

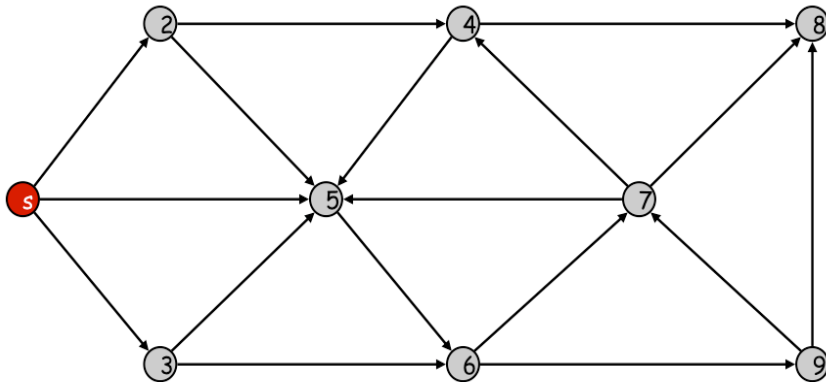
Depth First Search (DFS) + Topological Sort

Instructor: Krishna Venkatasubramanian

CSC 212

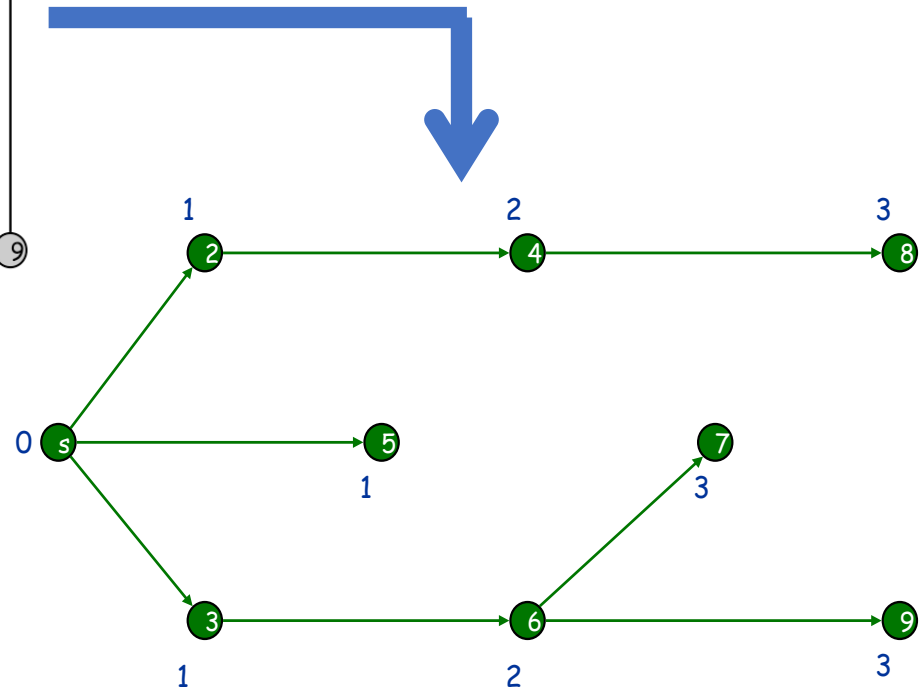
We Already Covered Breadth First Search(BFS)

- Traverses the graph one level at a time
 - Visit all outgoing edges from a node before you go deeper
- Needs a queue



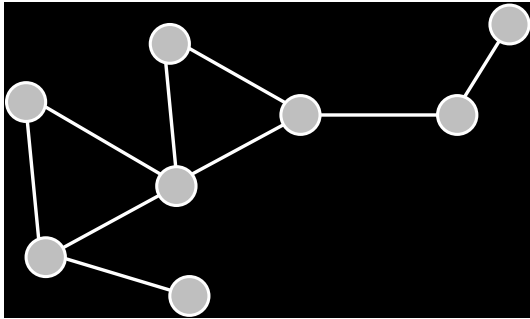
Time Complexity $O(V+E)$

BFS creates a tree called BFS-Tree

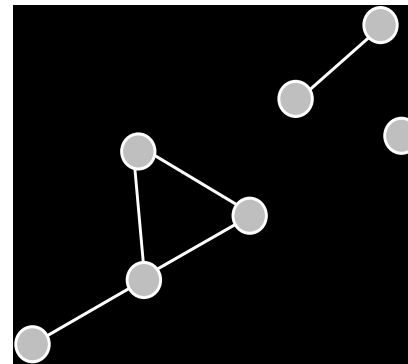


Connected Undirected Graph

- **$G = (V, E)$ is called connected iff**
 - From any node u , we can reach all other nodes



Connected graph



Un-Connected graph

- ***What is the time complexity to decide if G is a connected graph?***
 - Take any node from G and apply BFS
 - If you reached all nodes $\rightarrow G$ is connected
 - Otherwise $\rightarrow G$ is not connected

Time Complexity $O(V+E)$

Depth First Search (DFS)

- Traverse the graph by going deeper whenever possible
- DFS uses a stack, hence can be implemented using recursion
- While traversing keep some useful information
 - **u.color:**
 - White → u has not been visited yet
 - Gray → u is visited but its descendent are not completed yet
 - Black → u and its all descendent are visited
 - **u.startTime (u.d)** = the first time u is visited
 - **u.endTime (u.f)** = the last time u will be seen (after all descendent of u are processed)

DFS: Pseudocode

U is just discovered...make it gray

DFS-VISIT(u)

```
1  color[u] ← GRAY    ▷ White vertex u has just been discovered.
2  time ← time + 1
3  d[u] = time         ← U start time
4  for each v in Adj[u] ▷ Explore edge(u, v).
5      do if color[v] = WHITE
6          then  $\pi[v] \leftarrow u$ 
7              DFS-VISIT(v)
8  color[u] = BLACK    ▷ Blacken u; it is finished.
9  f[u] ← time + 1
```

Time is a global variable

If v is not seen before, recursively visit v

DFS(G)

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

Notice: We maintain 4 arrays during the traversal

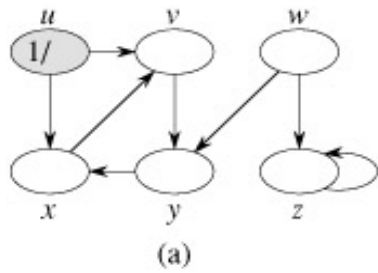
$d[u] \rightarrow$ First time u is seen

$f[u] \rightarrow$ Last time u is seen

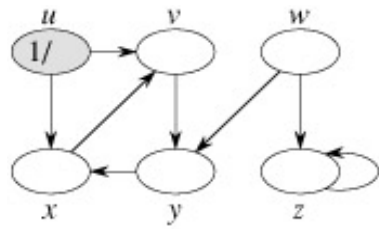
$color[u] \rightarrow \{White, Gray, Black\}$

$\pi[u] \rightarrow$ parent of node u

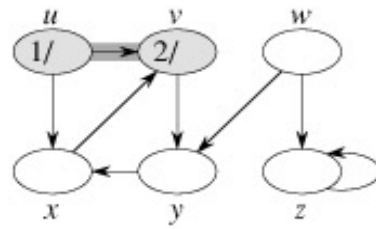
Depth First Search (DFS): Example



Depth First Search (DFS): Example

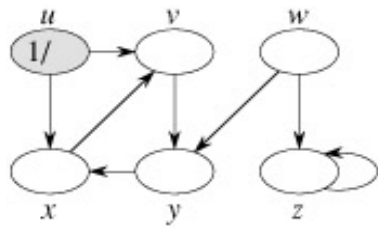


(a)

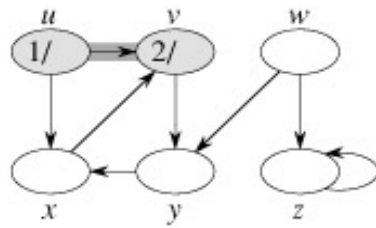


(b)

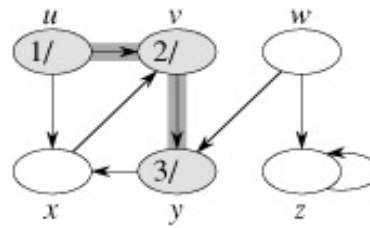
Depth First Search (DFS): Example



(a)

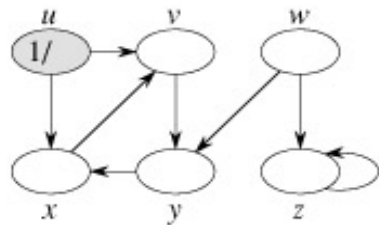


(b)

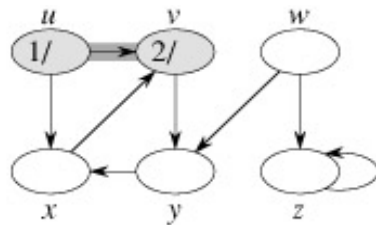


(c)

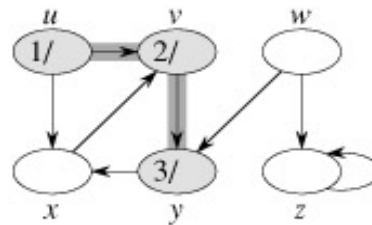
Depth First Search (DFS): Example



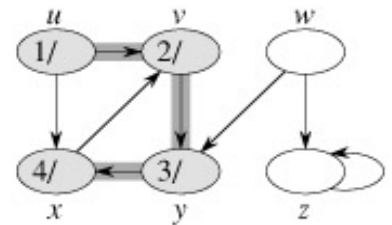
(a)



(b)

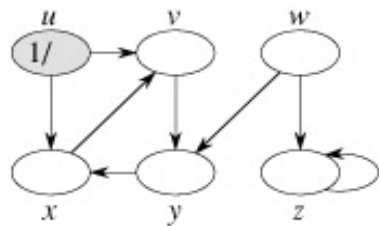


(c)

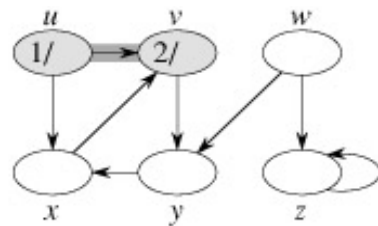


(d)

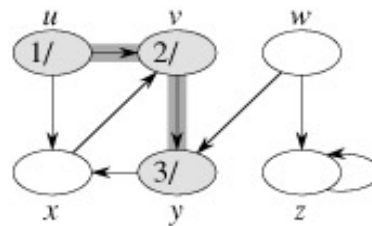
Depth First Search (DFS): Example



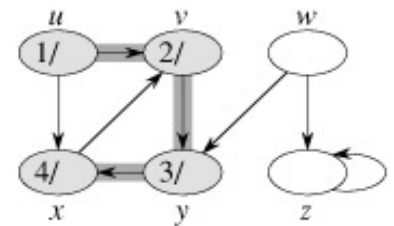
(a)



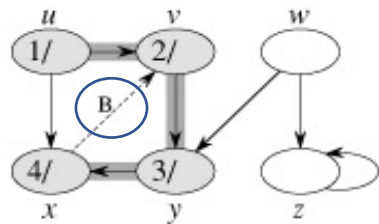
(b)



(c)

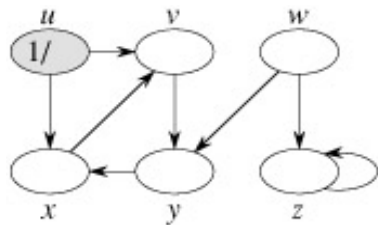


(d)

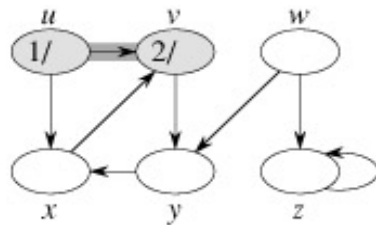


(e)

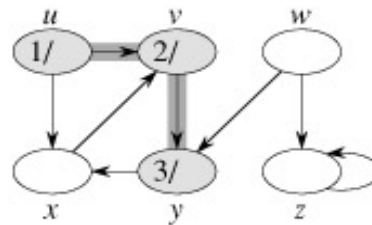
Depth First Search (DFS): Example



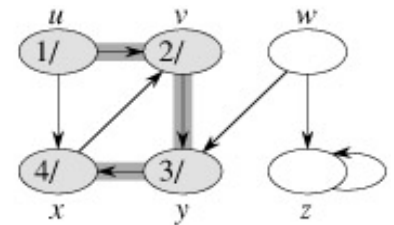
(a)



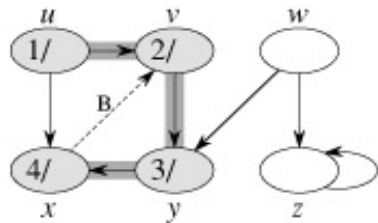
(b)



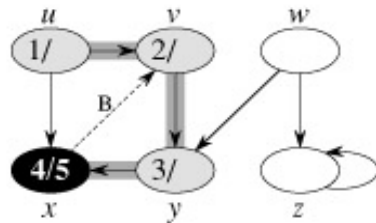
(c)



(d)

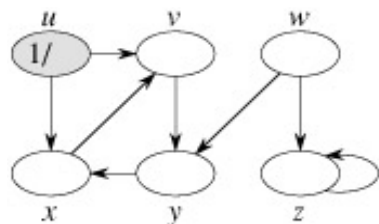


(e)

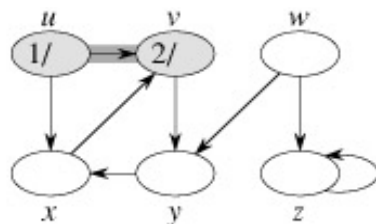


(f)

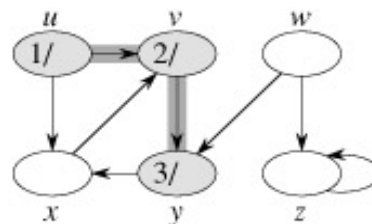
Depth First Search (DFS): Example



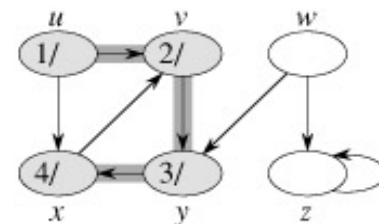
(a)



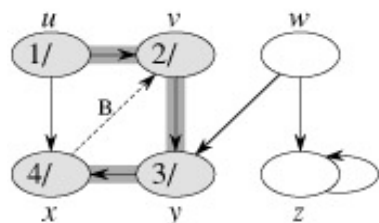
(b)



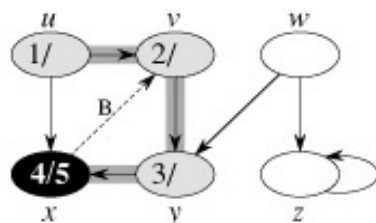
(c)



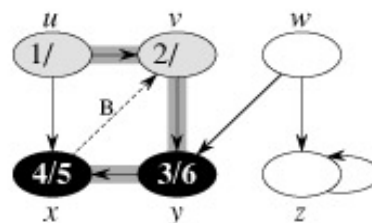
(d)



(e)

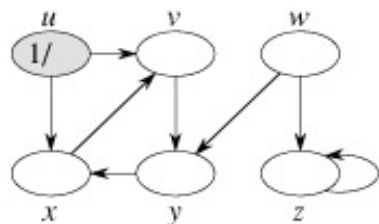


(f)

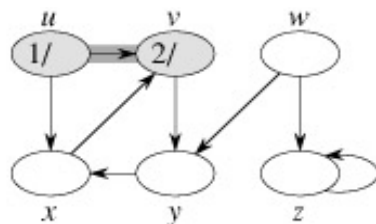


(g)

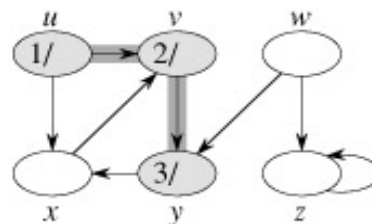
Depth First Search (DFS): Example



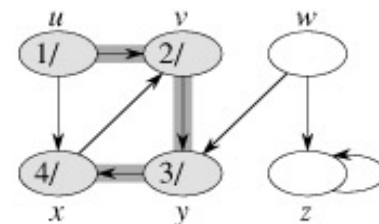
(a)



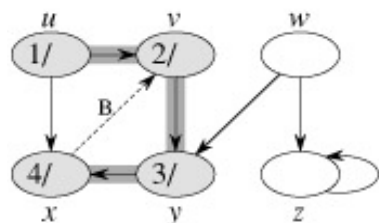
(b)



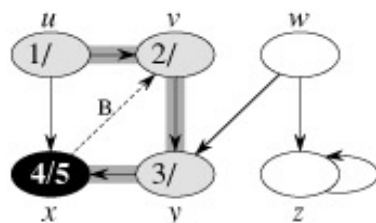
(c)



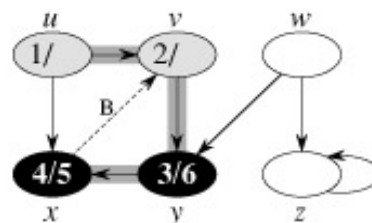
(d)



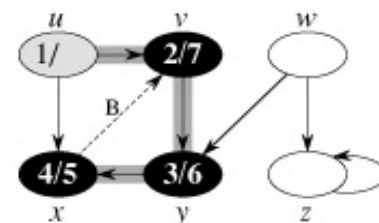
(e)



(f)

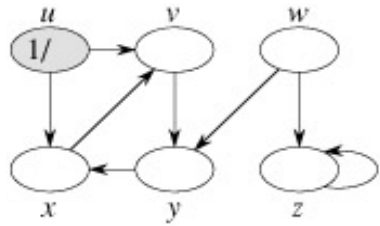


(g)

