BFS and DFS

- For many applications, one must systematically search through a graph to determine the structure of the graph.
- Two common elementary algorithms for tree-searching are
  - **Breadth-first search (BFS)**, and
  - **Depth-first search (DFS)**.
- Both of these algorithms work on directed or undirected graphs.
- Many advanced graph algorithms are based on the ideas of BFS or DFS.
- Each of these algorithms traverses edges in the graph, discovering new vertices as it proceeds.
- The difference is in the order in which each algorithm discovers the edges.
- Instead of talking about the differences, we will just look at each algorithm in turn.

Breadth-First Search

- Let $G = (V, E)$ be a graph.
- Let $s \in V$ be a special vertex called the **source**.
- A breadth-first search of G will:
  - Determine all vertices reachable from s.
  - Determine the distance (that is, the fewest number of edges) from s to all vertices reachable.
  - Determine a shortest path from s to all vertices reachable.
  - Construct a **breadth-first tree** rooted at s. That is, a tree whose edges form the shortest paths from s to all vertices reachable.
- Breadth-first search is so called because it locates all vertices of distance i before it locates any of distance $i + 1$.

How BFS Works

- For each node x in the graph we will need to store
  - the list of adjacent vertices,
  - the distance from s to x ($d(x)$),
  - predecessor of x ($p(x)$), and
  - the color of x ($c(x)$).
- The color of a vertex indicates the status of a vertex:
  - A **white** vertex has not been discovered.
  - A **gray** vertex has been discovered, but it may have undiscovered neighbors.
  - A **black** vertex has been discovered, and it has no undiscovered neighbors.
- The algorithm starts with
  - $d(x) = \infty$, $p(x) = NIL$, and $c(x) = white$ for $x \neq s$, and
  - $d(s) = 0$, $p(s) = NIL$, and $c(s) = gray$.

How BFS Works Continued

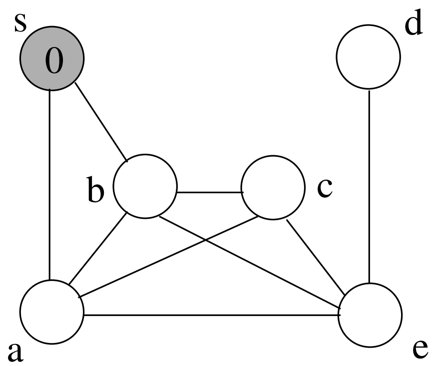
- Gray vertices are stored in a **queue**, Q.
- When the algorithm begins, s is on the queue.
- The algorithm proceeds as follows:
  - Remove the first element, x, from the queue ($x = Dequeue(Q)$).
  - For each neighbor y of x
    - If y is *white*.
      - Color y *gray* ($c(y) = gray$).
      - Let the distance of y be one more than the distance of x ($d(y) = d(x) + 1$).
      - Let x be the predecessor of y ($p(y) = x$).
      - Place y at the end of the queue ($Enqueue(Q, y)$).
    - If y is *gray* or *black*, do nothing.
  - Color x *black*.
  - Repeat until the queue is empty.
- Now let's look at the whole algorithm.

BFS Algorithm

- Let $G = (V, E)$ be a graph, and $s \in V$ be some vertex.
- The breadth-first search algorithm is as follows:

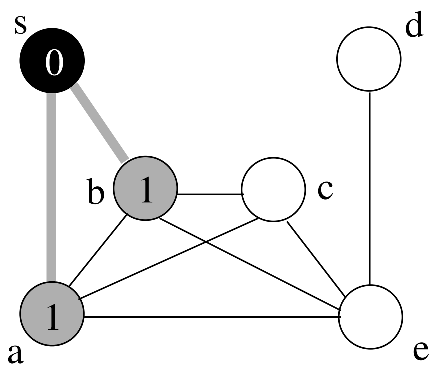
```
BFS(G,s)
   ForAll u in V-{s}
        c(u)=white
        d(u)=Max_Int
        p(u)=NIL
   c(s)=gray
   d(s)=0
   p(s)=NIL
   Enqueue(Q,s)
   while(NotEmpty(Q))
        u=Dequeue(Q)
        ForAll v adjacent to u
             if c(v) = white then
                c(v)=gray
                d(v)=d(u)+1
                p(v)=u
                Enqueue(Q,v)
        color(u) = black
```

