

$$f_{n-1}f_{n+1} = f_n^2 + (-1)^n,$$

which can be rewritten as

$$f_n^2 = f_{n-1}f_{n+1} - (-1)^n$$

Then

$$\begin{array}{rcl} f_n f_{n+2} & = & f_n(f_{n+1} + f_n) \\ & = & f_n f_{n+1} + f_n^2 \\ & = & f_n f_{n+1} + f_{n-1}f_{n+1} - (-1)^n \\ & = & f_{n+1}(f_n + f_{n-1}) + (-1)^{n+1} \\ & = & f_{n+1}f_{n+1} + (-1)^{n+1} \\ & = & f_{n+1}^2 + (-1)^{n+1}, \end{array}$$

and so the assertion follows by induction. □

☞ *There is an important but subtle point that should be made. Whether you assume $P(k)$ or $P(k-1)$ is true, you must specify the values of k precisely based on your choice. For instance, if you assume $P(k)$ is true for all $k > a$, you have a problem. Although you known $P(a)$ is true (because it is a base case), when you assume $P(k)$ is true for $k > a$, the smallest k can be is $a+1$. In other words, when you prove $P(k) \to P(k+1)$, you leave out $P(a) \to P(a+1)$. But that means you can't get anywhere from the base case, so the whole proof is invalid.*

366 Example What is wrong with the following (supposed) proof that $a^n = 1$ for $n \geq 0$:

Proof: *Base case:* Since $a^0 = 1$, the statement is true for $n = 0$.
Inductive step: Assume $a^j = 1$ for $0 \leq j \leq k$. Then

$$a^{k+1} = \frac{a^k \cdot a^k}{a^{k-1}} = \frac{1 \cdot 1}{1} = 1.$$

Summary: Therefore by PMI, $a^n = 1$ for all $n \geq 0$. □

Solution: The base case is correct, and there is nothing wrong with the summary, assuming the inductive step is correct. The fact that $a^k = 1$ and $a^{k-1} = 1$ are correct by the inductive hypothesis. Since $j \geq 1$, the algebra is correct. So what is wrong? Notice that if we had allowed $j = 0$, a^{-1} would be in the denominator, but we don't know whether or not $a^{-1} = 1$. Thus we needed to assume $j > 0$. As it turns out, that is precisely where the problem lies. We proved that $P(0)$ is true and that $P(k) \to P(k+1)$ is true when $k > 0$. Thus, we know that $P(1) \to P(2)$, and $P(2) \to P(3)$, etc., but we never showed that $P(0) \to P(1)$ because, of course, it isn't true. The induction doesn't work without $P(0) \to P(1)$.

Induction proofs are both intuitive and non-intuitive. One the one hand, when you talk through the idea, it seems to make sense. On the other hand, it almost seems like you are using *circular reasoning*. It is important to understand that induction proofs do *not* rely on circular reasoning. Circular reasoning is when you assume p in order to prove p. But here we are not doing that. We are assuming $P(k)$ and using that fact to prove $P(k+1)$, a different statement. However, we are *not* assuming that $P(k)$ is true for all $k \geq a$. We are proving that ***if we assume that*** $P(k)$ ***is true***, then $P(k+1)$ is true. The difference between these statements may seem subtle, but it is important.

Exercises

367 Problem Prove by induction that if n non-parallel straight lines on the plane intersect at a common point, they divide the plane into $2n$ regions.

368 Problem Demonstrate by induction that no matter how n straight lines divide the plane, it is always possible to colour the regions produced in two colours so that any two adjacent regions have different colours.

369 Problem Prove, by induction on n, that $1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n = 2 + (n-1)2^{n+1}$.

Answers

367 The assertion is clear for $n = 1$ since a straight line divides the plane into two regions. Assume P_{n-1}, that is, that $n-1$ non-parallel straight lines intersecting at a common point divide the plane into $2(n-1) = 2n-2$ regions. A new line non-parallel to them but passing through a common point will lie between two of the old lines, and divide the region between them into two more regions, producing then $2n-2+2 = 2n$ regions, demonstrating the assertion.

368 For $n = 1$ straight lines this is clear. Assume P_{n-1}, the proposition that this is possible for $n-1 > 1$ lines is true. So consider the plane split by $n-1$ lines into regions and coloured as required. Consider now a new line added to the $n-1$ lines. This line splits the plane into two regions, say I and II. We now do the following: in region I we leave the original coloration. In region II we switch the colours. We now have a coloring of the plane in the desired manner. For, either the two regions lie completely in region I or completely in region II, and they are coloured in the desired manner by the induction hypothesis. If one lies in region I and the other in region II, then they are coloured in the prescribed manner because we switched the colours in the second region.

369 For $n = 1$ we have $1 \cdot 2 = 2 + (1-1)2^2$, and so the statement is true for $n = 1$. Assume the statement is true for n, that is, assume

$$P(n) : 1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n = 2 + (n-1)2^{n+1}.$$

We need to show

$$P(n+1) : 1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + (n+1) \cdot 2^{n+1} = 2 + n2^{n+2}.$$

Adding $(n+1)2^{n+1}$ to both sides of $P(n)$ and simplifying the right side, we obtain

$$\begin{aligned} 1 \cdot 2 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n + (n+1)2^{n+1} &= 2 + (n-1)2^{n+1} + (n+1)2^{n+1} \\ &= 2 + 2n2^{n+1} \\ &= 2 + n2^{n+2}, \end{aligned}$$

proving $P(n+1)$. The result follows by induction.

6.2 Recursion

You have seen examples of recursion if you have seen Russian Matryoshka dolls (Google it), two almost parallel mirrors, a video camera pointed at the monitor, or a picture of a painter painting a picture of a painter painting a picture of a painter... More importantly for us, recursion is a very useful tool to implement algorithms.

370 Definition An algorithm is *recursive* if it calls itself.

Some problems can be solved by combining solutions of smaller instances of the given problem. Recursion can be useful in these cases. Examples that you may have already seen include *binary search*, *Quicksort*, and *Mergesort*.

If a subroutine/function simply called itself as a part of its execution, it would result in infinite recursion. This is a bad thing. Therefore, when using recursion, one must ensure that at some point, the subroutine/function terminates without calling itself. Before getting into more details, let's see an example.

371 Example Notice that

$$\begin{array}{rcllcl} 1! & = & 1 & & & \\ 2! & = & 2\times 1 & & = & 2\times 1! \\ 3! & = & 3\times 2\times 1 & & = & 3\times 2! \\ 4! & = & 4\times 3\times 2\times 1 & & = & 4\times 3! \\ & & \multicolumn{4}{l}{\quad\text{and in general, when } n>1} \\ n! & = & n\times(n-1)\times\cdots\times 2\times 1 & & = & n\times(n-1)! \end{array}$$

In other words, we can define $n!$ recursively as follows:

$$n! = \begin{cases} 1 & \text{when } n=1 \\ n*(n-1)! & \text{otherwise} \end{cases}$$

This leads to the following recursive algorithm to compute $n!$.

```
// Returns n!, assuming n>=0.
int factorial(int n) {
   if (n<=1)
      return 1;
   else
      return n*factorial(n-1);
}
```

Notice that if $n \leq 1$, `factorial` does not make a recursive call. Also notice that when a recursive call is made to `factorial`, the argument is smaller. Both of these are of critical importance, as we will see next.

Every recursive algorithm needs

❶ *Base case(s)*: One or more cases which are solved non-recursively. In other words, when an algorithm gets to the base case, it does not call itself again. This is also called a *stopping case* or *terminating condition*.

❷ *Inductive case(s)*: One or more recursive rule for all cases except the base case.

❸ *Progress:* The inductive case(s) should always progress toward the base case. Often this means the arguments will get smaller until they approach the base case, but sometimes it is more complicated than this.

In general, we can solve a problem with recursion if we can:

❶ Find one or more simple cases of the problem that can be solved directly.

❷ Find a way to break up the problem into smaller instances of the *same* problem.

❸ Find a way to combine the smaller solutions.

372 Example Implement an algorithm `countdown(int n)` that outputs the integers from n down to 1, where $n > 0$. So, for example, `countdown(5)` would output “5 4 3 2 1”.

Solution: One way to do this is with a simple loop:[4]

```
void countdown(int n) {
    for(i=n;i>0;i--)
       print(i);
}
```

Of course, if we did this, we wouldn’t learn anything about recursion. So, let’s consider how to do it with recursion. Notice that `countdown(n)` outputs n followed by the numbers from $n-1$ down to 1. But the numbers $n-1$ down to 1 are the output from `countdown(n-1)`. This leads to the following recursive algorithm:

```
void countdown(int n) {
    print(n);
    countdown(n-1):
}
```

To see if this is correct, we can trace through the execution of `countdown(3)` (see the table to the right). Unfortunately, `countdown` will never terminate. We are supposed to stop printing when $n = 1$, but we didn’t take that into account. To fix this, we can modify it so that a call to `countdown(0)` produces no output and does not call `countdown` again.

Execution of	outputs	then executes
`countdown(3)`	3	`countdown(2)`
`countdown(2)`	2	`countdown(1)`
`countdown(1)`	1	`countdown(0)`
`countdown(0)`	0	`countdown(-1)`
`countdown(-1)`	-1	`countdown(-2)`
⋮	⋮	⋮

Calls to `countdown(n)` should also produce no output when $n < 0$. The following algorithm takes care of both problems and is our final solution.

```
void countdown(int n) {
    if(n>0) {
        print(n);
        countdown(n-1):
    }
}
```

[4] For simplicity, we will sometimes use `print` to output results and not worry about spacing, etc. You can think of this as being equivalent to Java’s `System.out.print(i+" ")` or C++’s `cout<<i<<" "`, or C’s `printf("%d ",i)`.

Notice that when $n \leq 0$, `countdown(n)` does nothing, making $n \leq 0$ the *base cases*. When $n > 0$, `countdown(n)` calls `countdown(n-1)`, making $n > 0$ the *inductive cases*. Finally, when `countdown(n)` makes a recursive call it is to `countdown(n-1)`, so the inductive cases *progress* to the base case.

Although recursion is a great technique to solving many problems, care must be taken when using it. Not only is it easy to make simple mistakes like we did in the last example, but recursive algorithms often take more memory than iterative ones.

373 Example Consider our algorithms for $n!$. The iterative one from Example 72 uses memory to store four numbers: n, f, i, and return value.[5] The recursive one from Example 371 uses memory to store two numbers: n and the return value. Although the recursive algorithm uses less memory, it is called multiple times, and every call needs its own memory. For instance, a call to `factorial(3)` will call `factorial(2)` which will call `factorial(1)`. Thus, computing 3! requires enough memory to store 6 numbers, which is more than the 4 required by the iterative algorithm. In general, the recursive algorithm to compute $n!$ will need to store $2n$ numbers, whereas the iterative one will still just need 4.

Since computers have a finite amount of memory, and since every call to a function requires its own memory, there is a limit to how many recursive calls can be made in practice. In fact some languages, including Java, have a defined limit to how deep the recursion can be. Even for those that don't have a limit, if you run out of memory, you can certainly expect bad things to happen. This is one of the reasons recursion is avoided when possible. Good compilers remove recursion whenever possible, but it is not always possible.

There are many possible errors that one can make when implementing recursive algorithms. Let's see a few of the most common ones.

374 Example The following algorithm is supposed to sum the numbers from 1 to n:

```
void Sum1toN(int n) {
    if (n == 0)
        return(0);
    else
        return(n + Sum1toN(n-1));
}
```

Unfortunately the algorithm will go into infinite recursion if $n < 0$. Like our original solution to the `countdown` problem, the mistake here is an *improper base case*.

375 Example It is easy to get things backwards when recursion is involved. For instance, one of these routines prints from 1 up to n, the other from n down to 1. We leave it to you to figure out which is which.

```
void PrintN(int n) {
     if (n > 0) {
        PrintN(n-1);
        print(n);
        }
     }
```

```
void NPrint(int n) {
   if (N > 0) {
      print(n);
      NPrint(n-1);
      }
   }
```

[5] I won't get technical here, but memory needs to be allocated for the value returned by a function.

The next example is a classic example of a more subtle problem that can occur with recursive algorithms.

376 Example Recall the *Fibonacci sequence*, defined by the recurrence relation

$$f_n = \begin{cases} 0 & \text{if } n\text{=}0 \\ 1 & \text{if } n\text{=}1 \\ f_{n-1} + f_{n-2} & \text{if } n > 1. \end{cases}$$

Let's see an iterative and a recursive algorithm to compute f_n. The iterative algorithm (on the left) starts with f_0 and f_1 and computes each f_i based on f_{i-1} and f_{i-2} for i from 2 to n. As it goes, it needs to keep track of the previous two values. The recursive algorithm (on the right) just uses the definition and is pretty straightforward.

```
int Fib(int n) {
   int fib, fibm1, fibm2, index;
   if (n <= 1) return(n);
   else {
      fibm2 = 0;
      fibm1 = 1;
      index  = 1;
      while (index < n) {
          fib = fibm1 + fibm2;
          fibm2 = fibm1;
          fibm1 = fib;
          index = index + 1;
          }
      return(fib);
     }
}
```

```
 int FibR(int n) {
    if (n <= 1)
       return(n);
    else
       return(FibR(n-1) + FibR(n-2));
    }
}
```

Which algorithm is better? It is pretty clear that the recursive algorithm is much shorter and was a lot easier to write. It is also a lot easier to make a mistake implementing the iterative algorithm. However, it turns out that in this case the iterative version is actually *much better* when it comes to efficiency. In fact, computing f_{40} takes virtually no time with the iterative algorithm, but several seconds with the recursive algorithm. The recursive algorithm gets much worse as n increases. We will see why later.

We conclude this section by summarizing some of the advantages and disadvantages of recursion. The advantages include:

❶ Recursion often mimics the way we think about a problem, thus the recursive solutions can be very intuitive to program.

❷ Often recursive algorithms to solve problems are much shorter than iterative ones. This can make the code easier to understand, modify, and/or debug.

❸ The best known algorithms for many problems are based on a divide-and-conquer approach:

- Divide the problem into a set of smaller problems
- Solve each small problem separately

- Put the results back together for the overall solution

These divide-and-conquer techniques are often best thought of in terms of recursive functions.

Perhaps the main disadvantage of recursion is the extra time and space required. We have already discussed the extra space. The extra time comes from the fact that when a recursive call is made, the operating system has to record how to restart the calling subroutine later on, pass the parameters from the calling subroutine to the called subroutine (often by pushing the parameters onto a stack controlled by the system), set up space for the called subroutine's local variables, etc. The bottom line is that calling a function is not "free".

Example 376 demonstrates a more subtle disadvantage of recursion—the potential for hidden inefficiencies. On the other hand, if such inefficiencies are found, there are techniques that can often easily remove them (e.g. a technique called memoization[6]).

Exercises

377 Problem Prove that the recursive `countdown(n)` algorithm from Example 372 works correctly.

Answers

377 Notice that if $n \leq 0$, `countdown(0)` prints nothing, so it works in that case. For $k \geq 0$, assume `countdown(k)` works correctly.[7] Then `countdown(k+1)` will print 'k' and call `countdown(k)`. By the inductive hypothesis, `countdown(k)` will print '$kk-1\ldots 21$', so `countdown(k+1)` will print '$k+1kk-1\ldots 21$', so it works properly. By PMI, `countdown(n)` works for all $n \geq 0$.

6.3 Recurrence Relations

Recall that a *recurrence relation* is simply a sequence that is recursively defined. More formally, a recurrence relation is a formula that defines a_n in terms of a_i, for one or more values of $i < n$.[8]

378 Example We previously saw that we can define $n!$ by $0! = 1$, and if $n > 0$, $n! = n \cdot (n-1)!$. This is a recurrence relation for the sequence $n!$.

Similarly, recall the n-th Fibonacci number, given by $f_0 = f_1 = 1$ and for $n > 1$, $f_n = f_{n-1} + f_{n-2}$. This is recurrence relation for the sequence of Fibonacci numbers.

379 Example Each of the following are recurrence relations.

$$\begin{aligned} t_n &= n \cdot t_{n-1} + 4 \cdot t_{n-3} \\ r_n &= r_{n/2} + 1 \\ a_n &= a_{n-1} + 2 \cdot a_{n-2} + 3 \cdot a_{n-3} + 4 \cdot a_{n-4} \\ p_n &= p_{n-1} \cdot p_{n-2} \\ s_n &= s_{n-3} + n^2 - 4n + 32 \end{aligned}$$

[6] No, that's not a typo. Google it.

[7] We are letting $n = 0$ be the base case. You could also let $n = 1$ be the base case, but then you would need to prove that `countdown(1)` works.

[8] You might see recurrence relations written in terms of subscripted letters, like a_n, or using functional notation, like $a(n)$. Although there are technical differences between these notations, you can think of them as being essentially equivalent.

Notice that recurrence relations have 2 types of terms: *recursive* term(s) and the *non-recursive* terms. These are synonymous with the *inductive* and *base* cases of recursive algorithms. In the previous example, the recursive term of s_n is s_{n-3} and the non-recursive term is $n^2 - 4n + 32$.

In computer science, recurrence relations are used to analyze recursive algorithms. We won't get too technical yet, but let's see a simple example.

380 Example How many multiplications are required to compute f_n using the algorithm from Example 371?

> **Solution:** Let M_n be the number of multiplications needed to compute f_n using the algorithm from Example 371. From the code, it is obvious that $M_1 = 0$. If $n > 1$, the algorithm uses one multiplication and then makes a recursive call to f_{n-1}. The recursive call does M_{n-1} multiplications. Therefore, $M_n = M_{n-1} + 1$.

Given a recurrence relation for a_n, you can't just plug in n and get an answer. For instance, if $a_n = n \cdot a_{n-1}$, and $a_1 = 1$, what is $a(100)$? Not obvious, is it? That is the reason why *solving* recurrence relations is so important. As mentioned previously, solving a recurrence relation simply means finding a *closed form expression* for it.

381 Example It is not too difficult to see that the recurrence from Example 380 has the solution $M(n) = n - 1$. To prove it, notice that with this assumption, $M_{n-1} + 1 = (n-2) + 1 = n - 1 = M_n$, so the solution is consistent with the recurrence relation.
We can also prove it with induction: We know that $M_1 = 0$, so the base case of $n = 1$ is true. Assume $M_k = k - 1$. Then we have

$$M_{k+1} = M_k + 1 = (k-1) + 1 = k,$$

so the formula is correct for $k+1$. Thus, by PMI, the formula is correct for all $k \geq 1$.