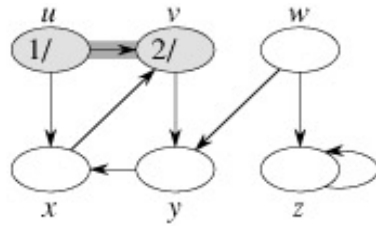


(h)

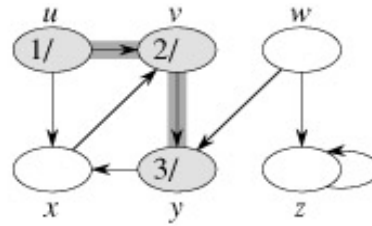
Depth First Search (DFS): Example



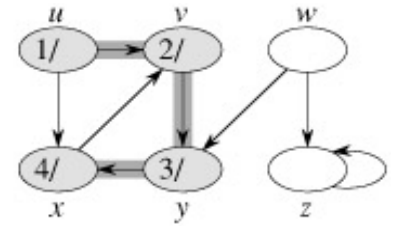
(a)



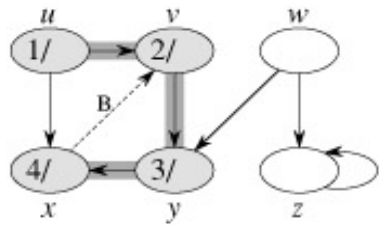
(b)



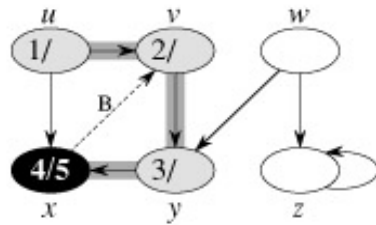
(c)



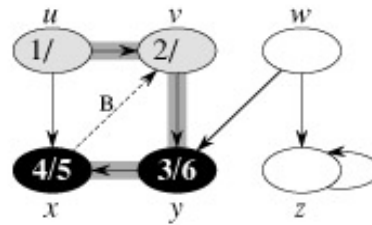
(d)



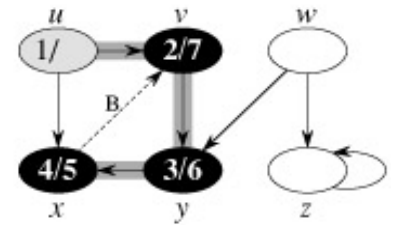
(e)



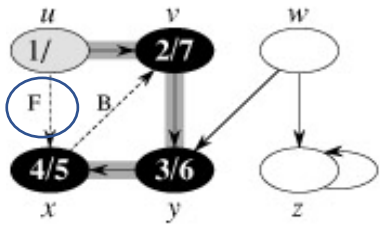
(f)



(g)

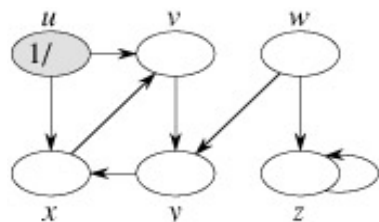


(h)

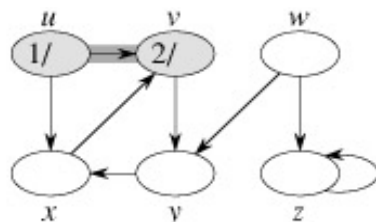


(i)

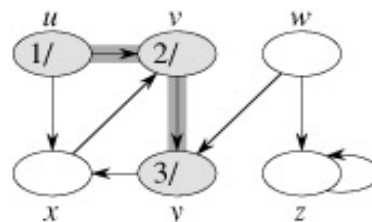
Depth First Search (DFS): Example



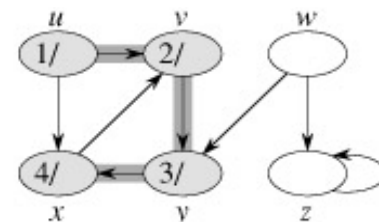
(a)



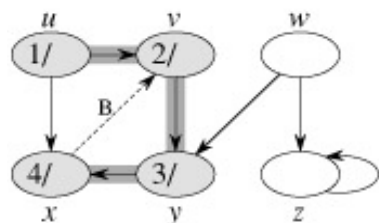
(b)



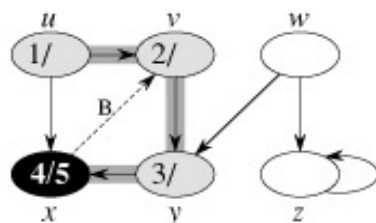
(c)



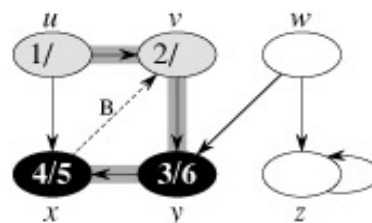
(d)



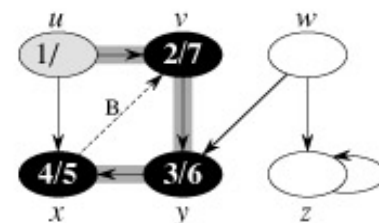
(e)



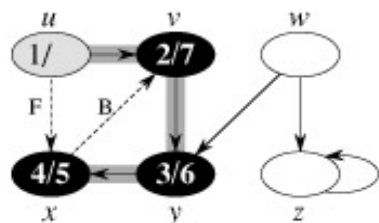
(f)



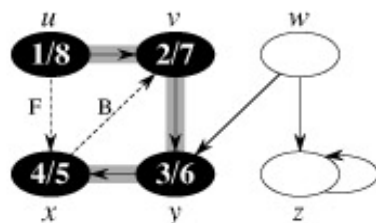
(g)



(h)

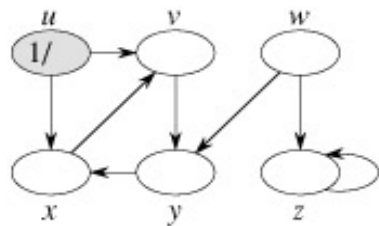


(i)

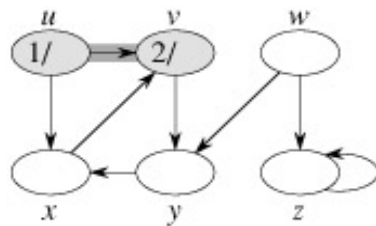


(j)

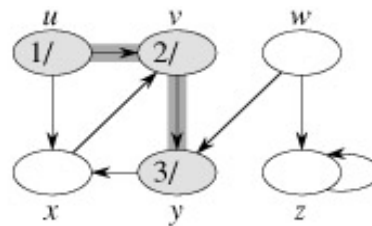
Depth First Search (DFS): Example



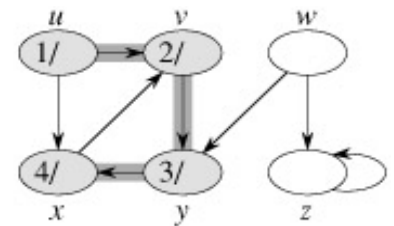
(a)



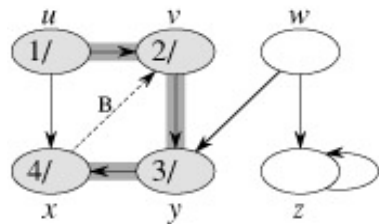
(b)



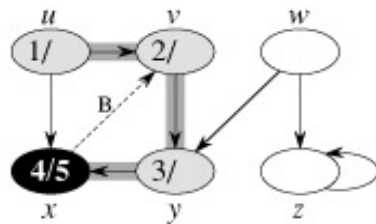
(c)



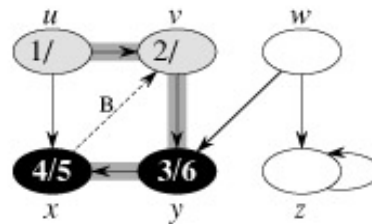
(d)



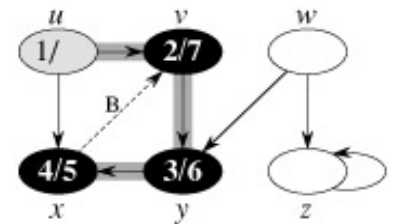
(e)



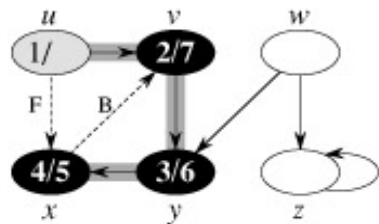
(f)



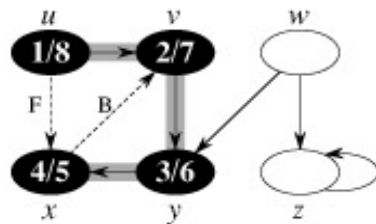
(g)



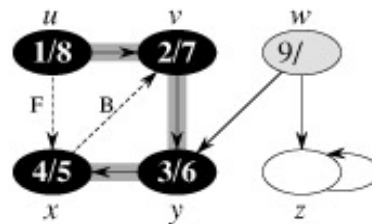
(h)



(i)

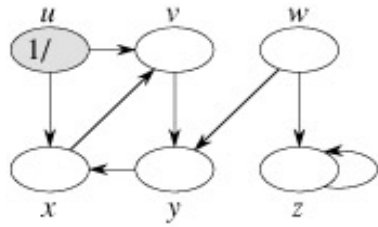


(j)

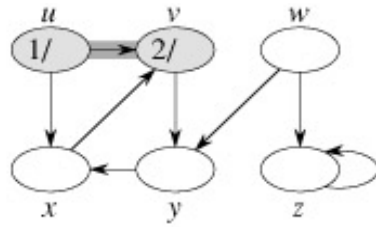


(k)

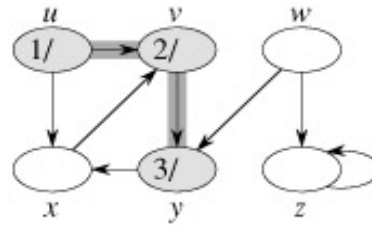
Depth First Search (DFS): Example



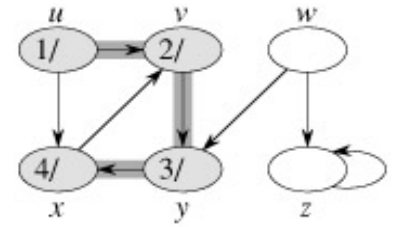
(a)



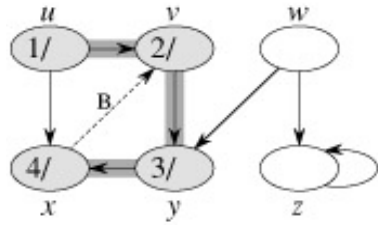
(b)



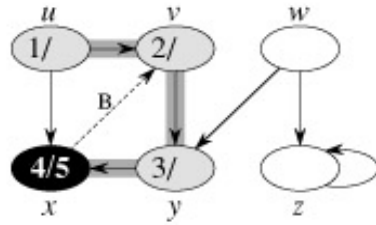
(c)



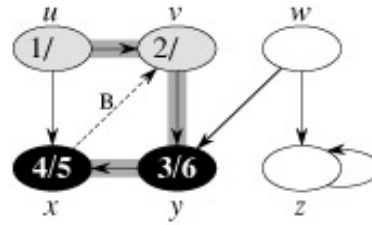
(d)



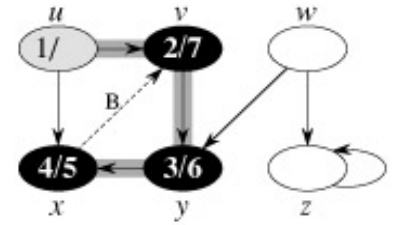
(e)



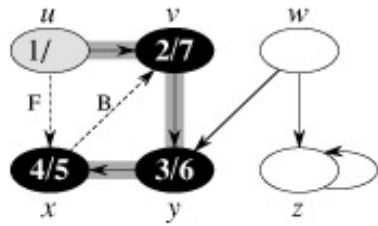
(f)



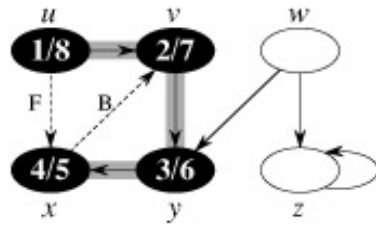
(g)



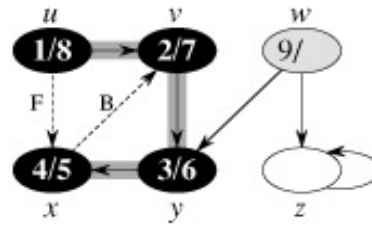
(h)



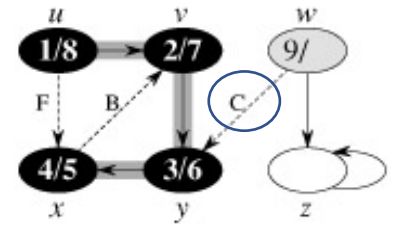
(i)



(j)

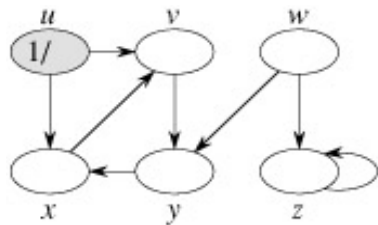


(k)

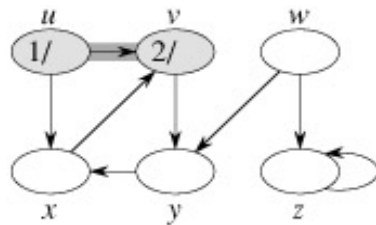


(l)

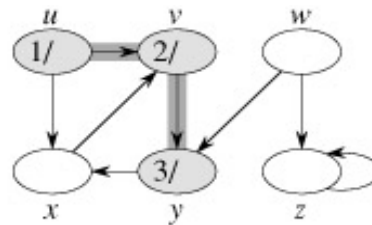
Depth First Search (DFS): Example



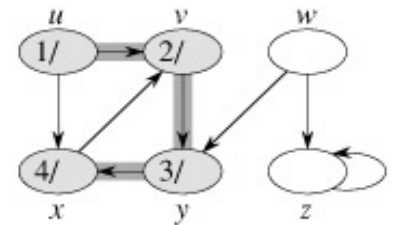
(a)



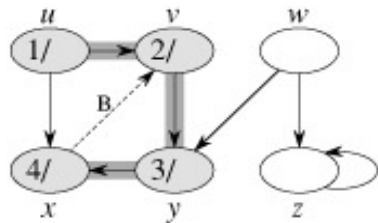
(b)



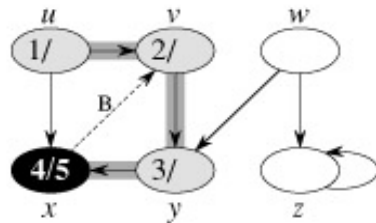
(c)



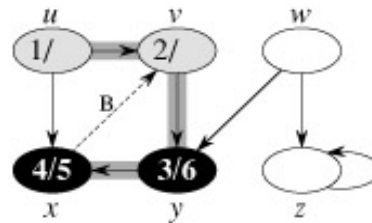
(d)



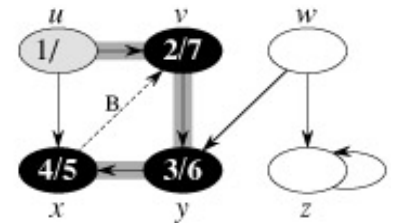
(e)



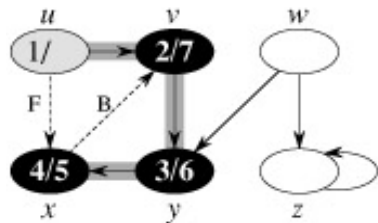
(f)



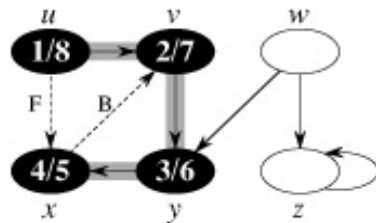
(g)



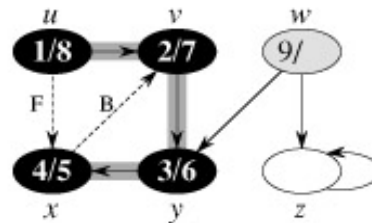
(h)



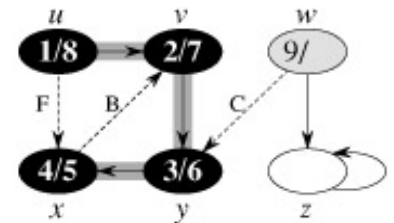
(i)



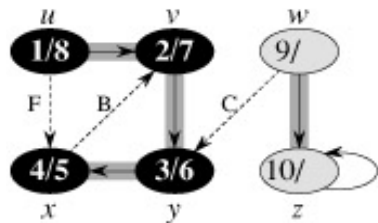
(j)



(k)

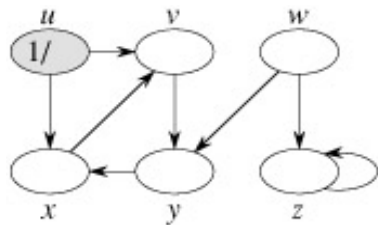


(l)

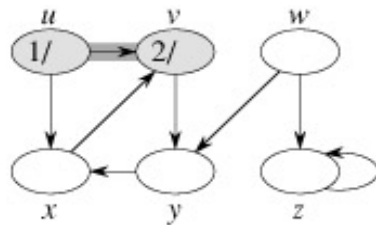


(m)