BFS Example



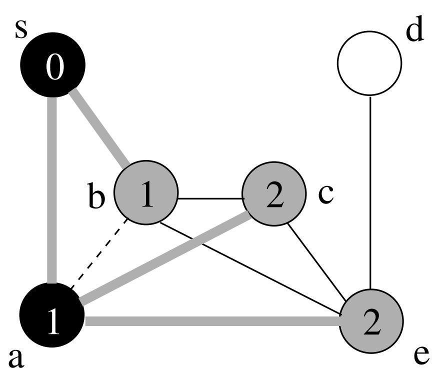
Q=[s]

v	adjacent	d	p	color
s	a,b	0	nil	gray
a	s,b,c,e	inf	nil	white
b	s,a,c,e	inf	nil	white
c	b,a,e	inf	nil	white
d	e	inf	nil	white
e	a,b,c,d	inf	nil	white



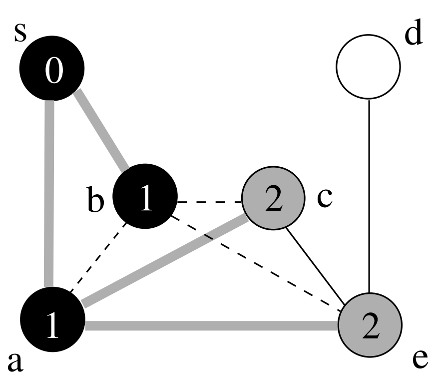
Q=[a,b]

v	adjacent	d	p	color
s	a,b	0	nil	black
a	s,b,c,e	1	s	gray
b	s,a,c,e	1	s	gray
c	b,a,e	inf	nil	white
d	e	inf	nil	white
e	a,b,c,d	inf	nil	white



Q=[b,c,e]

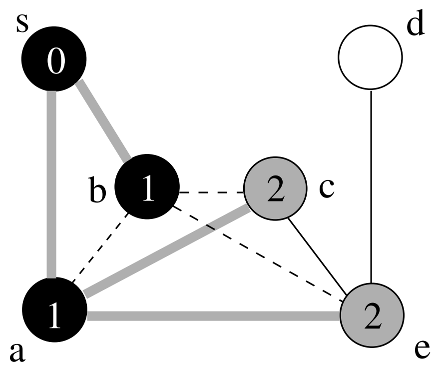
v	adjacent	d	p	color
s	a,b	0	nil	black
a	s,b,c,e	1	s	black
b	s,a,c,e	1	s	gray
c	b,a,e	2	a	gray
d	e	inf	nil	white
e	a,b,c,d	2	a	gray



Q=[c,e]

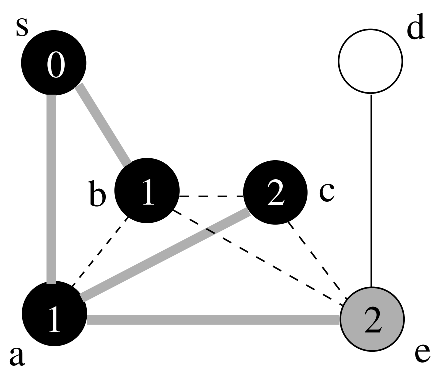
v	adjacent	d	p	color
s	a,b	0	nil	black
a	s,b,c,e	1	s	black
b	s,a,c,e	1	s	black
c	b,a,e	2	a	gray
d	e	inf	nil	white
e	a,b,c,d	2	a	gray