The last example demonstrates an important fact about recurrence relations used to analyze algorithms. The recursive terms come from when a recursive function calls itself. The non-recursive terms come from the other work that is done by the algorithm, including any splitting or combining of data that must be done.

382 Example Consider the *binary search* algorithm to find an item on a sorted list. Informally, the algorithm works as follows. We want to find a value v in a sorted array of size n.

- We compare the middle value m of the array to v.
- If the $m = v$, we are done.
- Else if $m < v$, we binary search the left half of the array.
- Else $(m > v)$, we binary search the right half of the array.
- Now, we have the same problem, but only half the size.

Here is an implementation of the algorithm:

```
    boolean binarySearch(int[] A,int First,int Last,int Value) {
        if(Last>=First) {
            int mid=(Last+First)/2;
            if(Value==A[mid])
                return true;
            else if(Value<A[mid])
                return binarySearch(A,First,mid-1,Value);
            else
                return binarySearch(A,mid+1,Last,Value);
        } else {
            return false;
        }
    }
```

Find a recurrence relation for the worst-case complexity of `binarySearch`.

Solution: Let $T(n)$ be the complexity of `binarySearch` for an array of size n. Notice that the only things done in the algorithm are to find the middle element, make a few comparisons, perhaps make a recursive call, and return a value. Aside from the recursive call, the amount of work done is constant. Notice that at most one recursive call is made, and that the array passed in is half the size. Therefore $T(n) = T(n/2)+1$.[9] We'll see how to solve this recurrence shortly.

383 Example Give a recurrence relation for the complexity of the following algorithm:

```
int Nothing(int n) {
     if(n>5) {
        return Nothing(n-1)+Nothing(n-1)+Nothing(n-5)+Nothing(sqrt(n));
     }
     else {
        return n;
     }
}
```

Solution: It is not hard to see that if $T(n)$ is the running time for `Nothing(n)`, then

$$T(n) = 2T(n-1)+T(n-5)+T(\sqrt{n})+O(1).$$

There is no general method to solve recurrences. There are many strategies, however. In the next few sections we will discuss four common techniques: *substitution method*, *iteration method*, *Master method* and *characteristic equation method* for linear recurrences.

6.3.1 Substitution Method

The *substitution method* might be better called the *guess and prove it by induction method*. Why? Because to use it, you first have to figure out what you think the solution is, and then you need to actually prove it. Because of the close tie between recurrence relations and induction, it is the most natural technique to use. Let's see an example.

[9] Technically, the recurrence relation is $T(n) = T(\lfloor n/2 \rfloor)+1$ since $n/2$ might not be an integer. It turns out that most of the time we can ignore the floors/ceilings and still obtain the correct answer.

384 Example Consider the recurrence

$$S(n) = \begin{cases} 1 & \text{when } n = 1 \\ S(n-1) + n & \text{otherwise} \end{cases}$$

Prove that the solution is $S(n) = \dfrac{n(n+1)}{2}$.

Proof: When $n = 1$, $S(1) = 1 = \frac{1(1+1)}{2}$. Assume that for $0 \leq j < k$, $S(j) = \frac{j(j+1)}{2}$. Then

$$\begin{aligned} S(k) &= S(k-1) + k & \text{(Definition of } S(k)) \\ &= \frac{(k-1)(k)}{2} + k & \text{(Inductive hypothesis)} \\ &= \frac{k^2 - k}{2} + k & \text{(The rest is just algebra)} \\ &= \frac{k^2 - k + 2k}{2} \\ &= \frac{k^2 + k}{2} \\ &= \frac{k(k+1)}{2} \end{aligned}$$

Thus, $S(n) = \frac{n(n+1)}{2}$ for all $n \geq 1$. □

385 Example Solve the recurrence

$$H_n = \begin{cases} 1 & \text{when } n = 1 \\ 2H_{n-1} + 1 & \text{otherwise} \end{cases}$$

Proof: Notice that $H_1 = 1$, $H_2 = 2 \cdot 1 + 1 = 3$, $H_3 = 2 \cdot 3 + 1 = 7$, and $H_4 = 2 \cdot 7 + 1 = 15$. It sure looks like $H_n = 2^n - 1$, but now we need to prove it. Since $H_1 = 1 = 2^1 - 1$, we have our base case of $n = 1$. Assume $H_n = 2^n - 1$. Then

$$\begin{aligned} H_{n+1} &= 2H_n + 1 \\ &= 2(2^n - 1) + 1 \\ &= 2^{n+1} - 1, \end{aligned}$$

and the result follows by induction. □

386 Example Why was the recursive algorithm to compute f_n from Example 376 so bad?

Solution: Let's count the number of additions `FibR(n)` computes since that is the main thing that the algorithm does.[10] Let $F(n)$ be the number of additions required to compute f_n using `FibR(n)`. Since `FibR(n)` performs one addition and then calls `FibR(n-1)` and `FibR(n-2)`, it is easy to see that

$$F(n) = F(n-1) + F(n-2) + 1,$$

[10] Alternatively, we could count the number of recursive calls made. This is reasonable since the amount of work done by the algorithm, aside from the recursive calls, is constant. Therefore, the time it takes to compute f_n is proportional to the number of recursive calls made. This would produce a slightly different answer, but they would be comparable.

where $F(0) = F(1) = 0$ is clear from the algorithm. We could use the method for linear recurrences that will be outlined later to solve this, but the algebra gets a bit messy. Instead, Let's see if we can figure it out by computing some values.

$$\begin{aligned}
F(0) &= 0 \\
F(1) &= 0 \\
F(2) &= F(1)+F(0)+1=1 \\
F(3) &= F(2)+F(1)+1=2 \\
F(4) &= F(3)+F(2)+1=4 \\
F(5) &= F(4)+F(3)+1=7 \\
F(6) &= F(5)+F(4)+1=12 \\
F(7) &= F(6)+F(5)+1=20
\end{aligned}$$

No pattern is evident unless you add one to each of these. If you do, you will get $1,1,2,3,5,8,13,21$, etc., which looks a lot like the Fibonacci sequence starting with f_1. So it appears $F(n) = f_{n+1} - 1$. To verify this, first notice that $F(0) = 0 = f_1 - 1$ and $F(1) = 0 = f_2 - 1$. Assume it holds for all values less than k. Then

$$\begin{aligned}
F(k) &= F(k-1)+F(k-2)+1 \\
&= f_k - 1 + f_{k-1} - 1 + 1 \\
&= f_k + fk - 1 - 1 \\
&= f_{k+1} - 1.
\end{aligned}$$

The result follows by induction.

So what does this mean? It means in order to compute f_n, `FibR(n)` performs $f_{n+1}+1$ additions. In other words, it computes f_n by adding a bunch of 0s and 1s, which doesn't seem very efficient. Since f_n grows exponentially (we'll see this in Example 403), then $F(n)$ does as well. That pretty much explains what is wrong with the recursive algorithm.

6.3.2 Iteration Method

With the iteration method, we expand the recurrence and express it as a summation dependent only on n and initial conditions. Then we evaluate the summation.

387 Example Solve the recurrence

$$R(n) = \begin{cases} 1 & \text{when } n = 1 \\ 2R(n/2) + n/2 & \text{otherwise} \end{cases}$$

Proof: We have

$$\begin{aligned}
R(n) &= 2R(n/2)+n/2 \\
&= 2(2R(n/4)+n/4)+n/2 \\
&\quad = 2^2R(n/4)+n/2+n/2 \\
&\quad = 2^2R(n/4)+n \\
&= 2^2(2R(n/8)+n/8)+n \\
&\quad = 2^3R(n/8)+n/2+n \\
&\quad = 2^3R(n/8)+3n/2 \\
&= 2^3(2R(n/16)+n/16)+3n/2 \\
&\quad = 2^4R(n/16)+n/2+3n/2 \\
&\quad = 2^4R(n/16)+2n \\
&\vdots \\
&= 2^kR(n/(2^k))+kn/2 \\
&= 2^{\log_2 n}R(n/(2^{\log_2 n}))+(\log_2 n)n/2 \\
&= nR(n/n)+(\log_2 n)n/2 \\
&= nR(1)+(\log_2 n)n/2 \\
&= n+(\log_2 n)n/2
\end{aligned}$$

□

Using this method requires a little abstract thinking and pattern recognition. It also requires good algebra skills. Care must be taken when doing algebra, especially with the non-recursive terms. Sometimes you should add/multiply (depending on context) them all together, and other times you should leave them as is. The problem is that it takes experience (i.e. practice) to determine which one is better in a given situation. The key is flexibility. If you try doing it one way and don't see a pattern, try another way.

Here is my suggestion for using this method

❶ Iterate enough times so you are certain of what the pattern is. Typically this means at least 3 or 4 iterations.

❷ As you iterate, make adjustments to your algebra as necessary so you can see the pattern. For instance, whether you write 2^3 or 8 can make a difference in seeing the pattern.

❸ Once you see the pattern, generalize it, writing what it should look like after k iterations.

❹ Determine the value of k that will get you to the base case, and then plug it in.

❺ Simplify.

388 Example Solve the recurrence

$$H_n = \begin{cases} 1 & \text{when } n=1 \\ 2H_{n-1}+1 & \text{otherwise} \end{cases}$$

Solution:

$$\begin{aligned} H_n &= 2H_{n-1}+1 \\ &= 2(2H_{n-2}+1)+1 = 2^2H_{n-2}+2+1 \\ &= 2^2(2H_{n-3}+1)+2+1 = 2^3H_{n-3}+2^2+2+1 \\ &\vdots \\ &= 2^{n-1}H_1+2^{n-2}+2^{n-3}+\cdots+2+1 \\ &= 2^{n-1}+2^{n-2}+2^{n-3}+\cdots+2+1 \\ &= 2^n-1 \end{aligned}$$

Thus, $H_n = 2^n - 1$.

6.3.3 Master Method

We will omit the proof of the following theorem which is particularly certain recursive algorithms.

389 Theorem (Master Theorem) Let $T(n)$ be a monotonically increasing function satisfying

$$\begin{aligned} T(n) &= aT(n/b)+f(n) \\ T(1) &= c \end{aligned}$$

where $a \geq 1$, $b \geq 2$, and $c > 0$. If $f(n) = \theta(n^d)$, where $d \geq 0$, then

$$T(n) = \begin{cases} \Theta(n^d) & if a < b^d \\ \Theta(n^d \log n) & if a = b^d \\ \Theta(n^{\log_b a}) & if a > b^d \end{cases}$$

390 Example Solve the recurrence

$$T(n) = 4T(n/2)+n.$$

Solution: We have $a = 4, b = 2$, and $d = 1$. Since $4 > 2^1$, $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$ by the third case of the Master Theorem.

391 Example Solve the recurrence

$$T(n) = 4T(n/2)+n^2.$$

Solution: We have $a = 4, b = 2$, and $d = 2$. Since $4 = 2^2$, we have $T(n) = \Theta(n^2 \log n)$ by the second case of the Master Theorem.

392 Example Solve the recurrence

$$T(n) = 4T(n/2)+n^3.$$

Solution: Here, $a = 4, b = 2$, and $d = 3$. Since $4 < 2^3$, we have $T(n) = \Theta(n^3)$ by the first case of the Master Theorem.

Wow. That was easy.[11] But the ease of use of the Master Method comes with a cost. Notice that we do not get an *exact* solution, but only an *asymptotic* solution. Depending on the context, this may be good enough. If you need an exact numerical solution, the Master Method will do you no good. But when analyzing algorithms, typically we are more interested in the asymptotic behavior. In that case, it works great.

393 Example Let's redo one from a previous section. Solve the recurrence

$$R(n)=\begin{cases}1 & \text{when } n=1\\ 2R(n/2)+n/2 & \text{otherwise}\end{cases}$$

Solution: Here, we have $a=2$, $b=2$, and $d=1$. Since $2=2^1$, $R(n)=\Theta(n^1\log n)=\Theta(n\log n)$. Recall that in Example 387 we showed that $R(n)=n+(\log_2 n)n/2$. Since $n+(\log_2 n)n/2=\Theta(n\log n)$, our solution is consistent.

394 Example What is the time complexity if *binary search*?

Solution: We previously saw that the number of comparisons needed for binary search is $T(n)=T(n/2)+1$. Here we have $a=1$, $b=2$, and $d=0$. Since $1=2^0$, the second case of the Master Theorem tells is that $T(n)=\Theta(n^0\log n)=\Theta(\log n)$.

6.3.4 Linear Recurrence Relations

Although in my mind linear recurrence relations are of the least importance of these four methods for computer scientists, we will discuss them briefly, both for completeness sake, and because we can talk about the Fibonacci number again.

395 Definition Let $c_0, c_2, \ldots, c_k$ be real constants and $f:\mathbb{N}\rightarrow\mathbb{R}$ a function. A recurrence relation of the form

$$c_0a_n+c_1a_{n+1}+c_2a_{n+2}+\cdots+c_ka_{n+k}=f(n), \qquad n\geq 0. \tag{6.1}$$

is called a *linear recurrence relation* (or *linear difference equation*). If f is identically zero, we say that the equation is *homogeneous*, and otherwise we say the equation is *nonhomogeneous*.

Notice that equation 6.1 above is equivalent to

$$a_{n+k}=c'_0a_n+c'_1a_{n+1}+c'_2a_{n+2}+\cdots+c'_{k-1}a_{n+k-1}+f(n),$$

where $c'_i=-c_i/c_{n+k}$, which is the form we are more used to. Since the technique used to solve these involves factoring polynomials, it is more convenient to have all of the terms on the same side of the equation, which is why we think of them in the form of equation 6.1 when solving them.

The *order* of the recurrence is the difference between the highest and the lowest subscripts. For example

$$u_{n+2}-u_{n+1}=2$$

is of the first order, and

$$u_{n+4}+9u_n^2=n^5$$

is of the fourth order.

There is a general technique that can be used to solve linear homogeneous recurrence relations. However, we will restrict our discussion to first and second order recurrences.

[11] Almost too easy...

First Order Recurrences

We outline a method for solving first order linear recurrence relations of the form

$$x_n = ax_{n-1} + f(n), a \neq 1,$$

where f is a polynomial.

1. First solve the homogeneous recurrence $x_n = ax_{n-1}$ by "raising the subscripts" in the form $x^n = ax^{n-1}$. This we call the *characteristic equation*. Canceling this gives $x = a$. The solution to the homogeneous equation $x_n = ax_{n-1}$ will be of the form $x_n = Aa^n$, where A is a constant to be determined.
2. Test a solution of the form $x_n = Aa^n + g(n)$, where g is a polynomial of the same degree as f.

396 Example Let $x_0 = 7$ and $x_n = 2x_{n-1}, n \geq 1$. Find a closed form for x_n.

Solution: Raising subscripts we have the characteristic equation $x^n = 2x^{n-1}$. Canceling, $x = 2$. Thus we try a solution of the form $x_n = A2^n$, were A is a constant. But $7 = x_0 = A2^0$ and so $A = 7$. The solution is thus $x_n = 7(2)^n$.

Here is a different method that sometimes works. We have:

$$\begin{array}{lcl} x_0 & = & 7 \\ x_1 & = & 2x_0 \\ x_2 & = & 2x_1 \\ x_3 & = & 2x_2 \\ \vdots & \vdots & \vdots \\ x_n & = & 2x_{n-1} \end{array}$$

Multiplying both columns,

$$x_0x_1 \cdots x_n = 7 \cdot 2^n x_0x_1x_2 \cdots x_{n-1}.$$

Canceling the common factors on both sides of the equality,

$$x_n = 7 \cdot 2^n.$$

397 Example Let $x_0 = 7$ and $x_n = 2x_{n-1} + 1, n \geq 1$. Find a closed form for x_n.

Solution: By raising the subscripts in the homogeneous equation we obtain $x^n = 2x^{n-1}$ or $x = 2$. A solution to the homogeneous equation will be of the form $x_n = A(2)^n$. Now $f(n) = 1$ is a polynomial of degree 0 (a constant) and so we test a particular constant solution C. The general solution will have the form $x_n = A2^n + B$. Now, $7 = x_0 = A2^0 + B = A + B$. Also, $x_1 = 2x_0 + 7 = 15$ and so $15 = x_1 = 2A + B$. Solving the simultaneous equations

$$A + B = 7,$$

$$2A + B = 15,$$

we find $A = 8, B = -1$. So the solution is $x_n = 8(2^n) - 1 = 2^{n+3} - 1$.

398 Example Let $x_0 = 2, x_n = 9x_{n-1} - 56n + 63$. Find a closed form for this recursion.

Solution: By raising the subscripts in the homogeneous equation we obtain the characteristic equation $x^n = 9x^{n-1}$ or $x = 9$. A solution to the homogeneous equation will be of the form $x_n = A(9)^n$. Now $f(n) = -56n + 63$ is a polynomial of degree 1 and so we test a particular solution of the form $Bn + C$. The general solution will have the form $x_n = A9^n + Bn + C$. Now $x_0 = 2, x_1 = 9(2) - 56 + 63 = 25, x_2 = 9(25) - 56(2) + 63 = 176$. We thus solve the system

$$2 = A + C,$$

$$25 = 9A + B + C,$$

$$176 = 81A + 2B + C.$$

We find $A = 2, B = 7, C = 0$. The general solution is $x_n = 2(9^n) + 7n$.

399 Example Let $x_0 = 1, x_n = 3x_{n-1} - 2n^2 + 6n - 3$. Find a closed form for this recursion.

Solution: By raising the subscripts in the homogeneous equation we obtain the characteristic equation $x^n = 3x^{n-1}$ or $x = 9$. A solution to the homogeneous equation will be of the form $x_n = A(3)^n$. Now $f(n) = -2n^2 + 6n - 3$ is a polynomial of degree 2 and so we test a particular solution of the form $Bn^2 + Cn + D$. The general solution will have the form $x_n = A3^n + Bn^2 + Cn + D$. Now $x_0 = 1, x_1 = 3(1) - 2 + 6 - 3 = 4, x_2 = 3(4) - 2(2)^2 + 6(2) - 3 = 13, x_3 = 3(13) - 2(3)^2 + 6(3) - 3 = 36$. We thus solve the system

$$1 = A + D,$$

$$4 = 3A + B + C + D,$$

$$13 = 9A + 4B + 2C + D,$$

$$36 = 27A + 9B + 3C + D.$$

We find $A = B = 1, C = D = 0$. The general solution is $x_n = 3^n + n^2$.

400 Example Find a closed form for $x_n = 2x_{n-1} + 3^{n-1}, x_0 = 2$.

Solution: We test a solution of the form $x_n = A2^n + B3^n$. Then $x_0 = 2, x_1 = 2(2) + 3^0 = 5$. We solve the system

$$2 = A + B,$$

$$7 = 2A + 3B.$$

We find $A = 1, B = 1$. The general solution is $x_n = 2^n + 3^n$.