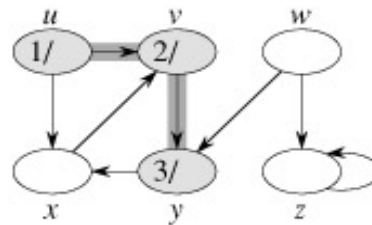
Depth First Search (DFS): Example



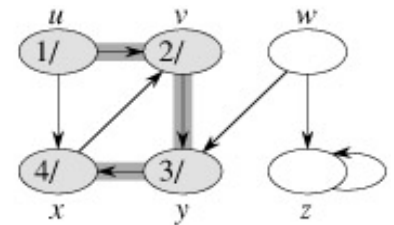
(a)



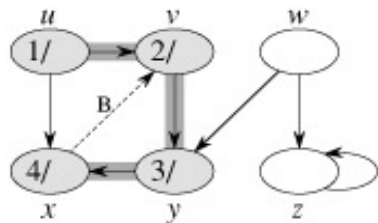
(b)



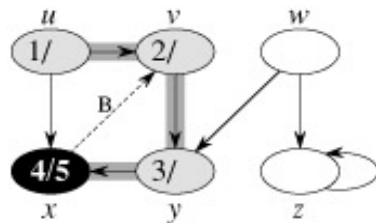
(c)



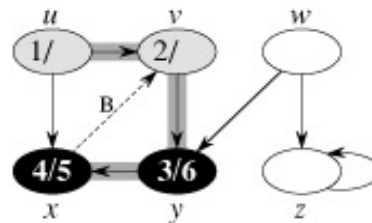
(d)



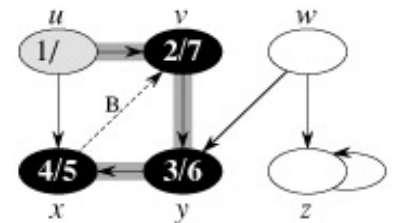
(e)



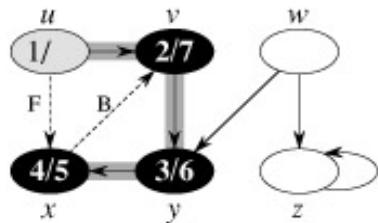
(f)



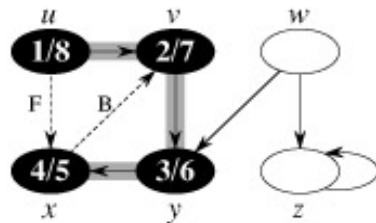
(g)



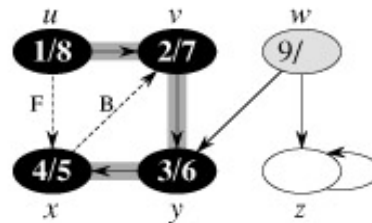
(h)



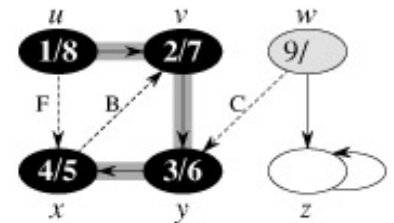
(i)



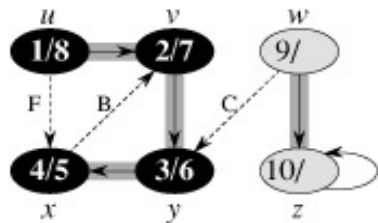
(i)



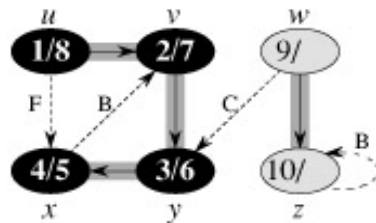
(k)



(1)

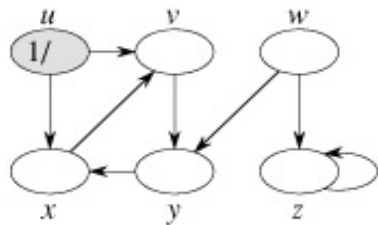


(m)

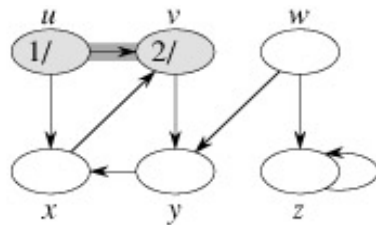


(n)

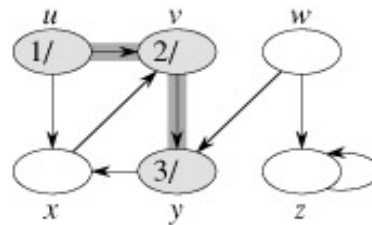
Depth First Search (DFS): Example



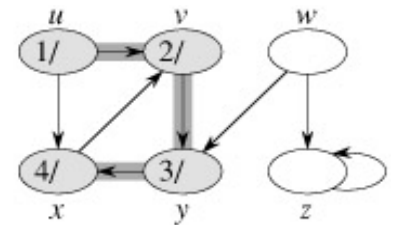
(a)



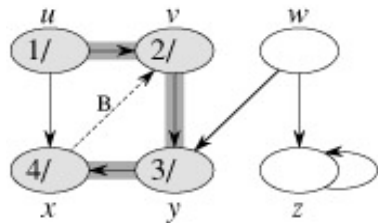
(b)



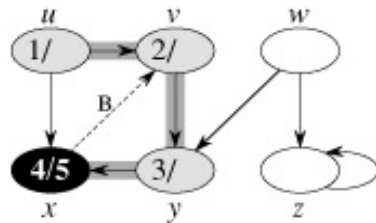
(c)



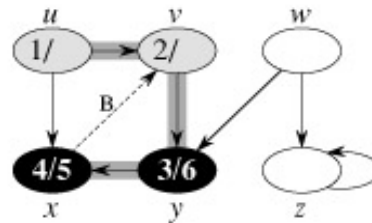
(d)



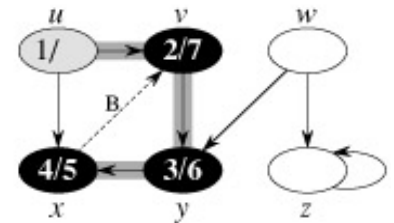
(e)



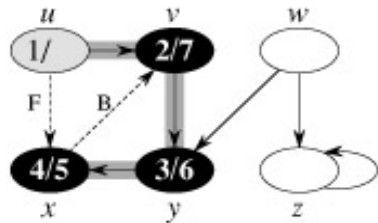
(f)



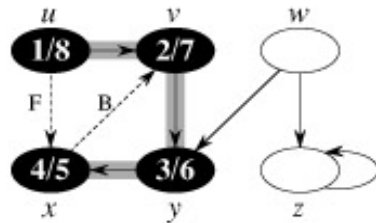
(g)



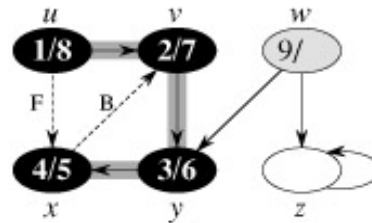
(h)



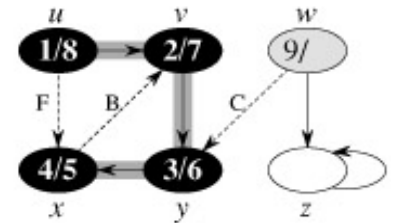
(i)



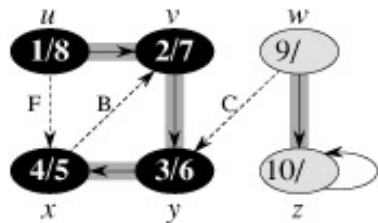
(i)



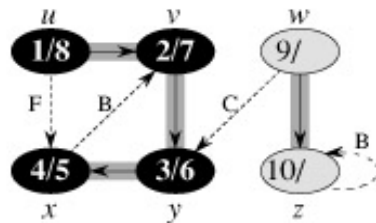
(k)



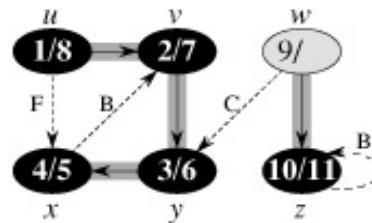
(1)



(m)

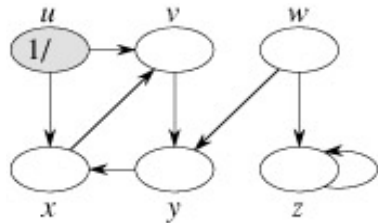


(n)

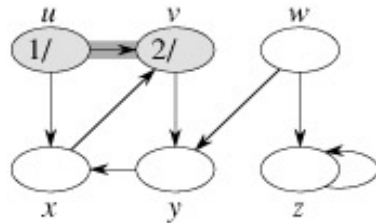


(o)

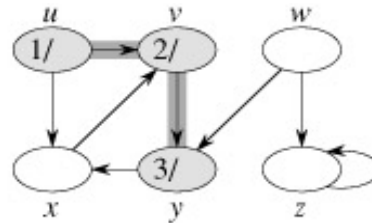
Depth First Search (DFS): Example



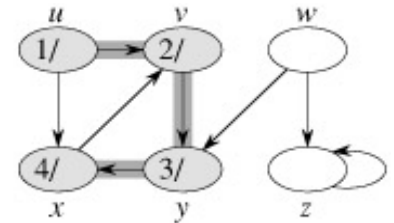
(a)



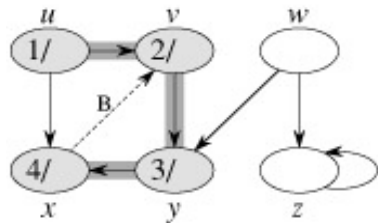
(b)



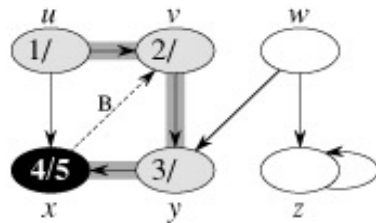
(c)



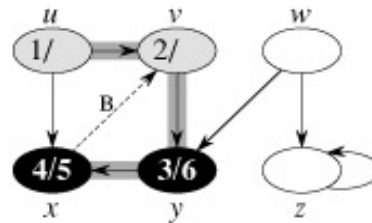
(d)



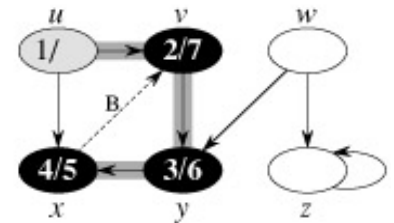
(e)



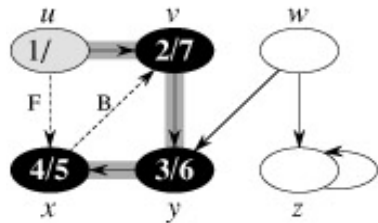
(f)



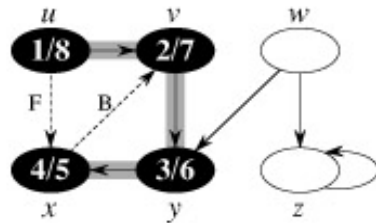
(g)



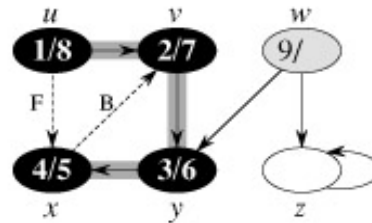
(h)



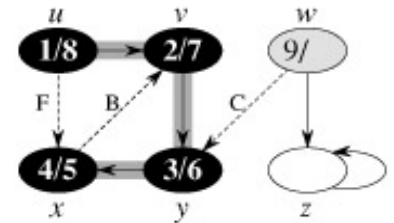
(i)



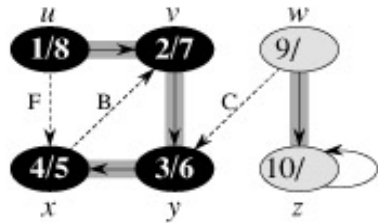
(j)



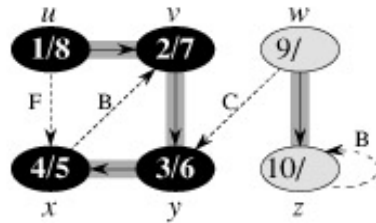
(k)



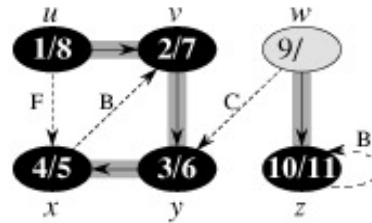
(l)



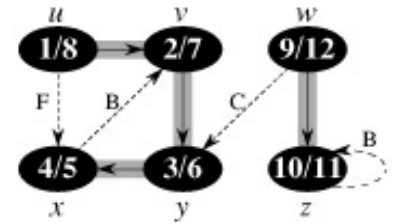
(m)



(n)



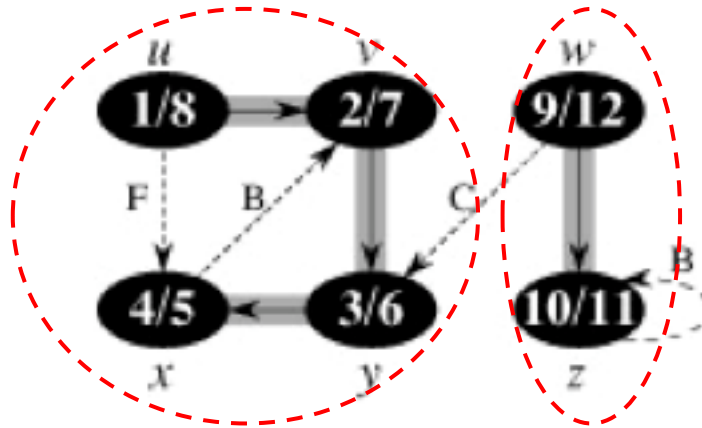
(o)



(p)

DFS: Forest

- The DFS may create **multiple disconnected trees** called **forest**



DFS: Time Complexity

DFS-VISIT(*u*)

```
1  color[u] ← GRAY      ▷White vertex u has just been discovered.
2  time ← time +1
3  d[u]=time
4  for each v in Adj[u]  ▷Explore edge(u, v).
5      do if color[v] = WHITE
6          then π[v] ← u
7              DFS-VISIT(v)
8  color[u]=BLACK      ▷ Blacken u; it is finished.
9  f [u] ▷ time ← time +1
```

Each node is recursively called once → $O(V)$

The For Loop (Step 4) visits the outgoing edges of each node → Total $O(E)$

Total Complexity $O(V + E)$

Also written in a more precise form as $O(|V| + |E|)$

DFS (*G*)

```
1  for each vertex u in G.V
2      color[u] = WHITE
3      π(u) <- NIL
4  time <- 0
5  for each vertex un in G.V
6      if color[u] = WHITE
7          DFS-VISIT(u)
```

Analyzing The Collected Info.

