PROBLEM SOLVING 6

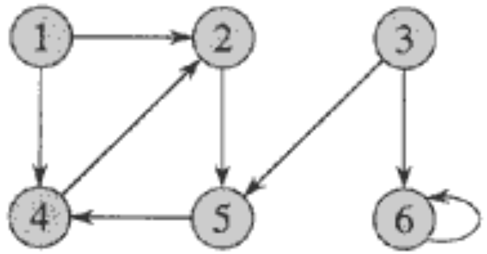
BFS, DFS, Topological Sort

CS340

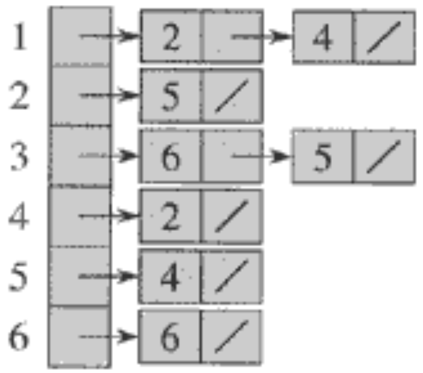
Graphs Interview Questions

- What kind of graph have we already worked with?
- How many edges in a graph?
  - Minimum number of edges for a connected graph?
  - How many edges does a complete graph with n vertices have?
  - Why is the total number of edges only half the sum of the degrees of the vertices?

Representation of a directed graph



(a)



(b)

$$\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}$$

(c)

Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Representation of graphs

- Adjacency list
  - Array *Adj* of |V| lists, one per vertex.
  - Preferred for sparse graphs (why?)
  - Weights and other info can be stored with each vertex
  - Time: to list all vertices adjacent to u: Θ(degree(u))
  - Time: to determine whether (u, v) ∊ E: O(degree(u))
- Adjacency matrix
  - 1 or weight in matrix represents an edge
  - Preferred for dense graphs
  - Preferred if we need to determine quickly if an edge exists (why?)
  - Time: to list all vertices adjacent to u: Θ(V)
  - Time: to determine whether (u, v) ∊ E: Θ(1)

Interview Questions: Graph Data Structure

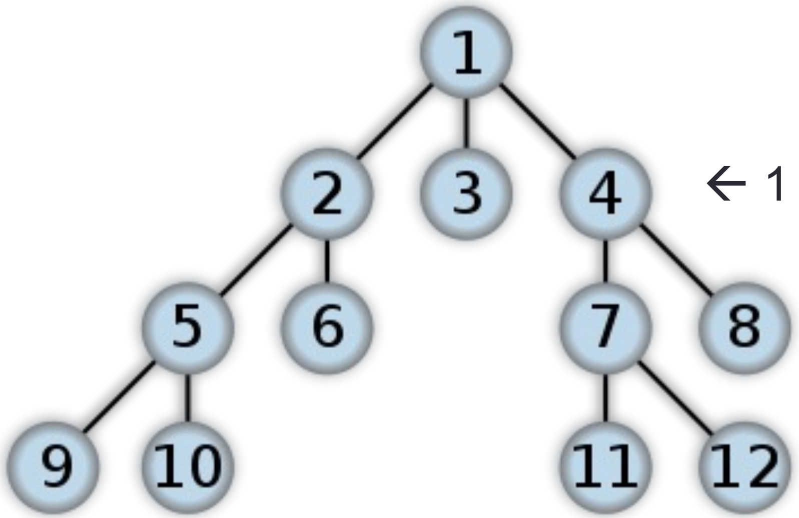
- Suppose that instead of a linked list, each array entry Adj[u] is a hash table containing the vertices v for which (u,v) ∈ E. If all edge lookups are equally likely, what is the expected time to determine whether an edge is in the graph?
- What disadvantages does this scheme have? Suggest an alternate data structure for each edge list that solves these problems.
- Does your alternative have disadvantages compared to the hash table?

Interview question

- On a real-life network, eg Facebook
  - Facebook has V vertices (= around a billion)
  - Each person has O(degree(v)) friends

- What is the magnitude of O(degree(v)) ?
- Does the degree of an individual node v grow as the number of Facebook users V grows?