We now tackle the case when $a = 1$. In this case, we simply consider a polynomial g of degree 1 higher than the degree of f.

401 Example Let $x_0 = 7$ and $x_n = x_{n-1} + n, n \geq 1$. Find a closed formula for x_n.

Solution: By raising the subscripts in the homogeneous equation we obtain the characteristic equation $x^n = x^{n-1}$ or $x = 1$. A solution to the homogeneous equation will be of the form $x_n = A(1)^n = A$, a constant. Now $f(n) = n$ is a polynomial of degree 1 and so we test a particular solution of the form $Bn^2 + Cn + D$, one more degree than that of f. The general solution will have the form $x_n = A + Bn^2 + Cn + D$. Since A and D are constants, we may combine them to obtain $x_n = Bn^2 + Cn + E$. Now, $x_0 = 7, x_1 = 7 + 1 = 8, x_2 = 8 + 2 = 10$. So we solve the system

$$7 = E,$$

$$8 = B + C + E,$$

$$10 = 4B + 2C + E.$$

We find $B = C = \frac{1}{2}, E = 7$. The general solution is $x_n = \frac{n^2}{2} + \frac{n}{2} + 7$.

Second Order Recurrences

All the recursions that we have so far examined are first order recursions, that is, we find the next term of the sequence given the preceding one. Let us now briefly examine how to solve some second order recursions.

We now outline a method for solving second order homogeneous linear recurrence relations of the form

$$x_n = ax_{n-1} + bx_{n-2}.$$

1. Find the characteristic equation by "raising the subscripts" in the form $x^n = ax^{n-1} + bx^{n-2}$. Canceling this gives $x^2 - ax - b = 0$. This equation has two roots r_1 and r_2.
2. If the roots are different, the solution will be of the form $x_n = A(r_1)^n + B(r_2)^n$, where A, B are constants.
3. If the roots are identical, the solution will be of the form $x_n = A(r_1)^n + Bn(r_1)^n$.

402 Example Let $x_0 = 1, x_1 = -1, x_{n+2} + 5x_{n+1} + 6x_n = 0$.

Solution: The characteristic equation is $x^2 + 5x + 6 = (x + 3)(x + 2) = 0$. Thus we test a solution of the form $x_n = A(-2)^n + B(-3)^n$. Since $1 = x_0 = A + B, -1 = -2A - 3B$, we quickly find $A = 2, B = -1$. Thus the solution is $x_n = 2(-2)^n - (-3)^n$.

403 Example Find a closed form for the Fibonacci recursion $f_0 = 0$, $f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$.

Solution: The characteristic equation is $f^2 - f - 1 = 0$, whence a solution will have the form

$$f_n = A\left(\frac{1+\sqrt{5}}{2}\right)^n + B\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

The initial conditions give

$$0 = A + B,$$

$$1 = A\left(\frac{1+\sqrt{5}}{2}\right) + B\left(\frac{1-\sqrt{5}}{2}\right) = \frac{1}{2}(A+B) + \frac{\sqrt{5}}{2}(A-B) = \frac{\sqrt{5}}{2}(A-B)$$

This gives $A = \frac{1}{\sqrt{5}}, B = -\frac{1}{\sqrt{5}}$. We thus have the *Cauchy-Binet Formula:*

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n \tag{6.2}$$

404 Example Solve the recursion $x_0 = 1, x_1 = 4, x_n = 4x_{n-1} - 4x_{n-2} = 0$.

Solution: The characteristic equation is $x^2 - 4x + 4 = (x-2)^2 = 0$. There is a multiple root and so we must test a solution of the form $x_n = A2^n + Bn2^n$. The initial conditions give

$$1 = A,$$

$$4 = 2A + 2B.$$

This solves to $A = 1, B = 1$. The solution is thus $x_n = 2^n + n2^n$.

Exercises

405 Problem (Lines on the Plane) Find a recurrence relation for the number of regions into which the plane is divided by n straight lines if every pair of lines intersect, but no three lines intersect.

406 Problem Solve the recursion $a_n = 1 + \sum_{k=1}^{n-1} a_k$ for $n \geq 2$ and $a_1 = 1$.

407 Problem Let $x_0 = 1, x_n = 3x_{n-1} - 2n^2 + 6n - 3$. Find a closed form for this recursion.

408 Problem Find a closed form for $x_n = 2x_{n-1} + 3^{n-1}, x_0 = 2$.

409 Problem Let $x_0 = 2, x_n = 9x_{n-1} - 56n + 63$. Find a closed form for this recursion.

410 Problem Let $x_0 = 7$ and $x_n = x_{n-1} + n, n \geq 1$. Find a closed formula for x_n.

411 Problem There are two urns, one is full of water and the other is empty. On the first stage, half of the contains of urn I is passed into urn II. On the second stage 1/3 of the contains of urn II is passed into urn I. On stage three, 1/4 of the contains of urn I is passed into urn II. On stage four 1/5 of the contains of urn II is passed into urn I, and so on. What fraction of water remains in urn I after the 1978th stage?

412 Problem (Derangements) An absent-minded secretary is filling n envelopes with n letters. Find a recurrence relation for the number D_n of ways in which she never stuffs the right letter into the right envelope.

Answers

405 Let a_n be this number. Clearly $a_1 = 2$. The nth line is cut by the previous $n-1$ lines at $n-1$ points, adding n new regions to the previously existing a_{n-1}. Hence

$$a_n = a_{n-1} + n,\ a_1 = 2.$$

Notice that

$$\begin{array}{lll} a_2 & = & a_1+2, \\ a_3 & = & a_2+3, \\ a_4 & = & a_3+4, \\ \vdots & \vdots & \vdots \\ a_{n-1} & = & a_{n-2}+(n-1), \\ a_n & = & a_{n-1}+n, \end{array}$$

Add these equalities together, we get

$$a_2+a_3+a_4+\cdots+a_{n-1}+a_n = a_1+a_2+a_3+a_4+\cdots+a_{n-1}+(2+3+\cdots+n).,$$

Solving for a_n yields

$$a_n = a_1+\left(\frac{n(n+1)}{2}-1\right) = \frac{n^2+n+2}{2}.$$

406 Observe that

$$a_n - a_{n-1} = \left(1+\sum_{k=1}^{n-1} a_k\right) - \left(1+\sum_{k=1}^{n-2} a_k\right) = a_{n-1}.$$

This means that $a_n = 2a_{n-1}$ and so

$$\begin{array}{lll} a_n & = & 2a_{n-1} \\ a_{n-1} & = & 2a_{n-2} \\ \vdots & \vdots & \vdots \\ a_2 & = & 2a_1 \end{array}$$

Multiplying all these equalities,

$$a_n a_{n-1}\cdots a_2 = 2^{n-1} a_{n-1} a_{n-2}\cdots a_1 \implies a_n = 2^{n-1}a_1 = 2^{n-1}.$$

407 $x_n = 3^n+n^2$. We leave it to the reader to verify this.

408 $x_n = 2^n+3^n$. We leave it to the reader to verify this.

409 $x_n = 2(9^n)+7n$. We leave it to the reader to verify this.

410 We have

$$\begin{array}{lll} x_0 & = & 7 \\ x_1 & = & x_0+1 \\ x_2 & = & x_1+2 \\ x_3 & = & x_2+3 \\ \vdots & \vdots & \vdots \\ x_n & = & x_{n-1}+n \end{array}$$

Adding both columns,

$$x_0+x_1+x_2+\cdots+x_n = 7+x_0+x_2+\cdots+x_{n-1}+(1+2+3+\cdots+n).$$

Cancelling and using the fact that $1+2+\cdots+n = \dfrac{n(n+1)}{2}$,

$$x_n = 7+\frac{n(n+1)}{2}.$$

411 Let $x_n, y_n, n = 0, 1, 2, \ldots$ denote the fraction of water in urns I and II respectively at stage n. Observe that $x_n + y_n = 1$ and that

$$\begin{array}{ll}
x_0 = 1; & y_0 = 0 \\
x_1 = x_0 - \frac{1}{2}x_0 = \frac{1}{2}; & y_1 = y_1 + \frac{1}{2}x_0 = \frac{1}{2} \\
x_2 = x_1 + \frac{1}{3}y_1 = \frac{2}{3}; & y_2 = y_1 - \frac{1}{3}y_1 = \frac{1}{3} \\
x_3 = x_2 - \frac{1}{4}x_2 = \frac{1}{2}; & y_1 = y_1 + \frac{1}{4}x_2 = \frac{1}{2} \\
x_4 = x_3 + \frac{1}{5}y_3 = \frac{3}{5}; & y_1 = y_1 - \frac{1}{5}y_3 = \frac{2}{5} \\
x_5 = x_4 - \frac{1}{6}x_4 = \frac{1}{2}; & y_1 = y_1 + \frac{1}{6}x_4 = \frac{1}{2} \\
x_6 = x_5 + \frac{1}{7}y_5 = \frac{4}{7}; & y_1 = y_1 - \frac{1}{7}y_5 = \frac{3}{7} \\
x_7 = x_6 - \frac{1}{8}x_6 = \frac{1}{2}; & y_1 = y_1 + \frac{1}{8}x_6 = \frac{1}{2} \\
x_8 = x_7 + \frac{1}{9}y_7 = \frac{5}{9}; & y_1 = y_1 - \frac{1}{9}y_7 = \frac{4}{9}
\end{array}$$

A pattern emerges (which may be proved by induction) that at each odd stage n we have $x_n = y_n = \frac{1}{2}$ and that at each even stage we have (if $n = 2k$) $x_{2k} = \frac{k+1}{2k+1}, y_{2k} = \frac{k}{2k+1}$. Since $\frac{1978}{2} = 989$ we have $x_{1978} = \frac{990}{1979}$.

412 Number the envelopes $1, 2, 3, \cdots, n$. We condition on the last envelope. Two events might happen. Either n and r (for some $1 \leq r \leq n-1$) trade places or they do not.

In the first case, the two letters r and n are misplaced. Our task is just to misplace the other $n-2$ letters, $(1, 2, \cdots, r-1, r+1, \cdots, n-1)$ in the slots $(1, 2, \cdots, r-1, r+1, \cdots, n-1)$. This can be done in D_{n-2} ways. Since r can be chosen in $n-1$ ways, the first case can happen in $(n-1)D_{n-2}$ ways.

In the second case, let us say that letter r, $(1 \leq r \leq n-1)$ moves to the n-th position but n moves not to the r-th position. Since r has been misplaced, we can just ignore it. Since n is not going to the r-th position, we may relabel n as r. We now have $n-1$ numbers to misplace, and this can be done in D_{n-1} ways.

As r can be chosen in $n-1$ ways, the total number of ways for the second case is $(n-1)D_{n-1}$. Thus $D_n = (n-1)D_{n-2} + (n-1)D_{n-1}$.

6.4 Analyzing Recursive Algorithms

In Section 6.3 we already saw a few examples of analyzing recursive algorithms. We will provide a few more examples in this section. In case it isn't clear, the most common method to analyze a recursive algorithm is to develop and solve a recurrence relation for its running time. Let's see some examples.

413 Example What is the worst-case running time of Mergesort?

> **Solution:** The algorithm for `Mergesort` is below. Let $T(n)$ be the worst-case running time of `Mergesort` on an array of size $n = right - left$. Recall that `Merge` takes two sorted arrays and merges them into one sorted array in time $\Theta(n)$, where n is the number of elements in both arrays. Since the two recursive calls to `Mergsort` are on arrays of half the size, they each require time $T(n/2)$ in the worst-case. The other operations take constant time, as indicated below.

Analysis of Mergesort

Algorithm	Time required
`Mergesort(int[] A,int left,int right) {`	$T(n)$
`   if (left < right) {`	C_1
`       int mid = (left + right)/2;`	C_2
`       Mergesort(A, left, mid);`	$T(n/2)$
`       Mergesort(A, mid + 1, right);`	$T(n/2)$
`       Merge(A, left, mid, right);`	$\Theta(n)$
`   }`	
`}`	

Given this, we can see that

$$\begin{aligned} T(n) &= C_1 + C_2 + T(n/2) + T(n/2) + \Theta(n) \\ &= 2T(n/2) + \Theta(n). \end{aligned}$$

For simplicity, we will write this as $T(n) = 2T(n/2) + cn$ for some constant c.

We could use the Master Theorem to prove that $T(n) = \Theta(n\log n)$, but that would be too easy. Instead, we will use induction to prove that $T(n) = O(n\log n)$, and leave the Ω-bound to the reader.

By definition, $T(n) = O(n\log n)$ if and only if there exists constants k and n_0 such that $T(n) \leq kn\log n$ for all $n \geq n_0$.

For the base case, notice that $T(2) = a$ for some constant a, and $a \leq k2\log 2 = 2k$ as long as we pick $k \geq a/2$. Now, assume that $T(n/2) \leq k(n/2)\log(n/2)$. Then

$$\begin{aligned} T(n) &= 2T(n/2) + cn \\ &\leq 2(k(n/2)\log(n/2) + cn \\ &= kn\log(n/2) + cn \\ &= kn\log n - kn\log 2 + cn \\ &= kn\log n + (c-k)n \\ &\leq kn\log n \quad \text{if } k \geq c \end{aligned}$$

As long as we pick $k = \max\{a/2, c\}$, we have $T(n) \leq kn\log n$, so $T(n) = O(n\log n)$ as desired.

414 Example (Towers of Hanoi) The following legend is attributed to French mathematician Edouard Lucas in 1883. The tower of Brahma had 64 disks of gold resting on three diamond needles. At the beginning of time, God placed these disks on the first needle and ordained that a group of priests should transfer them to the third needle according to the following rules:

1. The disks are initially stacked on peg A, in decreasing order (from bottom to top).

2. The disks must be moved to another peg in such a way that only one disk is moved at a time and without stacking a larger disk onto a smaller disk.

When they finish, the Tower will crumble and the world will end. How many moves does it take to solve the *Towers of Hanoi* problem with n disks?

Solution: The usual (and best) algorithm to solve the *Towers of Hanoi* is as follows:

- Move the top $n-1$ disk to from peg 1 to peg 2.
- Move the last disk from peg 1 to peg 3.
- Move the top $n-1$ disks from peg 2 to peg 3.

The only question is how to move the top $n-1$ disks. The answer is simple: using the same algorithm (with the peg numbers switched). Don't worry if you don't see why this works. Our main concern here is analyzing the algorithm.

Let $H(n)$ be the time required to solve the *Towers of Hanoi* problem with n disks. Assuming moving a single disk takes 1 operations (so $H(1) = 1$), the above algorithm requires

$$H(n) = H(n-1) + 1 + H(n-1) = 2H(n-1) + 1$$

operations. As with the first example, we want a closed form for $H(n)$. But we already showed that $H(n) = 2^n - 1$ in Examples 385 and 388.

6.4.1 The Average Complexity of Quicksort

In this section we give a proof that the average case running time of randomized quicksort is $\Theta(n \log n)$. This proof gets its own section because the analysis is fairly involved. This proof is based on the one presented in Section 8.4 of the classic *Introduction to Algorithms* by Cormen, Leiserson, and Rivest. The algorithm they give is slightly different, and they include some interesting insights, so read their proof/discussion if you get a chance.

There are several slight variations of the quicksort algorithm, and although the exact running times are different for each, the asymptotic running times are all the same. We begin by presenting the following version of `Quicksort`, written in C++.

```
Quicksort(int A[],int l, int r) {
   if (r > l) {
     int p = RPartition(A,l,r);
     Quicksort(A,l,p-1);
     Quicksort(A,p+1,r);
     }
}
```

```
int RPartition(int A[], int l, int r) {
    int piv=l+(rand()%(r-l+1);
    Swap(A[l],A[piv]);
    int i = l+1;
    int j = r;
    while (1) {
        while (A[i] <= A[l] && i<r)
            ++i;
        while (A[j] >= A[l] && j>l)
            --j;
        if (i >= j) {
           Swap(A[j],A[l]);
           return j;
           }
        else Swap(A[i],A[j]);
        }
}
```

We will base our analysis on this version of `Quicksort`. It is straightforward to see that the runtime of `RPartition` is $\Theta(n)$. (A proof of this is left to the reader). We start by developing a recurrence relation for the average case runtime of `Quicksort`.

415 Theorem Let $T(n)$ be the average case runtime of `Quicksort` on an array of size n. Then

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n).$$

Proof: Since the pivot element is chosen randomly, it is equally likely that the pivot will end up at any position from l to r. That is, the probability that the pivot ends up at location $l+i$ is $1/n$ for each $i=0,\ldots,r-l$. If we average over all of the possible pivot locations, we obtain

$$\begin{aligned}
T(n) &= \frac{1}{n}\left(\sum_{k=0}^{n-1}(T(k)+T(n-k-1))\right)+\Theta(n)\\
&= \frac{1}{n}\sum_{k=0}^{n-1}T(k)+\frac{1}{n}\sum_{k=0}^{n-1}T(n-k-1)+\Theta(n)\\
&= \frac{1}{n}\sum_{k=0}^{n-1}T(k)+\frac{1}{n}\sum_{k=0}^{n-1}T(k)+\Theta(n)\\
&= \frac{2}{n}\sum_{k=0}^{n-1}T(k)+\Theta(n)\\
&= \frac{2}{n}\sum_{k=1}^{n-1}T(k)+\Theta(n).
\end{aligned}$$

The last step holds since $T(0)=0$. □

We will need the following result in order to solve the recurrence relation.

