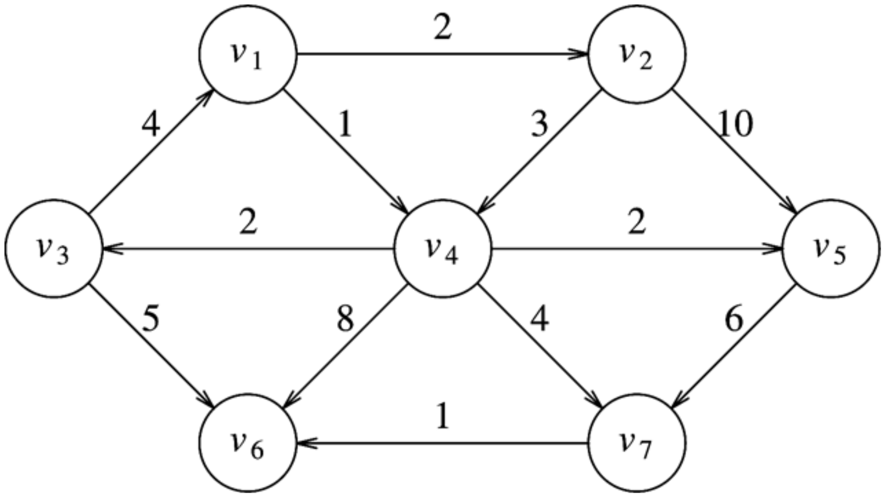
CS 10C: INTRO TO DATA STRUCTURES AND ALGORITHMS

Ryan Rusich
rusichr@cs.ucr.edu
Department of Computer Science
and Engineering
UC Riverside

Graphs



Graph

- A Graph $G = (V,E)$
  - A set V of vertices and a set E edges
- A graph is a way representing connections/relationships between pairs of objects in V
- Each edge is a pair (v,w), where $v, w \in V$.
- Edges are either directed or undirected
  - Directed referred to as *ordered*
    - Directed graphs are called Digraphs
    - Directed graphs with no cycles (acyclic) are called DAGs
  - Undirected referred to as unordered

Graph

- An edge (v,v) is a loop
- Vertices can have incoming and outgoing edges
  - indegree – number of incoming edges of a vertex v
  - outdegree – number of outgoing edges of a vertex v
- A path in a graph is a sequence of vertices $w_1, w_2, w_3, \ldots, w_N$ such that $(w_i, w_{i+1}) \in E$ for $1 \leq i < N$
  - A **simple path** is a path where all vertices are distinct except possibly first/last
  - **path length:** $N-1$ for N vertices
- Edges can have an associated cost called the **weight**.

Graphs - Definitions

- An **undirected graph** is **connected** if there is a path from every vertex to every other vertex
- A **directed graph** that is connected is called **strongly connected.**
- **Weakly connected** - If directed graph is not strongly connected, but removing direction makes graph connected
- **Complete graph** – edge between every pair of vertices.

Graph Representations

- Adjacency matrix – $|V|^2$ matrix, with $\mathbf{1}$ if there is an edge between two vertices, $\mathbf{0}$ otherwise.
  - Good for dense graphs.
  - Wasted space if not dense.
  - If graph is sparse, adjacency list is better.

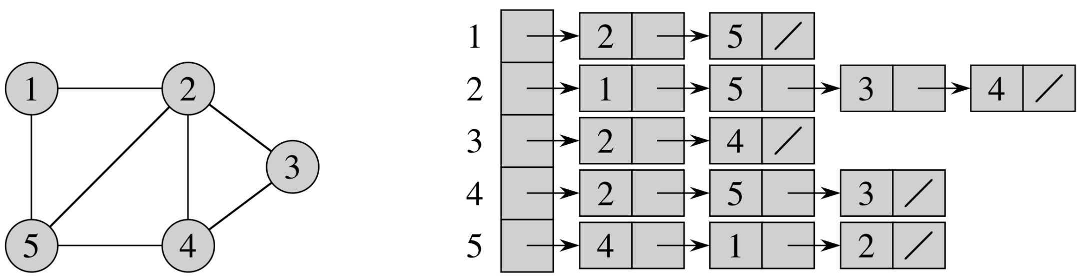
$$|E| = \Theta(|V|^2)$$

- Adjacency list – for each vertex, store a list of vertices that share an edge
  - Example…

Adjacency list

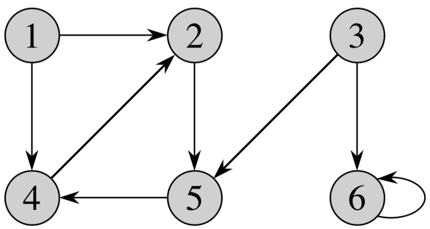
1	2, 4, 3
2	4, 5
3	6
4	6, 7, 3
5	4, 7
6	(empty)
7	6

Undirected Graph

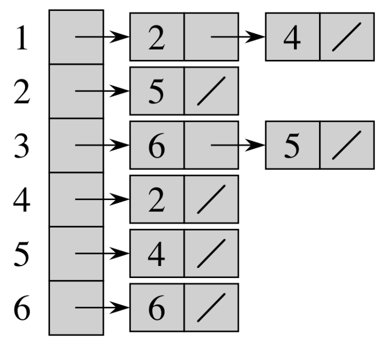


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Directed Graph



(a)