BFS Example Continued



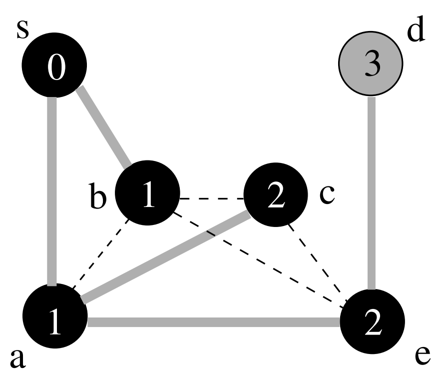
Q=[c,e]

v	adjacent	d	p	color
s	a,b	0	nil	black
a	s,b,c,e	1	s	black
b	s,a,c,e	1	s	black
c	b,a,e	2	a	gray
d	e	inf	nil	white
e	a,b,c,d	2	a	gray



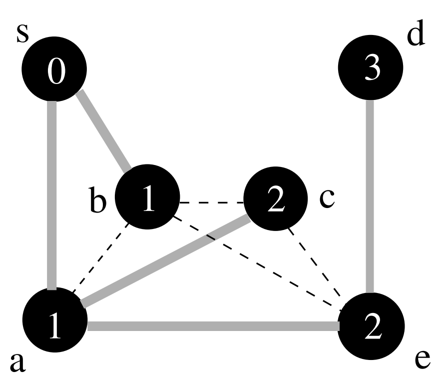
Q=[e]

v	adjacent	d	p	color
s	a,b	0	nil	black
a	s,b,c,e	1	s	black
b	s,a,c,e	1	s	black
c	b,a,e	2	a	black
d	e	inf	nil	white
e	a,b,c,d	2	a	gray



Q=[d]

v	adjacent	d	p	color
s	a,b	0	nil	black
a	s,b,c,e	1	s	black
b	s,a,c,e	1	s	black
c	b,a,e	2	a	black
d	e	3	e	grey
e	a,b,c,d	2	a	black



Q=[]

v	adjacent	d	p	color
s	a,b	0	nil	black
a	s,b,c,e	1	s	black
b	s,a,c,e	1	s	black
c	b,a,e	2	a	black
d	e	3	e	black
e	a,b,c,d	2	a	black

BFS Summary

- There are several things we should ask about BFS:
  - Why does BFS discover every vertex reachable from s?
  - Why does BFS give the distance from s to x, for every reachable vertex x?
  - Why/How does BFS give a shortest path from s to x, for every reachable vertex x?
  - How do we construct a breadth-first tree after we have executed BFS?
  - What is the complexity of BFS?
- It is not hard to convince yourself that the first 3 are true, although proving them is a little harder.
- It is not too difficult to see that the complexity is $O(V + E)$.
- We won't prove any of these things explicitly.

Depth-First Search

- Let $G = (V, E)$ be a graph.
- A depth-first search of G will
  - Find every vertex in G.
  - *Timestamp* every vertex with
    - the first time it was discovered, and
    - the last time it is examined.
  - Construct a **depth-first forest**. That is, a partition of G into **depth-first trees**.
- The timestamps are used by other graph algorithms.
- Depth-first search is so called because it continues along a path until it finds no new vertices, at which point it backtracks.
- Unlike BFS, DFS has no source vertex.
- DFS starts searching from *every* vertex of the graph eventually. We will see what this means.

How DFS Works

- For each node x in the graph we will need to store
  - the list of adjacent vertices,
  - the discover timestamp of x ($d(x)$).
  - the final timestamp of x ($f(x)$).
  - predecessor of x ($p(x)$), and
  - the color of x ($c(x)$).
- The color of a vertex indicates the status of a vertex:
  - A **white** vertex has not been discovered.
  - A **gray** vertex has been discovered, but it may have undiscovered neighbors.
  - A **black** vertex has been discovered, and it has no undiscovered neighbors.
- The algorithm starts with $p(x) = NIL$, and $c(x) = white$ for all $x \in V$.
- Since both $d(x)$ and $f(x)$ are guaranteed to be set in the algorithm, we need not initialize them.