416 Lemma For any $n\geq 3$,

$$\sum_{k=2}^{n-1}k\log k\leq\frac{1}{2}n^2\log n-\frac{1}{8}n^2.$$

Proof: We can write the sum as

$$\sum_{k=2}^{n-1}k\log k=\sum_{k=2}^{\lceil n/2\rceil-1}k\log k+\sum_{k=\lceil n/2\rceil}^{n-1}k\log k$$

Then we can bound $(k\log k)$ by $(k\log(n/2))=k(\log n-1)$ in the first sum, and by $(k\log n)$ in the second sum. This gives

$$\begin{aligned}
\sum_{k=2}^{n-1} k\log k &= \sum_{k=2}^{\lceil n/2\rceil -1} k\log k + \sum_{k=\lceil n/2\rceil}^{n-1} k\log k \\
&\leq \sum_{k=2}^{\lceil n/2\rceil -1} k(\log n - 1) + \sum_{k=\lceil n/2\rceil}^{n-1} k\log n \\
&= (\log n - 1)\sum_{k=2}^{\lceil n/2\rceil -1} k + \log n \sum_{k=\lceil n/2\rceil}^{n-1} k \\
&= \log n \sum_{k=2}^{\lceil n/2\rceil -1} k - \sum_{k=2}^{\lceil n/2\rceil -1} k + \log n \sum_{k=\lceil n/2\rceil}^{n-1} k \\
&= \log n \sum_{k=2}^{n-1} k - \sum_{k=2}^{\lceil n/2\rceil -1} k \\
&\leq \log n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2\rceil -1} k \\
&\leq (\log n)\frac{1}{2}(n-1)n - \frac{1}{2}(\frac{n}{2}-1)\frac{n}{2} \\
&= \frac{1}{2}n^2\log n - \frac{n}{2}\log n - \frac{1}{8}n^2 + \frac{n}{4} \\
&\leq \frac{1}{2}n^2\log n - \frac{1}{8}n^2.
\end{aligned}$$

The last step holds since

$$\frac{n}{4} \leq \frac{n}{2}\log n,$$

when $n \geq 3$. □

Now we are ready for the final analysis.

417 Theorem Let $T(n)$ be the average case runtime of `Quicksort` on an array of size n. Then

$$T(n) = \Theta(n\log n).$$

Proof: We need to show that $T(n) = O(n\log n)$ and $T(n) = \Omega(n\log n)$. To prove that $T(n) = O(n\log n)$, we will show that for some constant a,

$$T(n) \leq an\log n \text{ for all } n \geq 2.^{12}$$

When $n = 2$,

$$an\log n = a2\log 2 = 2a,$$

[12] We pick 2 for the base case since $n\log n$=0 if $n = 1$, so we cannot make the inequality hold. Another solution would be to show that $T(n) \leq an\log n + b$. In this case, b can be chosen so that the inequality holds for $n = 1$.

and a can be chosen large enough so that $T(2) \leq 2a$. Thus, the inequality holds for the base case. Assume that $T(1) = C$, for some constant C. For $2 < k < n$, assume $T(k) \leq ak\log k$. Then

$$\begin{aligned}
T(n) &= \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n) \\
&\leq \frac{2}{n}\sum_{k=2}^{n-1} ak\log k + \frac{2}{n}T(1) + \Theta(n) && \text{(by assumption)} \\
&= \frac{2a}{n}\sum_{k=2}^{n-1} k\log k + \frac{2}{n}C + \Theta(n) \\
&\leq \frac{2a}{n}\sum_{k=2}^{n-1} k\log k + C + \Theta(n) && (\text{since } \tfrac{2}{n} \leq 1) \\
&\leq \frac{2a}{n}\left(\frac{1}{2}n^2\log n - \frac{1}{8}n^2\right) + C + \Theta(n) && \text{(by Lemma 2)} \\
&= an\log n - \frac{a}{4}n + C + \Theta(n) \\
&= an\log n + \left(\Theta(n) + C - \frac{a}{4}n\right) \\
&\leq an\log n && (\text{choose } a \text{ so } \Theta(n) + C \leq \tfrac{a}{4}n)
\end{aligned}$$

We have shown that with an appropriate choice of a, $T(n) \leq an\log n$ for all $n \geq 2$, so $T(n) = O(n\log n)$.
We leave it to the reader to show that $T(n) = \Omega(n\log n)$. □

Homework

418 Problem Assuming the priests can move one disk per second, that they started moving disks 6000 years ago, and that the legend of the Towers of Hanoi is true, when will the world end?

419 Problem Prove that for all positive integers n, $\sum_{i=1}^{n} i \cdot i! = (n+1)! - 1$.

420 Problem Prove that for all positive integers n, $f_1^2 + f_2^2 + \cdots + f_n^2 = f_n f_{n+1}$, where f_n is the nth Fibonacci number.

421 Problem Explain why the following joke never ends: *Pete and Repete got in a boat. Pete fell off. Who's left?*.

422 Problem Binary palindromes can be defined recursively by $\lambda, 0, 1 \in P$, and whenever $p \in P$, then $1p1 \in P$ and $0p0 \in P$. (Note: λ is the notation sometimes used to denote the *empty string*—that is, the string of length 0. Also, $1p1$ means the binary string obtained by appending 1 to the begin and end of string p. Similarly for $0p0$.) Notice that there is 1 palindrome of length 0 (λ), 2 of length 1 (0, 1), 2 of length 2 (00, 11), 4 of length 3 (000, 010, 101, 111), etc.

1. Prove that the number of binary palindromes of length $2k$ (even length) is 2^k for all $k \geq 0$. (Hint: Use induction and don't over think it).

2. Prove that the number of binary palindromes of length $2k+1$ (odd length) is 2^{k+1} for all $k \geq 0$.

423 Problem Prove that the recursive algorithm `factorial(n)` from Example 371 works correctly for $n \geq 0$.

424 Problem Prove that the `RPartition` algorithm from Section 6.4.1 has complexity $\Theta(n)$.

This page intentionally left blank.

Chapter 7

Counting

In this chapter we provide a very brief introduction to a field called *combinatorics*. It turns out that combinatorial problems are notoriously deceptive. Sometimes they can seem much harder than they are, and at other times they seem easier than they are. In fact, there are many cases in which one combinatorial problem will be relatively easy to solve, but a very closely related problem that seems almost identical will be very difficult to solve.

When solving combinatorial problems, you need to make sure you fully understand what is being asked and make sure you are taking everything into account appropriately. I used to tell students that combinatorics was easy. I don't say that anymore. In some sense it is easy. But it *is* also easy to make mistakes.

7.1 The Multiplication and Sum Rules

We begin our study of combinatorial methods with the following two fundamental principles.

425 Rule (Sum Rule: Disjunctive Form) Let $E_1, E_2, \ldots, E_k$, be pairwise finite disjoint sets. Then

$$|E_1 \cup E_2 \cup \cdots \cup E_k| = |E_1| + |E_2| + \cdots + |E_k|.$$

Another way of putting the sum rule is this: If you have to accomplish some task and you can do it in one of n_1 ways, or one of n_2 ways, etc., up to one of n_k ways, and none of the ways of doing the task on any of the list are the same, then there are $n_1 + n_2 + \cdots + n_k$ ways of doing the task.

426 Rule (Product Rule) Let $E_1, E_2, \ldots, E_k$, be finite sets. Then

$$|E_1 \times E_2 \times \cdots \times E_k| = |E_1| \cdot |E_2| \cdots |E_k|.$$

Another way of putting the product rule is this: If you need to accomplish some task that takes k steps, and there are n_1 ways of accomplishing the first step, n_2 ways of accomplishing the second step, etc., and n_k ways of accomplishing the kth step, then there are $n_1 n_2 \cdots n_k$ ways of accomplishing the task.

427 Example I have 5 brown shirts, 4 green shirts, 10 red shirts, and 3 blue shirts. How many choices do I have if I intend to wear one shirt?

> **Solution:** Since each list of shirts is independent of the others, I can use the sum rule. Therefore I can choose any of my $5 + 4 + 10 + 4 = 22$ shirts.

428 Example I have 5 pairs of socks, 10 pairs of shorts, and 8 t-shirts. How many choices do I have if I intend to wear one of each?

> **Solution:** I can think of choosing what to wear as a task broken into 3 steps: I have to choose a pair of socks (5 ways), a pair of shorts (10 ways), and finally a t-shirt (8 ways). Thus I have $5\times 10\times 8 = 400$ choices.

429 Example If license plates are required to have 3 letters followed by 3 digits, how many license plates are possible?

> **Solution:** There are 26 choices for each of the first three characters, and 10 choices for each of the final three characters. Therefore, there are $26^3\cdot 10^3$ possible license plates.

430 Example What is the value of *sum* after each of the following segments of code?

```
int sum=0;
for(int i=0;i<n;i++) {
    for(int i=0;i<m;i++) {
        sum = sum + 1;
    }
}
```

```
int sum=0;
for(int i=0;i<n;i++) {
    sum = sum + 1;
}
for(int i=0;i<m;i++) {
    sum = sum + 1;
}
```

> **Solution:** In the code on the left, the inner loop executes m times, so every time the inner loop executes, *sum* gets m added to it. The outer loop executes n times, each time calling the inner loop. Therefore m is added to *sum* n times, so $sum = n\times m$ at the end.
>
> In the code on the right, The first loop adds n to *sum*, and then the second loop adds m to *sum*. Therefore, $sum = n+m$ at the end.

431 Example How many ordered pairs of integers (x,y) are there such that $0<|xy|\leq 5$?

> **Solution:** Let $E_k = \{(x,y)\in\mathbb{Z}^2 : |xy| = k\}$ for $k = 1,\ldots,5$. Then the desired number is
>
> $$|E_1|+|E_2|+\cdots+|E_{5|}.$$
>
> Then
>
> $$\begin{array}{rcl} E_1 & = & \{(-1,-1),(-1,1),(1,-1),(1,1)\} \\ E_2 & = & \{(-2,-1),(-2,1),(-1,-2),(-1,2),(1,-2),(1,2),(2,-1),(2,1)\} \\ E_3 & = & \{(-3,-1),(-3,1),(-1,-3),(-1,3),(1,-3),(1,3),(3,-1),(3,1)\} \\ E_4 & = & \{(-4,-1),(-4,1),(-2,-2),(-2,2),(-1,-4),(-1,4),(1,-4), \\ & & (1,4),(2,-2),(2,2),(4,-1),(4,1)\} \\ E_5 & = & \{(-5,-1),(-5,1),(-1,-5),(-1,5),(1,-5),(1,5),(5,-1),(5,1)\} \end{array}$$
>
> The desired number is therefore $4+8+8+12+8 = 40$.

432 Example The positive divisors of 400 are written in increasing order

$$1,2,4,5,8,\ldots,200,400.$$

How many integers are there in this sequence. How many of the divisors of 400 are perfect squares?

Solution: Since $400 = 2^4 \cdot 5^2$, any positive divisor of 400 has the form $2^a 5^b$ where $0 \leq a \leq 4$ and $0 \leq b \leq 2$. Thus there are 5 choices for a and 3 choices for b for a total of $5 \cdot 3 = 15$ positive divisors.

To be a perfect square, a positive divisor of 400 must be of the form $2^\alpha 5^\beta$ with $\alpha \in \{0, 2, 4\}$ and $\beta \in \{0, 2\}$. Thus there are $3 \cdot 2 = 6$ divisors of 400 which are also perfect squares.

It is easy to generalize Example 432 to obtain the following theorem. following theorem.

433 Theorem Let the positive integer n have the prime factorization

$$n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k},$$

where the p_i are distinct primes, and the a_i are integers ≥ 1. If $d(n)$ denotes the number of positive divisors of n, then

$$d(n) = (a_1 + 1)(a_2 + 1) \cdots (a_k + 1).$$

434 Example The integers from 1 to 1000 are written in succession. Find the sum of all the digits.

Solution: When writing the integers from 000 to 999 (with three digits), $3 \times 1000 = 3000$ digits are used. Each of the 10 digits is used an equal number of times, so each digit is used 300 times. The the sum of the digits in the interval 000 to 999 is thus

$$(0+1+2+3+4+5+6+7+8+9) \cdot 300 = 13500.$$

Therefore, the sum of the digits when writing the integers from 1 to 1000 is $13500 + 1 = 13501$.

Aliter: Pair up the integers from 0 to 999 as

$$(0, 999),\ (1, 998),\ (2, 997),\ (3, 996),\ \ldots, (499, 500).$$

Each pair has sum of digits 27 and there are 500 such pairs. Adding 1 for the sum of digits of 1000, the required total is

$$27 \cdot 500 + 1 = 13501.$$

435 Example The strictly positive integers are written in succession

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, \ldots$$

Which digit occupies the 3000-th position?

Solution: Upon using

$9 \cdot 1 = 9$	1-digit integers,
$90 \cdot 2 = 180$	2-digit integers,
$900 \cdot 3 = 2700$	3-digit integers,

a total of $9 + 180 + 2700 = 2889$ digits have been used, so the 3000-th digit must belong to a 4-digit integer. There remains to use $3000 - 2889 = 111$ digits, and $111 = 4 \cdot 27 + 3$, so the 3000-th digit is the third digit of the 28-th 4-digit integer, that is, the third digit of 4027, namely 2.

7.2 Pigeonhole Principle

The following theorem seems so obvious that it doesn't need to be stated. However, it often come in handy in unexpected situations.

436 Theorem (The Pigeonhole Principle) If n is a positive integer and $n+1$ or more objects are placed into n boxes, then one of the boxes contains at least two objects.

437 Example In any group of 13 people, there are always two who have their birthday on the same month. Similarly, if there are 32 people, at least two people were born on the same day of the month.

The pigeonhole principle can be generalized.

438 Theorem (The Generalized Pigeonhole Principle) If n objects are placed into k boxes, then there is at least one box that contains at least $\lceil n/k \rceil$ objects.

Proof: Assume not. Then each of the k boxes contains no more than $\lceil n/k \rceil - 1$ objects. Notice that $\lceil n/k \rceil < n/k+1$ (convince yourself that this is always true). Thus, the total number of objects in the k boxes is at most

$$k(\lceil n/k \rceil - 1) < k(n/k+1-1) = n,$$

contradicting the fact that there are n objects in the boxes. Therefore, some box contains at least $\lceil n/k \rceil$ objects. □

The Pigeonhole Principle is useful in proving *existence* problems, that is, we show that something exists without actually identifying it concretely.

439 Example Show that amongst any seven distinct positive integers not exceeding 126, one can find two of them, say a and b, which satisfy

$$b < a \leq 2b.$$

Solution: Split the numbers $\{1,2,3,\ldots,126\}$ into the six sets

$$\{1,2\},\{3,4,5,6\},\{7,8,\ldots,13,14\},\{15,16,\ldots,29,30\},$$

$$\{31,32,\ldots,61,62\} \text{ and } \{63,64,\ldots,126\}.$$

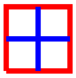
By the Pigeonhole Principle, two of the seven numbers must lie in one of the six sets, and obviously, any such two will satisfy the stated inequality.

440 Example Given any 9 integers whose prime factors lie in the set $\{3,7,11\}$ prove that there must be two whose product is a square.

Solution: For an integer to be a square, all the exponents of its prime factorisation must be even. Any integer in the given set has a prime factorisation of the form $3^a7^b11^c$. Now each triplet (a,b,c) has one of the following 8 parity patterns: (even, even, even), (even, even, odd), (even, odd, even), (even, odd, odd), (odd, even, even), (odd, even, odd), (odd, odd, even), (odd, odd, odd). In a group of 9 such integers, there must be two with the same parity patterns in the exponents. Take these two. Their product is a square, since the sum of each corresponding exponent will be even.

441 Example Prove that if five points are taken on or inside a unit square, there must always be two whose distance is $\leq \dfrac{\sqrt{2}}{2}$.

Solution: Split the square into four congruent squares as shown to the right. Two of the points must fall into one of the smaller squares, and the longest distance there is, by the Pythagorean Theorem, $\sqrt{(\frac{1}{2})^2 + (\frac{1}{2})^2} = \dfrac{\sqrt{2}}{2}$.



442 Example Given any set of ten natural numbers between 1 and 99 inclusive, prove that there are two disjoint nonempty subsets of the set with equal sums of their elements.

Solution: There are $2^{10} - 1 = 1023$ non-empty subsets that one can form with a given 10-element set. To each of these subsets we associate the sum of its elements. The maximum value that any such sum can achieve is $90 + 91 + \cdots + 99 = 945 < 1023$. Therefore, there must be at least two different subsets that have the same sum.

443 Example Prove that if 55 of the integers from 1 to 100 are selected, then two of them differ by 10.

Solution: First observe that if we choose $n+1$ integers from any set of $2n$ consecutive integers, there will always be some two that differ by n. This is because we can pair the $2n$ consecutive integers

$$\{a+1, a+2, a+3, \ldots, a+2n\}$$

into the n pairs

$$\{a+1, a+n+1\}, \{a+2, a+n+2\}, \ldots, \{a+n, a+2n\},$$

and if $n+1$ integers are chosen from this, there must be two that belong to the same group.

So now group the one hundred integers as follows:

$$\{1, 2, \ldots 20\}, \{21, 22, \ldots, 40\},$$

$$\{41, 42, \ldots, 60\},\ \{61, 62, \ldots, 80\}$$

and

$$\{81, 82, \ldots, 100\}.$$

If we select fifty five integers, then we must have selected at least $\lceil 55/5 \rceil = 11$ from one of the groups. From that group, by the above observation (let $n = 10$), there must be two that differ by 10.

444 Example Label one disc "**1**", two discs "**2**", three discs "**3**", ..., fifty discs "**50**". Put these $1 + 2 + 3 + \cdots + 50 = 1275$ labeled discs in a box. Discs are then drawn from the box at random without replacement. What is the minimum number of discs that must me drawn in order to guarantee drawing at least ten discs with the same label?

Solution: If we draw all the $1 + 2 + \cdots + 9 = 45$ labelled "**1**", ..., "**9**" and any nine from each of the discs "**10**", ..., "**50**", we have drawn $45 + 9 \cdot 41 = 414$ discs. The 415-th disc drawn will assure at least ten discs from a label.

7.3 Permutations and Combinations

Most counting problems we will be dealing with can be classified into one of four categories. We explain such categories by means of an example.

445 Example Consider the set $\{a,b,c,d\}$. Suppose we "select" two letters from these four. Depending on our interpretation, we may obtain the following answers.

❶ **Permutations with repetitions.** The *order* of listing the letters is important, and *repetition is* allowed. In this case there are $4 \cdot 4 = 16$ possible selections:

aa	*ab*	*ac*	*ad*
ba	*bb*	*bc*	*bd*
ca	*cb*	*cc*	*cd*
da	*db*	*dc*	*dd*

❷ **Permutations without repetitions.** The *order* of listing the letters is important, and *repetition is not* allowed. In this case there are $4 \cdot 3 = 12$ possible selections:

	ab	*ac*	*ad*
ba		*bc*	*bd*
ca	*cb*		*cd*
da	*db*	*dc*	

❸ **Combinations with repetitions.** The *order* of listing the letters is **not** important, and *repetition is* allowed. In this case there are $\frac{4 \cdot 3}{2} + 4 = 10$ possible selections:

aa	*ab*	*ac*	*ad*
	bb	*bc*	*bd*
		cc	*cd*
			dd

❹ **Combinations without repetitions.** The *order* of listing the letters is **not** important, and *repetition is not* allowed. In this case there are $\frac{4 \cdot 3}{2} = 6$ possible selections:

	ab	*ac*	*ad*
		bc	*bd*
			cd

Although most of the simple types of counting problems we want to solve can be reduced to one of these four, care must be taken. The previous example assumed that we had a set of *distinguishable* objects. When objects are not distinguishable, the situation is a more complicated.