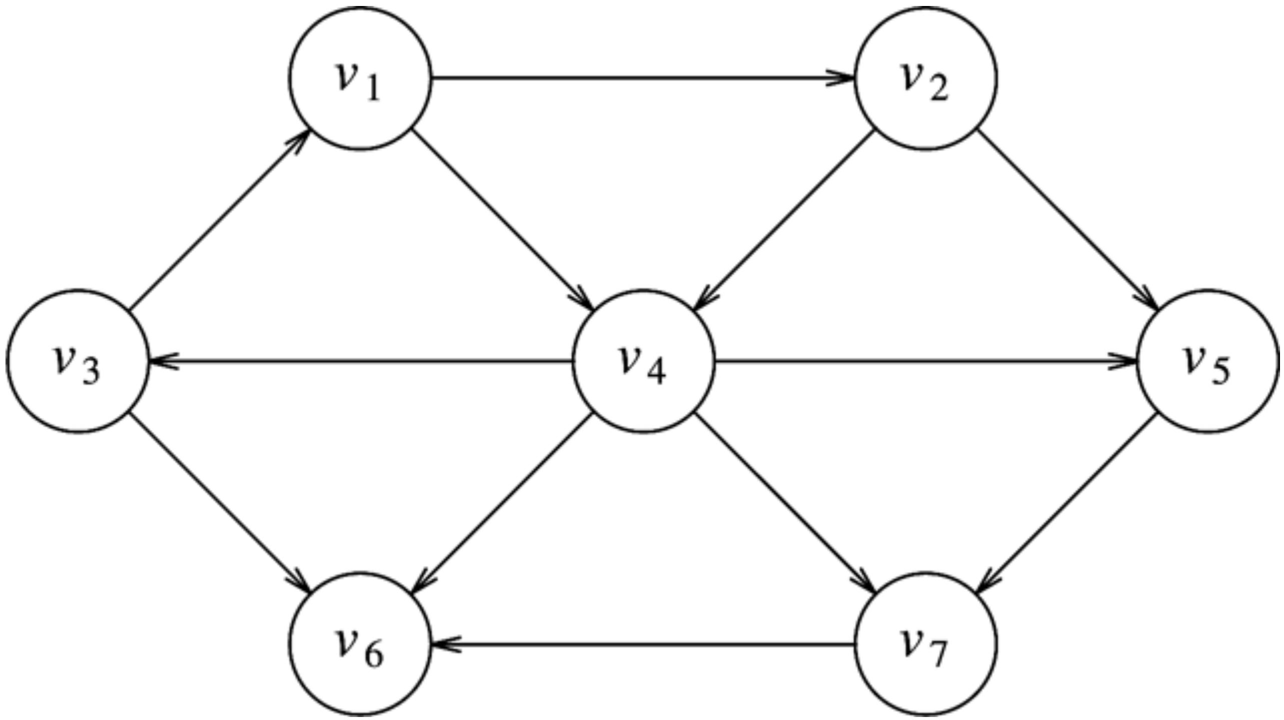


(b)

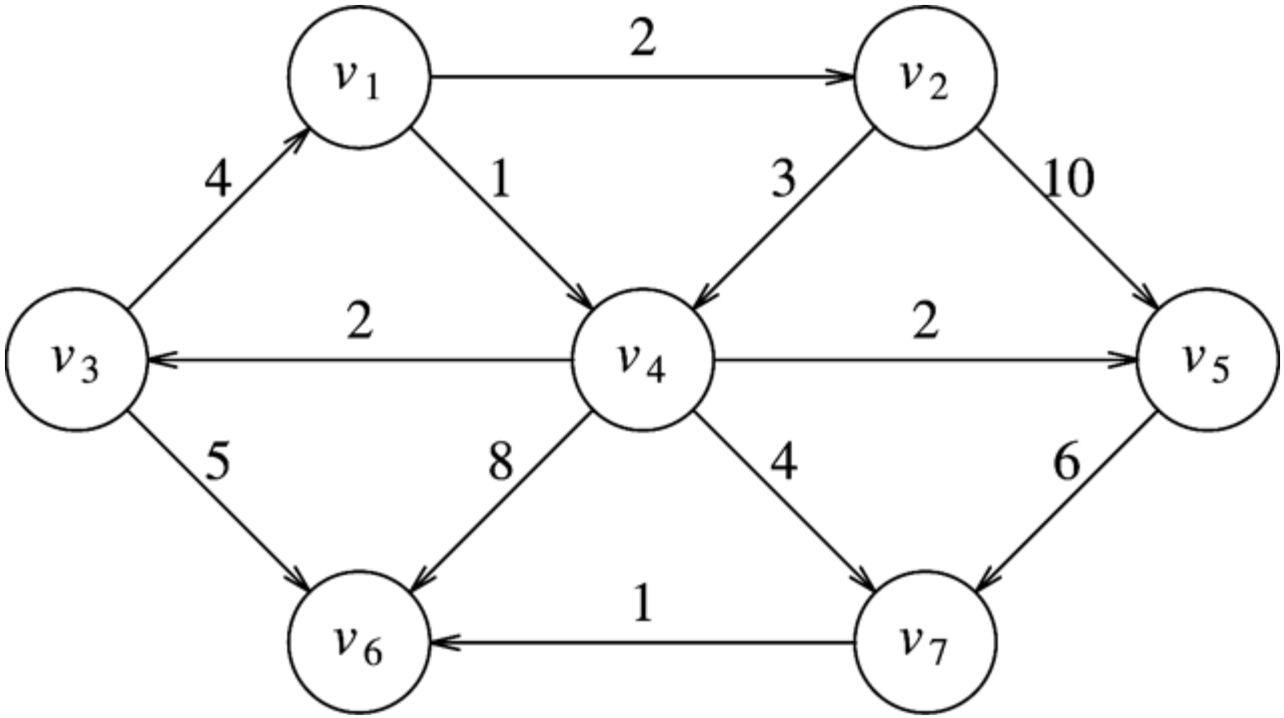
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