The "wave" of BFS



← 1 edge from s

← 2 edges from s

← 3 edges from s

Breadth-First Search

- Discovers every vertex that is reachable from the source
- Computes distance (in edges) from source to each reachable vertex
- Can construct a breadth-first tree of reachable vertices
- BFS may not reach all vertices

BFS

- All vertices start out white
- When a vertex is discovered (and put into the queue) it is colored gray. Gray vertices have not yet had their adjacency lists fully examined.
- When a vertex is removed from the queue it is colored black
- The queue consists only of gray vertices

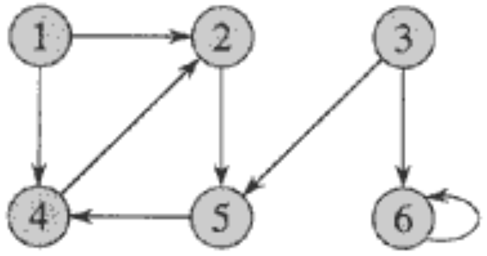
```
BFS($G, s$)
for each vertex $u \in V[G] - \{s\}$
    do $color[u] \leftarrow$ WHITE
       $d[u] \leftarrow \infty$
       $\pi[u] \leftarrow$ NIL
$color[s] \leftarrow$ GRAY
$d[s] \leftarrow 0$
$\pi[s] \leftarrow$ NIL
$Q \leftarrow \emptyset$
ENQUEUE($Q, s$)
while $Q \neq \emptyset$
    do $u \leftarrow$ DEQUEUE($Q$)
       for each $v \in Adj[u]$
           do if $color[v] = $ WHITE
                 then $color[v] \leftarrow$ GRAY
                      $d[v] \leftarrow d[u] + 1$
                      $\pi[v] \leftarrow u$
                      ENQUEUE($Q, v$)
       $color[u] \leftarrow$ BLACK
```

Running time

- Time O(V + E)
- V contribution: Initialization and while loop iterations in which every vertex is enqueued at most once.
- O(V+E): Every vertex u is also dequeued at most once in the main while loop. For each u, the inner for loop iterates through all the *neighbors* v of u.
  - How many *total* times does line 12 execute?
  - Recall: $2E = \sum_{v \in V} degree(v)$

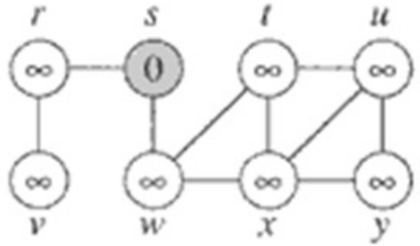
Interview Questions

- Show the d and π values that result from running breadth-first search on figure a, using vertex 3 as the source.
- Show the d and π values that result from running breadth-first search on figure a, using vertex u as the source.



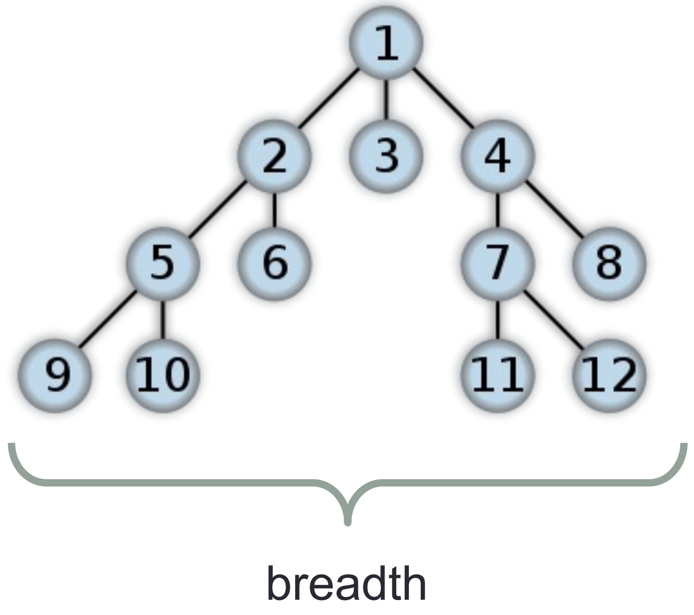
(a)