The next four sections provide more details and examples of each of the four interpretations from the previous example.

7.3.1 Permutations without Repetitions

446 Definition Let $x_1, x_2, \ldots, x_n$ be n distinct objects. A *permutation* of these objects is simply a rearrangement of them.

447 Example There are 24 permutations of the letters in $MATH$, namely

$$\begin{array}{cccccc}
MATH & MAHT & MTAH & MTHA & MHTA & MHAT \\
AMTH & AMHT & ATMH & ATHM & AHTM & AHMT \\
TAMH & TAHM & TMAH & TMHA & THMA & THAM \\
HATM & HAMT & HTAM & HTMA & HMTA & HMAT
\end{array}$$

448 Theorem Let $x_1, x_2, \ldots, x_n$ be n distinct objects. Then there are $n!$ permutations of them.

Proof: The first position can be chosen in n ways, the second object in $n-1$ ways, the third in $n-2$, etc. This gives

$$n(n-1)(n-2)\cdots 2\cdot 1 = n!.$$

□

449 Example A bookshelf contains 5 German books, 7 Spanish books and 8 French books. Each book is different from one another. How many different arrangements can be done of these books if

❶ we put no restrictions on how they can be arranged?

❷ books of each language must be next to each other?

❸ all the French books must be next to each other?

❹ no two French books must be next to each other?

Solution:

❶ We are permuting $5+7+8=20$ objects. Thus the number of arrangements sought is $20! = 2432902008176640000$.

❷ "Glue" the books by language, this will assure that books of the same language are together. We permute the 3 languages in 3! ways. We permute the German books in 5! ways, the Spanish books in 7! ways and the French books in 8! ways. Hence the total number of ways is $3!5!7!8! = 146313216000$.

❸ Align the German books and the Spanish books first. Putting these $5+7=12$ books creates $12+1=13$ spaces (we count the space before the first book, the spaces between books and the space after the last book). To assure that all the French books are next each other, we "glue" them together and put them in one of these spaces. Now, the French books can be permuted in 8! ways and the non-French books can be permuted in 12! ways. Thus the total number of permutations is

$$(13)8!12! = 251073478656000.$$

❹ Align the German books and the Spanish books first. Putting these $5+7=12$ books creates $12+1=13$ spaces (we count the space before the first book, the spaces between books and the space after the last book). To assure that no two French books are next to each other, we put them into these spaces. The first French book can be put into any of 13 spaces, the second into any of 12 remaining spaces, etc., and the eighth French book can be put into any 6 remaining spaces. Now, the non-French books can be permuted in 12! ways. Thus the total number of permutations is

$$(13)(12)(11)(10)(9)(8)(7)(6)12!,$$

which is 24856274386944000.

7.3.2 Permutations with Repetitions

We now consider permutations with repeated objects.

450 Example In how many ways may the letters of the word

$$MASSACHUSETTS$$

be permuted to form different strings?

Solution: We put subscripts on the repeats forming

$$MA_1S_1S_2A_2CHUS_3ET_1T_2S_4.$$

There are now 13 distinguishable objects, which can be permuted in 13! different ways by Theorem 448. But this counts some arrangements multiple times since in reality the duplicated letters are not distinguishable. Consider a single permutation of all of the distinguishable letters. If I permute the letters A_1A_2, I get the same permutation when ignoring the subscripts. The same thing is true of T_1T_2. Similarly, there are 4! permutations of $S_1S_2S_3S_4$, so there are 4! permutations that look the same (without the subscripts). Since I can do all of these independently, there are 2!2!4! permutations that look identical when the subscripts are removed. This is true of every permutation. Therefore, the actual number of permutations is

$$\frac{13!}{2!4!2!} = 64864800.$$

If you do not follow this example, I highly recommend trying this yourself by determining the number of permutations of a few smaller words with fewer repeats—for instance, try *TALL*, *SELLS* and *AEEEI*[1]. Using reasoning analogous to the one of example 450, we may prove the following theorem.

451 Theorem Let there be k types of objects: n_1 of type 1; n_2 of type 2; etc. Then the number of ways in which these $n_1+n_2+\cdots+n_k$ objects can be rearranged is

$$\frac{(n_1+n_2+\cdots+n_k)!}{n_1!n_2!\cdots n_k!}.$$

[1] O.K., I admit that this isn't a word. I couldn't come up with a 5 or six letter word with three repeats

452 Example In how many ways may we permute the letters of the word $MASSACHUSETTS$ in such a way that $MASS$ is always together, in this order?

> **Solution:** The particle $MASS$ can be considered as one block along with the remaining 9 letters A, C, H, U, S, E, T, T, S. There are two S's[2] and two T's and so the total number of permutations sought is
> $$\frac{10!}{2!2!} = 907200.$$

453 Example In how many ways may we write the number 9 as the sum of three positive integer summands? Here order counts, so, for example, $1+7+1$ is to be regarded different from $7+1+1$.

> **Solution:** We first look for answers with
> $$a+b+c=9, 1 \leq a \leq b \leq c \leq 7$$
> and we find the permutations of each triplet. We have

(a,b,c)	Number of permutations
$(1,1,7)$	$\frac{3!}{2!} = 3$
$(1,2,6)$	$3! = 6$
$(1,3,5)$	$3! = 6$
$(1,4,4)$	$\frac{3!}{2!} = 3$
$(2,2,5)$	$\frac{3!}{2!} = 3$
$(2,3,4)$	$3! = 6$
$(3,3,3)$	$\frac{3!}{3!} = 1$

> Thus the number desired is
> $$3+6+6+3+3+6+1 = 28.$$

454 Example In how many ways can the letters of the word **MURMUR** be arranged without letting two letters which are alike come together?

> **Solution:** If we started with, say , **MU** then the **R** could be arranged as follows:
> $$\begin{array}{|c|c|c|c|c|c|}\hline \mathbf{M} & \mathbf{U} & \mathbf{R} & & \mathbf{R} & \\ \hline\end{array}\,,$$
> $$\begin{array}{|c|c|c|c|c|c|}\hline \mathbf{M} & \mathbf{U} & \mathbf{R} & & & \mathbf{R} \\ \hline\end{array}\,,\text{or}$$
> $$\begin{array}{|c|c|c|c|c|c|}\hline \mathbf{M} & \mathbf{U} & & \mathbf{R} & & \mathbf{R} \\ \hline\end{array}\,.$$
> In the first case there are $2! = 2$ ways of putting the remaining **M** and **U**, in the second there are $2! = 2$ ways and in the third there is only $1!$ way. Thus starting the word with **MU** gives $2+2+1 = 5$ possible arrangements. In the general case, we can choose the first letter of the word in 3 ways, and the second in 2 ways. Thus the number of ways sought is $3 \cdot 2 \cdot 5 = 30$.[3]

[2] Remember, the other two S's are part of $MASS$, which we are now treating as a single object.

[3] It should be noted that this analysis worked because the three letters each occurred twice. If this was not the case we would have had to work harder to solve the problem.

455 Example In how many ways can the letters of the word **AFFECTION** be arranged, keeping the vowels in their natural order and not letting the two **F**'s come together?

> **Solution:** There are $\dfrac{9!}{2!}$ ways of permuting the letters of **AFFECTION**. The 4 vowels can be permuted in 4! ways, and in only one of these will they be in their natural order. Thus there are $\dfrac{9!}{2!4!}$ ways of permuting the letters of **AFFECTION** in which their vowels keep their natural order. If we treat FF as a single letter, there are 8! ways of permuting the letters so that the F's stay together. Hence there are $\dfrac{8!}{4!}$ permutations of **AFFECTION** where the vowels occur in their natural order and the FF's are together. In conclusion, the number of permutations sought is
>
> $$\frac{9!}{2!4!} - \frac{8!}{4!} = \frac{8!}{4!}\left(\frac{9}{2} - 1\right) = \frac{8\cdot 7\cdot 6\cdot 5\cdot 4!}{4!}\cdot\frac{7}{2} = 5880$$

7.3.3 Combinations without Repetitions

456 Definition Let n, k be non-negative integers with $0 \leq k \leq n$. The *binomial coefficient* $\dbinom{n}{k}$ (read "*n choose k*") is defined and denoted by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n\cdot(n-1)\cdot(n-2)\cdots(n-k+1)}{1\cdot 2\cdot 3\cdots k}.$$

☞ *Observe that in the last fraction, there are k factors in both the numerator and denominator. Also, observe the boundary conditions*

$$\binom{n}{0} = \binom{n}{n} = 1, \quad \binom{n}{1} = \binom{n}{n-1} = n.$$

457 Example We have

$$\begin{aligned}
\binom{6}{3} &= \frac{6\cdot 5\cdot 4}{1\cdot 2\cdot 3} = 20,\\
\binom{11}{2} &= \frac{11\cdot 10}{1\cdot 2} = 55,\\
\binom{12}{7} &= \frac{12\cdot 11\cdot 10\cdot 9\cdot 8\cdot 7\cdot 6}{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6\cdot 7} = 792,\\
\binom{110}{109} &= 110,\\
\binom{110}{0} &= 1.
\end{aligned}$$

If there are n kittens and you decide to take k of them home, you also decided *not* to take $n-k$ of them home. This idea leads to the following important theorem.

458 Theorem If $n,k \in \mathbb{Z}$, with $0 \leq k \leq n$, then

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!(n-(n-k))!} = \binom{n}{n-k}$$

Proof: Since $k = n-(n-k)$, the result is obvious. □

459 Example

$$\binom{11}{9} = \binom{11}{2} = 55,$$
$$\binom{12}{5} = \binom{12}{7} = 792.$$

460 Definition Let there be n distinguishable objects. A *k-combination* is a selection of k, $(0 \leq k \leq n)$ objects from the n made without regards to order.

461 Example The 2-combinations from the list $\{X,Y,Z,W\}$ are

$$XY, XZ, XW, YZ, YW, WZ.$$

462 Example The 3-combinations from the list $\{X,Y,Z,W\}$ are

$$XYZ, XYW, XZW, YWZ.$$

463 Theorem Let there be n distinguishable objects, and let k, $0 \leq k \leq n$. Then the numbers of k-combinations of these n objects is $\binom{n}{k}$.

Proof: Pick any of the k objects. They can be ordered in $n(n-1)(n-2)\cdots(n-k+1)$, since there are n ways of choosing the *first*, $n-1$ ways of choosing the *second*, etc. This particular choice of k objects can be permuted in $k!$ ways. Hence the total number of k-combinations is

$$\frac{n(n-1)(n-2)\cdots(n-k+1)}{k!} = \binom{n}{k}.$$

□

464 Example From a group of 10 people, we may choose a committee of 4 in $\binom{10}{4} = 210$ ways.

465 Example Three different integers are drawn from the set $\{1,2,\ldots,20\}$. In how many ways may they be drawn so that their sum is divisible by 3?

Solution: In $\{1,2,\ldots,20\}$ there are

6	numbers leaving remainder 0
7	numbers leaving remainder 1
7	numbers leaving remainder 2

The sum of three numbers will be divisible by 3 when (a) the three numbers are divisible by 3; (b) one of the numbers is divisible by 3, one leaves remainder 1 and the third leaves remainder 2 upon division by 3; (c) all three leave remainder 1 upon division by 3; (d) all three leave remainder 2 upon division by 3. Hence the number of ways is

$$\binom{6}{3}+\binom{6}{1}\binom{7}{1}\binom{7}{1}+\binom{7}{3}+\binom{7}{3}=384.$$

466 Example To count the number of shortest routes from A to B in Figure 7.1, observe that any shortest path must consist of 6 horizontal moves and 3 vertical ones for a total of $6+3=9$ moves. Once we choose which 6 of these 9 moves are horizontal the 3 vertical ones are determined. For instance, if I choose to go horizontal on moves 1, 2, 4, 6, 7, and 8, then moves 3, 5 and 9 must be vertical. Thus there are $\binom{9}{6}=84$ paths.

Another way to think about it is that we need to compute the number of permutations of $EEEEEENNN$, where E means move east, and N means move north. The number of permutations is $9!/(6!3!)=\binom{9}{6}$.

467 Example To count the number of shortest routes from A to B in Figure 7.2 that pass through point O we count the number of paths from A to O (of which there are $\binom{5}{3}=20$) and the number of paths from O to B (of which there are $\binom{4}{3}=4$). Thus the desired number of paths is $\binom{5}{3}\binom{4}{3}=(20)(4)=80$.

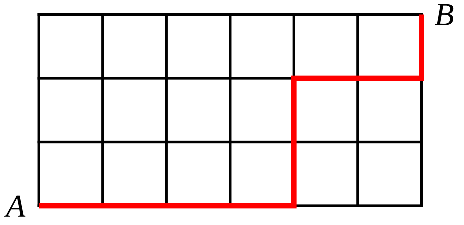


Figure 7.1: Example 466.

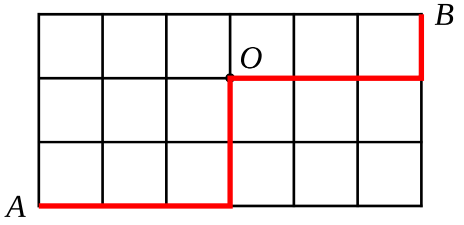


Figure 7.2: Example 467.

7.3.4 Combinations with Repetitions

468 Theorem (De Moivre) Let n be a positive integer. The number of positive integer solutions to

$$x_1+x_2+\cdots+x_r=n$$

is

$$\binom{n-1}{r-1}.$$

Proof: Write n as

$$n=1+1+\cdots+1+1,$$

where there are n 1s and $n-1$ +s. To decompose n in r summands we only need to choose $r-1$ pluses from the $n-1$, For instance, writing $n=7$ as $7=2+3+2$ is equivalent to $7=(1+1)+(1+1+1)+(1+1)$, where the +'s outside of the parentheses are the ones we chose. This proves the theorem. □

469 Example In how many ways may we write the number 9 as the sum of three positive integer summands? Here order counts, so, for example, $1+7+1$ is to be regarded different from $7+1+1$.

Solution: Notice that this is the same problem as Example 453. We are seeking integral solutions to

$$a+b+c=9, \quad a>0, b>0, c>0.$$

By Theorem 468 this is

$$\binom{9-1}{3-1}=\binom{8}{2}=28.$$

☞ *The solution in Example 469 was much easier than the solution in Example 453, demonstrating the fact that choosing the right tool for the job can make a huge difference. Sometimes recognizing the best tool for the job can be tricky. Of course, the more problems of this type you solve, the easier it gets. Similarly, having more tools in your bag gives you more options.*

This also demonstrates something that is true of a lot of combinatorial problems: There are often several valid ways of approaching them. But there are also a lot of invalid approaches, so be careful!

470 Example In how many ways can 100 be written as the sum of four positive integer summands?

Solution: We want the number of positive integer solutions to

$$a+b+c+d=100,$$

which by Theorem 468 is

$$\binom{99}{3}=156849.$$

The following corollary is similar to Theorem 468 except that the numbers are allowed to be 0.

471 Corollary Let n be a positive integer. The number of non-negative integer solutions to

$$y_1+y_2+\cdots+y_r=n$$

is

$$\binom{n+r-1}{r-1}.$$

Proof: Set $x_i-1=y_i$ for $i=1,\ldots,r$. Then $x_i\geq 1$, and equation

$$x_1-1+x_2-1+\cdots+x_r-1=n$$

is equivalent to

$$x_1+x_2+\cdots+x_r=n+r,$$

which from Theorem 468, has

$$\binom{n+r-1}{r-1}$$

solutions. □

472 Example Find the number of quadruples (a,b,c,d) of integers satisfying

$$a+b+c+d=100, \; a\geq 30, b>21, c\geq 1, d\geq 1.$$

Solution: Put $a'+29=a, b'+20=b$. Then we want the number of positive integer solutions to

$$a'+29+b'+21+c+d=100,$$

or

$$a'+b'+c+d=50.$$

By Theorem 468 this number is

$$\binom{49}{3}=18424.$$

473 Example In how many ways may 1024 be written as the product of three positive integers?

Solution: Observe that $1024=2^{10}$. We need a decomposition of the form $2^{10}=2^a2^b2^c$, that is, we need integers solutions to

$$a+b+c=10, \quad a\geq 0, b\geq 0, c\geq 0.$$

By Corollary 471 there are $\binom{10+3-1}{3-1}=\binom{12}{2}=66$ such solutions.

7.4 Binomial Theorem

It is well known that

$$(a+b)^2=a^2+2ab+b^2 \tag{7.1}$$

Multiplying this last equality by $a+b$ one obtains

$$(a+b)^3=(a+b)^2(a+b)=a^3+3a^2b+3ab^2+b^3$$

Again, multiplying

$$(a+b)^3=a^3+3a^2b+3ab^2+b^3 \tag{7.2}$$

by $a+b$ one obtains

$$(a+b)^4=(a+b)^3(a+b)=a^4+4a^3b+6a^2b^2+4ab^3+b^4$$

This generalizes, as we see in the next theorem.

474 Theorem (Binomial Theorem) Let x and y be variables and n be a nonnegative integer. Then

$$(x+y)^n=\sum_{i=0}^{n}\binom{n}{i}x^{n-i}y^i.$$

475 Example

$$\begin{aligned}(4x+5)^3 &= \binom{3}{0}(4x)^35^0+\binom{3}{1}(4x)^2(5)^1+\binom{3}{2}(4x)^1(5)^2+\binom{3}{3}(4x)^05^3\\ &= (4x)^3+3(4x)^2(5)+3(4x)(5)^2+5^3\\ &= 64x^3+240x^2+300x+125\end{aligned}$$

476 Example

$$\begin{aligned}(2x-y^2)^4 &= \binom{4}{0}(2x)^4+\binom{4}{1}(2x)^3(-y^2)+\binom{4}{2}(2x)^2(-y^2)^2+\binom{4}{3}(2x)(-y^2)^3+\binom{4}{4}(-y^2)^4\\ &= (2x)^4+4(2x)^3(-y^2)+6(2x)^2(-y^2)^2+4(2x)(-y^2)^3+(-y^2)^4\\ &= 16x^4-32x^3y^2+24x^2y^4-8xy^6+y^8\end{aligned}$$

The most important things to remember when using the binomial theorem are not to forget the binomial coefficients, and not to forget that the powers (i.e. x^{n-i} and y^i) apply to the whole term, including any coefficients. A specific case that is easy to forget is a negative sign on the coefficient. We skip a few steps (e.g. the step of explicitly writing out the binomial coefficient) in the next few examples.