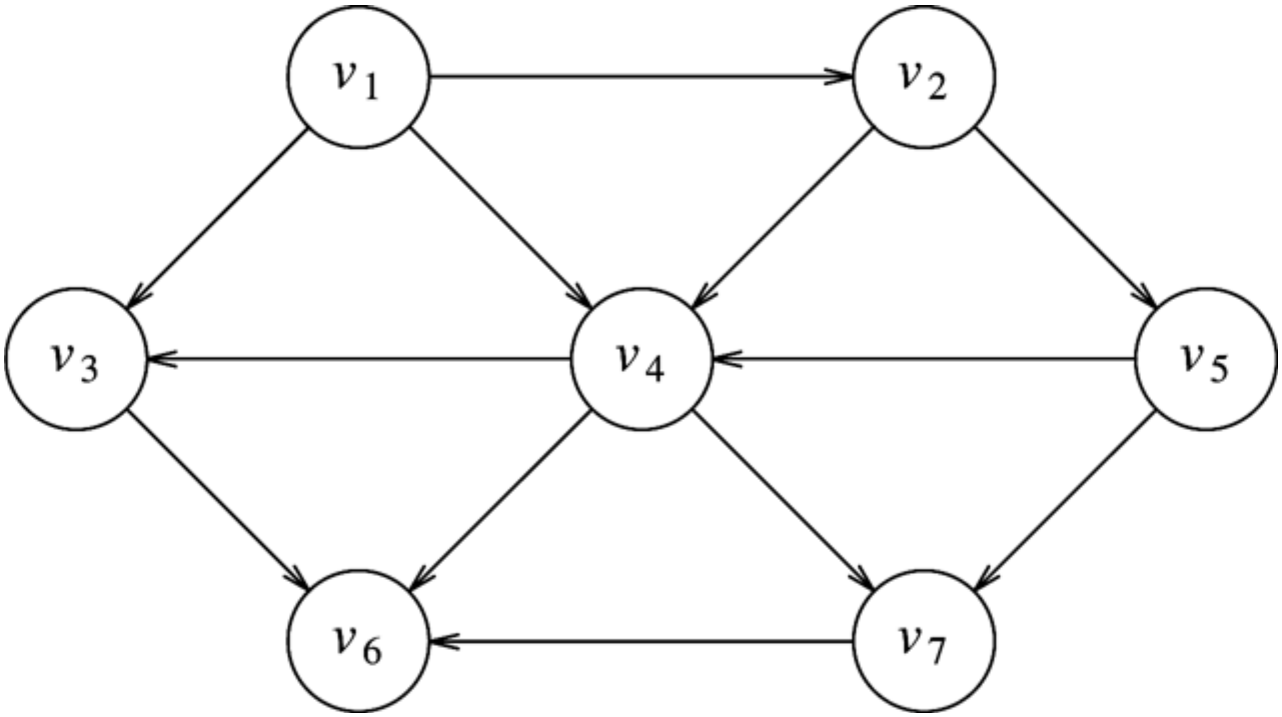
Directed Graph – Unweighted

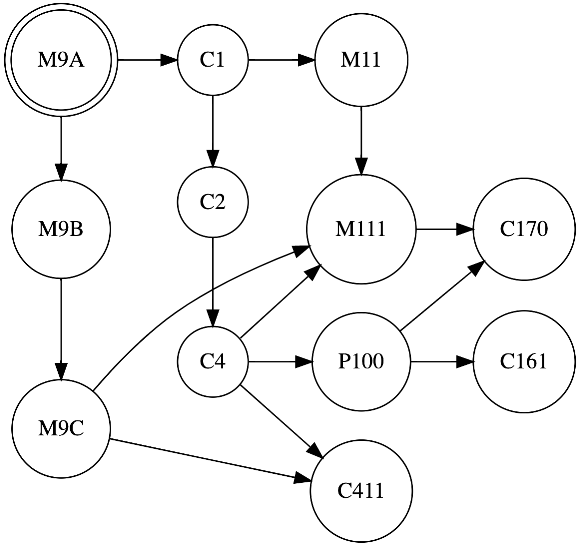


Directed Graph - Weighted



Directed Acyclic Graph

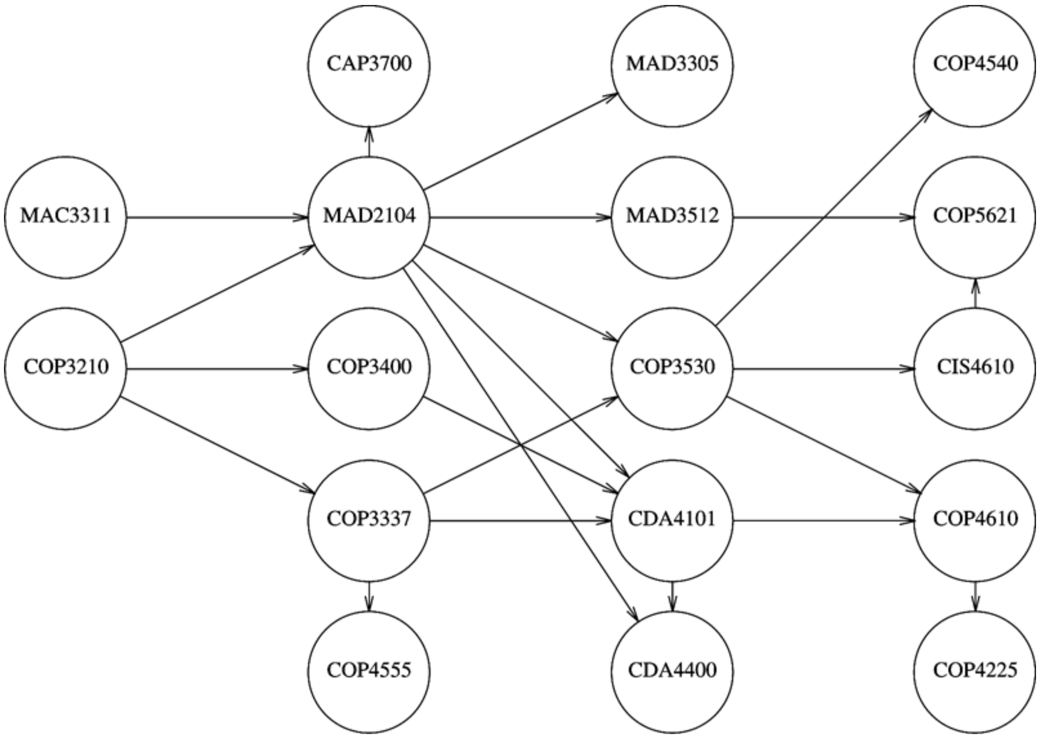



M9A
C1
M11
M9B
C2
M111
C170
C4
P100
C161
M9C
C411

Topological Sort

- Linear ordering of vertices for a DAG (Directed Acyclic Graph).
- Every vertex comes before all vertices to which it has outgoing edges.
- Every DAG has at least 1 topological sort
  - 1 indicates a Hamiltonian path
  - 2 or more, no Hamiltonian path
  - Hamiltonian path – every vertex is visited exactly once.
- Topological Sort - used in scheduling jobs or tasks

Scheduling

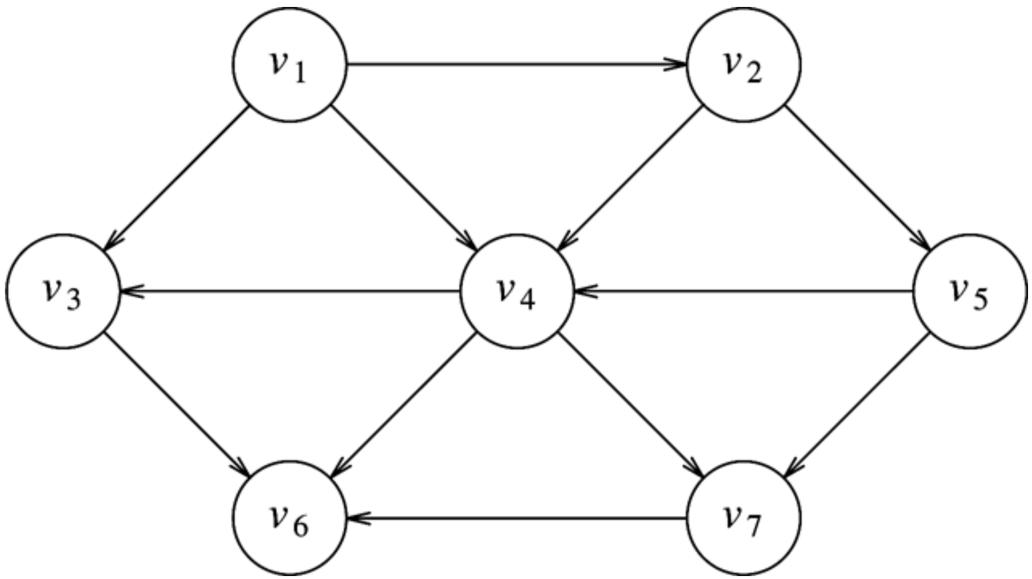

CAP3700
MAD3305
COP4540
MAC3311
MAD2104
MAD3512
COP5621
COP3210
COP3400
COP3530
CIS4610
COP3337
CDA4101
COP4610
COP4555
CDA4400
COP4225