- **The algorithm computes for each node u**
 - $u.startTime$ ($u.d$)
 - $u.endTime$ ($u.f$)
- **These intervals form a nested structure**

- **Parenthesis Theorem**

- If u has $(u.d, u.f)$, and v has $(v.d, v.f)$, then either:

- Case 1: u descendant of v in DFS

$$v.d < u.d < u.f < v.f$$

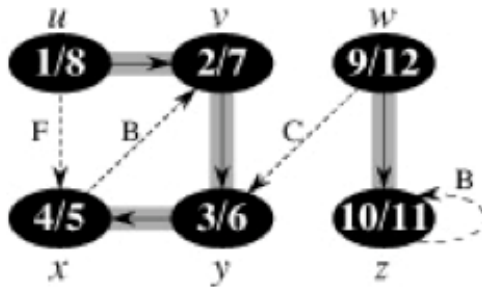
- Case 2: v descendant of u in DFS

$$u.d < v.d < v.f < u.f$$

- Case 3: u and v are disjoint (different trees in the forest)

$(u.d, u.f)$ and $(v.d, v.f)$ are not overlapping

Example



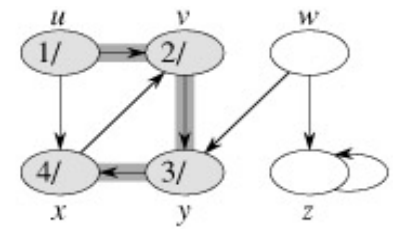
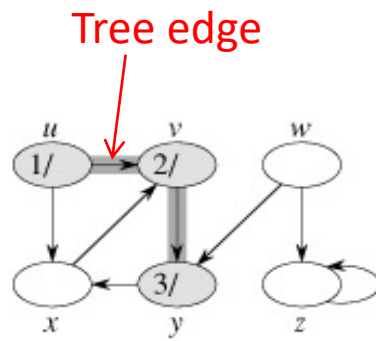
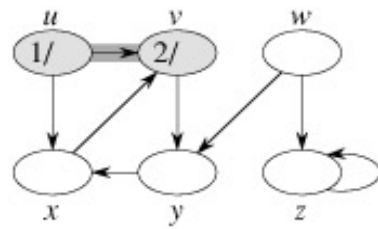
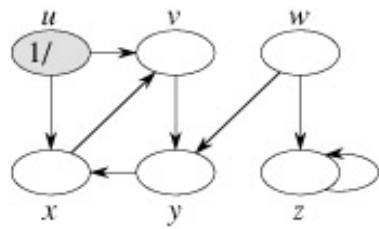
Based on the numbers (in the node) you can draw the DFS trees (plural)

- v, y, x are all descendants of u
 - In other words: u is an ancestor to v, y, x
- w and u are disjoint

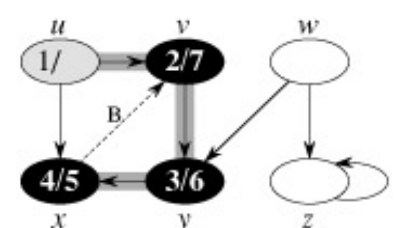
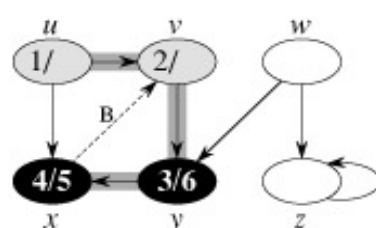
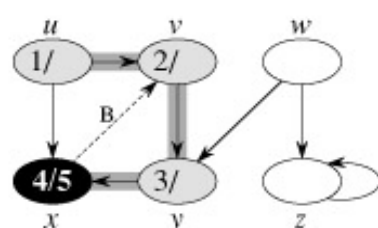
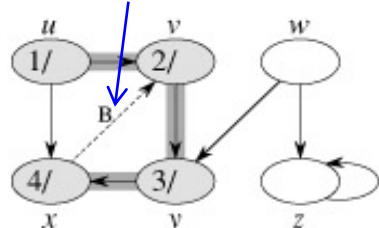
Classification of Graph Edges

- **While doing DFS, we can label the edges**
 - **Tree edges:** Included in the DFS trees
 - **Forward edges:** From ancestor to already-visited descendant
 - **Backward edges:** From descendant to already-visited ancestor
 - **Cross edges:** Any other edges

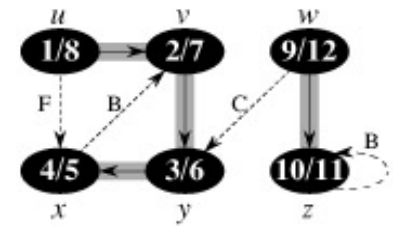
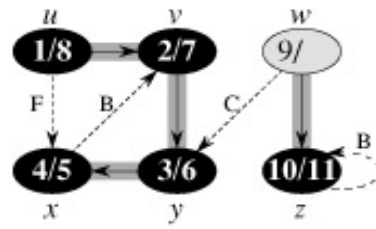
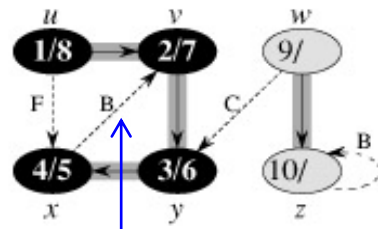
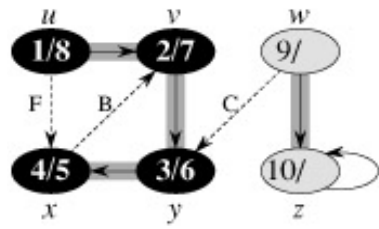
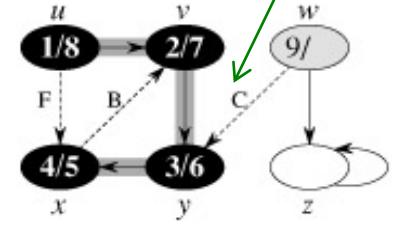
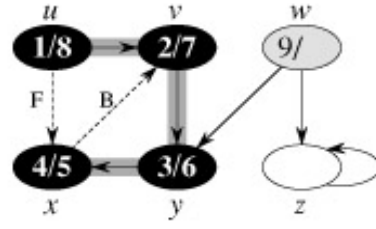
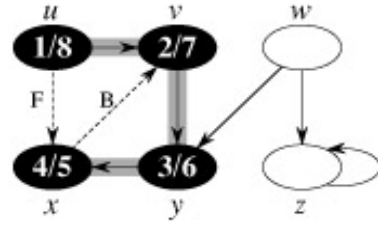
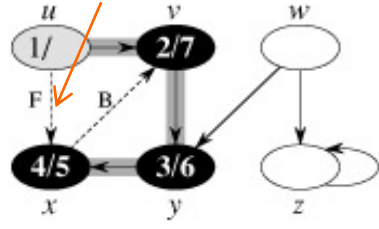
Depth First Search (DFS): Example



Backward edge

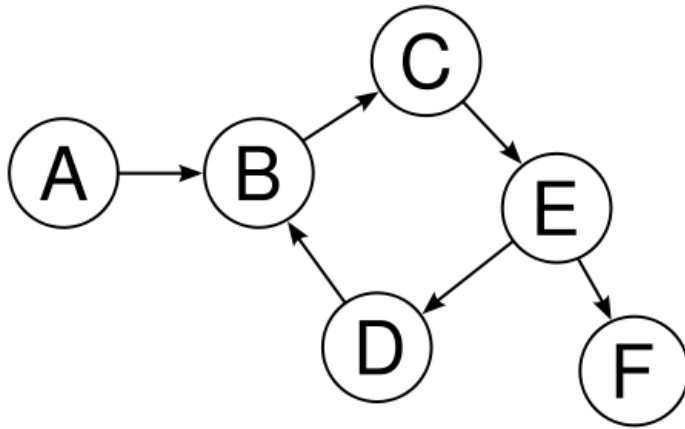


Forward edge

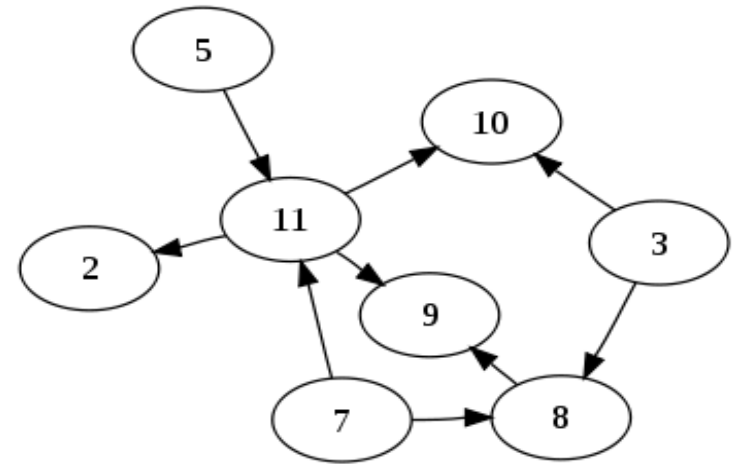


Backward edge

Cycles in Graphs



Cyclic graph (Graph containing cycles)



**Acyclic graph (Graph without cycles)
Called: Directed Acyclic Graph DAG**

- ***How to know if a graph has a cycle or not? What is the time complexity?***

Cycles in Graphs (Cont'd)

- **Answer**
 - Perform DFS on the graph
 - If there are **backward edges** → there is a cycle
 - Otherwise → there are no cycles

Topological Sort

Topological Sort

- **Sorting technique over DAGs (Directed Acyclic Graphs)**
- **It creates a linear sequence (ordering) for the nodes such that:**
 - **If u has an outgoing edge to $v \rightarrow$ then u must finish before v starts**
- **Very common in ordering jobs or tasks**

Topological Sort Example

A job consists of 10 tasks with the following precedence rules:

Must start with 7, 5, 4 or 9.

Task 1 must follow 7.

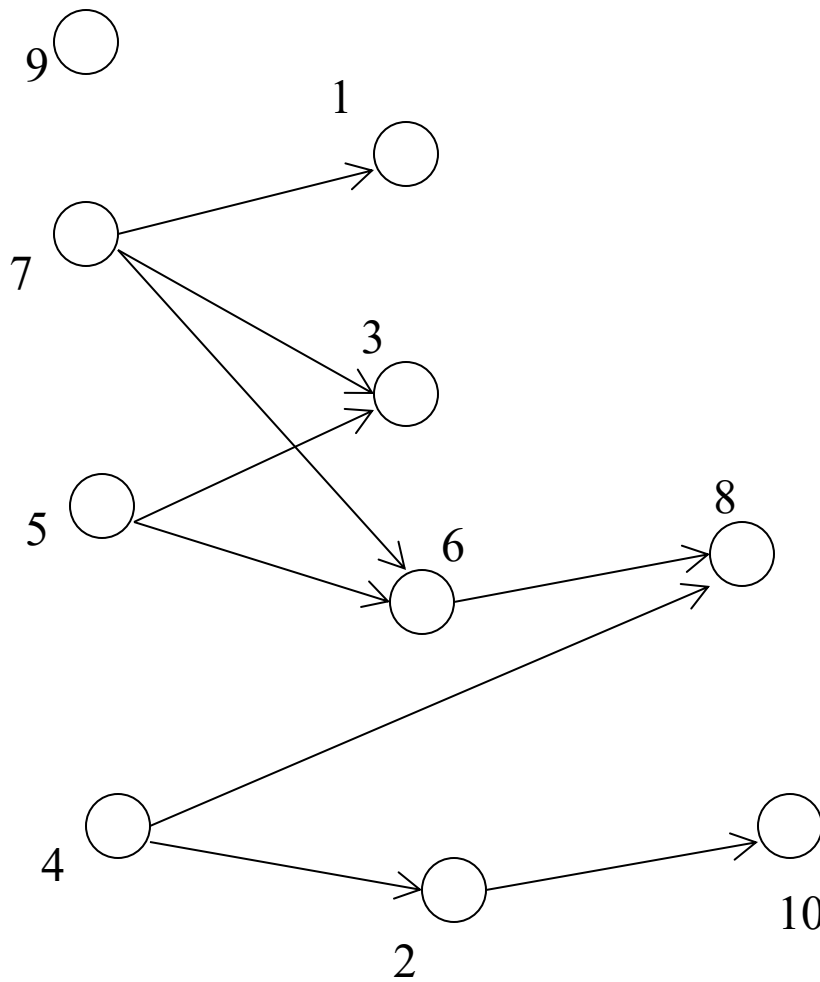
Tasks 3 & 6 must follow both 7 & 5.

8 must follow 6 & 4.

2 must follow 4.

10 must follow 2.

Make a directed graph and then do DFS.

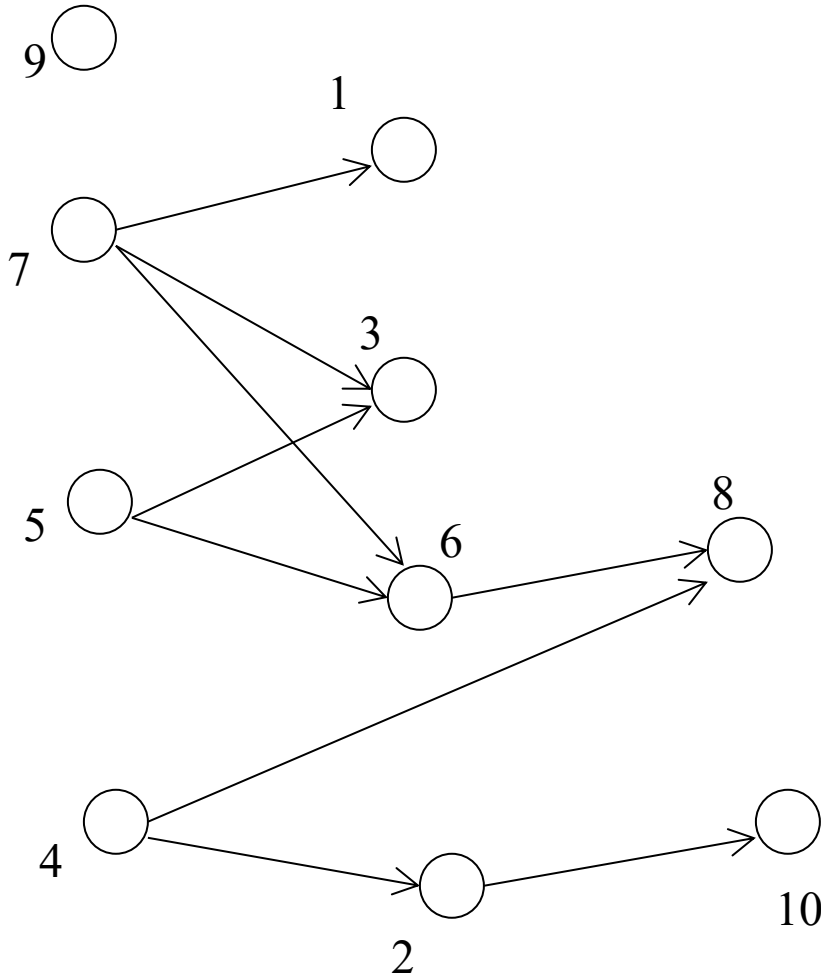


Tasks shown as a directed graph.

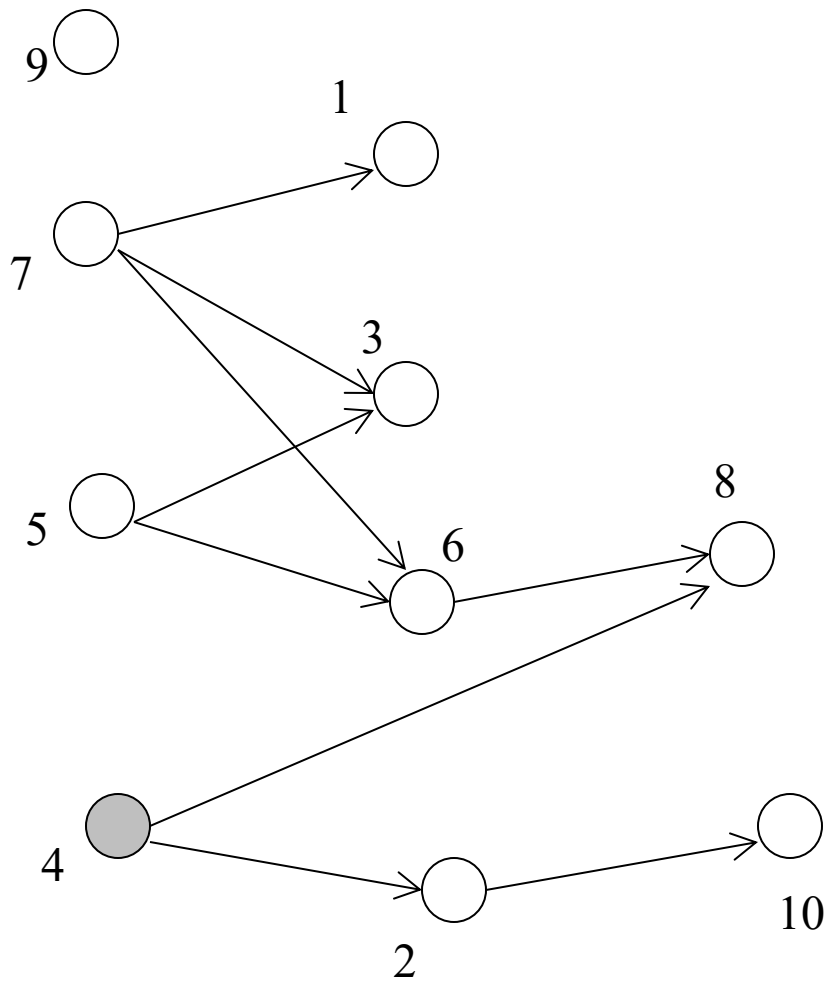
Topological Sort using DFS

- To create a topological sort from a DAG
 - 1- Final linked list is empty
 - 2- Run DFS
 - 3- When a node becomes black (finishes) insert it to the top of a linked list

