Computational Problems that need graph representations

AUBURN UNIVERSITY

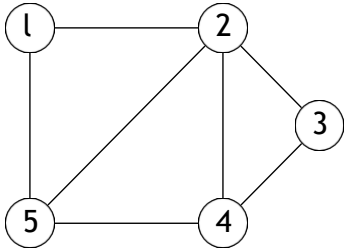
Hugh Kwon

Slides adapted from Dr. Debswapna Bhattacharya's class

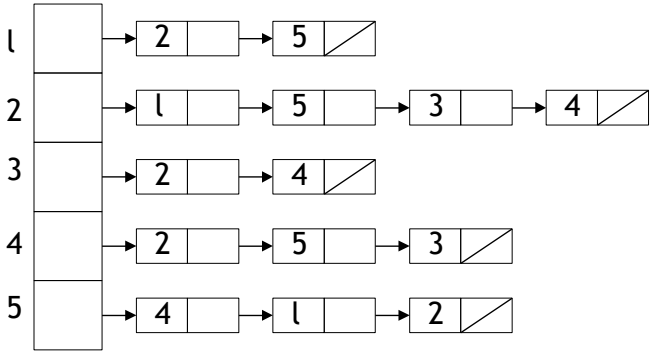
Conventions

- V – set of vertices or nodes
- E – set of edges
- |V| = # of vertices or nodes = n
- |E| = # of edges = m

- types: undirected, directed & weighted graphs
  - what is the max # of edges in a n-node undirected graph?
- adjacency matrix representation
- adjacency-list representation



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



Which representation to use?

- For a graph with n nodes and m edges,
  - the matrix representation needs n^2 array cells
  - the list representation uses n array cells and
    - 2m linked list nodes for an undirected graph or m nodes for a directed graph
- So,
  - the matrix representation is more time-efficient (faster access) than the list representation
  - if there are only O(n) edges (such a graph is called a sparse graph), then the matrix representation is space-inefficient
  - If there are $O(n^2)$ edges (such a graph is called a dense graph), the matrix representation is not space-inefficient

- A tree is a specific kind of undirected graph – one with no cycles.

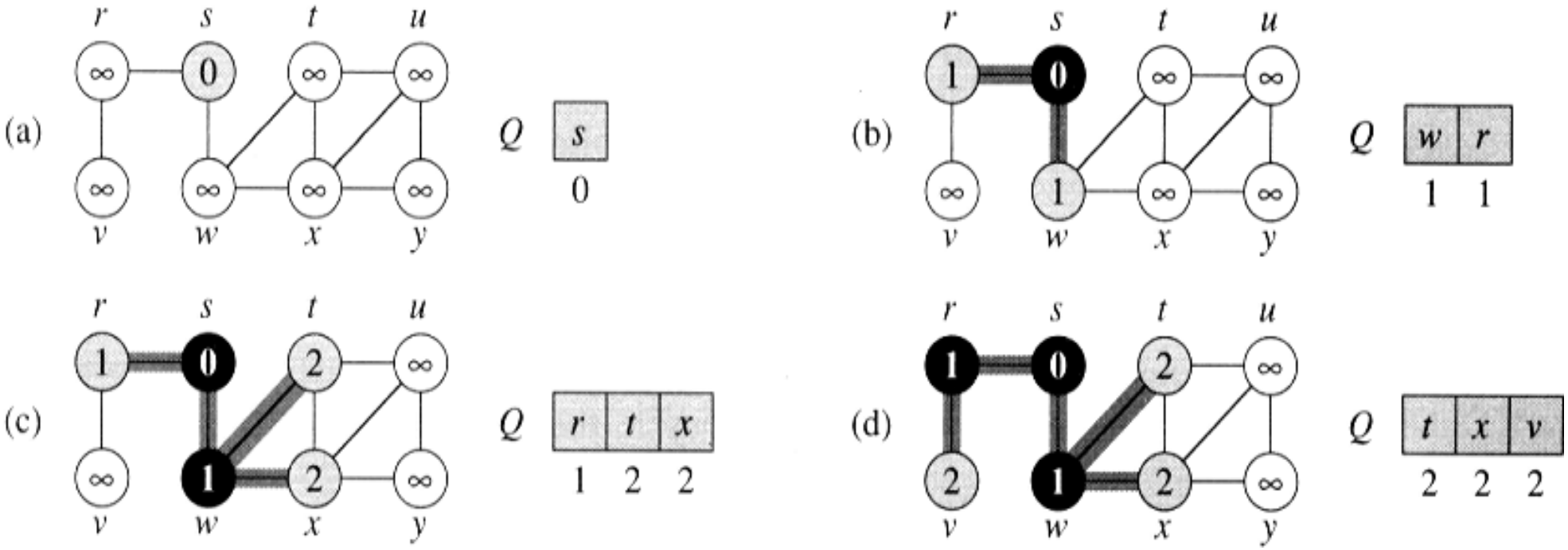
- An algorithm for searching a graph
- Starting from a vertex s, visits every vertex that is reachable from s
- Called breadth-first because it visits all vertices reachable from s in 1 step (by 1 edge) first, then those reachable in 2 steps, and so on.
- Produces a breadth-first tree