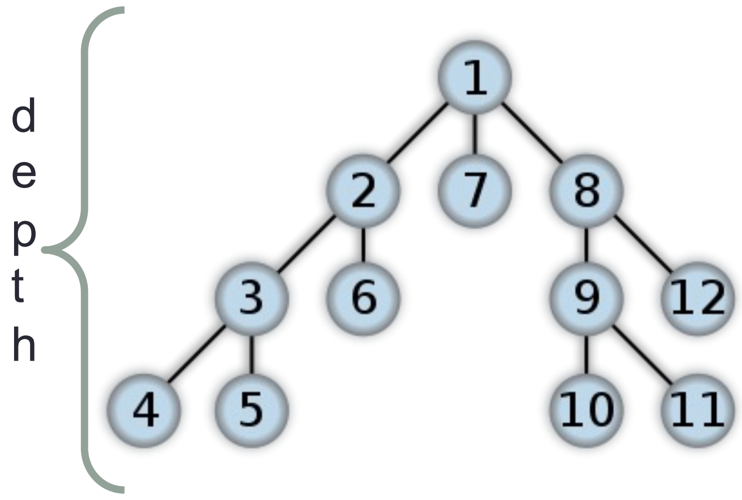
(a)



Depth first search

Search "deeper" whenever possible





*example shows discovery times

Depth first search

- Input: G = (V,E), directed or undirected.
  No source vertex is given!
- Output: **2 timestamps** on each vertex:
  - v.d **discovery time**
  - v.f **finishing time**
- These will be useful for other algorithms later on.
- Can also compute v.π

Depth first search

- Will methodically explore every edge.
  - Start over from different vertices as necessary.
- As soon as we discover a vertex, explore from it.
  - Unlike BFS, which puts a vertex on a queue so that we explore from it later.
- DFS may repeat from multiple source nodes
  - Unlike BFS, result is a forest of DFS trees

DFS, recursive version

```
DFS($G$)
1  for each vertex $u \in G.V$
2      $u.color$ = WHITE
3      $u.\pi$ = NIL
4  $time = 0$
5  for each vertex $u \in G.V$
6      if $u.color$ == WHITE
7          DFS-VISIT($G, u$)

DFS-VISIT($G, u$)
 1  $time = time + 1$              // white vertex $u$ has just been discovered
 2  $u.d = time$
 3  $u.color$ = GRAY
 4  for each $v \in G.Adj[u]$      // explore edge $(u, v)$
 5      if $v.color$ == WHITE
 6          $v.\pi = u$
 7          DFS-VISIT($G, v$)
 8  $u.color$ = BLACK              // blacken $u$; it is finished
 9  $time = time + 1$
10  $u.f = time$
```

Time complexity

- The procedure DFS-VISIT is called exactly once for each vertex v ∈ V , since the vertex u on which DFS-VISIT is invoked must be white and the first thing DFS-VISIT does is paint vertex u gray.
- During an execution of DFS-VISIT, the loop on lines 4–7 executes *Adj*[E] times = Θ(E).
- The running time of DFS is therefore Θ(V + E)
- Notice that BFS was O(V + E) because it was not certain that every vertex would be visited.

Classification of edges

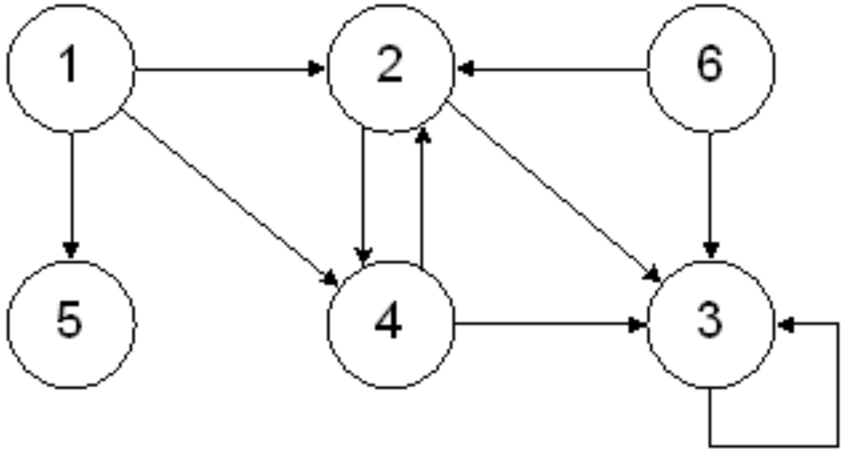
- **Tree** edge: in the depth-first forest. Found by exploring (u, v).
- **Back** edge: (u,v), where u is a descendant of v.
- **Forward** edge: (u,v), where v is a descendant of u, but not a tree edge.
- **Cross** edge: any other edge. Can go between vertices in same depth-first tree or in different depth-first trees.