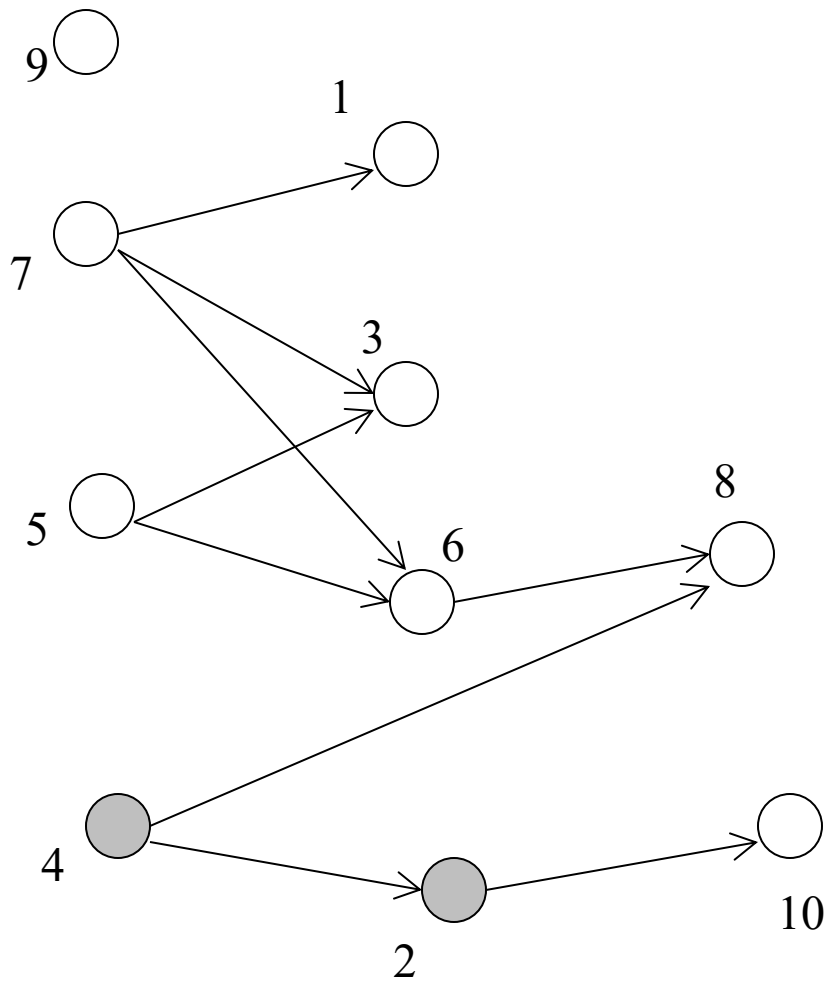
Example



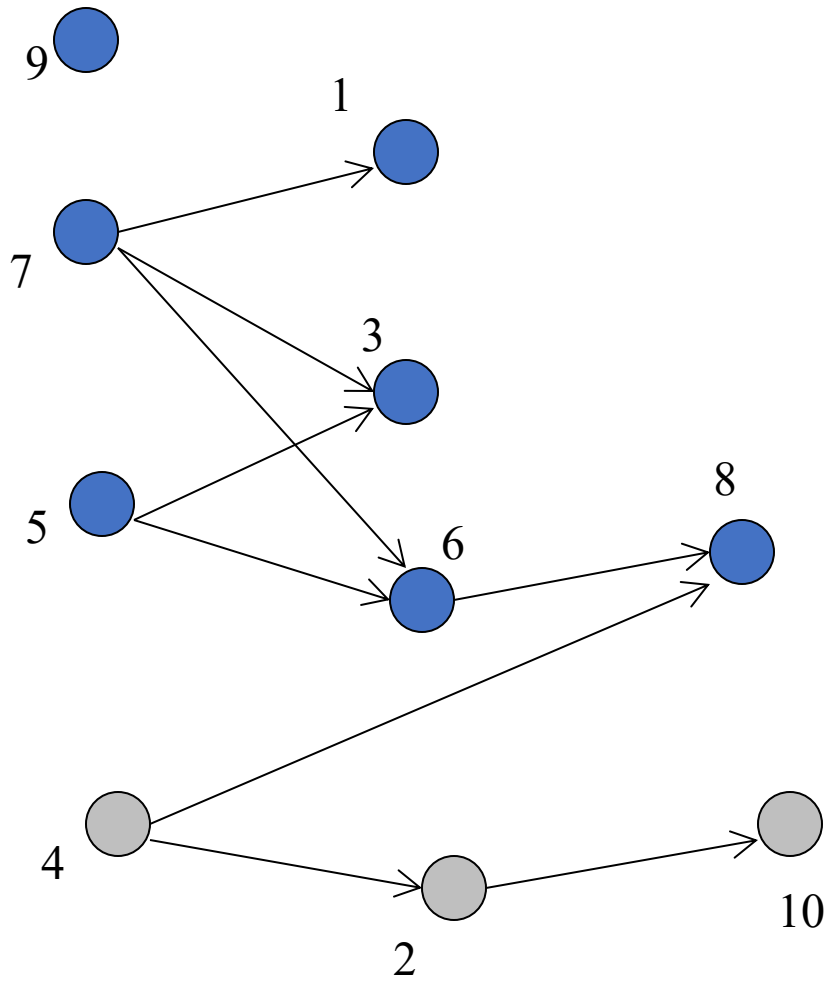
Example



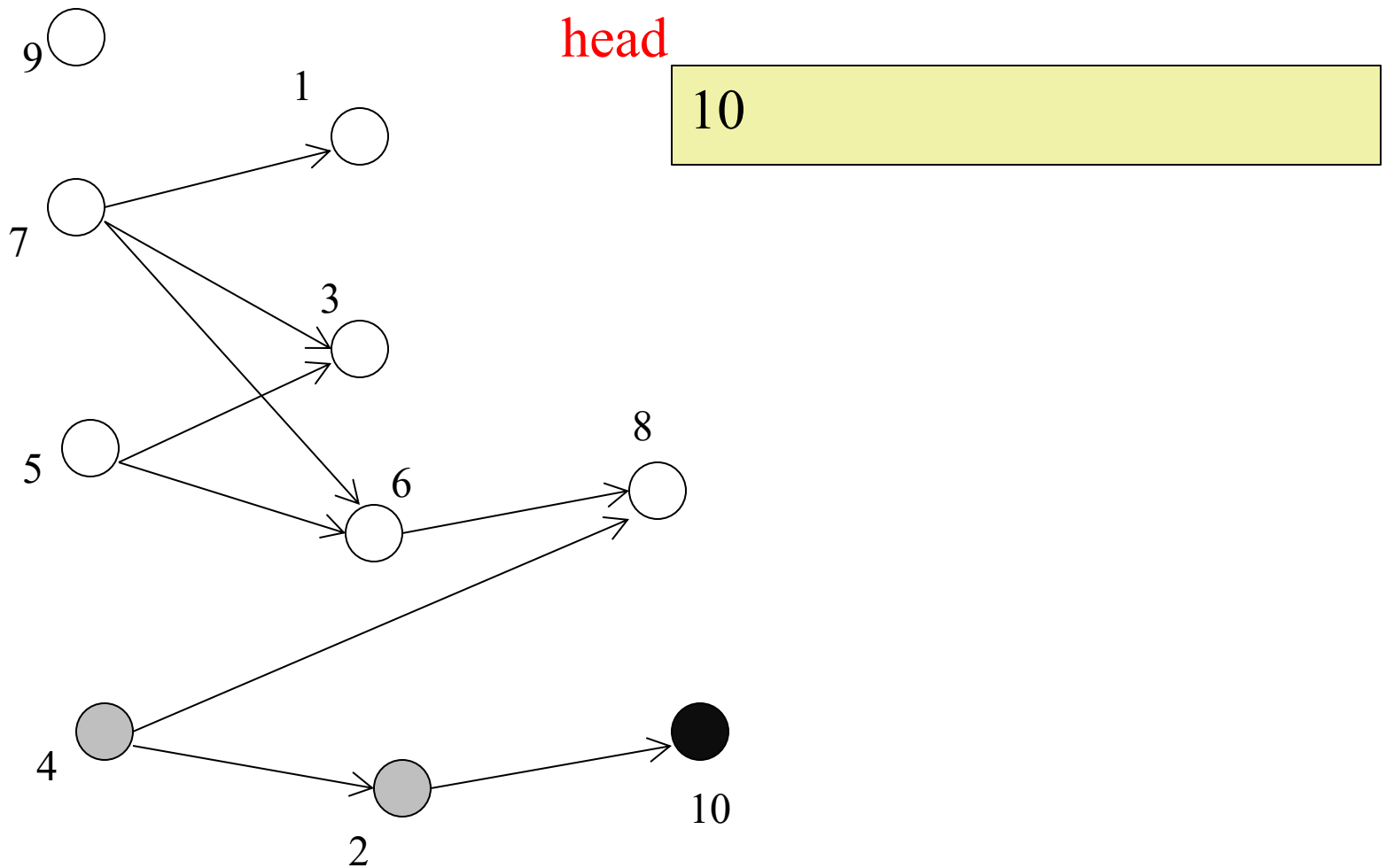
Example



Example

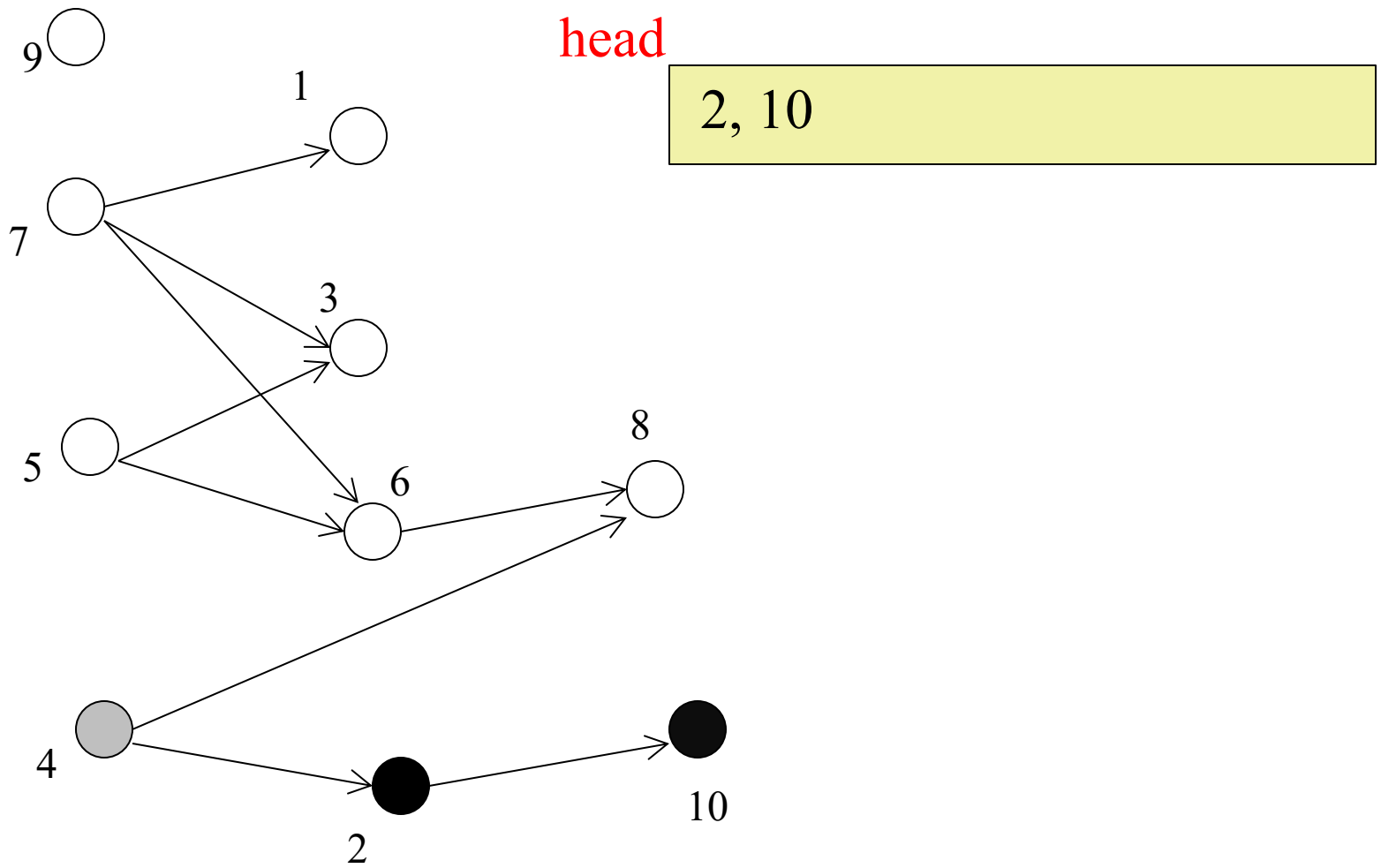


Example

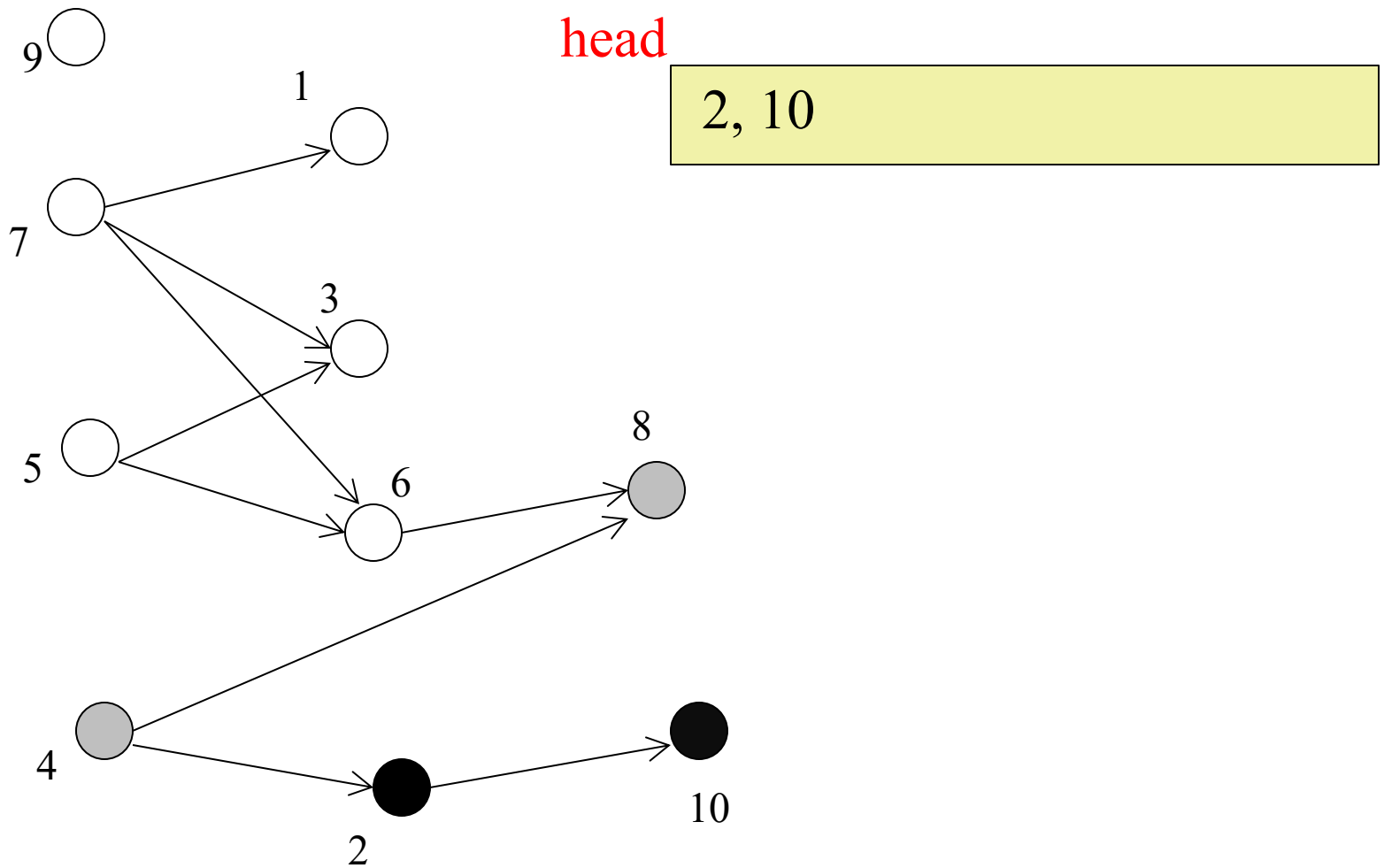


Note, you need a stack to run the DFS algorithm, that's not shown here. We only show the linked-list used for topological sort.