Breadth-first search

```
BFS(G, s)
1  for each vertex  u ∈ G.V − {s}
2        u.color = white
3        u.d = ∞
4        u.π = NIL
5  s.color = gray
6  s.d = 0
7  s.π = NIL
8  Q = φ
9  ENQUEUE(Q, s)
10 while  Q ≠ φ
11       u = DEQUEUE(Q)
12       for each  v ∈ G.Adj [u]
13            if  v.color  ==  white
14                 v.color = gray
15                 v.d  =  u.d  + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

Complexity: $O(|V|+|E|) = O(n+m)$

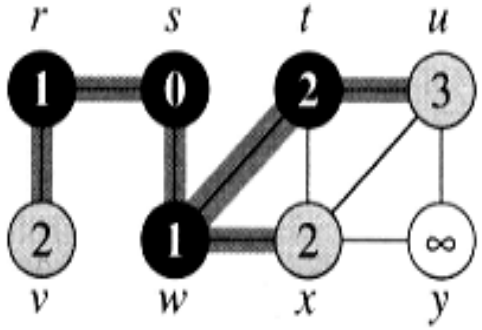
The operation of BFS



Thinking Assignment: Work out and understand this example from the text yourself

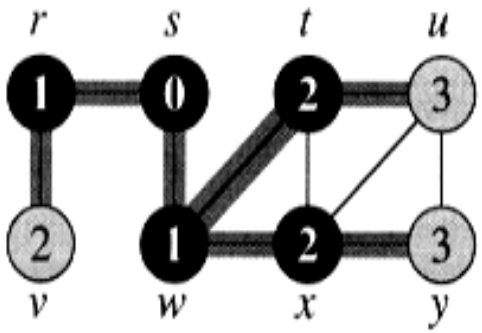
The operation of BFS

(e)



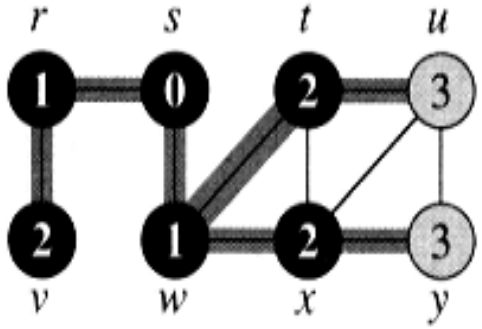
Q | x | v | u
2 2 3

(f)



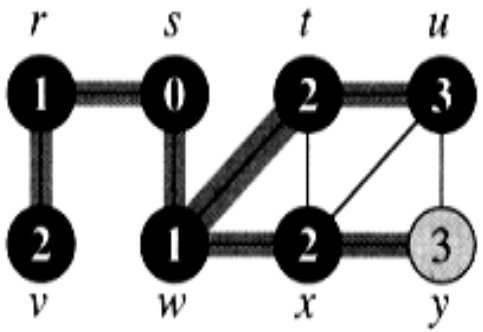
Q | v | u | y
2 3 3

(g)

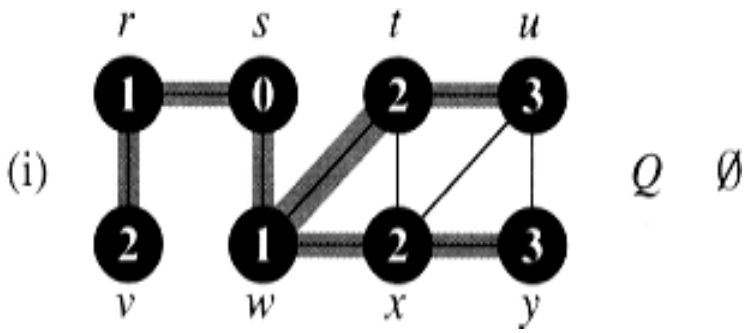


Q | u | y
3 3

(h)



Q | y
3



- Thinking Assignment: Draw the breadth-first tree created by BFS
- Thinking Assignment: What are the distances & shortest paths from s to every other node?