More of How DFS Works

- Unlike BFS, DFS is recursive.
- When DFS "visits" a *white* node $x \in V$, it does the following:
  - Color x gray ($c(x) = gray$).
  - Set discover time of x to the current time ($d(x) = time$).
  - Increment current time ($time = time + 1$).
  - For all y adjacent to x
    - If y is white,
      - Set x to be the predecessor of y ($p(y) = x$).
      - Recursively "visit" node y.
  - Color x black ($c(x) = black$).
  - Set finish time of x to the current time ($f(x) = time$).
  - Increment current time ($time = time + 1$).
- Notice that $time$ has to be a global variable.
- We will now see the algorithm in its entirety.

DFS Algorithm

- Let G be a graph.
- The depth-first algorithm is as follows:

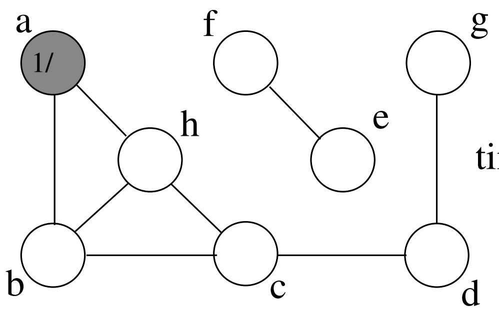
```
DFS(G)
   ForAll u in V
      c(u)=white
      p(u)=NIL
   time=0
   ForAll u in V
     If c(u)=white
         DFS-Visit(u)
```

- *DFS-Visit* is as follows:

```
DFS-Visit(u)
   c(u)=gray
   d(u)=time++
   ForAll v adjacent to u
       If c(v)=white
          p(v)=u
          DFS-Visit(v)
   c(u)=black
   f(u)=time++
```

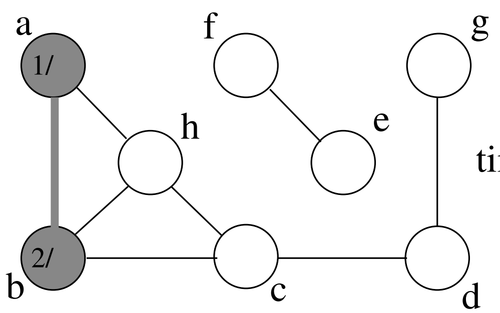
- An example should help make things clearer.

DFS Example



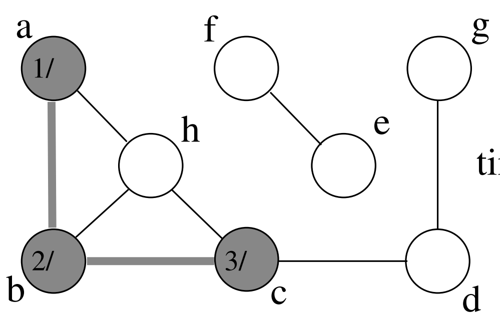
time=1

v	a	b	c	d	e	f	g	h
p	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
d	1							
f								
c	G	W	W	W	W	W	W	W



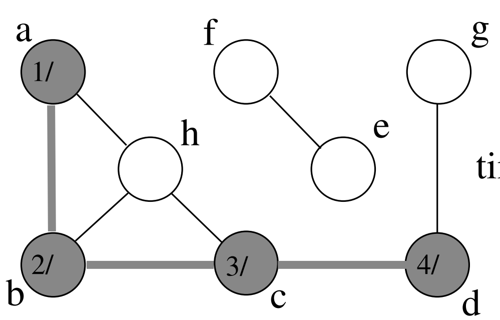
time=2

v	a	b	c	d	e	f	g	h
p	NIL	a	NIL	NIL	NIL	NIL	NIL	NIL
d	1	2						
f								
c	G	G	W	W	W	W	W	W