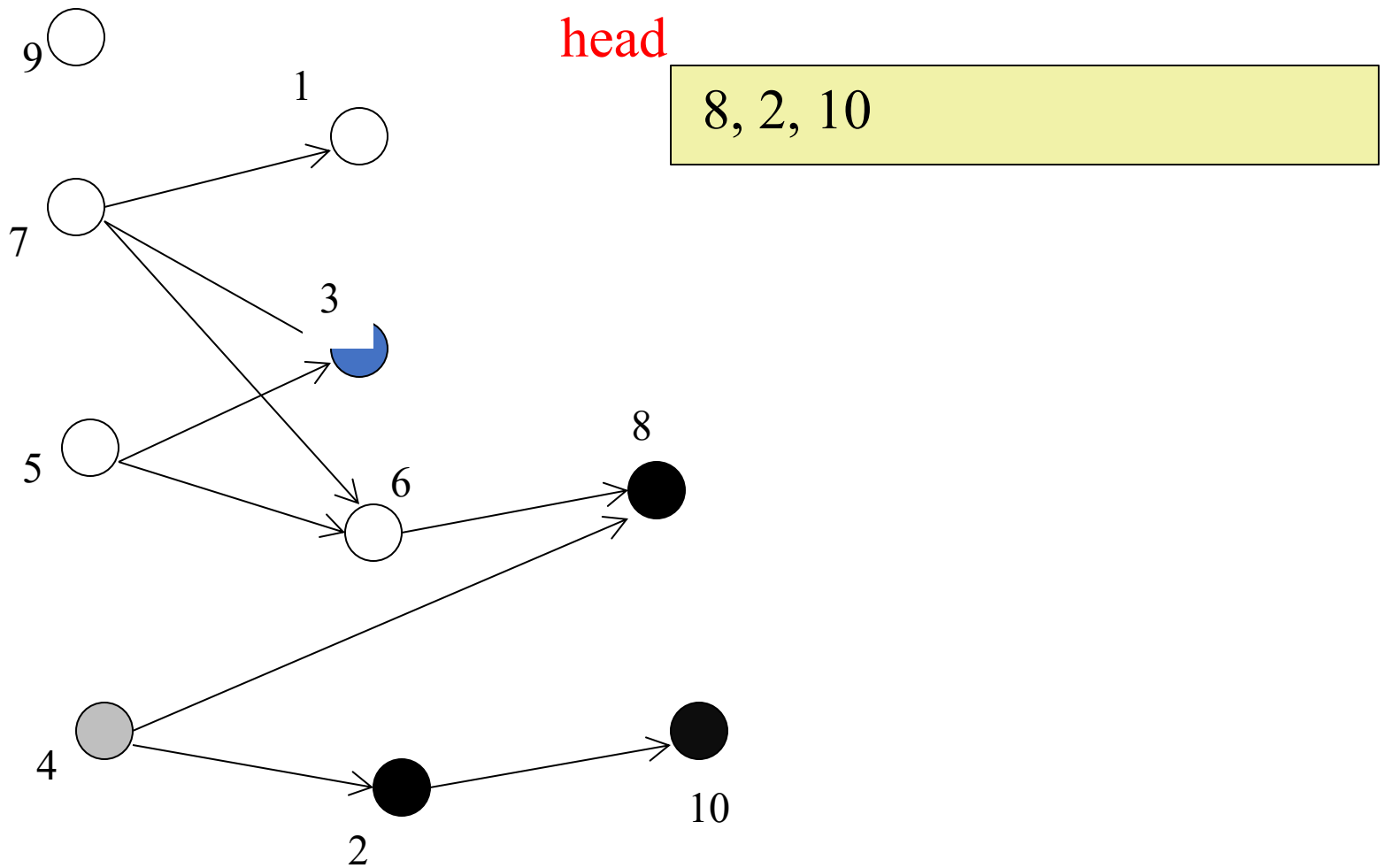
Example



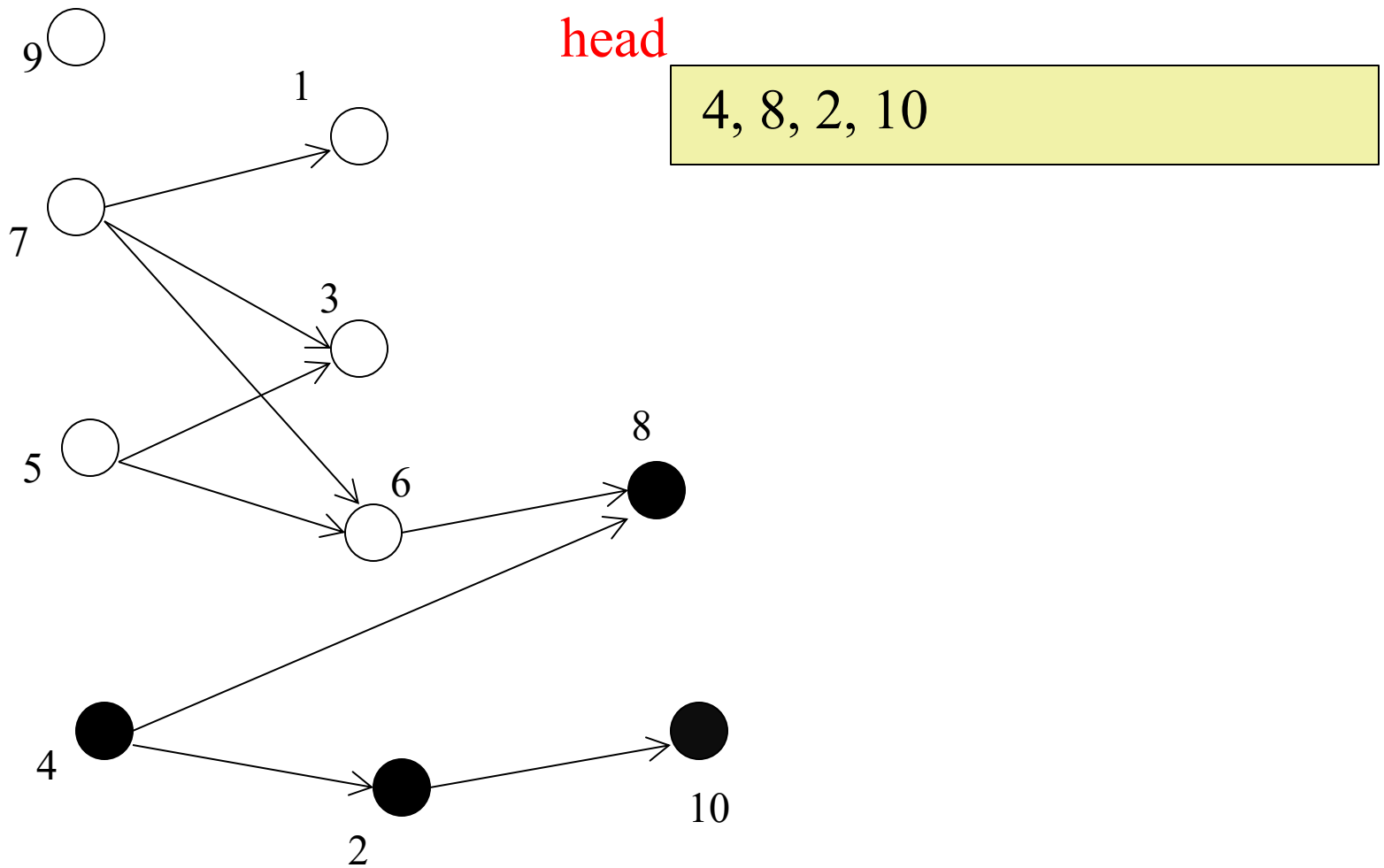
Example



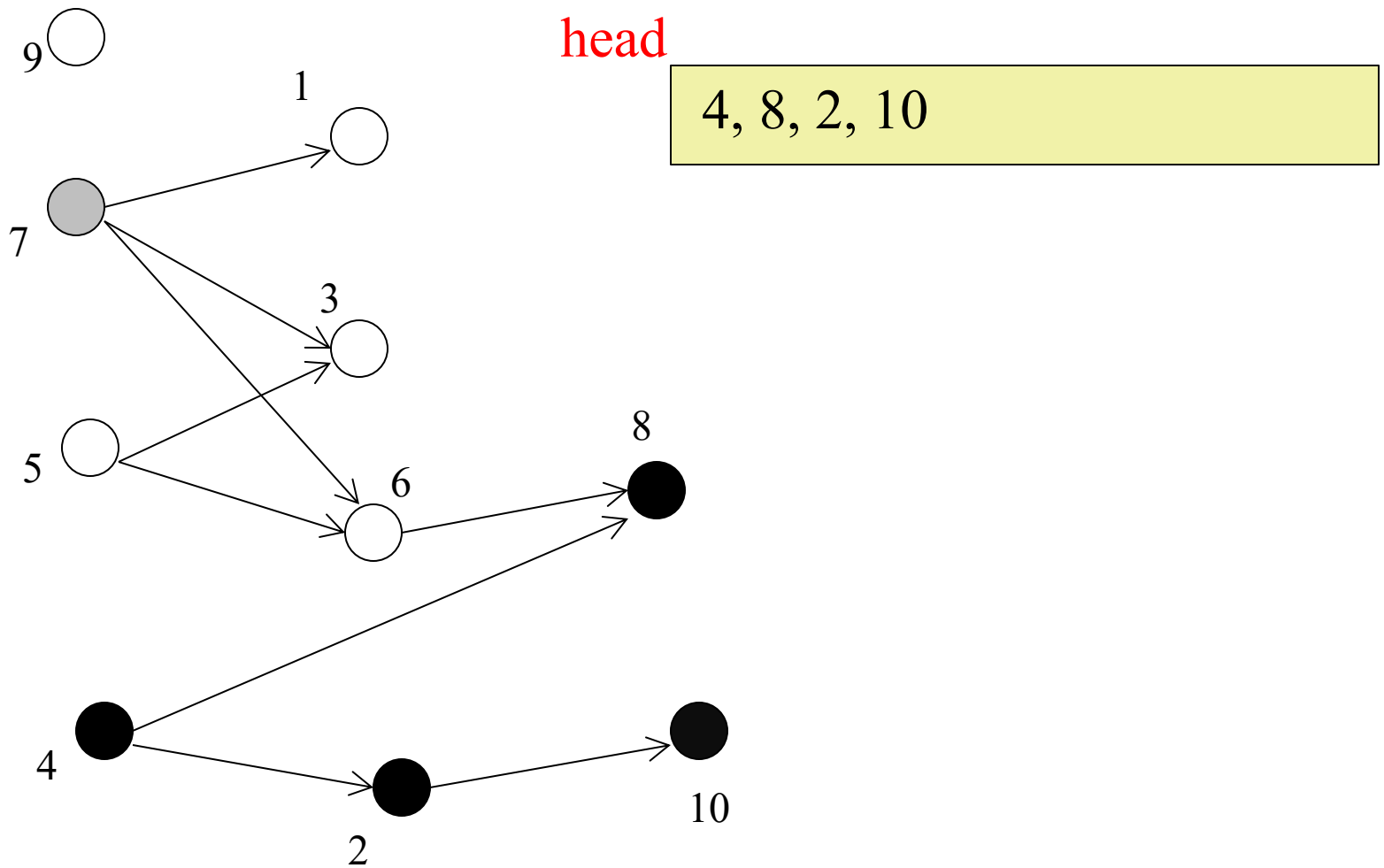
Example



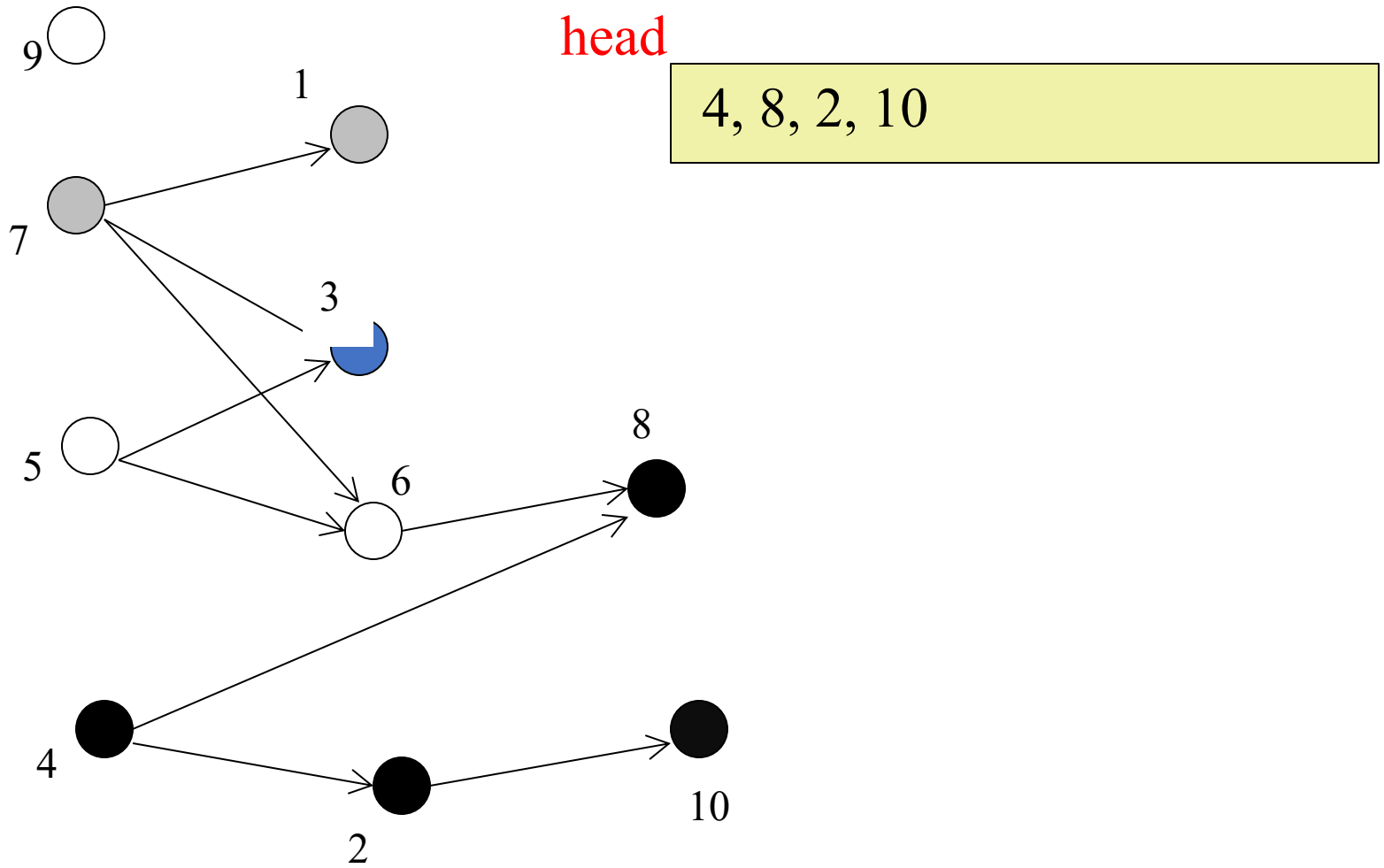
Example



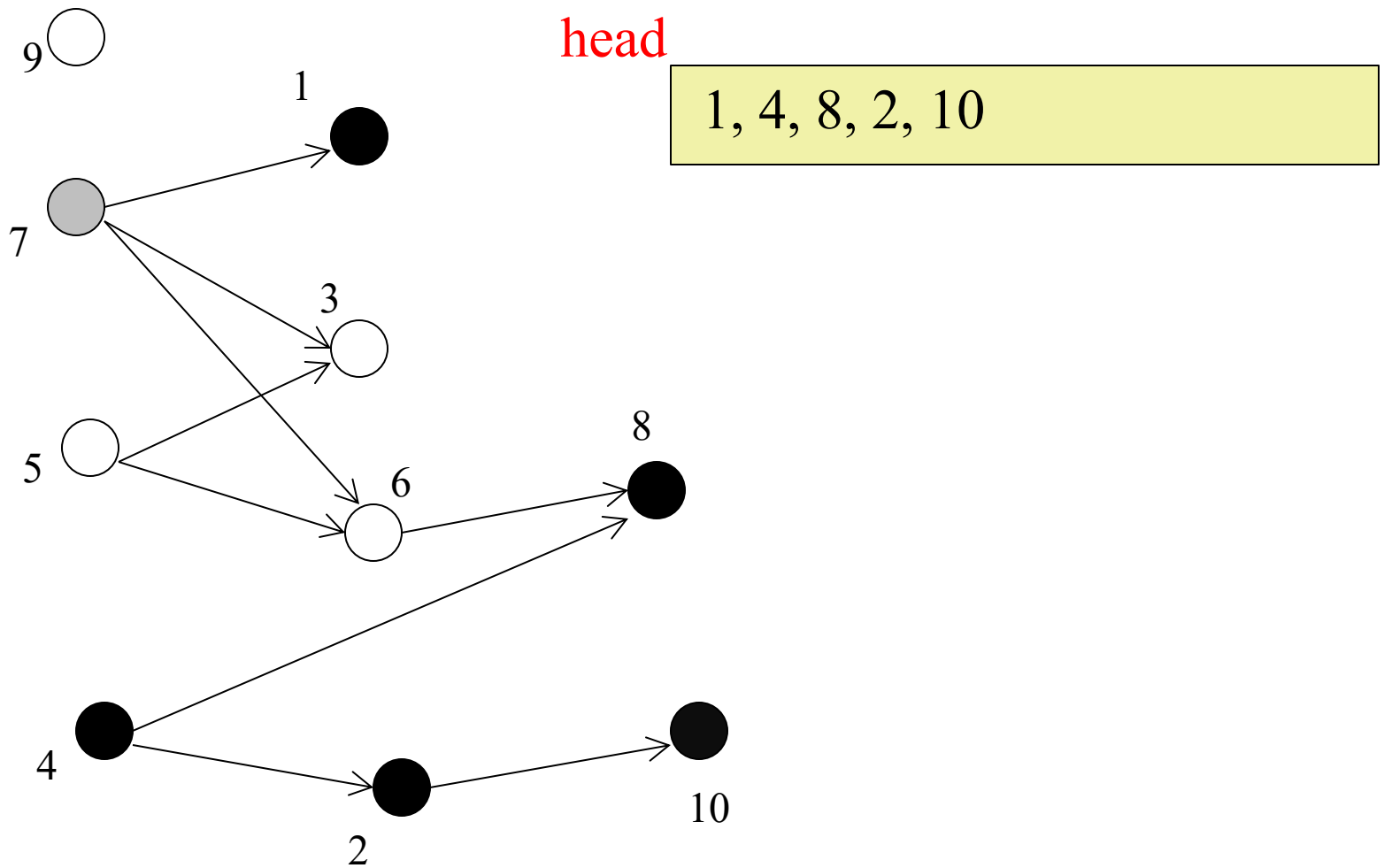
Example



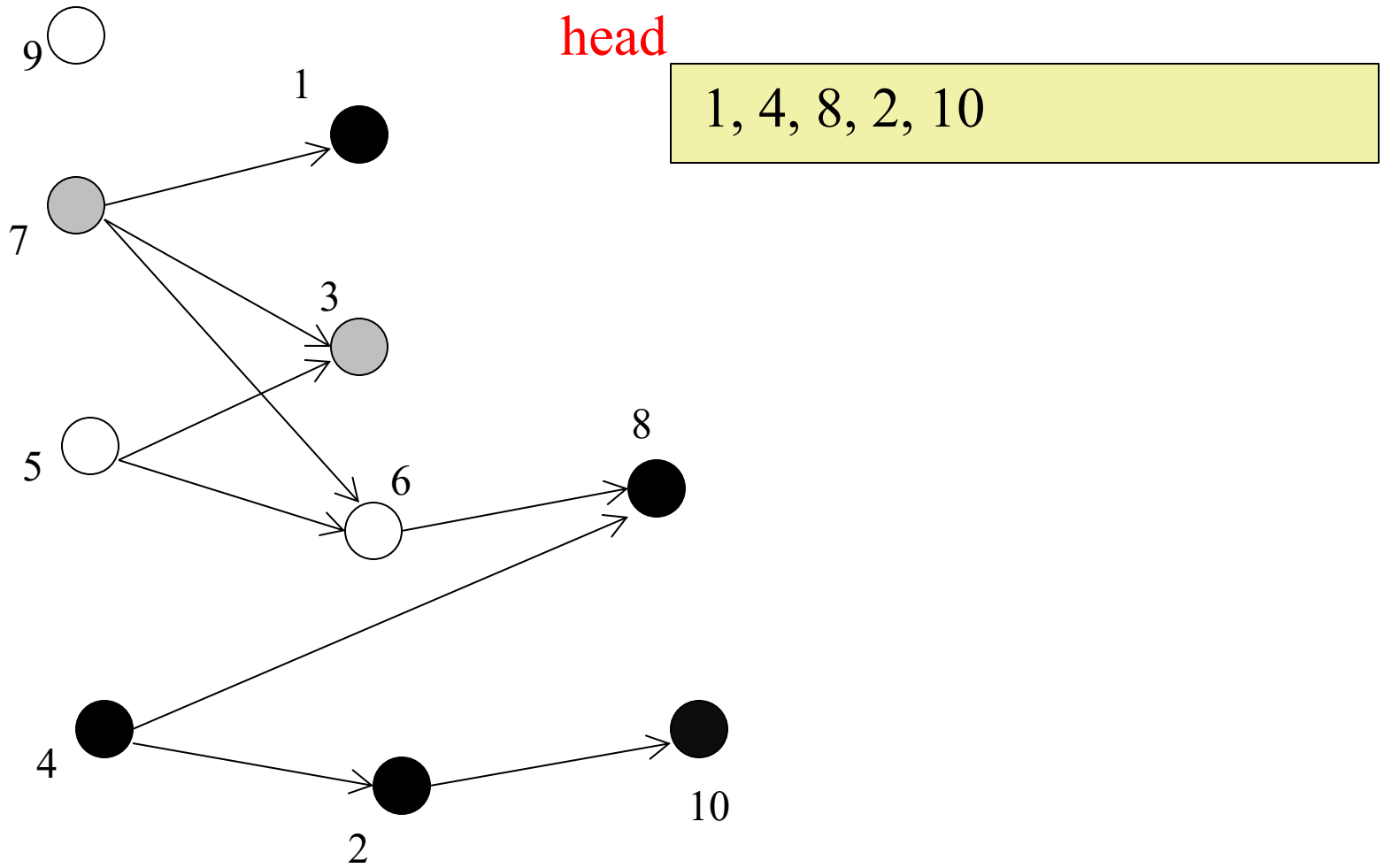
Example



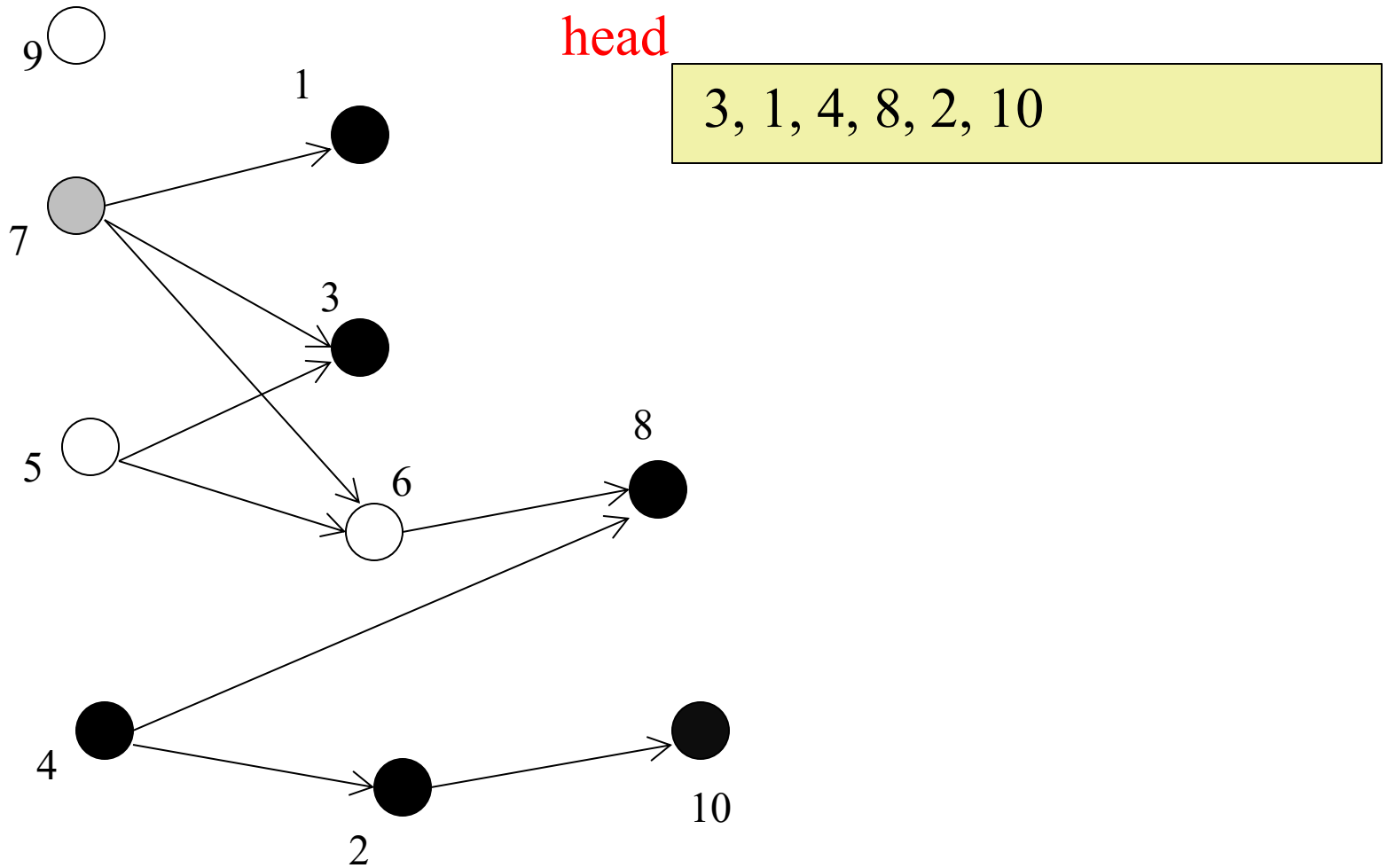
Example



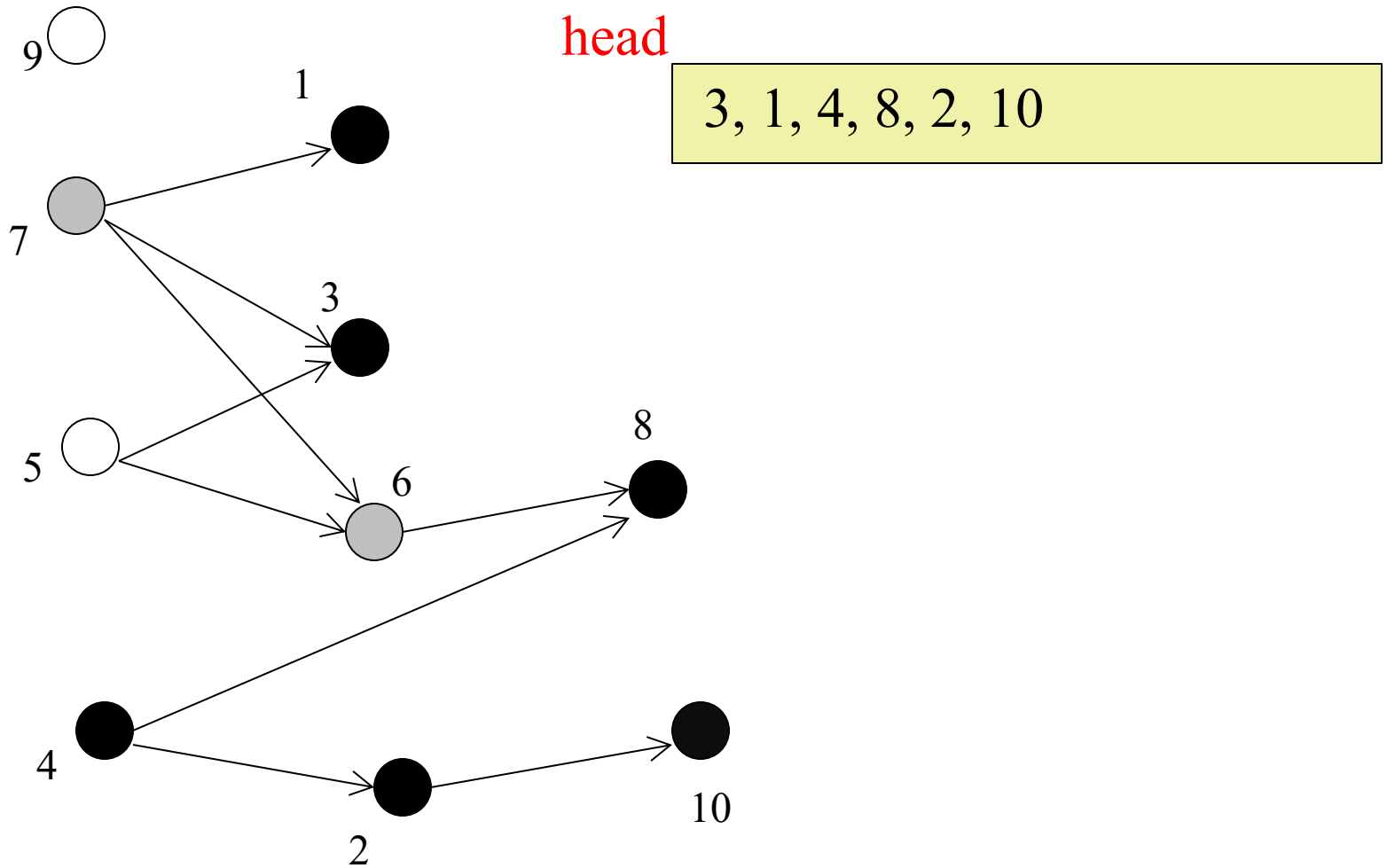