Topological Sort

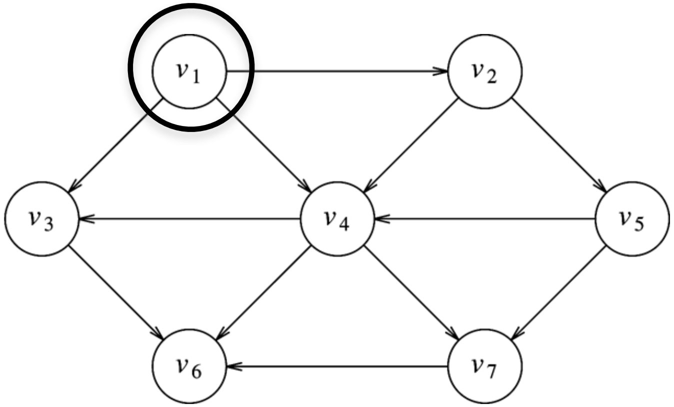
- **Simple algorithm:**
  1. Find an initial vertex v with no incoming edges
  2. Print v
  3. Remove v from graph
  4. Remove *v's* outgoing edges from graph, updating effected vertices
  5. Iterate (steps 1-4) over the remaining graph

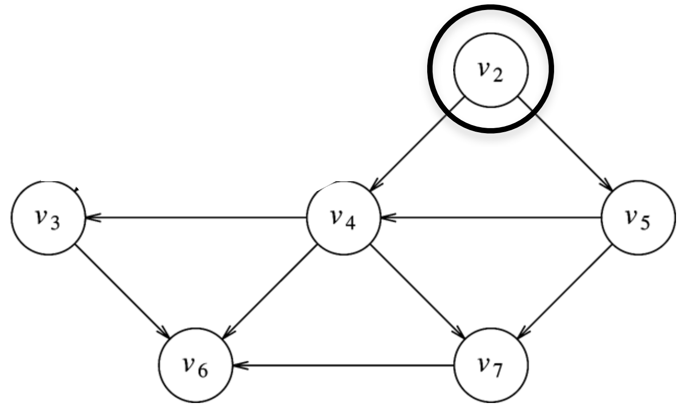
  Running Time : $O(|V|^2)$
  - Finding a vertex with in-degree zero, linear scan of V
  - There are $|V|$ calls to do this
- Speed up:
  - Use a Queue to store vertices with in-degree zero
  - Only have to remove the front of Queue, skip linear scan of V
  - **Running Time:** $O(|E| + |V|)$