```
PRINT_PATH(G, source, destination )
1  if source == destination
2        print source
3  else if  destination.π == NIL
4        print "no path from" source "to" destination "exists"
5  else PRINT-PATH(G, source, destination.π)
6        print destination
```

Understand and simulate this recursive algorithm by drawing its recursion tree on the previous graph with source=s and destination=y

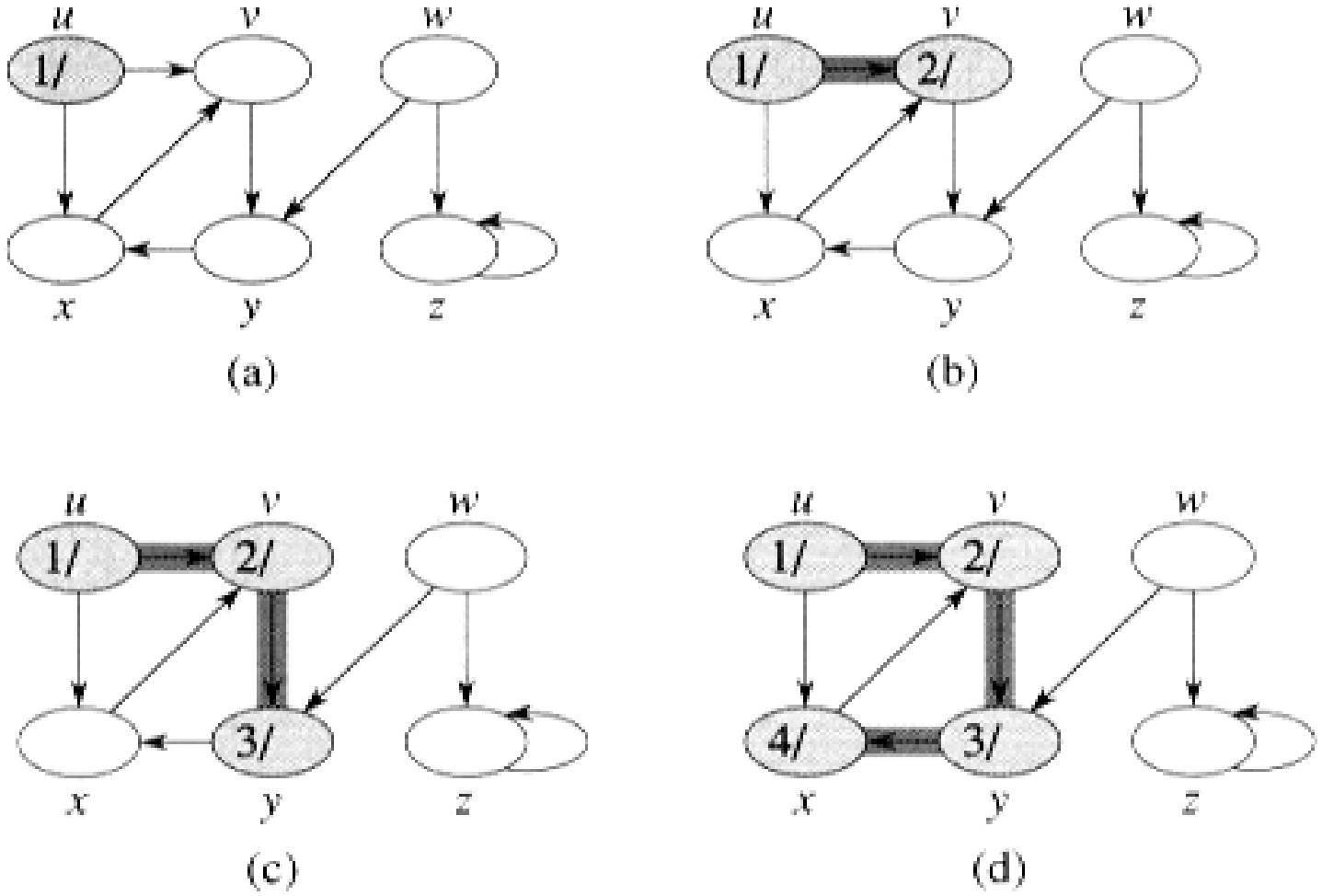
- Another graph search algorithm
- Depth-first: Go as deep as possible from s, then backtrack to find unvisited nodes
- Builds a depth-first forest (several depth-first trees) in the process
- Places two time stamps on each vertex v: v.d the time at which a vertex is “discovered” and v.f when it’s processing is “finished”; Each timestamp is an integer between 1 and 2n

```
DFS(G)
1   for each vertex  u ∈ G.V
2       u.color = white
3       u.π = NIL
4   time = 0
5   for each vertex  u ∈ G.V
6       if  u.color == white
7           DFS-VISIT(G,u)
```