Classification of edges

- When we first explore an edge (u,v), the color of
- vertex v tells us something about the edge:
- 1. WHITE indicates a tree edge,
- 2. GRAY indicates a back edge, and
- 3. BLACK indicates a forward or cross edge.

Interview Questions

- Show how **depth-first search** works. Assume that vertices are considered in numerical order, and assume that each adjacency list is ordered numerically. Show the discovery and finishing times for each vertex, and show the classification of each edge. Start searching at node 1. TOPOLOGICAL SORT, TOO?



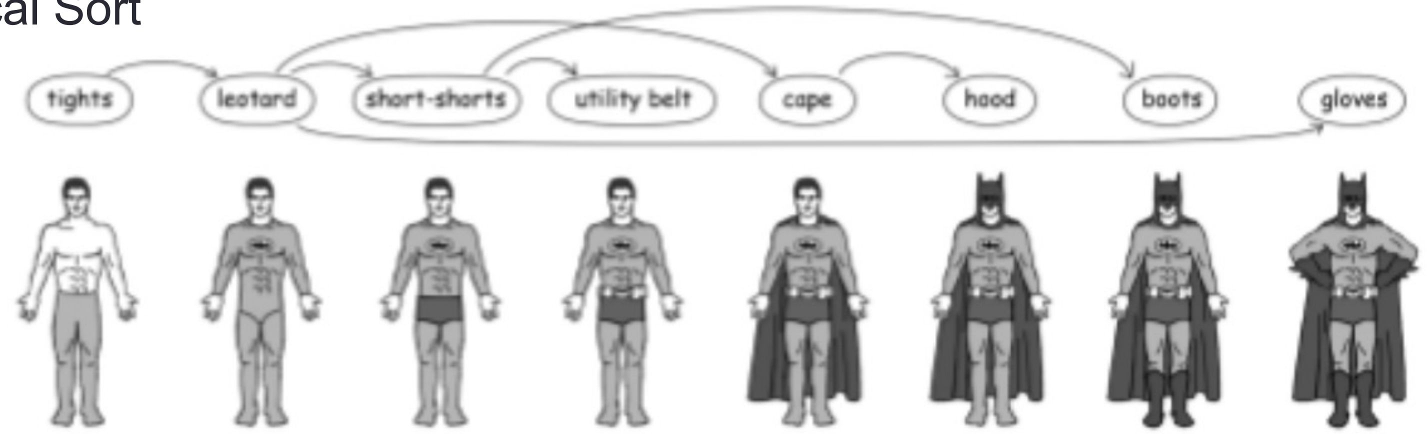
Interview Questions

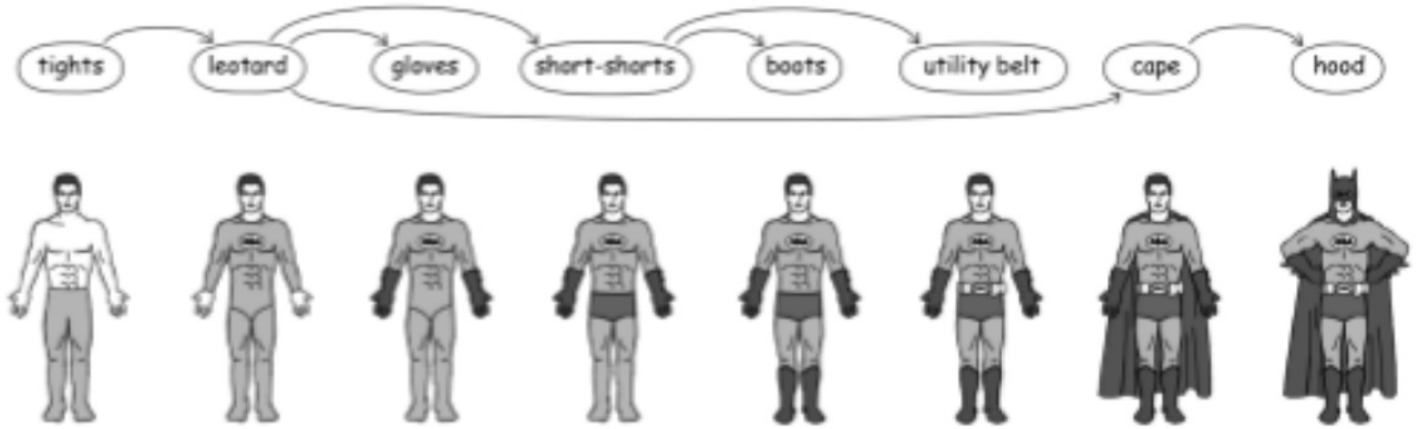
- Let G = (V,E) be a connected, undirected graph. Give an O(V+E)-time algorithm to compute a path in G that traverses each edge in E exactly once in each direction.
- Describe how you can find your way out of a maze if you are given a large supply of pennies. Does a bfs or dfs work better?