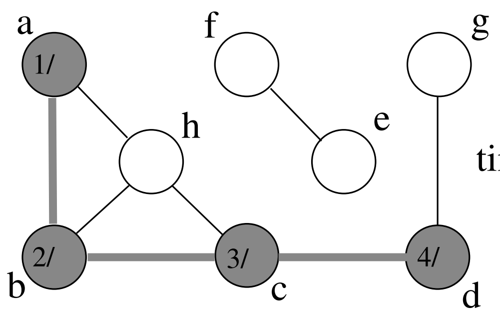
time=3

v	a	b	c	d	e	f	g	h
p	NIL	a	b	NIL	NIL	NIL	NIL	NIL
d	1	2	3					
f								
c	G	G	G	W	W	W	W	W



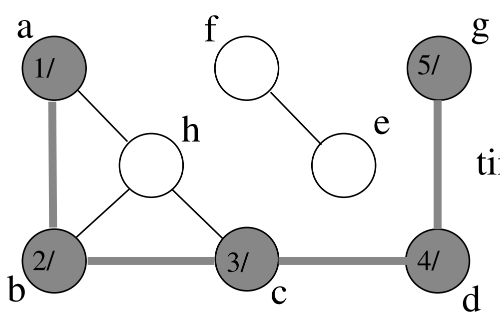
time=4

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	NIL	NIL
d	1	2	3	4				
f								
c	G	G	G	G	W	W	W	W



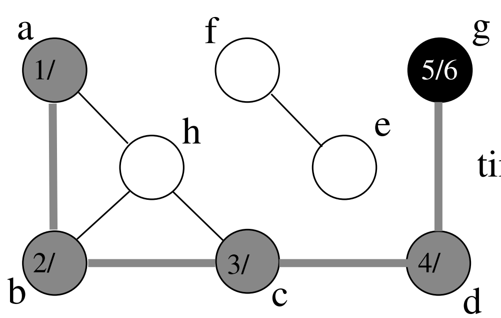
time=4

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	NIL	NIL
d	1	2	3	4				
f								
c	G	G	G	G	W	W	W	W



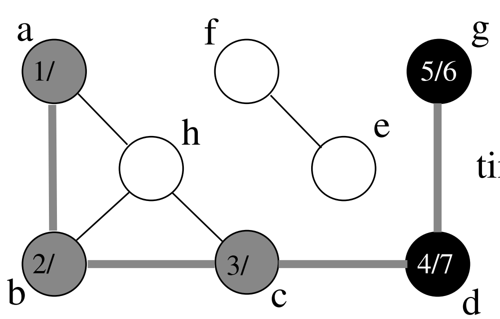
time=5

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	NIL
d	1	2	3	4			5	
f								
c	G	G	G	G	W	W	G	W



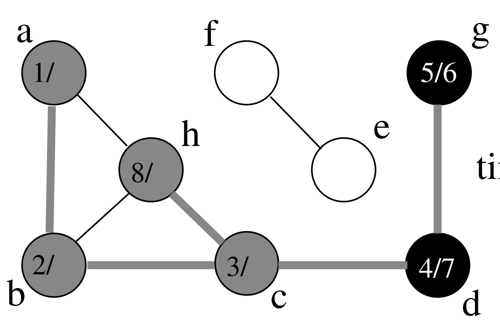
time=6

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	NIL
d	1	2	3	4			5	
f							6	
c	G	G	G	G	W	W	B	W



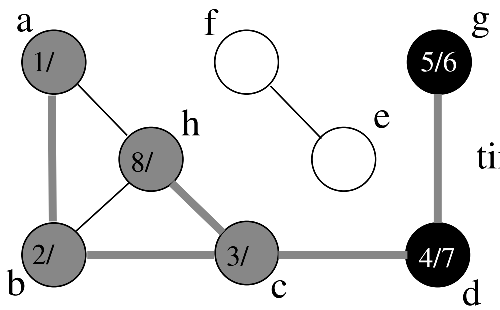
time=7

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	NIL
d	1	2	3	4			5	
f				7			6	
c	G	G	G	B	W	W	B	W