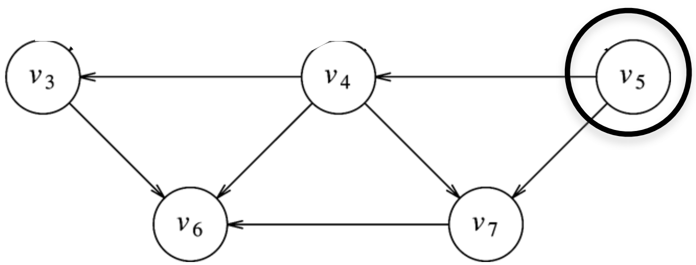
Topological Sort

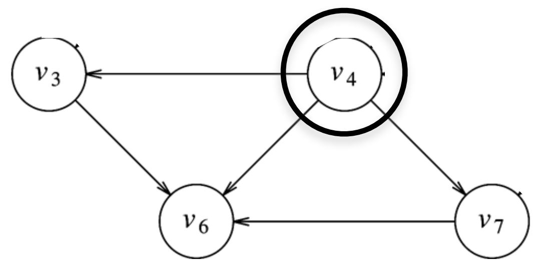


v1, v2, v5, v4, v3, v7, v6



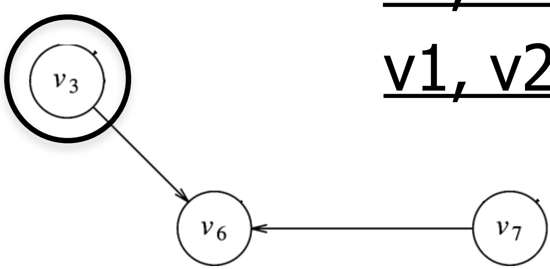


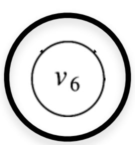




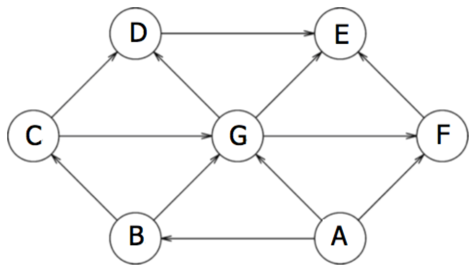
<u>v1, v2, v5, v4, v3, v7, v6</u>

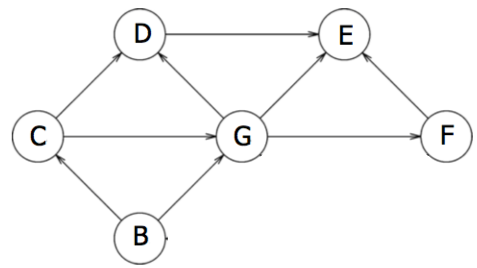
<u>v1, v2, v5, v4, v7, v3, v6</u>

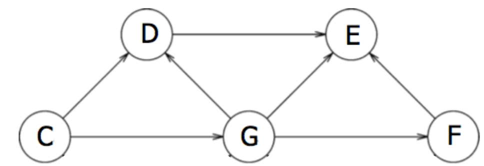


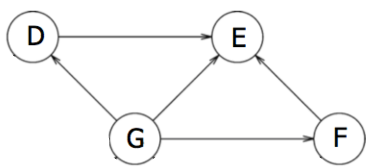


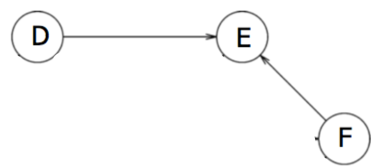
Topological Sort





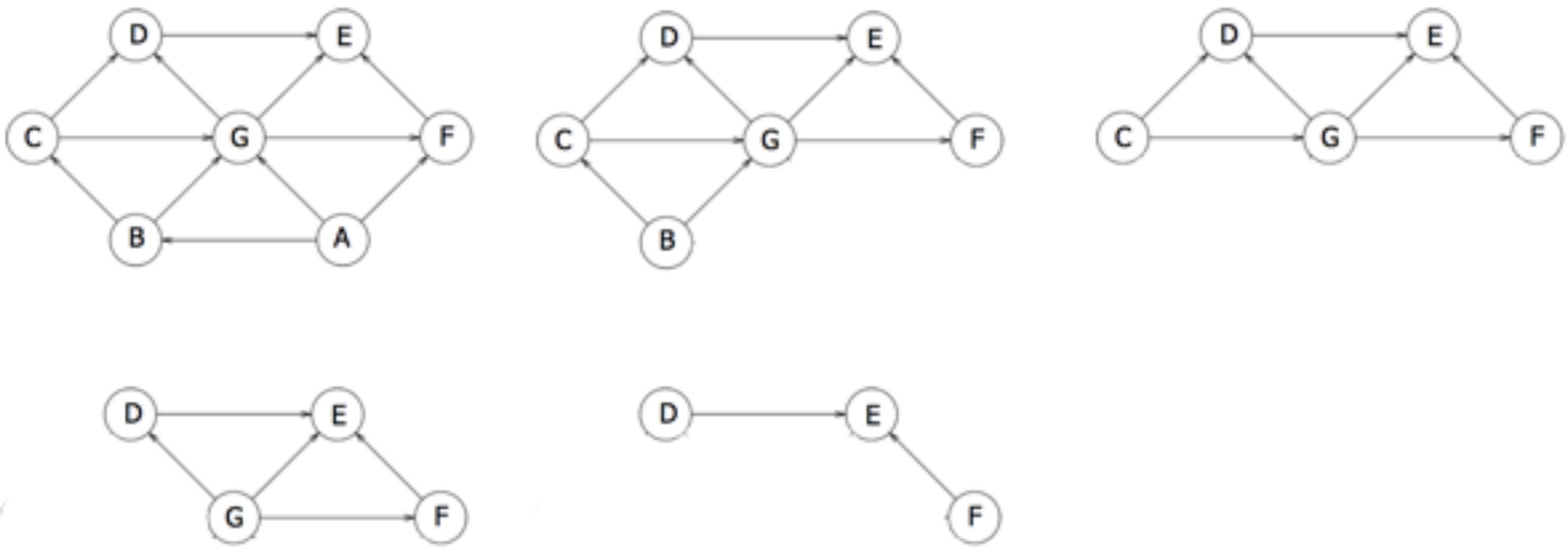






A, B, C, G, D, E, F

Topological Sort

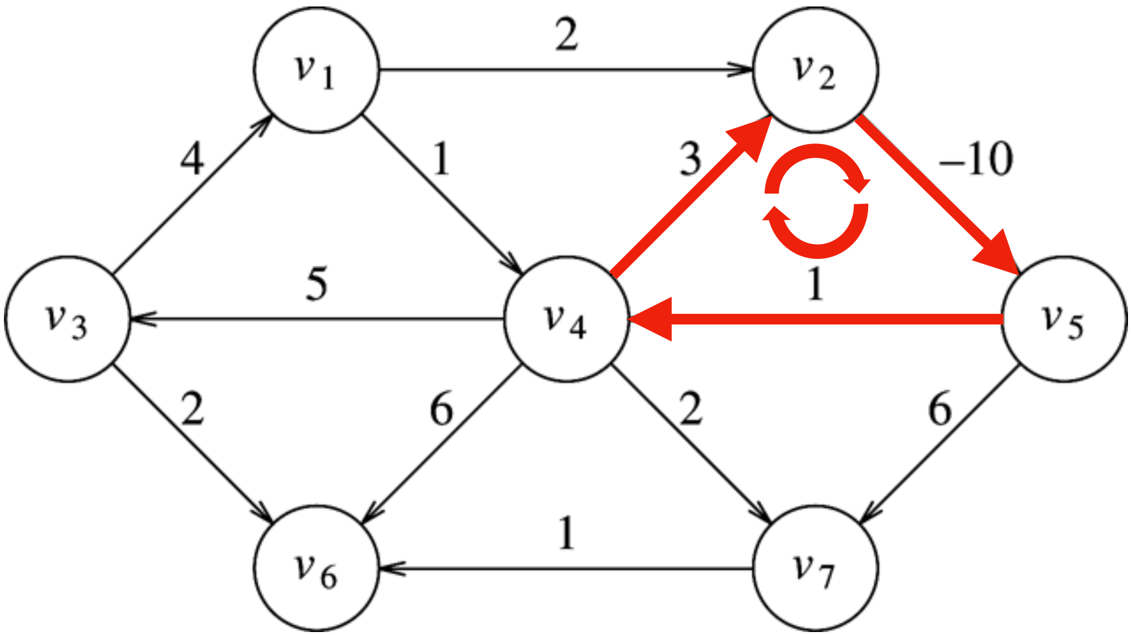


A, B, C, G, D, E, F

Single-Source Shortest Path

- **Problem:** find the path between two vertices that minimizes the number of edges or minimizes the sum of the weights of the edges.
  - Unweighted Graphs – number of edges
  - Weighted Graphs – sum of the costs
- **BFS** – solves for unweighted graphs
- **Dijkstra's** – solves for directed graph with positive weights
- **Bellman Ford** – solves for graphs that can include negative weights

Negative Cost Cycle



- Shortest Path is NOT defined.
- Reason: because you could complete cycle repeatedly and lower the cost.
- This is true for Dijkstra's and Bellman-Ford.

Single Pair Shortest Path Initialization Step

for each $u \in V - \{s\}$
$\quad u.d = \infty$
$s.d = 0$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

- Set all distances to infinite, since a path from source not yet found.
- Set distance to source equal to zero (no cost to go from a node to itself).