```
DFS-VISIT(G,u)
1.  time = time + 1
2.  u.d = time
3.  u.color = gray
4   for each  v ∈ G.Adj[u]
5       if  v.color == white
6           v.π = u
7           DFS-VISIT(G,v)
8   u.color = black
9   time = time + 1
10  u.f = time
```

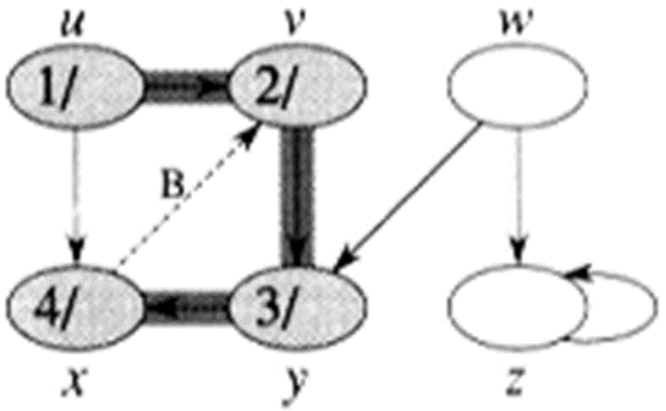
Complexity: $\Theta(|V|+|E|) = \Theta(n+m)$

The operation of DFS

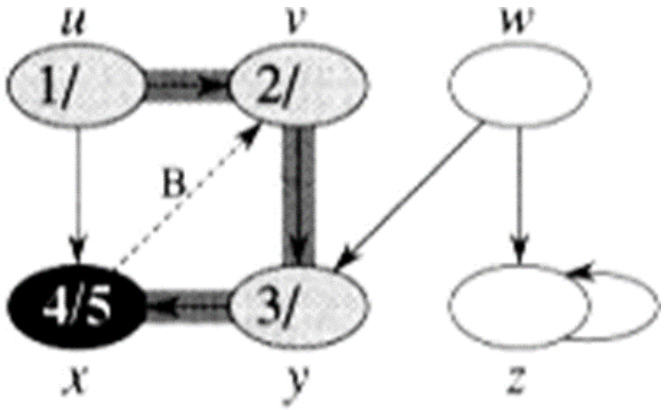


Thinking Assignment: Work out and understand this example from the text yourself

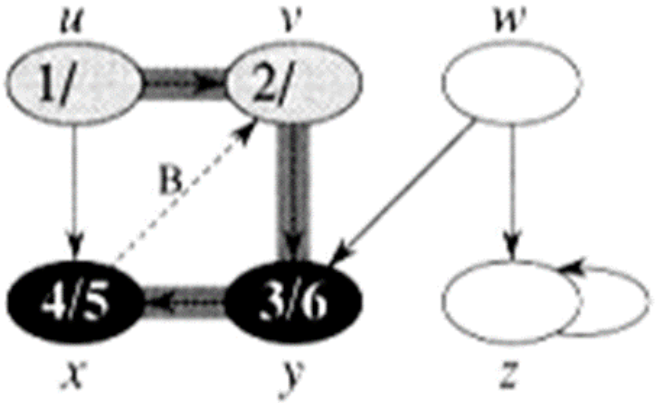
The progress of DFS



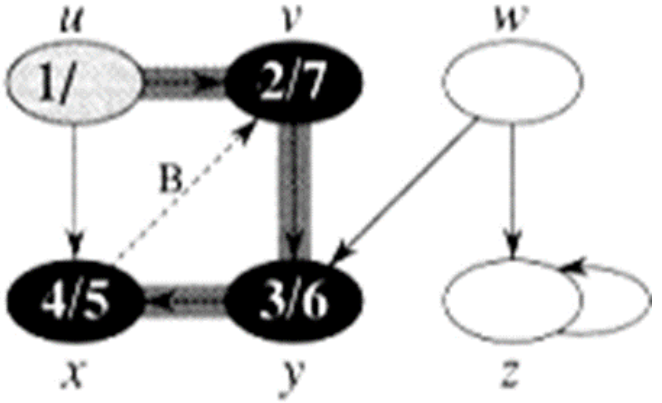
(e)



(f)

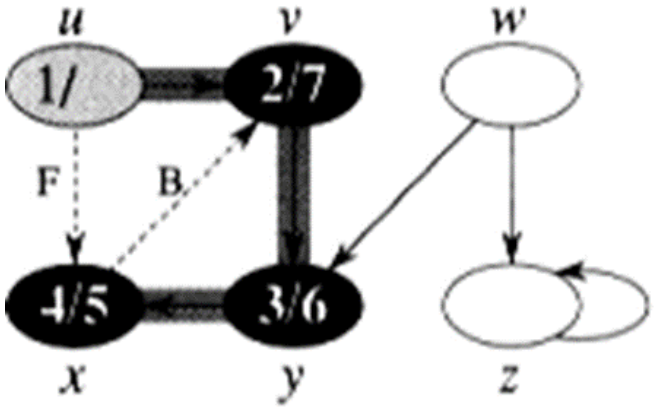


(g)

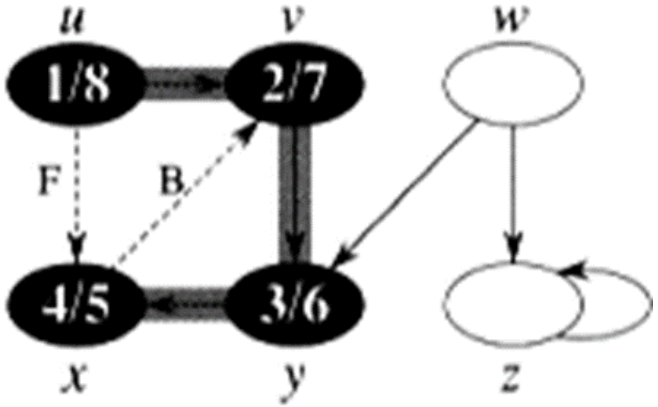


(h)

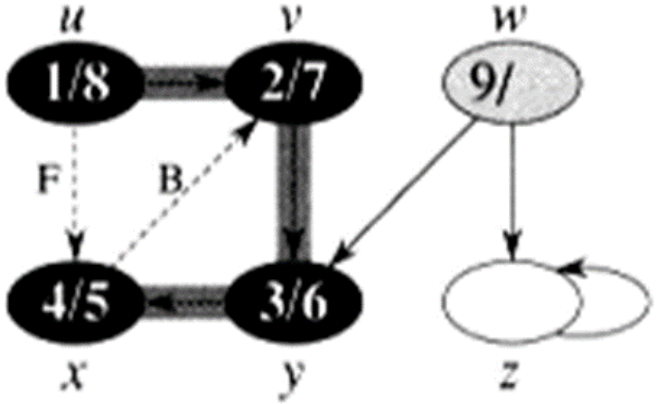
The progress of DFS



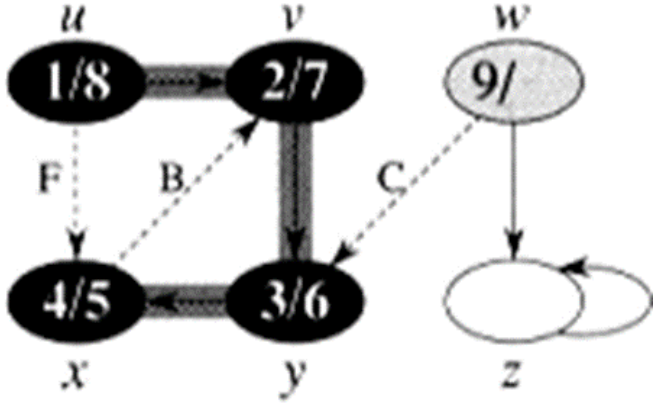
(i)



(j)

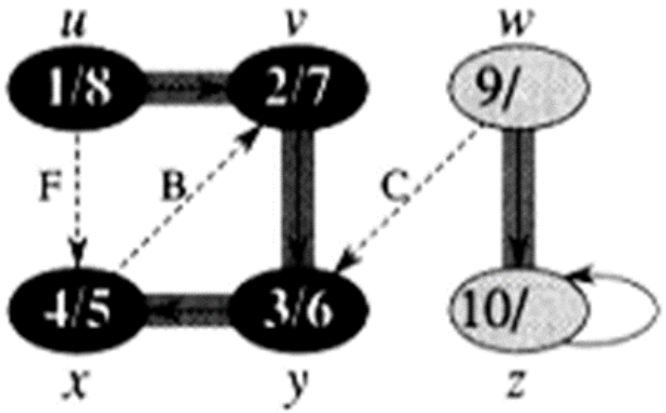


(k)

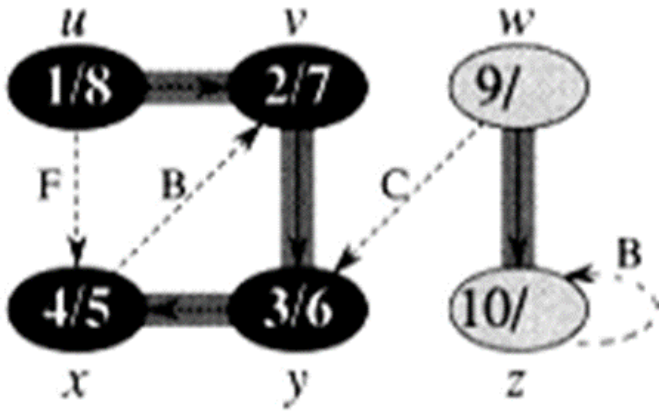


(l)

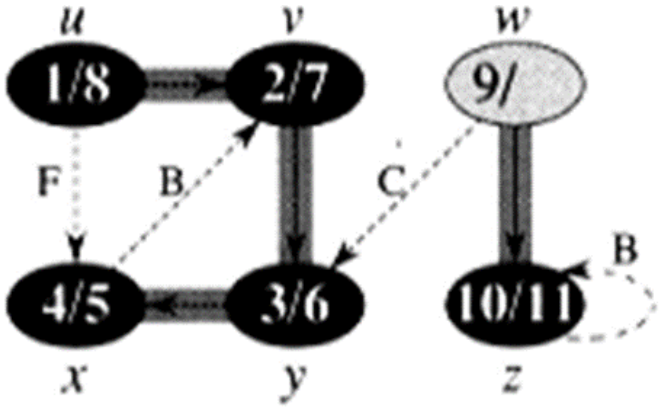
The progress of DFS



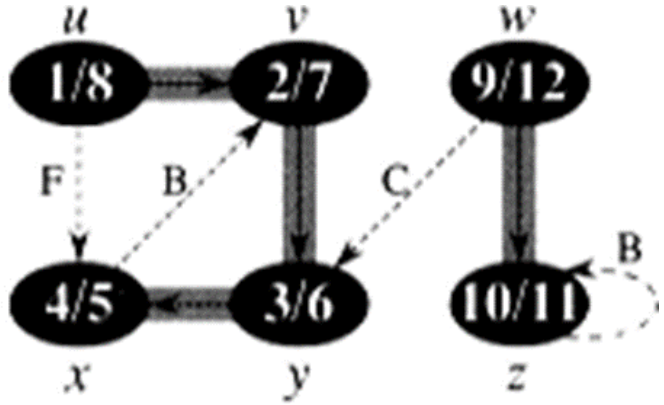
(m)



(n)

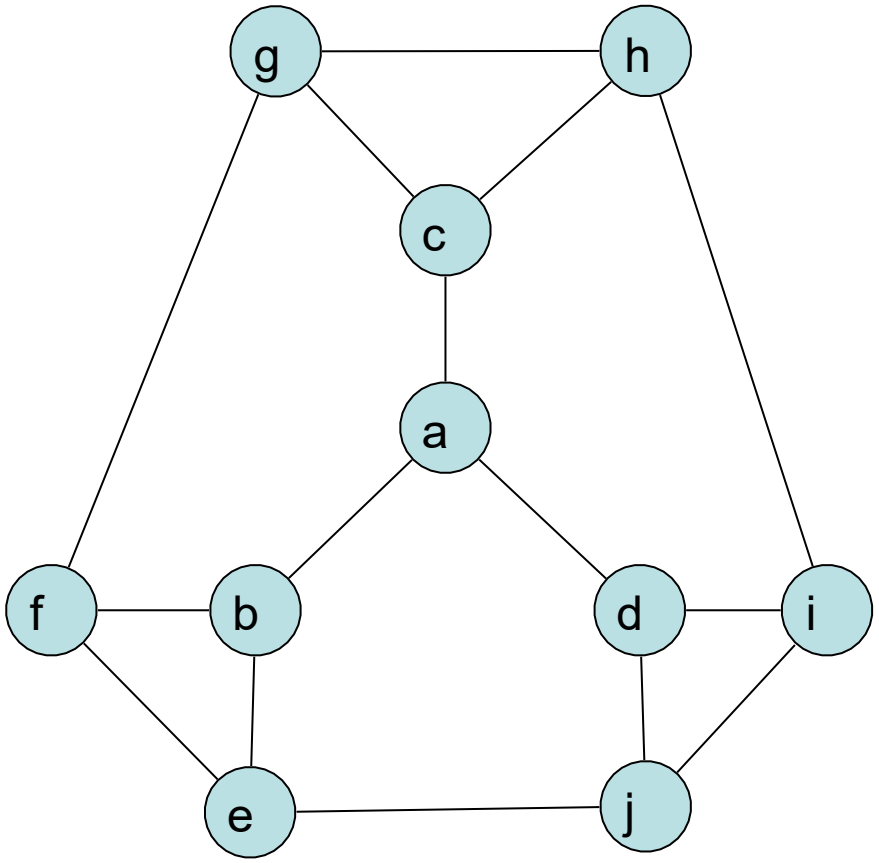


(o)