The Joys of DFS

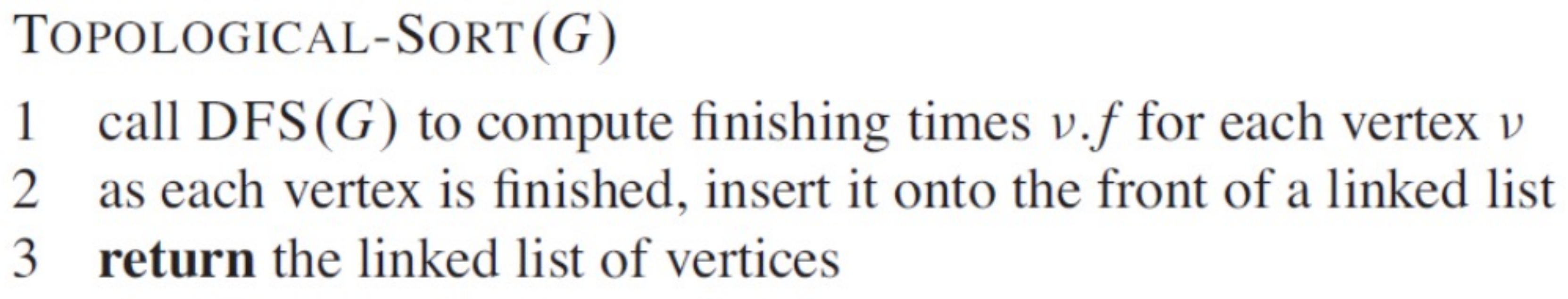
- Depth first search will turn out to be an incredibly useful algorithm. Tell how DFS can help solve the following problems:
- Minimum spanning tree and all pair shortest path tree (when will DFS work?).
- Detecting a cycle in a graph
- Path Finding (find a path between two given vertices u and z)
- Topological Sorting
- Finding Strongly Connected Components
- Solving puzzles with only one solution, such as mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.)

Topological Sort





Topological Sort

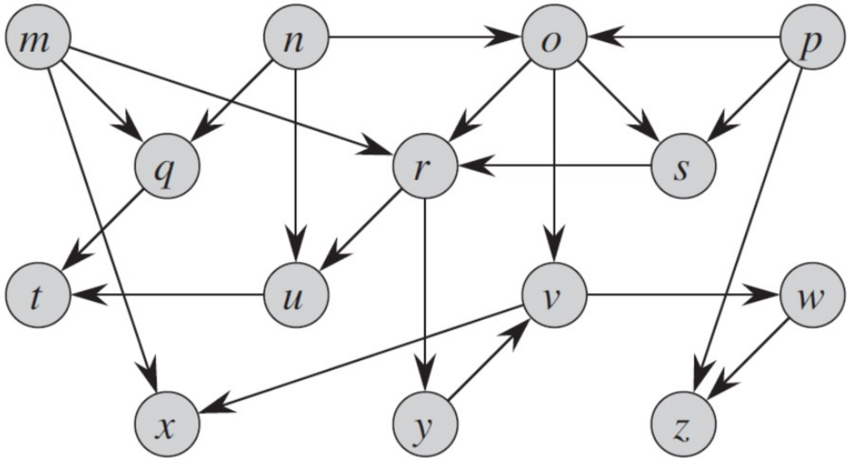


```
TOPOLOGICAL-SORT($G$)
1  call DFS($G$) to compute finishing times $v.f$ for each vertex $v$
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

- What is the time complexity?

Interview Questions

What is the ordering of vertices produced by a topological sort?



Interview Questions

- Give an algorithm that determines whether or not a given undirected graph contains a cycle. Your algorithm should run in O(V) time, independent of |E|.
- Another way to perform topological sorting on a directed acyclic graph is to repeatedly find a vertex of in-degree 0, output it, and remove it and all of its outgoing edges from the graph. Explain how to implement this idea so that it runs in time O(V+E). What happens to this algorithm if G has cycles?