Example



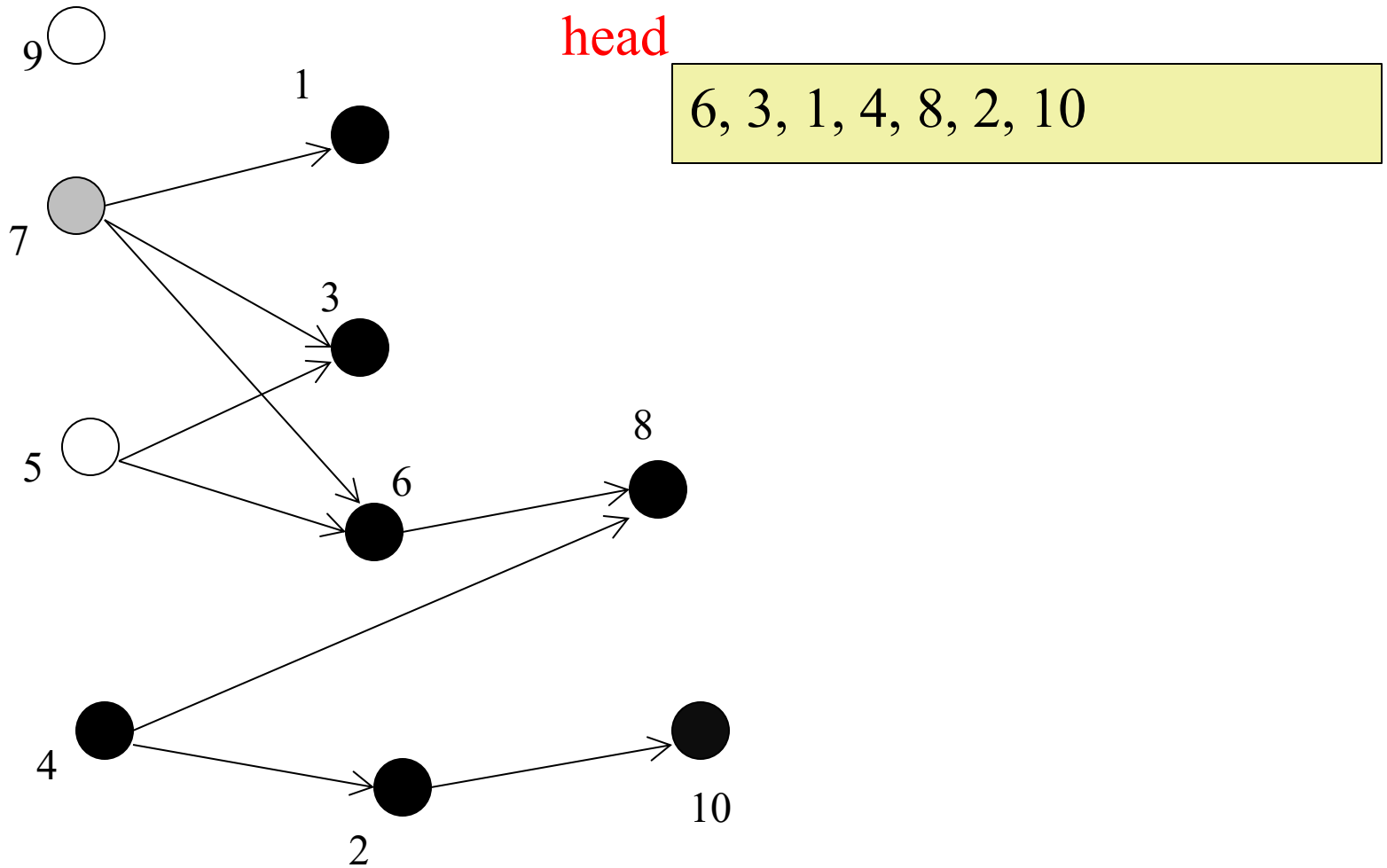
Example



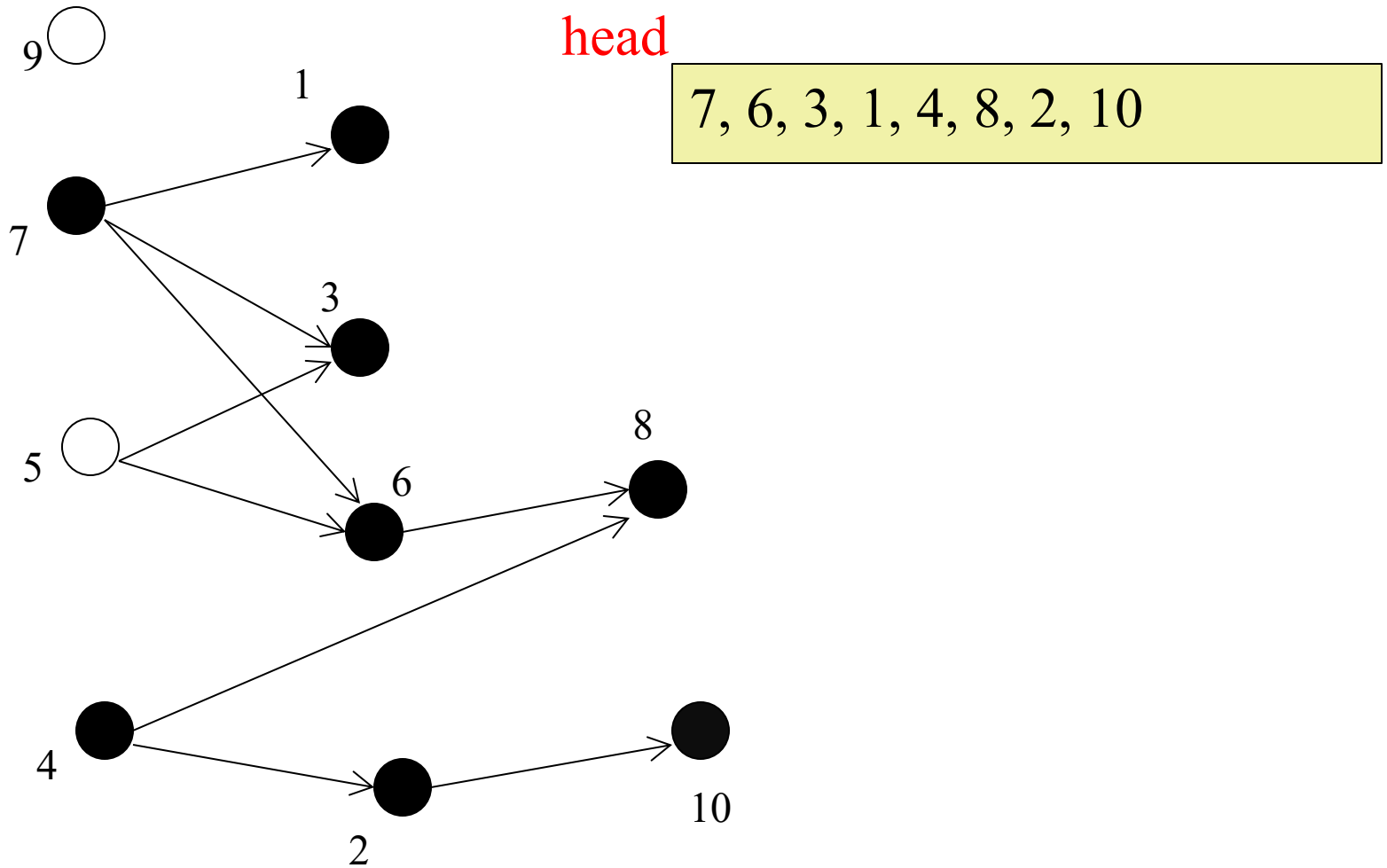
Example



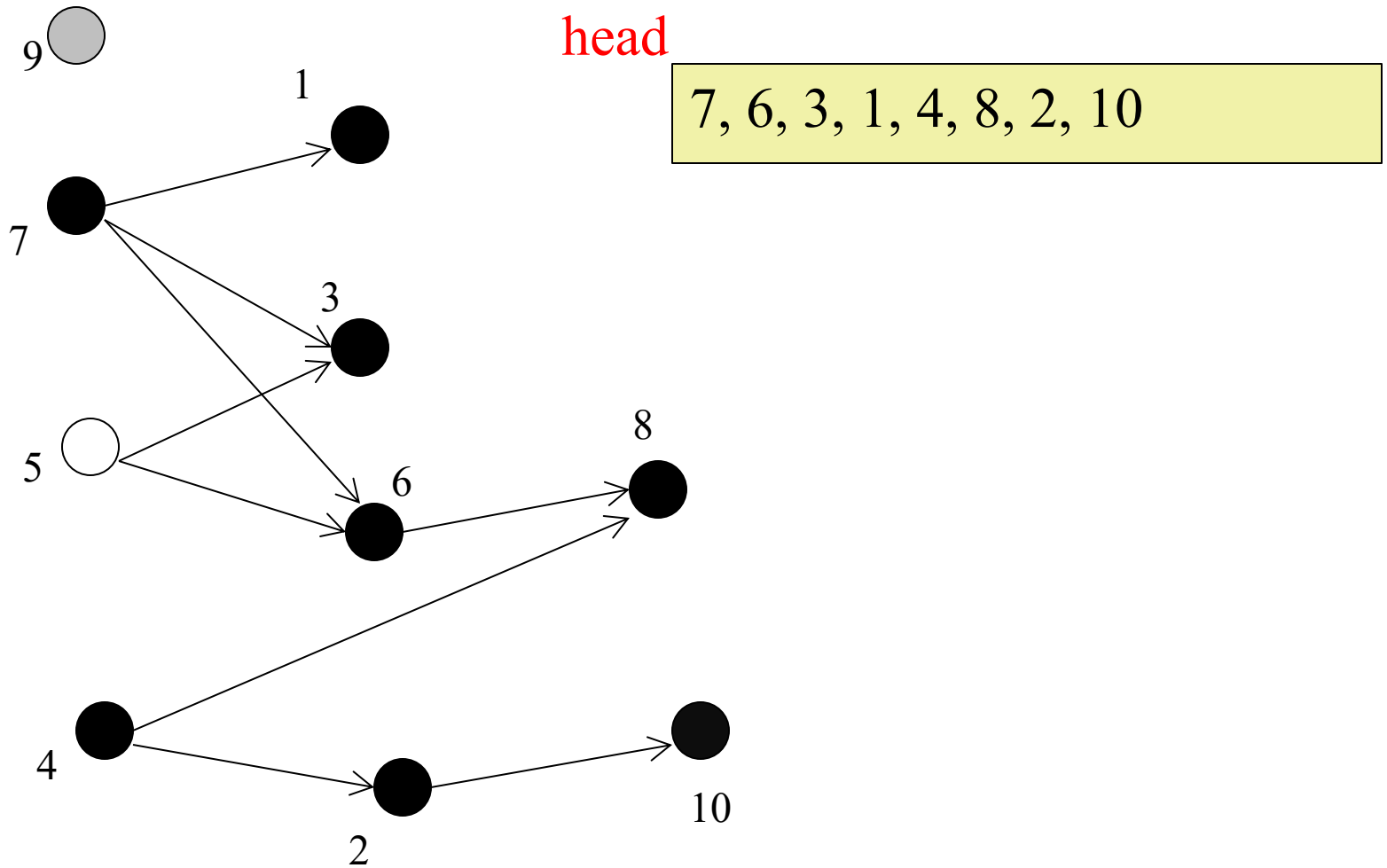
Example



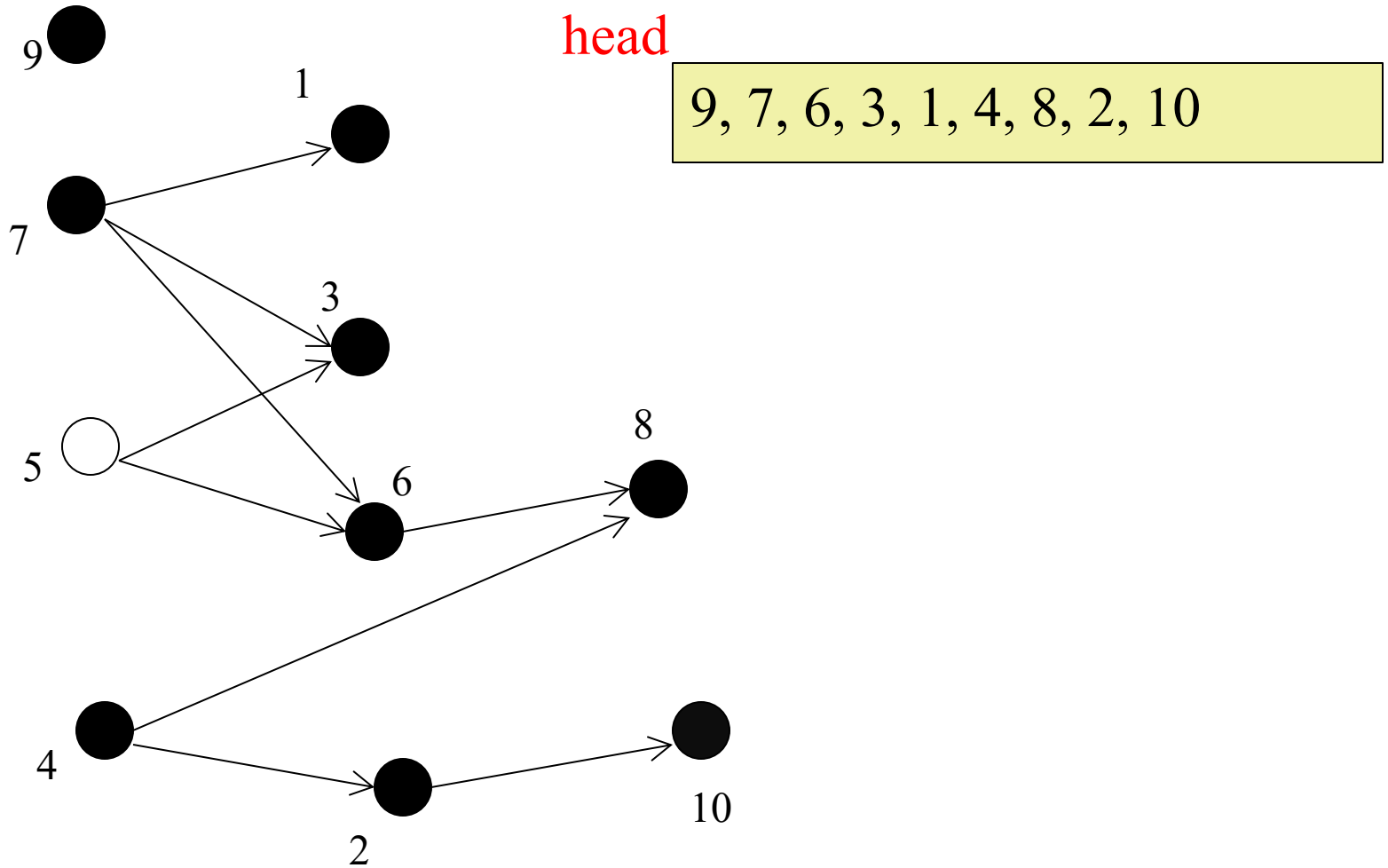
Example



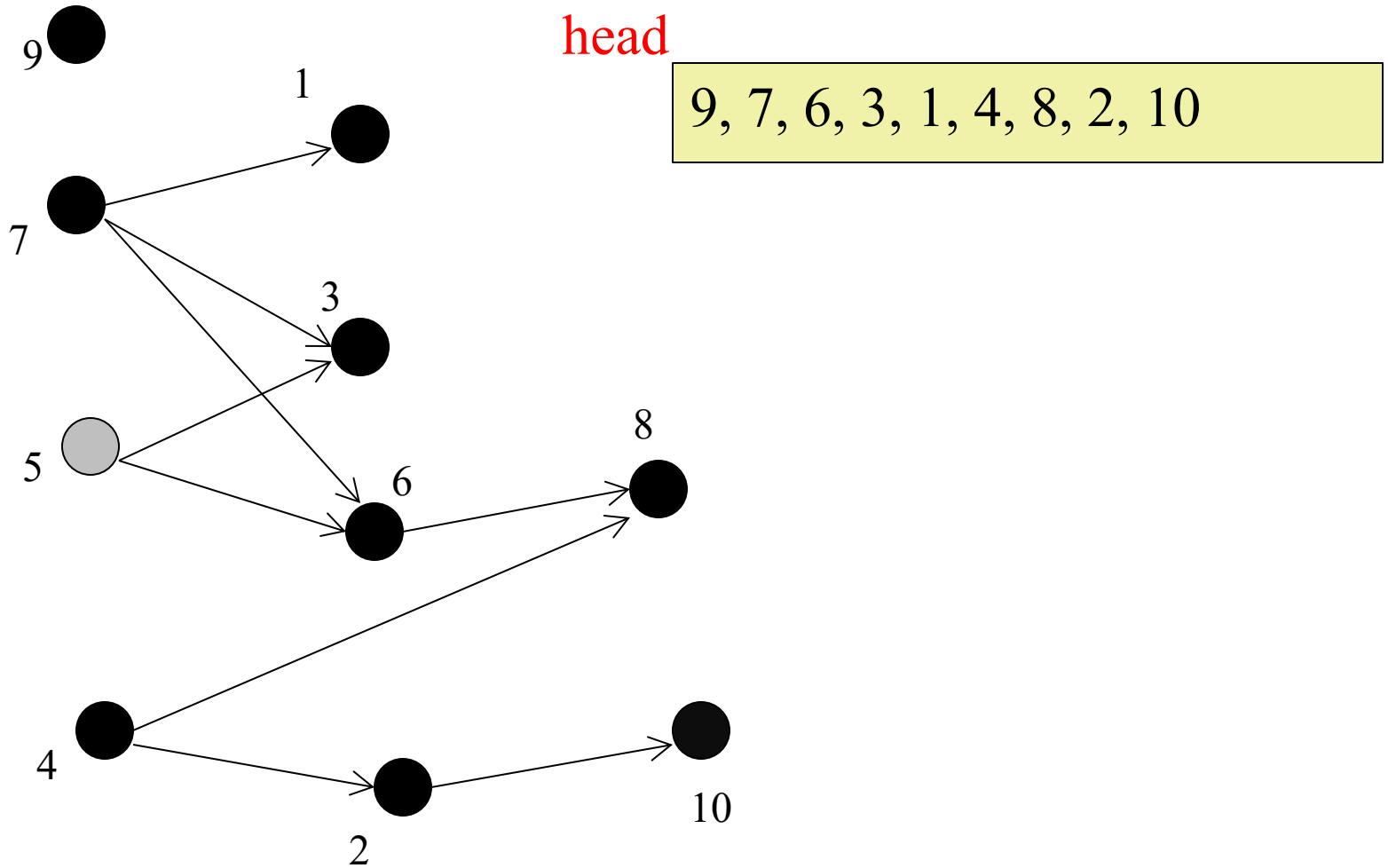
Example



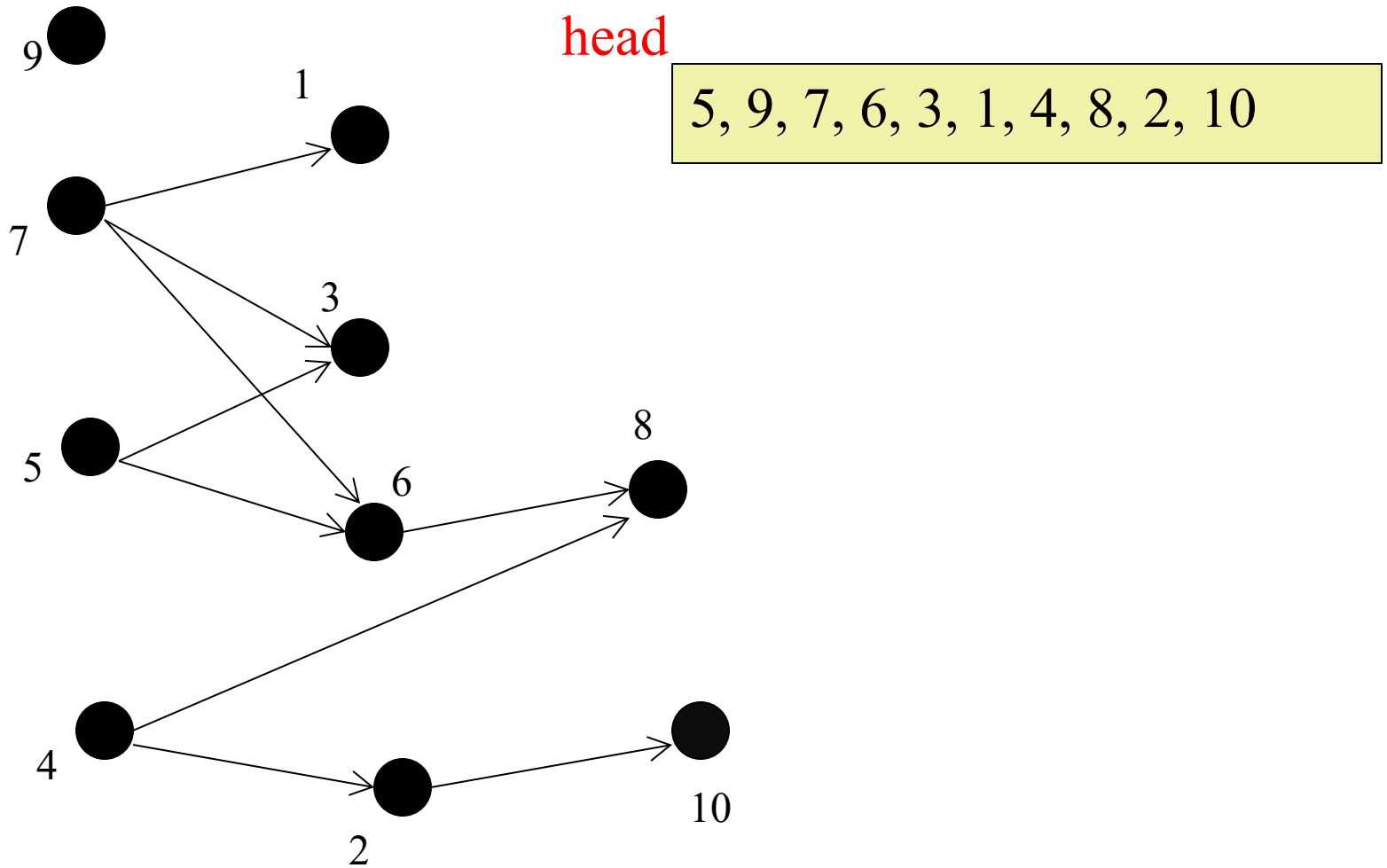
Example



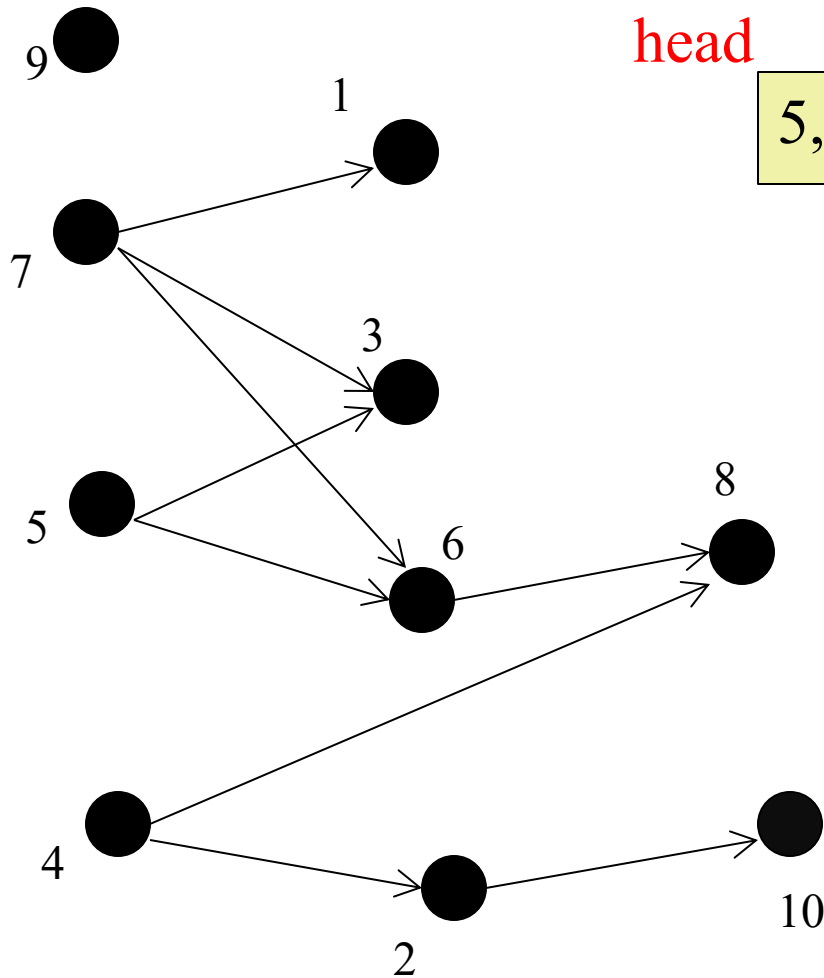
Example



Example



Topological Sort: Summary



5, 9, 7, 6, 3, 1, 4, 8, 2, 10

The final order or jobs

Time complexity = DFS
complexity $O(V + E)$

**There can be many
orders that meet the
requirements**

Quiz 5 next Tuesday

- Quiz 5 will be held next Nov 26
- The quiz will primarily focus on Graph algorithms covered BFS and DFS
- My office hours at 1pm today is cancelled.
- Assignment 2 due Tuesday (Nov 26) midnight



That's all Folks!
Any Question?