(p)

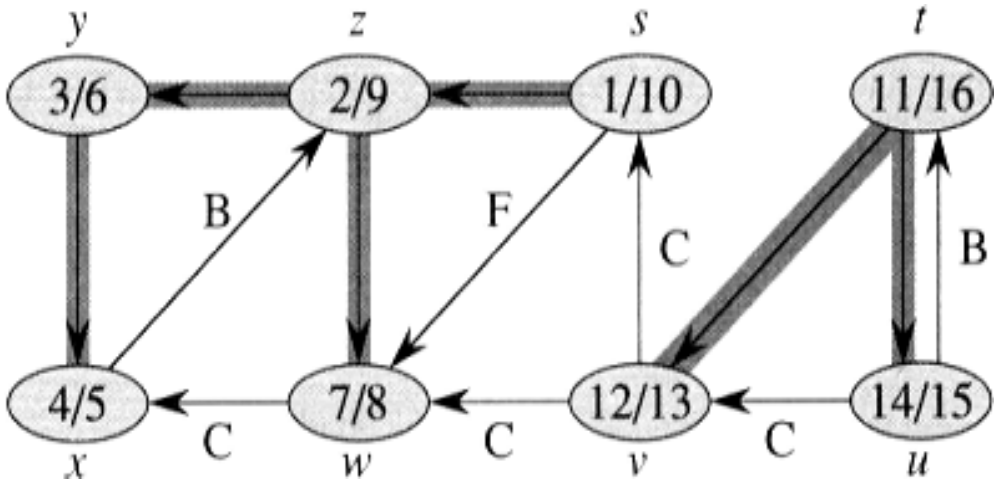
- Depth-first forest consists of trees that represent the edges of the original graph that the DFS algorithm traversed
- These edges are called tree edges
- The remaining edges in the original graph can be classified as back, forward, or cross edges
- DFS of a directed graph may produce back, forward, or cross edges in addition to tree edges
- DFS of an undirected graph will produce only tree and possibly back edges

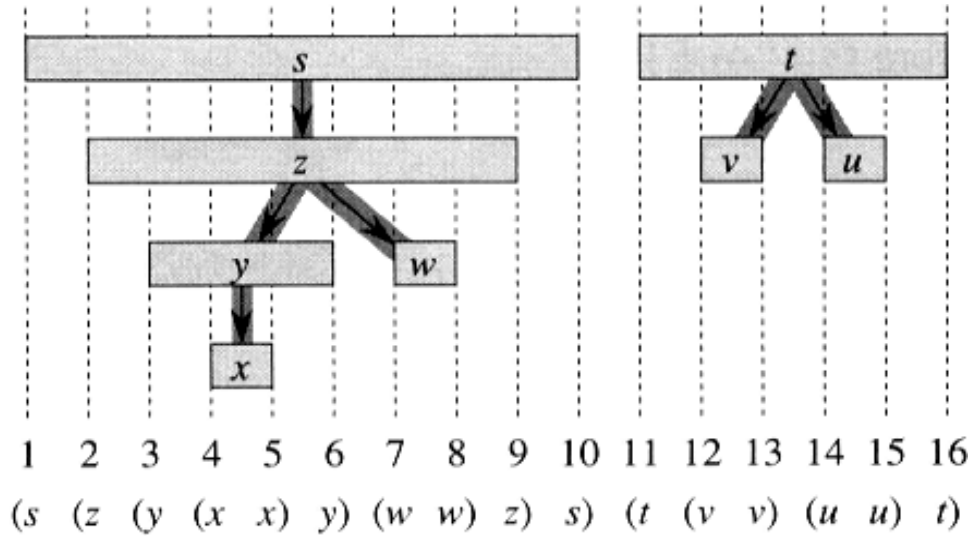


- DFS of an undirected graph will produce only tree and back edges.
  - Verify this statement by simulating DFS on the graph at left with “a” as the starting node.
  - Come up with an explanation as to why this is the case.

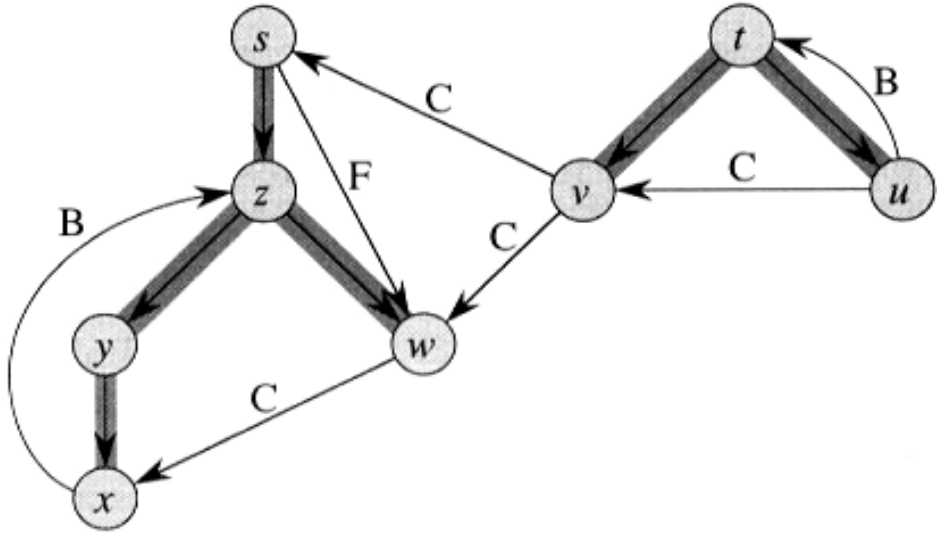
1. Produce a directed graph.
2. Simulate BFS and DFS on it.
3. Make it undirected.
4. Simulate BFS and DFS on it.
5. Compare the results of BFS on a directed graph and on the undirected version of the same graph. Did the algorithm traverse different paths? Produce different trees? Why?