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

$\text{BFS}(V, E, s)$

for each $u \in V - \{s\}$

$\quad u.d = \infty$

$s.d = 0$

$Q = \emptyset$

$\text{Enqueue}(Q, s)$

while $Q \neq \emptyset$

$\quad u = \text{Dequeue}(Q)$

$\quad$ **for** each $v \in G.Adj[u]$

$\quad\quad$ **if** $v.d == \infty$

$\quad\quad\quad v.d = u.d + 1$

$\quad\quad\quad \text{Enqueue}(Q, v)$

(u, v)
if v.d == infinity
then update v.d to u.d + 1

(u, v)
if v.d == infinity
then update v.d to 0 + 1

(u, v)
if v.d == infinity
then update v.d to 1 + 1

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

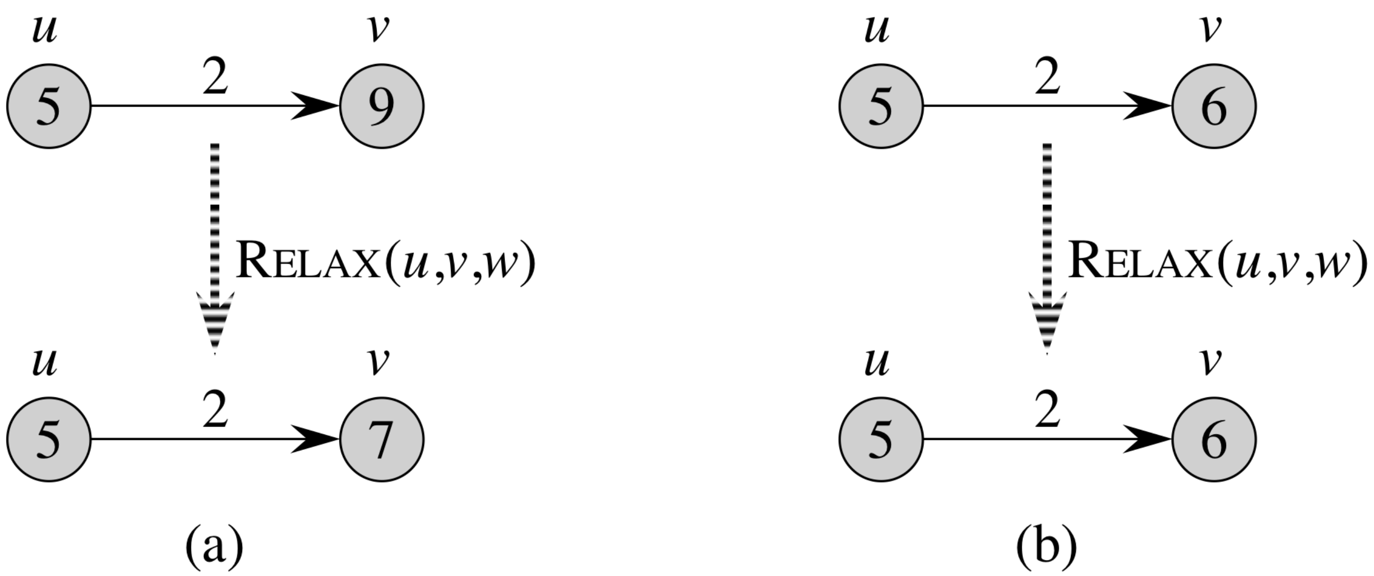
Dijkstra's - RELAX

$$
\begin{aligned}
&\textsc{Relax}(u, v, w) \\
&\quad \textbf{if } v.d > u.d + w(u, v) \\
&\qquad v.d = u.d + w(u, v) \\
&\qquad v.\pi = u
\end{aligned}
$$

- Single-pair shortest path update distance step for Dijkstra's is similar to BFS.
- Main difference is that the weights of the edges, e.g. $w(u, v)$, are positive values.
- Edge weights can be larger than 1.
- Dijkstra's - edge weights cannot be negative.

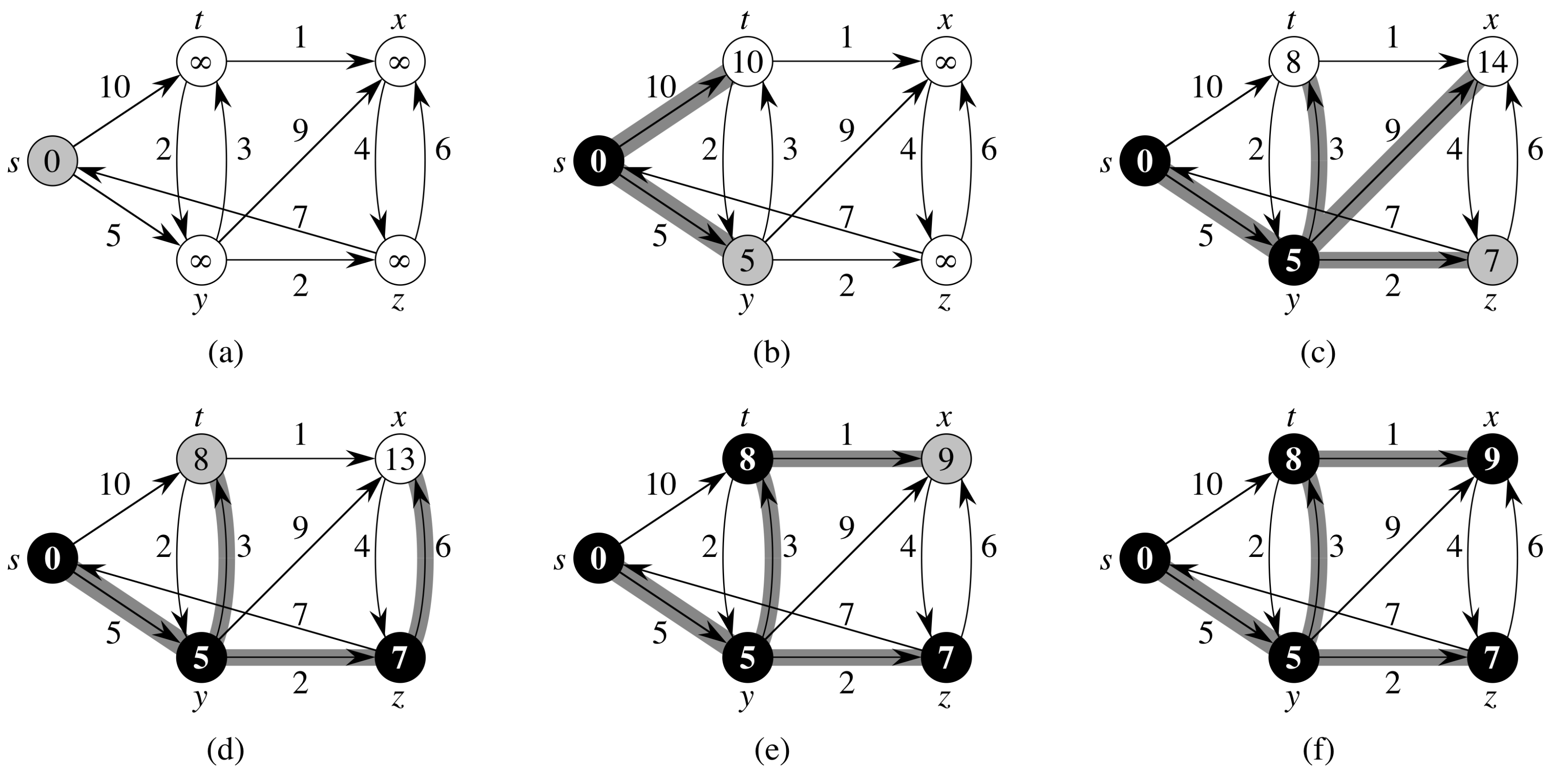
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