477 Example

$$\begin{aligned}(2+i)^5 &= 2^5+5(2)^4(i)+10(2)^3(i)^2+10(2)^2(i)^3+5(2)(i)^4+i^5\\ &= 32+80i-80-40i+10+i\\ &= -38+39i\end{aligned}$$

478 Example

$$\begin{aligned}(\sqrt{3}+\sqrt{5})^4 &= (\sqrt{3})^4+4(\sqrt{3})^3(\sqrt{5})+6(\sqrt{3})^2(\sqrt{5})^2+4(\sqrt{3})(\sqrt{5})^3+(\sqrt{5})^4\\ &= 9+12\sqrt{15}+90+20\sqrt{15}+25\\ &= 124+32\sqrt{15}\end{aligned}$$

479 Example Given that $a-b=2, ab=3$ find a^3-b^3.

Solution: One has

$$\begin{aligned}8 &= 2^3\\ &= (a-b)^3\\ &= a^3-3a^2b+3ab^2-b^3\\ &= a^3-b^3-3ab(a-b)\\ &= a^3-b^3-18,\end{aligned}$$

whence $a^3-b^3=26$.

If we ignore the variables in the Binomial Theorem and write down the coefficients for increasing values of n, a pattern, called *Pascal's Triangle* emerges.

Notice that each entry different from 1 is the sum of the two neighbors just above it. This leads to the following theorem

480 Theorem (Pascal's Identity) Let n and k be positive integers with $k \le n$. Then

$$\binom{n+1}{k}=\binom{n}{k-1}+\binom{n}{k}.$$

```
                                1
                             1     1
                          1     2     1
                       1     3     3     1
                    1     4     6     4     1
                 1     5    10    10     5     1
              1     6    15    20    15     6     1
           1     7    21    35    35    21     7     1
        1     8    28    56    70    56    28     8     1
     1     9    36    84   126   126    84    36     9     1
  1    10    45   120   210   252   210   120    45    10     1
                                ⋮
```

Figure 7.3: Pascal's Triangle

7.5 Inclusion-Exclusion

The Sum Rule 425 gives us the cardinality for unions of finite sets that are mutually disjoint. In this section we will drop the disjointness requirement and obtain a formula for the cardinality of unions of general finite sets.

The Principle of *Inclusion-Exclusion* is attributed to both Sylvester and to Poincaré. We will only consider the cases involving two and three sets, although the principle easily generalizes to k sets.

481 Theorem (Inclusion-Exclusion for Two Sets)

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Proof: Clearly there are $|A \cap B|$ elements that are in both A and B. Therefore, $|A| + |B|$ is the number of element in A and B, where the elements in $|A \cap B|$ are counted twice. From this it is clear that $|A \cup B| = |A| + |B| - |A \cap B|$. □

482 Example Of 40 people, 28 smoke and 16 chew tobacco. It is also known that 10 both smoke and chew. How many among the 40 neither smoke nor chew?

Solution: Let A denote the set of smokers and B the set of chewers. Then

$$|A \cup B| = |A| + |B| - |A \cap B| = 28 + 16 - 10 = 34,$$

meaning that there are 34 people that either smoke or chew (or possibly both). Therefore the number of people that neither smoke nor chew is $40 - 34 = 6$.

483 Example Consider the set A that are multiples of 2 no greater than 114. That is,

$$A = \{2, 4, 6, \ldots, 114\}.$$

❶ How many elements are there in A?

❷ How many are divisible by 3?

❸ How many are divisible by 5?

❹ How many are divisible by 15?

❺ How many are divisible by either 3, 5 or both?

❻ How many are neither divisible by 3 nor 5?

❼ How many are divisible by exactly one of 3 or 5?

Solution: Let $A_k \subset A$ be the set of those integers divisible by k.

❶ Notice that the elements are $2 = 2(1)$, $4 = 2(2)$, $\ldots$, $114 = 2(57)$. Thus $|A| = 57$.

❷ Notice that
$$A_3 = \{6, 12, 18, \ldots, 114\} = \{1 \cdot 6, 2 \cdot 6, 3 \cdot 6, \ldots, 19 \cdot 6\},$$
so $|A_3| = 19$.

❸ Notice that
$$A_5 = \{10, 20, 30, \ldots, 110\} = \{1 \cdot 10, 2 \cdot 10, 3 \cdot 10, \ldots, 11 \cdot 10\},$$
so $|A_5| = 11$.

❹ Notice that $A_{15} = \{30, 60, 90\}$, so $|A_{15}| = 3$.

❺ First notice that $A_3 \cap A_5 = A_{15}$. Then it is clear that the answer is $|A_3 \cup A_5| = |A_3| + |A_5| = |A_{15}| = 19 + 11 - 3 = 27$.

❻ We want
$$|A \setminus (A_3 \cup A_5)| = |A| - |A_3 \cup A_5| = 57 - 27 = 30.$$

❼ We want
$$\begin{array}{rcl} |(A_3 \cup A_5) \setminus (A_3 \cap A_5)| & = & |(A_3 \cup A_5)| - |A_3 \cap A_5| \\ & = & 27 - 3 \\ & = & 24. \end{array}$$

484 Example How many integers between 1 and 1000 inclusive, do not share a common factor with 1000, that is, are relatively prime to 1000?

Solution: Observe that $1000 = 2^3 5^3$, and thus from the 1000 integers we must weed out those that have a factor of 2 or of 5 in their prime factorization. If A_2 denotes the set of those integers divisible by 2 in the interval $[1, 1000]$ then clearly $|A_2| = \left\lfloor \frac{1000}{2} \right\rfloor = 500$. Similarly, if A_5 denotes the set of those integers divisible by 5 then $|A_5| = \left\lfloor \frac{1000}{5} \right\rfloor = 200$. Also $|A_2 \cap A_5| = \left\lfloor \frac{1000}{10} \right\rfloor = 100$. This means that there are $|A_2 \cup A_5| = 500 + 200 - 100 = 600$ integers in the interval $[1, 1000]$ sharing at least a factor with 1000, thus there are $1000 - 600 = 400$ integers in $[1, 1000]$ that do not share a factor prime factor with 1000.

We now derive a three-set version of the Principle of Inclusion-Exclusion.

485 Theorem (Inclusion-Exclusion for Three Sets)

$$\begin{aligned} |A\cup B\cup C| &= |A|+|B|+|C| \\ &\quad -|A\cap B|-|B\cap C|-|C\cap A| \\ &\quad +|A\cap B\cap C| \end{aligned}$$

Proof: Using the associativity and distributivity of unions of sets, we see that

$$\begin{aligned} |A\cup B\cup C| &= |A\cup(B\cup C)| \\ &= |A|+|B\cup C|-|A\cap(B\cup C)| \\ &= |A|+|B\cup C|-|(A\cap B)\cup(A\cap C)| \\ &= |A|+|B|+|C|-|B\cap C|-|A\cap B|-|A\cap C|+|(A\cap B)\cap(A\cap C)| \\ &= |A|+|B|+|C|-|B\cap C|-(|A\cap B|+|A\cap C|-|A\cap B\cap C|) \\ &= |A|+|B|+|C|-|A\cap B|-|B\cap C|-|C\cap A|+|A\cap B\cap C|. \end{aligned}$$

□

486 Example How many integers between 1 and 600 inclusive are not divisible by 3, nor 5, nor 7?

Solution: Let A_k denote the numbers in $[1,600]$ which are divisible by $k=3,5,7$. Then

$$\begin{aligned} |A_3| &= \lfloor\tfrac{600}{3}\rfloor = 200, \\ |A_5| &= \lfloor\tfrac{600}{5}\rfloor = 120, \\ |A_7| &= \lfloor\tfrac{600}{7}\rfloor = 85, \\ |A_{15|} &= \lfloor\tfrac{600}{15}\rfloor = 40 \\ |A_{21|} &= \lfloor\tfrac{600}{21}\rfloor = 28 \\ |A_{35|} &= \lfloor\tfrac{600}{35}\rfloor = 17 \\ |A_{105|} &= \lfloor\tfrac{600}{105}\rfloor = 5 \end{aligned}$$

By Inclusion-Exclusion there are $200+120+85-40-28-17+5=325$ integers in $[1,600]$ divisible by at least one of 3, 5, or 7. Those not divisible by these numbers are a total of $600-325=275$.

487 Example

How many integral solutions to the equation

$$a+b+c+d=100,$$

are there given the following constraints:

$$1\le a\le 10,\ b\ge 0,\ c\ge 2, 20\le d\le 30?$$

Solution: We use Inclusion-Exclusion. There are $\binom{80}{3} = 82160$ integral solutions to

$$a+b+c+d = 100, \quad a \geq 1, b \geq 0, c \geq 2, d \geq 20.$$

Let A be the set of solutions with

$$a \geq 11, b \geq 0, c \geq 2, d \geq 20$$

and B be the set of solutions with

$$a \geq 1, b \geq 0, c \geq 2, d \geq 31.$$

Then $|A| = \binom{70}{3}$, $|B| = \binom{69}{3}$, $|A \cap B| = \binom{59}{3}$ and so

$$|A \cup B| = \binom{70}{3} + \binom{69}{3} - \binom{59}{3} = 74625.$$

The total number of solutions to

$$a+b+c+d = 100$$

with

$$1 \leq a \leq 10, \ b \geq 0, \ c \geq 2, 20 \leq d \leq 30$$

is thus

$$\binom{80}{3} - \binom{70}{3} - \binom{69}{3} + \binom{59}{3} = 7535.$$

Exercises

488 Problem Telephone numbers in *Land of the Flying Camels* have 7 digits, and the only digits available are $\{0,1,2,3,4,5,7,8\}$. No telephone number may begin in 0, 1 or 5. Find the number of telephone numbers possible that meet the following criteria:

❶ You may repeat all digits.

❷ You may not repeat any of the digits.

❸ You may repeat the digits, but the phone number must be even.

❹ You may repeat the digits, but the phone number must be odd.

❺ You may not repeat the digits and the phone numbers must be odd.

489 Problem The number 3 can be expressed as a sum of one or more positive integers in four ways, namely, as 3, $1+2$, $2+1$, and $1+1+1$. Show that any positive integer n can be so expressed in 2^{n-1} ways.

490 Problem How many two or three letter initials for people are available if at least one of the letters must be a D and one allows repetitions?

491 Problem The sequence of palindromes, starting with 1 is written in ascending order

$$1,2,3,4,5,6,7,8,9,11,22,33,\ldots$$

Find the 1984-th positive palindrome.

492 Problem Would you believe a market investigator that reports that of 1000 people, 816 like candy, 723 like ice cream, 645 cake, while 562 like both candy and ice cream, 463 like both candy and cake, 470 both ice cream and cake, while 310 like all three? State your reasons!

493 Problem An auto insurance company has $10,000$ policyholders. Each policy holder is classified as

- young or old,
- male or female, and
- married or single.

Of these policyholders, 3000 are young, 4600 are male, and 7000 are married. The policyholders can also be classified as 1320 young males, 3010 married males, and 1400 young married persons. Finally, 600 of the policyholders are young married males.

How many of the company's policyholders are young, female, and single?

494 Problem In *Medieval High* there are forty students. Amongst them, fourteen like Mathematics, sixteen like theology, and eleven like alchemy. It is also known that seven like Mathematics and theology, eight like theology and alchemy and five like Mathematics and alchemy. All three subjects are favoured by four students. How many students like neither Mathematics, nor theology, nor alchemy?

495 Problem (Lewis Carroll in *A Tangled Tale.*) In a very hotly fought battle, at least 70% of the combatants lost an eye, at least 75% an ear, at least 80% an arm, and at least 85% a leg. What can be said about the percentage who lost all four members?

496 Problem An urn contains 28 blue marbles, 20 red marbles, 12 white marbles, 10 yellow marbles, and 8 magenta marbles. How many marbles must be drawn from the urn in order to assure that there will be 15 marbles of the same color?

497 Problem The nine entries of a 3×3 grid are filled with -1, 0, or 1. Prove that among the eight resulting sums (three columns, three rows, or two diagonals) there will always be two that add to the same number.

498 Problem Forty nine women and fifty one men sit around a round table. Demonstrate that there is at least a pair of men who are facing each other.

499 Problem An eccentric widow has five cats[4]. These cats have 16 kittens among themselves. What is the largest integer n for which one can say that at least one of the five cats has n kittens?

500 Problem Given any set of ten natural numbers between 1 and 99 inclusive, prove that there are two disjoint nonempty subsets of the set with equal sums of their elements.

[4] Why is it always eccentric widows who have multiple cats?

Answers

488 We have

❶ This is $5 \cdot 8^6 = 1310720$.

❷ This is $5 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 = 25200$.

❸ This is $5 \cdot 8^5 \cdot 4 = 655360$.

❹ This is $5 \cdot 8^5 \cdot 4 = 655360$.

❺ We condition on the last digit. If the last digit were 1 or 5 then we would have 5 choices for the first digit, and so we would have

$$5 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 2 = 7200$$

phone numbers. If the last digit were either 3 or 7, then we would have 4 choices for the last digit and so we would have

$$4 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 2 = 5760$$

phone numbers. Thus the total number of phone numbers is

$$7200 + 5760 = 12960.$$

489 $n = \underbrace{1+1+\cdots+1}_{n-1\ +'\text{s}}$. One either erases or keeps a plus sign.

490 $(26^2 - 25^2) + (26^3 - 25^3) = 2002$

491 It is easy to see that there are 9 palindromes of 1-digit, 9 palindromes with 2-digits, 90 with 3-digits, 90 with 4-digits, 900 with 5-digits and 900 with 6-digits. The last palindrome with 6 digits, 999999, constitutes the $9 + 9 + 90 + 90 + 900 + 900 = 1998$th palindrome. Hence, the 1997th palindrome is 998899, the 1996th palindrome is 997799, the 1995th palindrome is 996699, the 1994th is 995599, etc., until we find the 1984th palindrome to be 985589.

492 Let C denote the set of people who like candy, I the set of people who like ice cream, and K denote the set of people who like cake. We are given that $|C| = 816$, $|I| = 723$, $|K| = 645$, $|C \cap I| = 562$, $|C \cap K| = 463$, $|I \cap K| = 470$, and $\text{card}\,(C \cap I \cap K) = 310$. By Inclusion-Exclusion we have

$$\begin{aligned} |C \cup I \cup K| &= |C| + |I| + |K| \\ &\quad -|C \cap I| - |C \cap K| - |I \cap C| \\ &\quad +|C \cap I \cap K| \\ &= 816 + 723 + 645 - 562 - 463 - 470 + 310 \\ &= 999. \end{aligned}$$

The investigator miscounted, or probably did not report one person who may not have liked any of the three things.

493 Let Y, F, S, M stand for young, female, single, male, respectively, and let Ma stand for married. We have

$$\begin{aligned} |Y \cap F \cap S| &= |Y \cap F| - |Y \cap F \cap Ma| \\ &= |Y| - |Y \cap M| \\ &\qquad -(|Y \cap Ma| - \text{card}\,(Y \cap Ma \cap M)) \\ &= 3000 - 1320 - (1400 - 600) \\ &= 880. \end{aligned}$$

494 Let A be the set of students liking Mathematics, B the set of students liking theology, and C be the set of students liking alchemy. We are given that

$$|A| = 14, |B| = 16, |C| = 11, |A \cap B| = 7, |B \cap C| = 8, |A \cap C| = 5,$$

and

$$|A \cap B \cap C| = 4.$$

By the Principle of Inclusion-Exclusion,

$$|\overline{A} \cap \overline{B} \cap \overline{C}| = 40 - |A| - |B| - |C| + |A \cap B| + |A \cap C| + |B \cap C| - |A \cap B \cap C|$$

Substituting the numerical values of these cardinalities

$$40 - 14 - 16 - 11 + 7 + 5 + 8 - 4 = 15.$$

495 Let A denote the set of those who lost an eye, B denote those who lost an ear, C denote those who lost an arm and D denote those losing a leg. Suppose there are n combatants. Then

$$\begin{array}{lll} n & \geq & |A \cup B| \\ & = & |A| + |B| - |A \cap B| \\ & = & .7n + .75n - |A \cap B|, \end{array}$$

$$\begin{array}{lll} n & \geq & |C \cup D| \\ & = & |C| + |D| - |C \cap D| \\ & = & .8n + .85n - \text{card}\,(C \cap D)\,. \end{array}$$

This gives

$$|A \cap B| \geq .45n,$$

$$|C \cap D| \geq .65n.$$

This means that

$$\begin{array}{lll} n & \geq & |(A \cap B) \cup (C \cap D)| \\ & = & |A \cap B| + |C \cap D| - |A \cap B \cap C \cap D| \\ & \geq & .45n + .65n - |A \cap B \cap C \cap D|, \end{array}$$

whence

$$|A \cap B \cap C \cap D| \geq .45 + .65n - n = .1n.$$

This means that at least 10% of the combatants lost all four members.

496 If all the magenta, all the yellow, all the white, 14 of the red and 14 of the blue marbles are drawn, then in among these $8 + 10 + 12 + 14 + 14 = 58$ there are no 15 marbles of the same color. Thus we need 59 marbles in order to insure that there will be 15 marbles of the same color.

497 There are seven possible sums, each one a number in $\{-3, -2, -1, 0, 1, 2, 3\}$. By the Pigeonhole Principle, two of the eight sums must add up to the same.

498 Pick a pair of different sex facing one another, that is, forming a "diameter" on the table. On either side of the diameter there must be an equal number of people, that is, forty nine. If all the men were on one side of the diameter then we would have a total of $49 + 1 = 50$, a contradiction.

499 We have $\lceil\!\lceil \frac{16}{5} \rceil\!\rceil = 4$, so there is at least one cat who has four kittens.

500 There are $2^{10} - 1 = 1023$ non-empty subsets that one can form with a given 10-element set. To each of these subsets we associate the sum of its elements. The maximum value that any such sum can achieve is $90 + 91 + \cdots + 99 = 945 < 1023$. Therefore, there must be at least two different subsets that have the same sum.

Homework

501 Problem (Eötvös, 1947) Prove that amongst six people in a room there are at least three who know one another, or at least three who do not know one another.

502 Problem Suppose that the letters of the English alphabet are listed in an arbitrary order.

1. Prove that there must be four consecutive consonants.

2. Give a list to show that there need not be five consecutive consonants.

3. Suppose that all the letters are arranged in a circle. Prove that there must be five consecutive consonants.

503 Problem Bob has ten pockets and forty four silver dollars. He wants to put his dollars into his pockets so distributed that each pocket contains a different number of dollars.