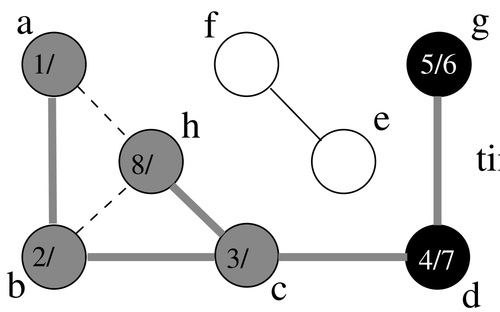
time=8

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4			5	8
f				7			6	
c	G	G	G	B	W	W	B	G



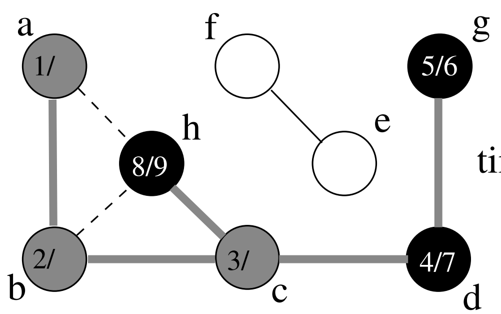
time=8

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4			5	8
f				7			6	
c	G	G	G	B	W	W	B	G



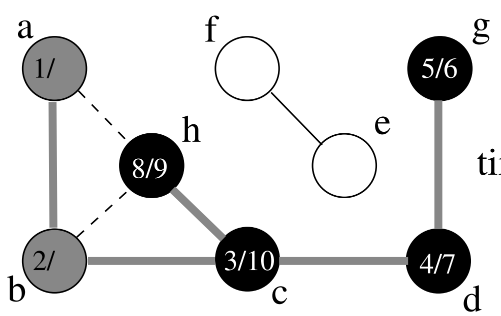
time=8

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4			5	8
f				7			6	
c	G	G	G	B	W	W	B	G



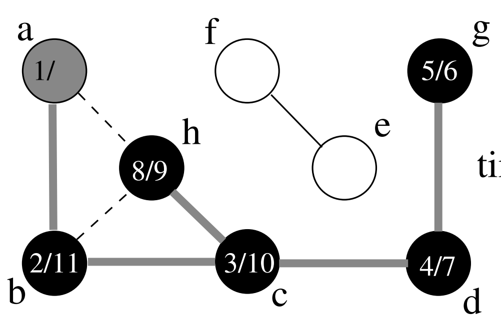
time=9

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4			5	8
f				7			6	9
c	G	G	G	B	W	W	B	B



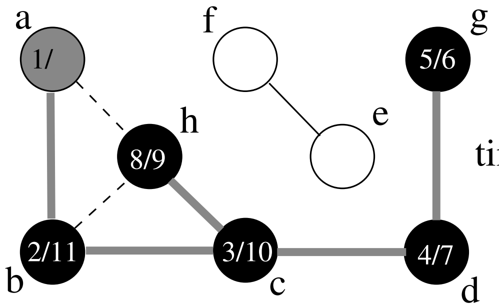
time=10

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4			5	8
f			10	7			6	9
c	G	G	B	B	W	W	B	B



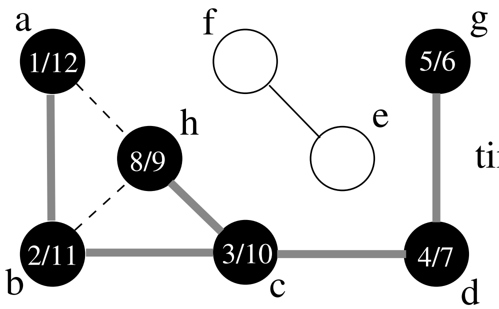
time=11

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4			5	8
f		11	10	7			6	9
c	G	B	B	B	W	W	B	B