Dijkstra's - RELAX



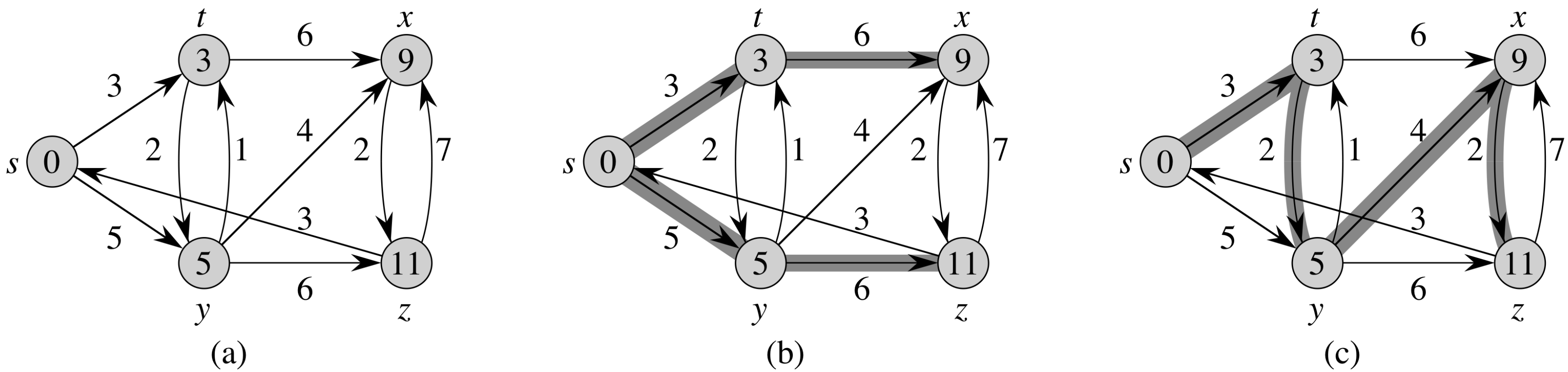
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

Dijkstra's - Example



Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

Non-Unique Shortest Path



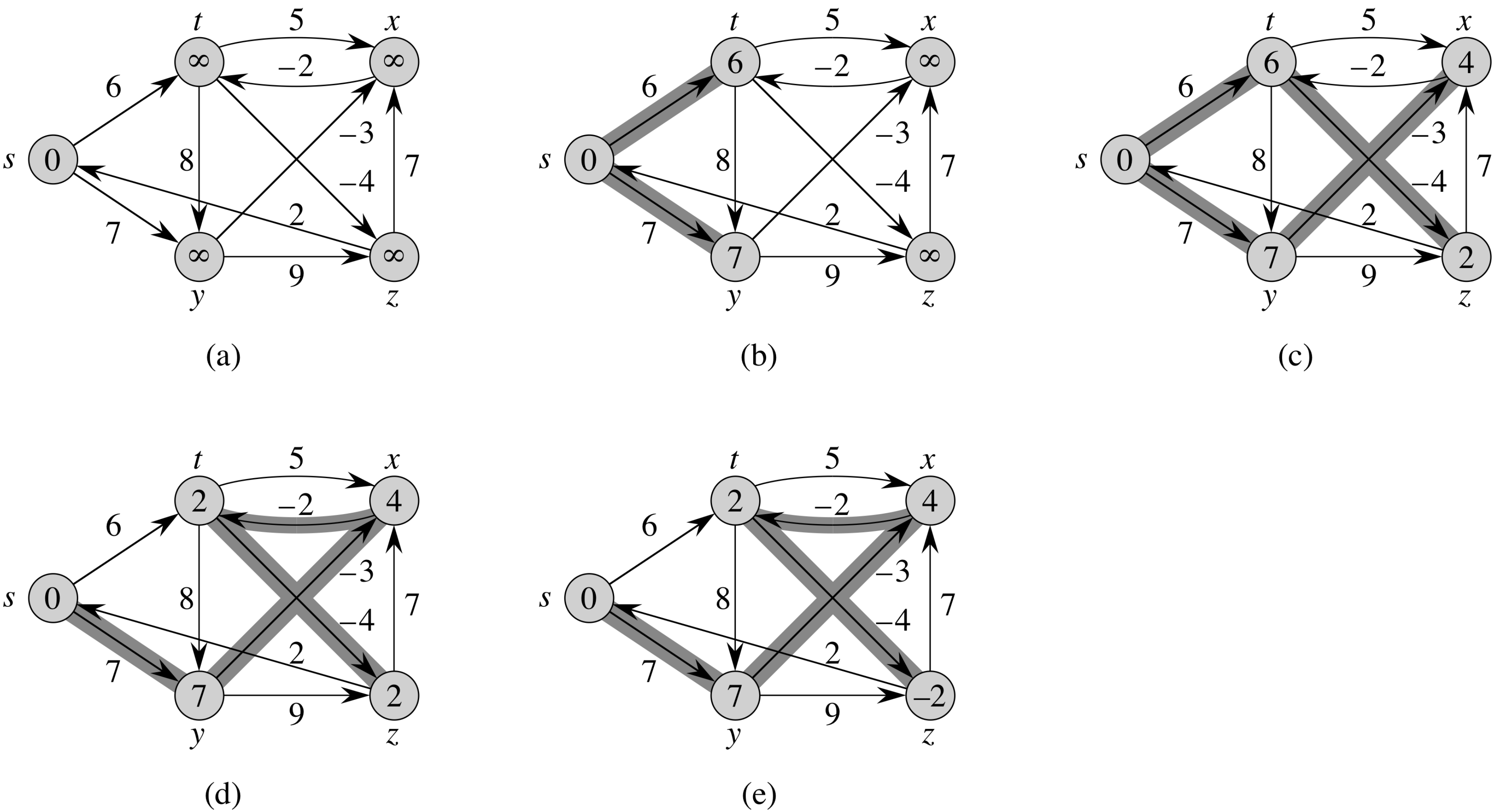
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