1. Can he do so?

2. Generalize the problem, considering p pockets and n dollars. The problem is most interesting when
$$n = \frac{(p-1)(p-2)}{2}.$$
Why?

504 Problem Expand

1. $(x-4y)^3$

2. $(x^3+y^2)^4$

3. $(2+3x)^3$

4. $(2i-3)^4$

5. $(2i+3)^4+(2i-3)^4$

6. $(2i+3)^4-(2i-3)^4$

7. $(\sqrt{3}-\sqrt{2})^3$

8. $(\sqrt{3}+\sqrt{2})^3+(\sqrt{3}-\sqrt{2})^3$

9. $(\sqrt{3}+\sqrt{2})^3-(\sqrt{3}-\sqrt{2})^3$

505 Problem Prove that
$$(a+b+c)^2 = a^2+b^2+c^2+2(ab+bc+ca)$$
Prove that
$$(a+b+c+d)^2 = a^2+b^2+c^2+d^2+2(ab+ac+ad+bc+bd+cd)$$
Generalize.

506 Problem Compute $(x+2y+3z)^2$.

507 Problem Given that $a+2b=-8,\ ab=4$, find (i) a^2+4b^2, (ii) a^3+8b^3, (iii) $\frac{1}{a}+\frac{1}{2b}$.

508 Problem The sum of the squares of three consecutive positive integers is 21170. Find the sum of the cubes of those three consecutive positive integers.

509 Problem What is the coefficient of x^4y^6 in

$$(x\sqrt{2}-y)^{10}?$$

510 Problem Expand and simplify

$$(\sqrt{1-x^2}+1)^7-(\sqrt{1-x^2}-1)^7.$$

511 Problem There are approximately 7,000,000,000 people on the planet. Assume that everyone has a name that consists of exactly k lower-case letters from the English alphabet.

1. If $k=8$, is it guaranteed that two people have the same name? Explain.
2. What is the maximum value of k that would guarantee that at least two people have the same name?
3. What is the maximum value of k that would guarantee that at least 100 people have the same name?
4. Now assume that names can be between 1 and k characters long. What is the maximum value of k that would guarantee that at least two people have the same name?

512 Problem Password cracking is the process of determining someone's password, typically using a computer. One way to crack passwords is to perform an exhaustive search that tries every possible string of a given length until it (hopefully) finds it. Assume your computer can test 10,000,000 passwords per second. How long would it take to crack passwords with the following restrictions? Give answers in seconds, minutes, hours, days, or years depending on how large the answer is (e.g. 12,344,440 seconds isn't very helpful). Start by determining how many possible passwords there are in each case.

1. 8 lower-case alphabetic characters.
2. 8 alphabetic characters (upper or lower).
3. 8 alphabetic (upper or lower) and numeric characters.
4. 8 alphabetic (upper or lower), numeric characters, and special characters (assume there are 32 allowable special characters).
5. 8 or fewer alphabetic (upper or lower) and numeric characters.
6. 10 alphabetic (upper or lower), numeric characters, and special characters (assume there are 32 allowable special characters).
7. 8 characters, with at least one upper-case, one lower-case, one number, and one special character.

513 Problem IP addresses are used to identify computers on a network. In IPv4, IP addresses are 32 bits long. They are usually written using dotted-decimal notation, where the 32 bits are split up into 4 8-bit segments, and each 8-bit segment is represented in decimal. So the IP address 10000001 11000000 00011011 00000100 is represented as 129.192.27.4. The *subnet mask* of a network is a string of k ones followed by $32-k$ zeros, where the value of k can be different on different networks. For instance, the subnet mask might be 11111111111111111111111100000000, which is 255.255.255.0 in dotted decimal. To determine the *netid*, an IP address is bitwise ANDed with the subnet mask. To determine the *hostid*, an IP address is bitwise ANDed with the bitwise complement of the subnet mask. Since every computer on a network needs to have a different *hostid*, the number of possible *hostid*s determines the maximum number of computers that can be on a network.

Assume that the subnet mask on my computer is currently 255.255.255.0 and my IP address is 209.140.209.27.

1. What are the *netid* and *hostid* of my computer?

2. How many computers can be on the network that my computer is on?

3. In 2010, Hope College's network was not split into subnetworks like it is currently, so all of the computers were on a single network that had a subnet mask of 255.255.240.0. How many computers could be on Hope's network in 2010?

514 Problem Prove that $\sum_{k=0}^{n}\binom{n}{k}=2^n$ by counting the number of binary strings of length n in two ways.

515 Problem You get a new job and your boss gives you 2 choices for your salary. You can either make \$100 per day or you can start at \$.01 on the first day and have your salary doubled every day. You know that you will work for k days. For what values of k should you take the first offer? The second? Explain.

516 Problem The 300-level courses in the CS department are split into three groups: Foundations (361, 385), Applications (321, 342, 392), and Systems (335, 354, 376). In order to get a BS in computer science at Hope you need to take at least one course from each group.

1. How many different ways are there of satisfying this requirement by taking exactly 3 courses?

2. If you take four 300-level courses, how many different possibilities do you have that satisfy the requirements?

3. How many ways are there to take 300-level courses that satisfy the requirements?

4. What is the fewest number of 300-level courses you need to take to guarantee that you will satisfy the requirement no matter which courses you choose?

517 Problem In March of every year people fill out brackets for the NCAA Basketball Tournament. They pick the winner of each game in each round. We will assume the tournament starts with 64 teams (it has become a little more complicated than this recently). The first round of the tournament consists of 32 games, the second 16 games, the third 8, the fourth 4, the fifth 2, and the final 1. So the total number of games is $32+16+8+4+2+1=63$. You can arrive at the number of games in a different way. Every game has a loser who is out of the tournament. Since only 1 of the 64 teams remains at the end, there must be 63 losers, so there must be 63 games. Notice that we can also write $1+2+4+8+16+32=63$ as $\sum_{k=0}^{5}2^k=2^6-1$.

1. Use a combinatorial proof to show that for any $n > 0$, $\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$. (That is, define an appropriate set and count the cardinality of the set in two ways to obtain the identity.)

2. When you fill out a bracket you are picking who you think the winner will be of each game. How many different ways are there to fill out a bracket? (Hint: If you don't over think it, this is pretty easy.)

3. If everyone on the planet (7,000,000,000) filled out a bracket, is it guaranteed that two people will have the same bracket? Explain.

4. Assume that everyone on the planet fills out k different brackets and that no brackets are repeated (either by an individual or by anybody else). How large would k have to be before it is guaranteed that somebody has a bracket that correctly predicts the winner of every game?

5. Assume every pair of people on the planet gets together to fill out a bracket (so everyone has 6,999,999 brackets, one with every other person on the planet). What is the smallest and largest number of possible repeated brackets?

518 Problem Mega Millions has 56 white balls numbered 1-56 and one red ball numbered 1-46. To play you pick 5 white balls and 1 red ball. Then 5 of the 56 balls and 1 of the 46 balls are drawn randomly (or so they would have us believe). You win if you match all 6 balls.

1. How many different draws are possible?

2. If everyone in the U.S.A. bought a ticket (about 314,000,000), is it guaranteed that two people have the same numbers? Three people?

3. If everyone in the U.S.A. bought a ticket, what is the maximum number of people that are guaranteed to share the jackpot?

4. Which is more likely: Winning Mega Millions or picking every winner in the NCAA Basketball Tournament (see previous question)?

5. (hard) What is the largest value of k such that you are more likely to pick at least k winners in the NCAA Basketball Tournament than you are to win Mega Millions?

519 Problem I am implementing a data structure that consists of k lists. I want to store a total of n objects in this data structure, with each item being stored on one of the lists. All of the lists will have the same capacity (e.g. perhaps each list can hold up to 10 elements).
Write a method `minimumCapacity(int n, int k)` that computes the minimum capacity each of the k lists must have to accommodate n objects. In other words, if the capacity is less than this, then there is no way the objects can all be stored on the lists. You may assume integer arithmetic truncates (essentially giving you the *floor* function), but that there is no *ceiling* function available.

520 Problem How many license plates can be made using either three letters followed by three digits or four letters followed by two digits?

521 Problem How many license plates can be made using 4 letters and 3 numbers if the letters cannot be repeated?

522 Problem How many bit strings of length 8 either begin with three 1s or end with four 0s?

523 Problem How many alphabetic strings are there whose length is at most 5?

524 Problem How many bit strings are there of length at least 4 and at most 6?

525 Problem How many subsets with 4 or more elements does a set of size 30 have?

526 Problem Prove that at a gathering of $n \geq 2$ people, there are two people who have shaken hands with the same number of people.

527 Problem Given a group of ten people, prove that at least 4 are male or at least 7 are female.

528 Problem My family wants to take a group picture. There are 7 men and 5 women, and we want none of the women to stand next to each other. How many different ways are there for us to line up?

529 Problem My family (7 men and 5 women) wants to select a group of 5 of us to plan Christmas. We want at least 1 man and 1 woman in the group. How many ways are there for us to select the members of this group?

530 Problem Compute each of the following: $\binom{8}{4}$, $\binom{9}{9}$, $\binom{7}{3}$, 8!, and 5!

531 Problem For what value(s) of k is $\binom{18}{k}$ largest? smallest?

532 Problem For what value(s) of k is $\binom{19}{k}$ largest? smallest?

533 Problem A computer network consists of 10 computers. Each computer is directly connected to zero or more of the other computers. Prove that there are at least two computers in the network that are directly connected to the same number of other computers.

534 Problem Simplify the following expression so it does not involve any factorials or binomial coefficients: $\binom{x}{y} / \binom{x+1}{y-1}$.

535 Problem What is the coefficient of x^6y^9 in $(3x-2y)^{15}$?

536 Problem Prove that for any positive integer n, $\displaystyle\sum_{k=0}^{n}(-2)^k\binom{n}{k} = (-1)^n$. (Hint: *Don't* use induction.)

537 Problem Write a method `choose(int n, int k)` (in a Java-like language) that computes $\binom{n}{k}$. Your implementation should be as efficient as possible.

This page intentionally left blank.

Chapter 8

Graph Theory

8.1 Simple Graphs

538 Definition A *simple graph (network)* $G = (V, E)$ consists of a non-empty set V (called the *vertex (node)* set) and a set E (possibly empty) of unordered pairs of elements (called the *edges* or *arcs*) of V.

Vertices are usually represented by means of dots on the plane, and the edges by means of lines connecting these dots. See figures 8.1 through 8.4 for some examples of graphs.

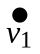


Figure 8.1: K_1, a graph with $|V| = 1$ and $|E| = 0$.

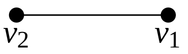


Figure 8.2: K_2, a graph with $|V| = 2$ and $|E| = 1$.

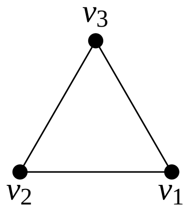


Figure 8.3: K_3, a graph with $|V| = 3$ and $|E| = 3$.

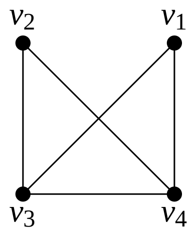


Figure 8.4: A graph with $|V| = 3$ and $|E| = 5$.

539 Definition If v and v' are vertices of a graph G which are joined by an edge e, we say that v is *adjacent* to v' and that v and v' are *neighbours*, and we write $e = vv'$. We say that vertex v is *incident* with an edge e if v is an endpoint of e. In this case we also say that e is incident with v.

540 Definition The *degree* of a vertex is the number of edges incident to it.

Depending on whether $|V|$ is finite or not, the graph is finite or infinite. In these notes we will only consider finite graphs.

Our definition of a graph does not allow that two vertices be joined by more than one edge. If this were allowed we would obtain a *multigraph*. Neither does it allow *loops* , which are edges incident to only one vertex. A graph with loops is a *pseudograph*.

541 Definition The *complete graph* with n vertices K_n is the graph where any two vertices are adjacent. Thus K_n has $\binom{n}{2}$ edges.

Figure 8.1 shows K_1, figure 8.2 shows K_2, figure 8.3 shows K_3, and figure 8.5 shows K_4, figure 8.6 shows K_5.

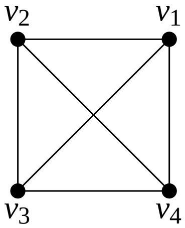


Figure 8.5: K_4.

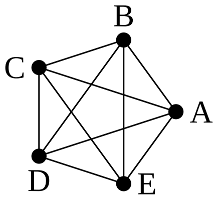


Figure 8.6: K_5.

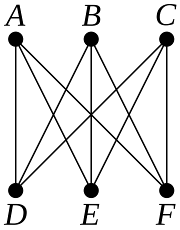


Figure 8.7: $K_{3,3}$.

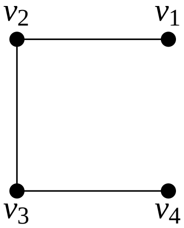


Figure 8.8: P_3.

542 Definition Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ is an *independent set* of vertices if $uv \not\in E$ for all u, v in S (S may be empty). A *bipartite graph* with bipartition X, Y is a graph such that $V = X \cup Y$, $X \cap Y = \varnothing$, and X and Y are independent sets. X and Y are called the *parts* of the bipartition.

543 Definition $K_{m,n}$ denotes the *complete bipartite graph* with $m+n$ vertices. One part, with m vertices, is connected to every other vertex of the other part, with n vertices.

544 Definition A $u-v$ *walk* in a graph $G = (V, E)$ is an alternating sequence of vertices and edges in G with starting vertex u and ending vertex v such that every edge joins the vertices immediately preceding it and immediately following it.

545 Definition A $u-v$ *trail* in a graph $G = (V, E)$ is a $u-v$ walk that does not repeat an edge, while a $u-v$ *path* is a walk that which does not repeat any vertex.

546 Definition P_n denotes a *path* of length n. It is a graph with n edges, and $n+1$ vertices $v_0 v_1 \cdots v_n$, where v_i is adjacent to v_{i+1} for $n = 0, 1, \ldots, n-1$.

547 Definition C_n denotes a *cycle* of length n. It is a graph with n edges, and n vertices $v_1 \cdots v_n$, where v_i is adjacent to v_{i+1} for $n = 1, \ldots, n-1$, and v_1 is adjacent to v_n.

548 Definition Q_n denotes the *n-dimensional cube* (or *hypercube*). It is a simple graph with 2^n vertices, which we label with n-tuples of 0's and 1's. Vertices of Q_n are connected by an edge if and only if they differ by exactly one coordinate. Observe that Q_n has $n2^{n-1}$ edges.

Figure 8.7 shows $K_{3,3}$, figure 8.8 shows P_3, figure 8.9 shows C_5, figure 8.10 shows Q_2, and figure 8.11 shows Q_3.

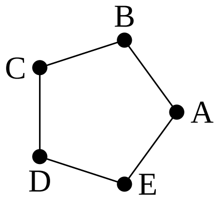


Figure 8.9: C_5.

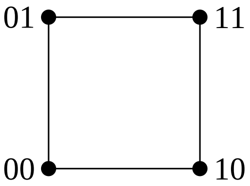


Figure 8.10: Q_2.

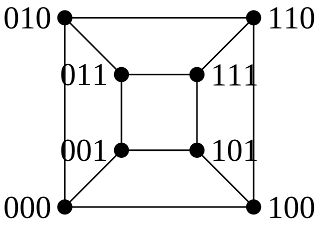


Figure 8.11: Q_3.

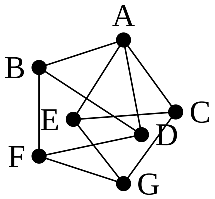


Figure 8.12: Example 550.

549 Definition A *subgraph* $G_1 = (V_1, E_1)$ of a graph $G = (V, E)$ is a graph with $V_1 \subseteq V$ and $E_1 \subseteq E$.

We will now give a few examples of problems whose solutions become simpler when using a graph-theoretic model.

550 Example If the points of the plane are coloured with three different colours, red, white, and blue, say, show that there will always exist two points of the same colour which are 1 unit apart.

> **Solution:** In figure 8.12 all the edges have length 1. Assume the property does not hold and that A is coloured red, B is coloured white, D coloured blue. Then F must both be coloured red. Since E and C must not be red, we also conclude that G is red. But then F and G are at distance 1 apart and both coloured red which contradicts our assumption that the property did not hold.

551 Example A wolf, a goat, and a cabbage are on one bank of a river. The ferryman wants to take them across, but his boat is too small to accommodate more than one of them. Evidently, he can neither leave the wolf and the goat, or the cabbage and the goat behind. Can the ferryman still get all of them across the river?

> **Solution:** Represent the position of a single item by 0 for one bank of the river and 1 for the other bank. The position of the three items can now be given as an ordered triplet, say (W, G, C). For example, $(0, 0, 0)$ means that the three items are on one bank of the river, $(1, 0, 0)$ means that the wolf is on one bank of the river while the goat and the cabbage are on the other bank. The object of the puzzle is now seen to be to move from $(0, 0, 0)$ to $(1, 1, 1)$, that is, traversing Q_3 while avoiding certain edges. One answer is
>
> $$000 \rightarrow 010 \rightarrow 011 \rightarrow 001 \rightarrow 101 \rightarrow 111.$$
>
> This means that the ferryman (i) takes the goat across, (ii) returns and that the lettuce over bringing back the goat, (iii) takes the wolf over, (iv) returns and takes the goat over. Another one is
>
> $$000 \rightarrow 010 \rightarrow 110 \rightarrow 100 \rightarrow 101 \rightarrow 111.$$
>
> This means that the ferryman (i) takes the goat across, (ii) returns and that the wolf over bringing back the goat, (iii) takes the lettuce over, (iv) returns and takes the goat over. The graph depicting both answers can be seen in figure 8.13. You can find a pictorial representation at `http://www.cut-the-knot.org/ctk/GoatCabbageWolf.shtml`.

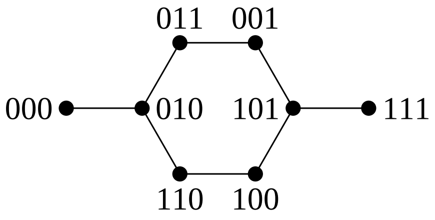


Figure 8.13: Example 551.

552 Example Prove that amongst six people in a room there are at least three who know one another, or at least three who do not know one another.