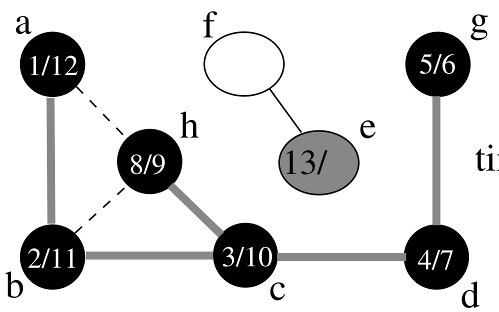
time=11

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4			5	8
f		11	10	7			6	9
c	G	B	B	B	W	W	B	B



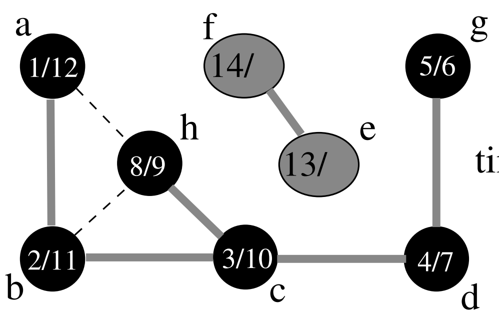
time=12

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4			5	8
f	12	11	10	7			6	9
c	B	B	B	B	W	W	B	B



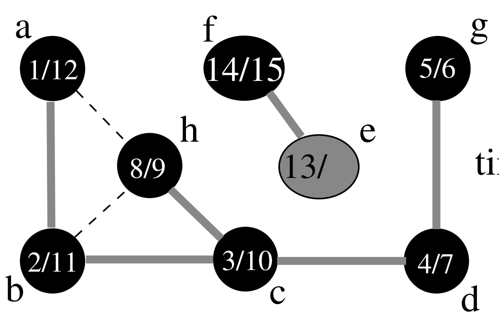
time=13

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	NIL	d	c
d	1	2	3	4	13		5	8
f	12	11	10	7			6	9
c	B	B	B	B	G	W	B	B



time=14

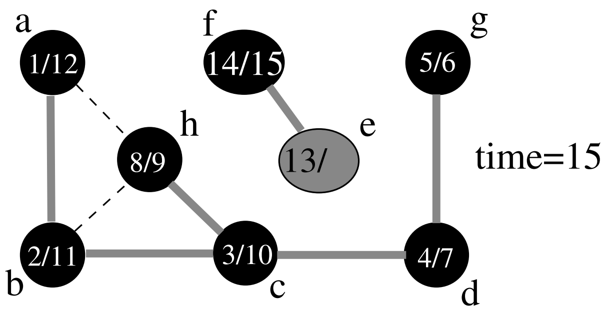
v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	e	d	c
d	1	2	3	4	13	14	5	8
f	12	11	10	7			6	9
c	B	B	B	B	G	G	B	B



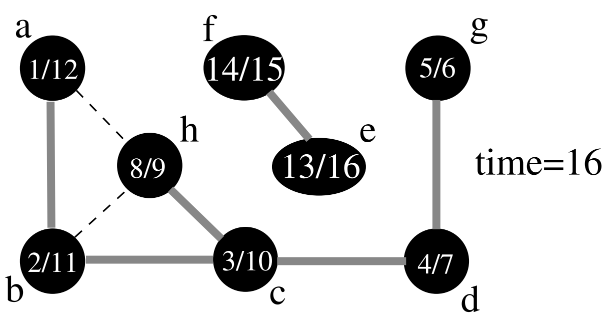
time=15

v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	e	d	c
d	1	2	3	4	13	14	5	8
f	12	11	10	7		15	6	9
c	B	B	B	B	G	B	B	B

