Data Structures: Lists, Queues, Stacks, and Trees

Some Examples Problems

1. *Prove that the maximum number of nodes of depth i in a binary tree is 2^i.*
   Proof: We will prove this by induction. There is at most $1 = 2^0$ nodes at level 0 of a binary tree (the root). Assume there are at most 2^i nodes at level i of a binary tree. Since each node can have at most 2 children, the number of nodes at level $i+1$ is at most $2 \cdot 2^i = 2^{i+1}$. We have the result by induction. □

2. *Prove that the maximum number of nodes in a binary tree of height i is $2^{i+1} - 1$.*
   Proof: We will prove this by induction. A binary tree of height 0 has at most $1 = 2^1 - 1$ nodes. Assume that a binary tree of height i has at most $2^{i+1} - 1$ nodes. We need to show that any binary tree of height $i+1$ has at most $2^{i+2} - 1$ nodes.

   Let T be a binary tree of height $i+1$, and let T' be the binary tree produced by removing all of the vertices at level $i+1$. Since T' has height i, it has at most $2^{i+1} - 1$ nodes, by assumption. T had at most 2^{i+1} nodes at level $i+1$ (see problem 1 above), so the maximum number of nodes T can have is $2^{i+1} + 2^{i+1} - 1 = 2 \cdot 2^{i+1} - 1 = 2^{i+2} - 1$. Thus, by induction, any binary tree of height i has at most $2^{i+1} - 1$ nodes. □

3. **5.5-4** *from page 96 of the text: Show by induction that the number of degree-2 nodes in any binary tree is 1 less than the number of leaves.*
   Proof: A binary tree with 1 node has 1 leaf and 0 nodes of degree 2, so the theorem holds for $v = 1$. Assume that every binary tree with v nodes has 1 less degree-2 node than leaves.

   Let T be a binary tree with $v+1$ nodes, a degree-2 nodes, and b leaves. Let x be a leaf node with parent y. There are 2 cases to consider

   Case 1: *y is a degree-1 node.* If y has degree 1, then removing x from T results in a tree T' having v nodes, a degree-2 nodes, and b leaves. Since T' has v nodes, it has 1 less degree-2 node than leaves. Thus, $a = b - 1$.

   Case 2: *y is a degree-2 node.* If y has degree 2, then removing x from T results in a tree T' having v nodes, $a - 1$ degree-2 nodes, and $b - 1$ leaves. Since T' has v nodes, it has 1 less degree-2 node than leaves. Thus, $a - 1 = b - 2$, or $a = b - 1$.

   In either case, $a = b - 1$. Thus T has 1 less degree-2 nodes than leaves. We have the result by induction.

4. **11.1-7** *from page 204 of the text. Show how to implement a stack using two queues. Analyze the running time of the stack operations.*

Solution: We need to implement the stack operations **pop** and **push** using two queues A and B, and the operations *enqueue* and *dequeue*. Below is one possible (although perhaps not the best) way to accomplish the task.

```
Push(x) [                Pop() [
   Enqueue(A,x)             x=Dequeue(A)
   ]                        While (A not empty) [
                                  Enqueue(B,x)
                                  x=Dequeue(A)  ]
                            While (B not empty) [
                                  y=Dequeue(B)
                                  Enqueue(A,y)  ]
                            Return(x)
                            ]
```

Push takes time $O(1)$ in any case, while *Pop* takes time $O(n)$ in the worst case. If the stack has an average of k elements at a time, then *Pop* takes time $O(k)$ on average.

5. **11.2-3** *from page 208 of the text. Implement a queue using a singly linked list L.*
Solution: Assume we have a linked list L that contains 2 pointers, *head* and *tail*. The head points to the first node as usual, and the tail pointer points to the last node. We will insert elements at the tail, and remove them from the head. We can implement a queue (omitting the boundary conditions) as follows:

```
Enqueue(x) [                    Dequeue() [
     next[tail[L]]=x               x=next[head[L]]
     tail=x                        next[head[L]]=next[x]
     ]                             Return(key[x]);
                                   ]
```

6. **11.2-5** *from page 208 of the text. Implement the* UNION *operation in time $O(1)$. (See book for more details)*
Solution: We will assume that S_1 and S_2 are stored as linked lists with *tail* pointers (as well as the usual *head* pointer). The following pseudo-code will perform the UNION operation in time $O(1)$.

```
Union(S1,S2) [
   S3=new Linked-List
   head[S3]=head[S1]
   tail[S3]=tail[S2]
   next[tail[S1]]=next[head[S2]]
   ]
```