Solution: In graph-theoretic terms, we need to show that every colouring of the edges of K_6 into two different colours, say red and blue, contains a monochromatic triangle (that is, the edges of the triangle have all the same colour). Consider an arbitrary person of this group (call him Peter). There are five other people, and of these, either three of them know Peter or else, three of them do not know Peter. Let us assume three do know Peter, as the alternative is argued similarly. If two of these three people know one another, then we have a triangle (Peter and these two, see figure 8.14, where the acquaintances are marked by solid lines). If no two of these three people know one another, then we have three mutual strangers, giving another triangle (see figure 8.15).

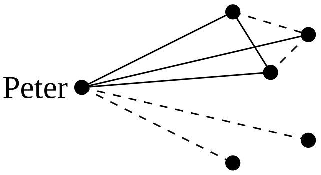


Figure 8.14: Example 552.

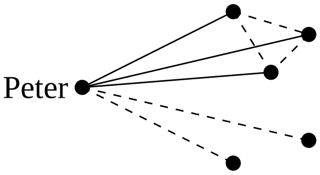


Figure 8.15: Example 552.

553 Example Mr. and Mrs. Landau invite four other married couples for dinner. Some people shook hands with some others, and the following rules were noted: (i) a person did not shake hands with himself, (ii) no one shook hands with his spouse, (iii) no one shook hands more than once with the same person. After the introductions, Mr. Landau asks the nine people how many hands they shook. Each of the nine people asked gives a different number. How many hands did Mrs. Landau shake?

Solution: The given numbers can either be $0,1,2,\ldots,8$, or $1,2,\ldots,9$. Now, the sequence $1,2,\ldots,9$ must be ruled out, since if a person shook hands nine times, then he must have shaken hands with his spouse, which is not allowed. The only permissible sequence is thus $0,1,2,\ldots,8$. Consider the person who shook hands 8 times, as in figure 8.16. Discounting himself and his spouse, he must have shaken hands with everybody else. This means that he is married to the person who shook 0 hands! We now consider the person that shook 7 hands, as in figure 8.17. He didn't shake hands with himself, his spouse, or with the person that shook 0 hands. But the person that shook hands only once did so with the person shaking 8 hands. Thus the person that shook hand 7 times is married to the person that shook hands once. Continuing this argument, we see the following pairs $(8,0)$, $(7,1)$, $(6,2)$, $(5,3)$. This leaves the person that shook hands 4 times without a partner, meaning that this person's partner did not give a number, hence this person must be Mrs. Landau! Conclusion: Mrs. Landau shook hands four times. A graph of the situation appears in figure 8.18.

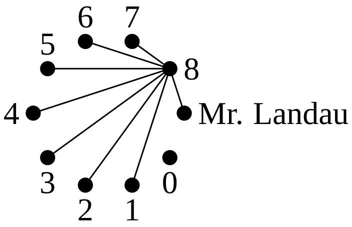


Figure 8.16: Example 553.

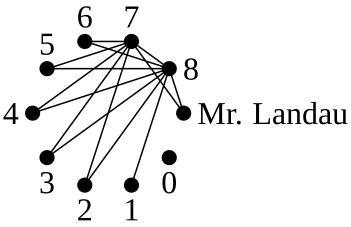


Figure 8.17: Example 553.

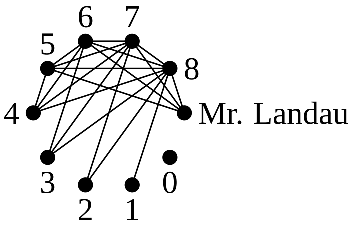


Figure 8.18: Example 553.

8.2 Graphic Sequences

554 Definition A sequence of non-negative integers is *graphic* if there exists a graph whose degree sequence is precisely that sequence.

555 Example The sequence $1,1,1$ is graphic, since K_3 is a graph with this degree sequence, and in general, so is the sequence $\underbrace{n,n,\ldots,n}_{n+1\ n'\text{s}}$, since K_{n+1} has this degree sequence. The degree sequence $1,\underbrace{2,2,\ldots,2}_{n\text{ twos}},1$ is graphic, since P_{n+1} has this sequence. The degree sequence $\underbrace{2,2,\ldots,2}_{n\text{ twos}}$ is graphic, since C_n has this sequence. The sequence $0,1,2,3,4,5,6,7,8$ is graphic, but the sequence $1,2,3,4,5,6,7,8,9$ is not according to Example 553.

8.3 Connectivity

556 Definition A graph is *connected* if there is a path between every pair of vertices. A *component* of a graph is a maximal connected subgraph.

557 Definition A *forest* is a graph with no cycles (acyclic). A *tree* is a connected acyclic graph. A *spanning tree* of a graph of a connected graph G is a subgraph of G which is a tree and having exactly the same of vertices as G.

8.4 Traversability

We start with the following, which is valid not only for simple graphs, but also for multigraphs and pseudographs.

558 Theorem (Handshake Lemma) Let $G=(V,E)$ be a graph. Then

$$\sum_{v\in V}\deg v = 2|E|.$$

Proof: If the edge connects two distinct vertices, as sum traverses through the vertices, each edge is counted twice. If the edge is a loop, then every vertex having a loop contributes 2 to the sum. This gives the theorem. □

559 Corollary Every graph has an even number of vertices of odd degree.

Proof: The sum of an odd number of odd numbers is odd. Since the sum of the degrees of the vertices in a simple graph is always even, one cannot have an odd number of odd degree vertices. □

560 Definition A *trail* is a walk where all the edges are distinct. An *Eulerian trail* on a graph G is a trail that traverses every edge of G. A *tour* of G is a closed walk that traverses each edge of G at least once. An *Euler tour* on G is a tour traversing each edge of G exactly once, that is, a closed Euler trail. A graph is *Eulerian* if it contains an Euler tour.

561 Theorem A nonempty connected graph is Eulerian if and only if has no vertices of odd degree.

Proof: Assume first that G is Eulerian, and let C be an Euler tour of C starting and ending at vertex u. Each time a vertex v is encountered along C, two of the edges incident to v are accounted for. Since C contains every edge of G, $d(v)$ is then even for all $v \neq u$. Also, since C begins and ends in u, $d(u)$ must also be even.

Conversely, assume that G is a connected nonEulerian graph with at least one edge and no vertices of odd degree. Let W be the longest walk in G that traverses every edge at most once:

$$W = v_0, v_0v_1, v_1, v_1v_2, v_2, \ldots, v_{n-1}, v_{n-1}v_n, v_n.$$

Then W must traverse every edge incident to v_n, otherwise, W could be extended into a longer walk. In particular, W traverses two of these edges each time it passes through v_n and traverses $v_{n-1}v_n$ at the end of the walk. This accounts for an odd number of edges, but the degree of v_n is even by assumption. Hence, W must also begin at v_n, that is, $v_0 = v_n$. If W were not an Euler tour, we could find an edge not in W but incident to some vertex in W since G is connected. Call this edge uv_i. But then we can construct a longer walk:

$$u, uv_i, v_i, v_iv_{i+1}, \ldots, v_{n-1}v_n, v_n, v_0v_1, \ldots, v_{i-1}v_i, v_i.$$

This contradicts the definition of W, so W must be an Euler tour. □

The following problem is perhaps the originator of graph theory.

562 Example (Königsberg Bridge Problem) The town of Königsberg (now called Kaliningrad) was built on an island in the Pregel River. The island sat near where two branches of the river join, and the borders of the town spreaded over to the banks of the river as well as a nearby promontory. Between these four land masses, seven bridges had been erected. The townsfolk used to amuse themselves by crossing over the bridges and asked whether it was possible to find a trail starting and ending in the same location allowing one to traverse each of the bridges exactly once. Figure 8.19 has a graph theoretic model of the town, with the seven edges of the graph representing the seven bridges. By Theorem 561, this graph is not Eulerian so it is impossible to find a trail as the townsfolk asked.

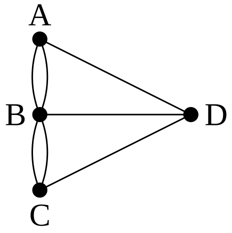


Figure 8.19: Example 562.

563 Definition A *Hamiltonian cycle* in a graph is a cycle passing through every vertex. G is *Hamiltonian* if it contains a Hamiltonian cycle.

Unlike Theorem 561, there is no simple characterisation of all graphs with a Hamiltonian cycle. We have the following one way result, however.

564 Theorem (Dirac's Theorem, 1952) Let $G = (V, E)$ be a graph with $n = |E| \geq 3$ edges whose every vertex has degree $\geq \frac{n}{2}$. Then G is Hamiltonian.

Proof: Arguing by contradiction, suppose G is a maximal non-Hamiltonian with with $n \geq 3$, and that G has more than 3 vertices. Then G cannot be complete. Let a and b be two non-adjacent vertices of G. By definition of G, $G+ab$ is Hamiltonian, and each of its Hamiltonian cycles must contain the edge ab. Hence, there is a Hamiltonian path $v_1v_2\ldots v_n$ in G beginning at $v_1 = a$ and ending at $v_n = b$. Put

$$S = \{v_i : av_{i+1} \in E\} \qquad \text{and} \qquad \{v_j : v_jb \in E\}.$$

As $v_n \in S\cap T$ we must have $|S\cup T| = n$. Moreover, $S\cap T = \varnothing$, since if $v_i\S\cap T$ then G would have the Hamiltonian cycle

$$v_1v_2\cdots v_iv_nv_{n-1}\cdots v_{i+1}v_1,$$

as in figure 8.20, contrary to the assumption that G is non-Hamiltonian. But then

$$d(a)+d(b) = |S|+|T| = |S\cup T|+|S\cap T| < n.$$

But since we are assuming that $d(a) \geq \dfrac{n}{2}$ and $d(b) \geq \dfrac{n}{2}$, we have arrived at a contradiction. □

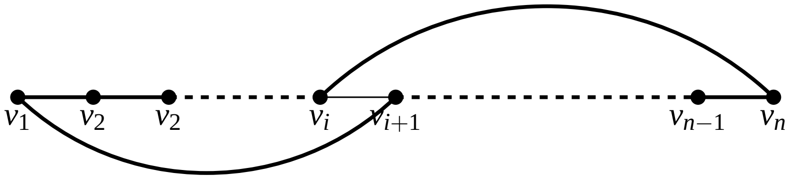


Figure 8.20: Theorem 564

8.5 Planarity

565 Definition A graph is *planar* if it can be drawn in a plane with no intersecting edges.

566 Example K_4 is planar, as shown in figure 8.21.

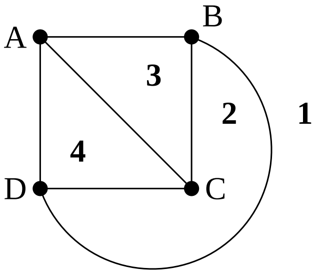


Figure 8.21: Example 568.

567 Definition A *face* of a planar graph is a region bounded by the edges of the graph.

568 Example From figure 8.21, K_4 has 4 faces. Face **1** which extends indefinitely, is called the *outside face*.

569 Theorem (Euler's Formula) For every drawing of a connected planar graph with v vertices, e edges, and f faces the following formula holds:

$$v-e+f=2.$$

Proof: The proof is by induction on e. Let $P(e)$ be the proposition that $v-e+f=2$ for every drawing of a graph G with e edges. If $e=0$ and it is connected, then we must have $v=1$ and hence $f=1$, since there is only the outside face. Therefore, $v-e+f=1-0+1=2$, establishing $P(0)$.

Assume now $P(e)$ is true, and consider a connected graph G with $e+1$ edges. Either

❶ G has no cycles. Then there is only the outside face, and so $f=1$. Since there are $e+1$ edges and G is connected, we must have $v=e+2$. This gives $(e+2)-(e+1)+1=2-1+1=2$, establishing $P(e+1)$.

❷ or G has at least one cycle. Consider a spanning tree of G and an edge uv in the cycle, but not in the tree. Such an edge is guaranteed by the fact that a tree has no cycles. Deleting uv merges the two faces on either side of the edge and leaves a graph G' with only e edges, v vertices, and f faces. G' is connected since there is a path between every pair of vertices within the spanning tree. So $v-e+f=2$ by the induction assumption $P(e)$. But then

$$v-e+f=2 \implies (v)-(e+1)+(f+1)=2 \implies v-e+f=2,$$

establishing $P(e+1)$.

This finishes the proof. □

570 Theorem Every simple planar graph with $v\geq 3$ vertices has at $e\leq 3v-6$ edges. Every simple planar graph with $v\geq 3$ vertices and which does not have a C_3 has $e\leq 2v-4$ edges.

Proof: If $v=3$, both statements are plainly true so assume that G is a maximal planar graph with $v\geq 4$. We may also assume that G is connected, otherwise, we may add an edge to G. Since G is simple, every face has at least 3 edges in its boundary. If there are f faces, let F_k denote the number of edges on the k-th face, for $1\leq k\leq f$. We then have

$$F_1+F_2\cdots+F_f\geq 3f.$$

Also, every edge lies in the boundary of at most two faces. Hence if E_j denotes the number of faces that the j-th edge has, then

$$2e\geq E_1+E_2+\cdots+E_e.$$

Since $E_1+E_2+\cdots+E_e=F_1+F_2\cdots+F_f$, we deduce that $2e\geq 3f$. By Euler's Formula we then have $e\leq 3v-6$.

The second statement follows for $v=4$ by inspecting all graphs G with $v=4$. Assume then that $v\geq 5$ and that G has no cycle of length 3. Then each face has at least four edges on its boundary. This gives $2e\geq 4f$ and by Euler's Formula, $e\leq 2v-4$. □

571 Example K_5 is not planar by Theorem 570 since K_5 has $\binom{5}{2} = 10$ edges and $10 > 9 = 3(5) - 6$.

572 Example $K_{3,3}$ is not planar by Theorem 570 since $K_{3,3}$ has $3 \cdot 3 = 9$ edges and $9 > 8 = 2(6) - 4$.

573 Definition A *polyhedron* is a convex, three-dimensional region bounded by a finite number of polygonal faces.

574 Definition A *Platonic solid* is a polyhedron having congruent regular polygon as faces and having the same number of edges meeting at each corner.

By puncturing a face of a polyhedron and spreading its surface into the plane, we obtain a planar graph.

575 Example (Platonic Solid Problem) How many Platonic solids are there? If m is the number of faces that meet at each corner of a polyhedron, and n is the number of sides on each face, then, in the corresponding planar graph, there are m edges incident to each of the v vertices. As each edge is incident to two vertices, we have $mv = 2e$, and if each face is bounded by n edges, we also have $nf = 2e$. It follows from Euler's Formula that

$$\frac{2e}{m} - e + \frac{2e}{n} = 2 \implies \frac{1}{m} + \frac{1}{n} = \frac{1}{e} + \frac{1}{2}.$$

We must have $n \geq 3$ and $m \geq 3$ for a nondegenerate polygon. Moreover, if either n or m were ≥ 6 then

$$\leq \frac{1}{3} + \frac{1}{6} = \frac{1}{2} < \frac{1}{e} + \frac{1}{2}.$$

Thus we only need to check the finitely many cases with $3 \leq n, m \leq 5$. The table below gives the existing polyhedra.

n	m	v	e	f	polyhedron
3	3	4	6	4	tetrahedron
4	3	8	12	6	cube
3	4	6	12	8	octahedron
3	5	12	30	20	icosahedron
5	3	20	30	12	dodecahedron

576 Example (Regions in a Circle) Prove that the chords determined by n points on a circle cut the interior into $1 + \binom{n}{2} + \binom{n}{4}$ regions provided no three chords have a common intersection.

Solution: By viewing the points on the circle and the intersection of two chords as vertices, we obtain a plane graph. Each intersection of the chords is determined by four points on the circle, and hence our graph has $v = \binom{n}{4} + n$ vertices. Since each vertex inside the circle has degree 4 and each vertex on the circumference of the circle has degree $n+1$, the Handshake Lemma (Theorem 558) we have a total of

$$e = \frac{1}{2}\left(4\binom{n}{4} + n(n+1)\right)$$

edges. Discounting the outside face, our graph has

$$f - 1 = 1 + e - v = 1 + 2\binom{n}{4} + \frac{n^2}{2} + \frac{n}{2} - \left(\binom{n}{4} + n\right) = 1 + \binom{n}{2} + \binom{n}{4}$$

faces or regions.

Exercises

577 Problem Seventeen people correspond by mail with one another—each one with all the rest. In their letters only three different topics are discussed. Each pair of correspondents deals with only one of these topics. Prove that there at least three people who write to each other about the same topic.

578 Problem If a given convex polyhedron has six vertices and twelve edges, prove that every face is a triangle.

579 Problem Prove, using induction, that the sequence

$$n,n,n-1,n-1,\ldots,4,4,3,3,2,2,1,1$$

is always graphic.

580 Problem Seven friends go on holidays. They decide that each will send a postcard to three of the others. Is it possible that every student receives postcards from precisely the three to whom he sent postcards? Prove your answer!

Answers

577 Choose a particular person of the group, say Charlie. He corresponds with sixteen others. By the Pigeonhole Principle, Charlie must write to at least six of the people of one topic, say topic I. If any pair of these six people corresponds on topic I, then Charlie and this pair do the trick, and we are done. Otherwise, these six correspond amongst themselves only on topics II or III. Choose a particular person from this group of six, say Eric. By the Pigeonhole Principle, there must be three of the five remaining that correspond with Eric in one of the topics, say topic II. If amongst these three there is a pair that corresponds with each other on topic II, then Eric and this pair correspond on topic II, and we are done. Otherwise, these three people only correspond with one another on topic III, and we are done again.

578 Let x be the average number of edges per face. Then we must have $xf = 2e$. Hence $x = \frac{2e}{f} = \frac{24}{8} = 3$. Since no face can have fewer than three edges, every face must have exactly three edges.

579 The sequence $1,1$ is clearly graphic. Assume that the sequence

$$n-1,n-1,\ldots,4,4,3,3,2,2,1,1$$

is graphic and add two vertices, u,v. Join v to one vertex of degree $n-1$, one of degree of $n-2$,, etc., one vertex of degree 1. Since v is joined to $n-1$ vertices, and u so far is not joined to any vertex, we have a sequence

$$n,n-1,n-1,n-1,n-2,n-2,\ldots,4,4,3,3,2,2,1,0.$$

Finally, join u to v to obtain the sequence

$$n,n,n-1,n-1,\ldots,4,4,3,3,2,2,1,1.$$

580 The sequence $3,3,3,3,3,3,3$ is not graphic, as the number of vertices of odd degree is odd. Thus the given condition is not realizable.

Homework

581 Problem Determine whether there is a simple graph with eight vertices having degree sequence $6,5,4,3,2,2,2,2$.

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index