DFS Example Finish

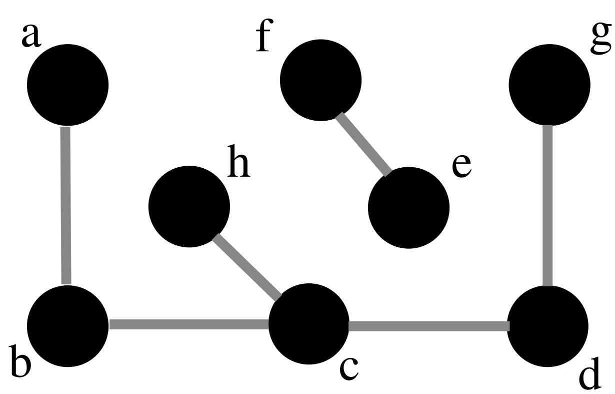


v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	e	d	c
d	1	2	3	4	13	14	5	8
f	12	11	10	7		15	6	9
c	B	B	B	B	G	B	B	B



v	a	b	c	d	e	f	g	h
p	NIL	a	b	c	NIL	e	d	c
d	1	2	3	4	13	14	5	8
f	12	11	10	7	16	15	6	9
c	B	B	B	B	B	B	B	B

- The depth-first forest for this example is

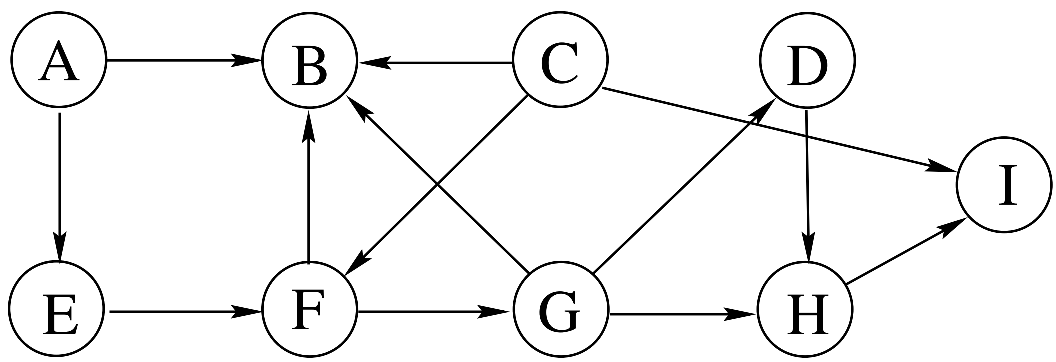


DFS Summary

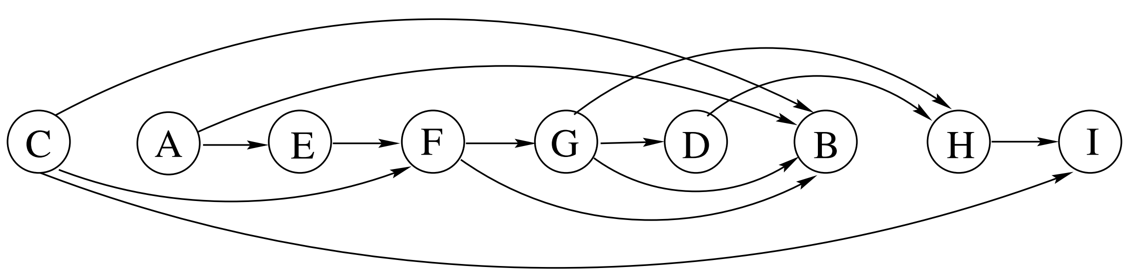
- The running time of DFS is $O(V + E)$.
- The edges $(x, p(x))$ are the edges in the depth-first forest.
- As stated previously, the timestamps are useful in other algorithms.
- There are several other things that can be said about depth-first searching, but we won't.
- Instead, we will discuss one application, the **Topological Sort**.

Topological Sort

- Let $G = (V, E)$ be a directed acyclic graph.
- A **topological sort** of G is a linear ordering of the vertices of G such that if u is before v, then (v, u) is not an edge.
- **Example:** Consider the following Graph:



- Here is one possible topological sort:



- Notice that no edges go from right to left.

Topological Sort Algorithm

- Let G be a directed acyclic graph.
- To topologically sort G, we use a slightly modified version of DFS.
- At the end of *DFS-Visit(u)*, place u at the head of a linked list.
- When DFS is done, the linked list contains the vertices of G topologically sorted.
- **Example:**