6. Compare the results of DFS on a directed graph and on the undirected version of the same graph. Did the algorithm traverse different paths? Produce different trees? Why?

Example: Graph, its depth-first forest and other edges





Thinking Assignment: Work out and understand this example from the text yourself



- To check if a graph has a cycle or not: a graph is acyclic if and only if the depth-first trees have no back edges.
- To check if the graph has one or more nodes whose removal will split the graph into separate pieces: if there are no such nodes (called articulation points) the graph is said to be biconnected. (thinking assignment: look up this algorithm)
- To sort the nodes of the graph so that constraints modeled by the graph edges are satisfied: topological sort. We will discuss this next.

- A topological sort of a directed acyclic graph $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.

- Thinking Assignment: Why is Topological Sort/Ordering defined only for an acyclic graph? What if the graph has a cycle?

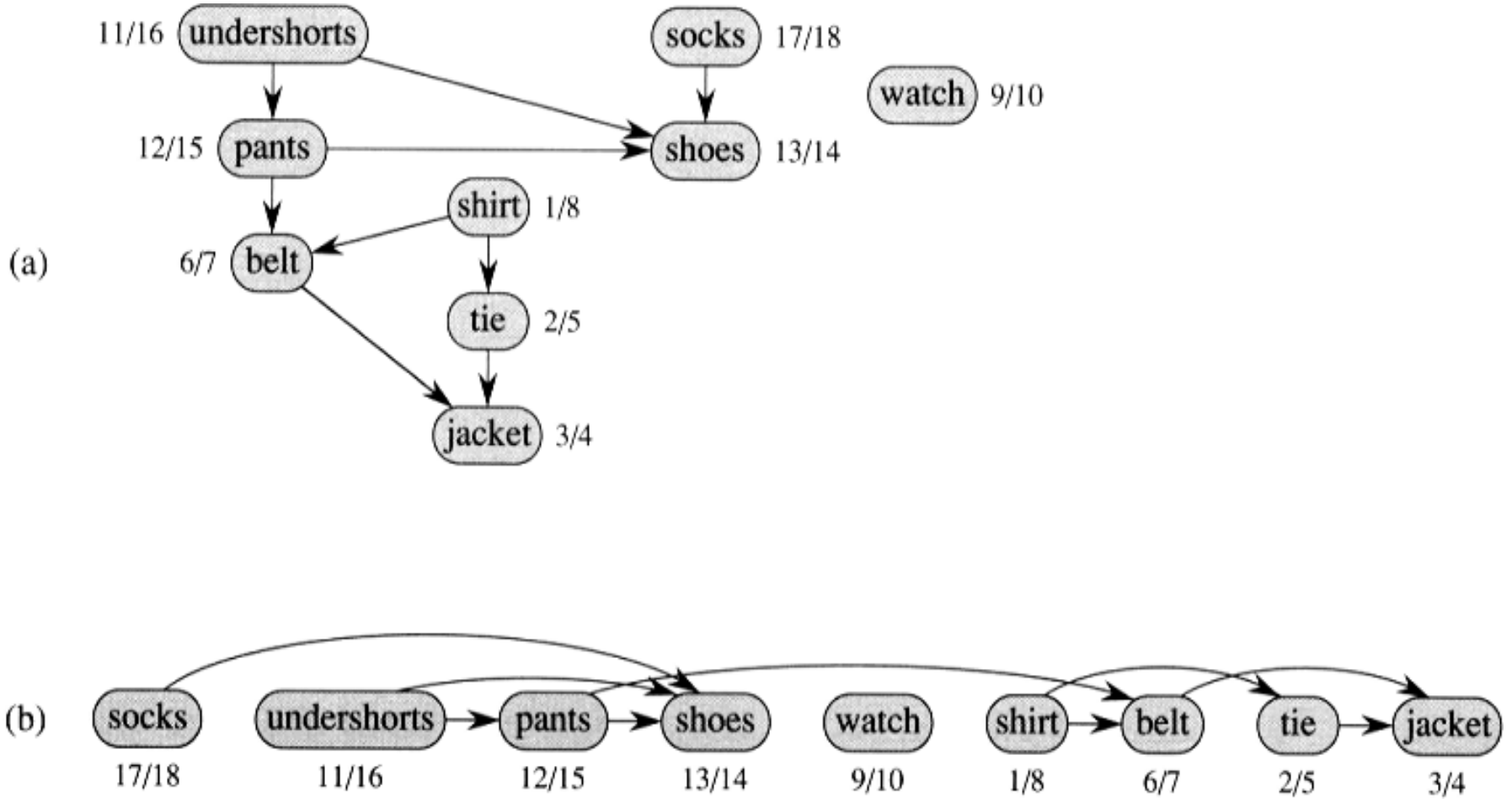
TOPOLOGICAL_SORT(*G*)

1 call DFS(*G*) to compute finishing time *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

Complexity: $\Theta(|V|+|E|) = \Theta(n+m)$


(a)
11/16 undershorts
socks 17/18
watch 9/10
12/15 pants
shoes 13/14
shirt 1/8
6/7 belt
tie 2/5
jacket 3/4
(b)
socks
undershorts
pants
shoes
watch
shirt
belt
tie
jacket
17/18
11/16
12/15
13/14
9/10
1/8
6/7
2/5
3/4

- Obtain and simulate the TS algorithm on the CSSE CS or SE major course prerequisite graph. What order of courses does it produce? Are there other correct orders of taking these courses? What changes to the DFS algorithm need to be made so that it produces a different topological order?

- How/Why does placing graph nodes in the reverse order of finishing times (i.e., the node to finish first ends up at the end of the topological order and the node to finish last ends up at the beginning of the order) ensure that the Topological Order property is not violated?

- Come up with a different Topological Algorithm that implements the following strategy:
  - In-degree of a node is the # of edges coming into it; if a node has in- degree zero, then it means that it is not dependent on any other nodes;
    1. therefore, determine the in-degree of all nodes in the graph and store those;
    2. if there is at least one such node (what does it mean if there are no such nodes?), output that as the first node in TO;
    3. then remove all outgoing edges from it by decrementing the in-degrees of all nodes connected by those outgoing edges;
    4. if that makes the in-degree of any node to be zero, output that as the next node in TO (what does it mean if there are no such nodes?);
    5. repeat steps 2-4 until all nodes have been output to the TO

Chapter 22 Elementary Graph Algorithms

- Read sections 22.1-22.4 (omit theorems, corollaries, lemmas and their proofs, but learn the results stated in the slides)
- Omit section 22.5

- Problems 22.1-1 : 22.1-8
- Problems 22.2-1 : 22.1-7 and 22.2-9
- Problems 22.3-1, 22.3-2, 22.3-4, 22.3-6 : 22.3-11
- Problems 22.4-1 : 22.4-5

AUBURN UNIVERSITY