Bellman-Ford - CLRS

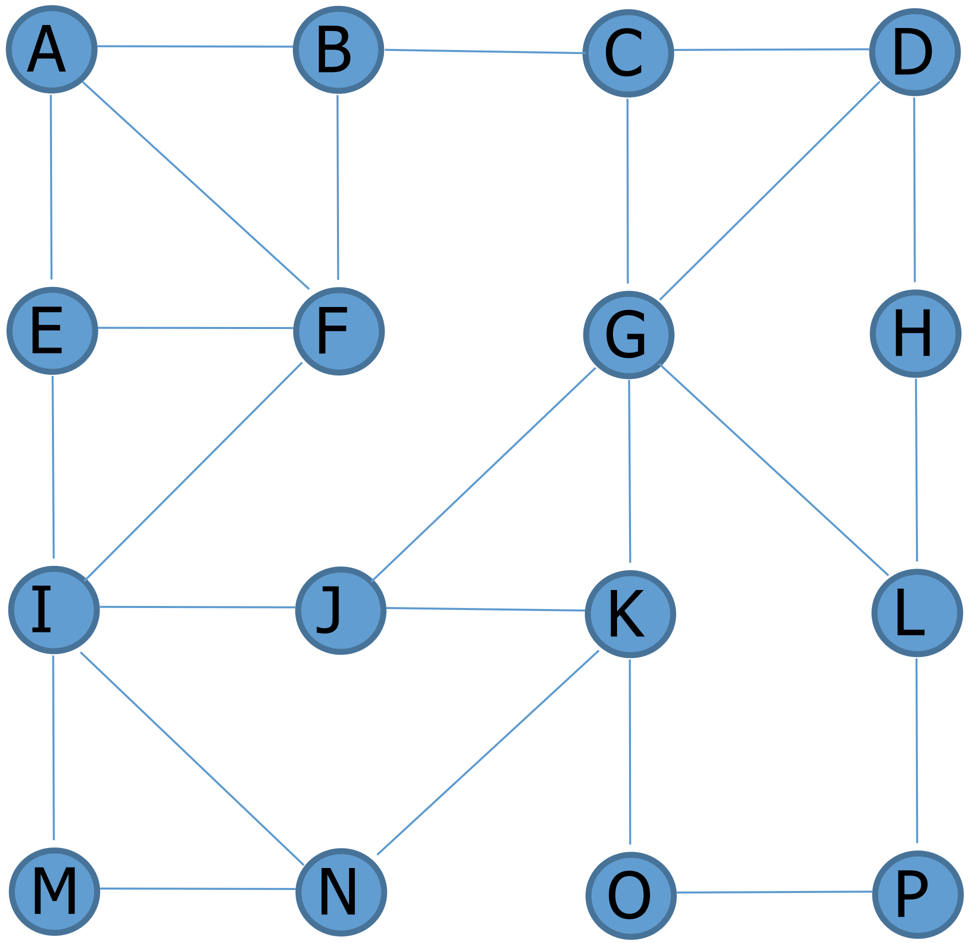
```
BELLMAN-FORD(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| - 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

Bellman Ford - Example

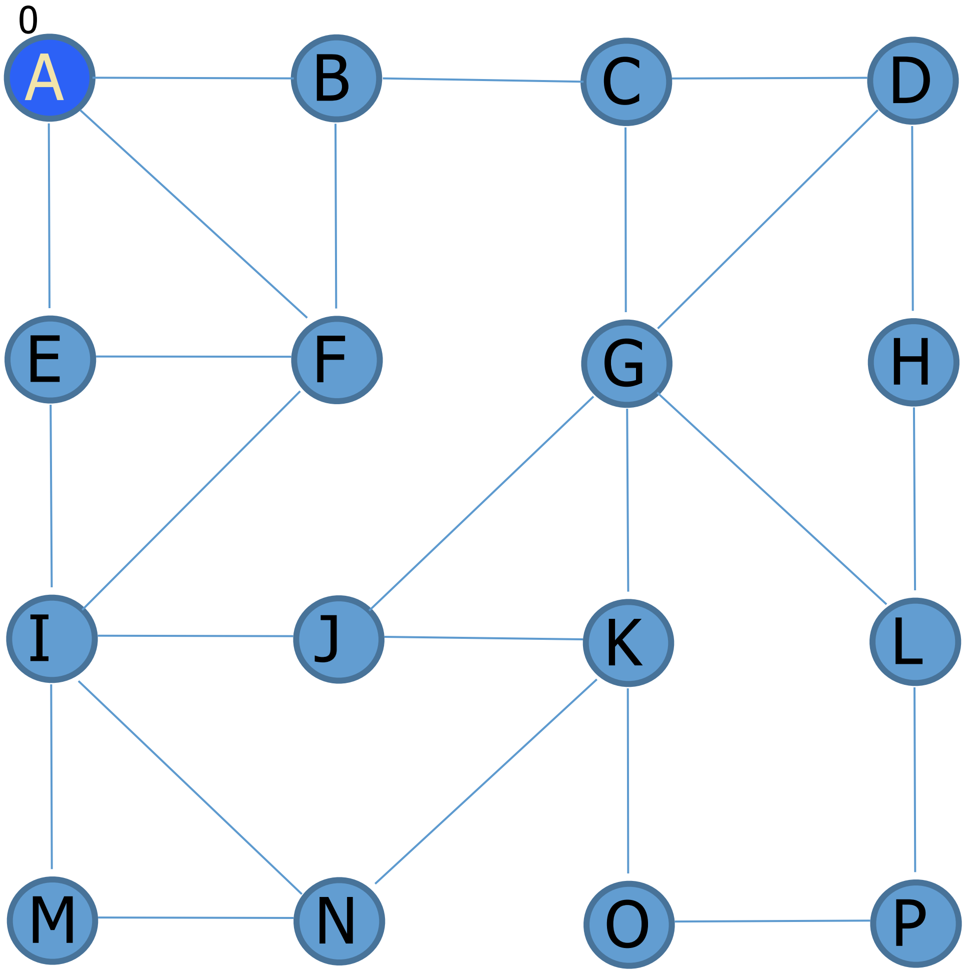


Breadth First Search (BFS)



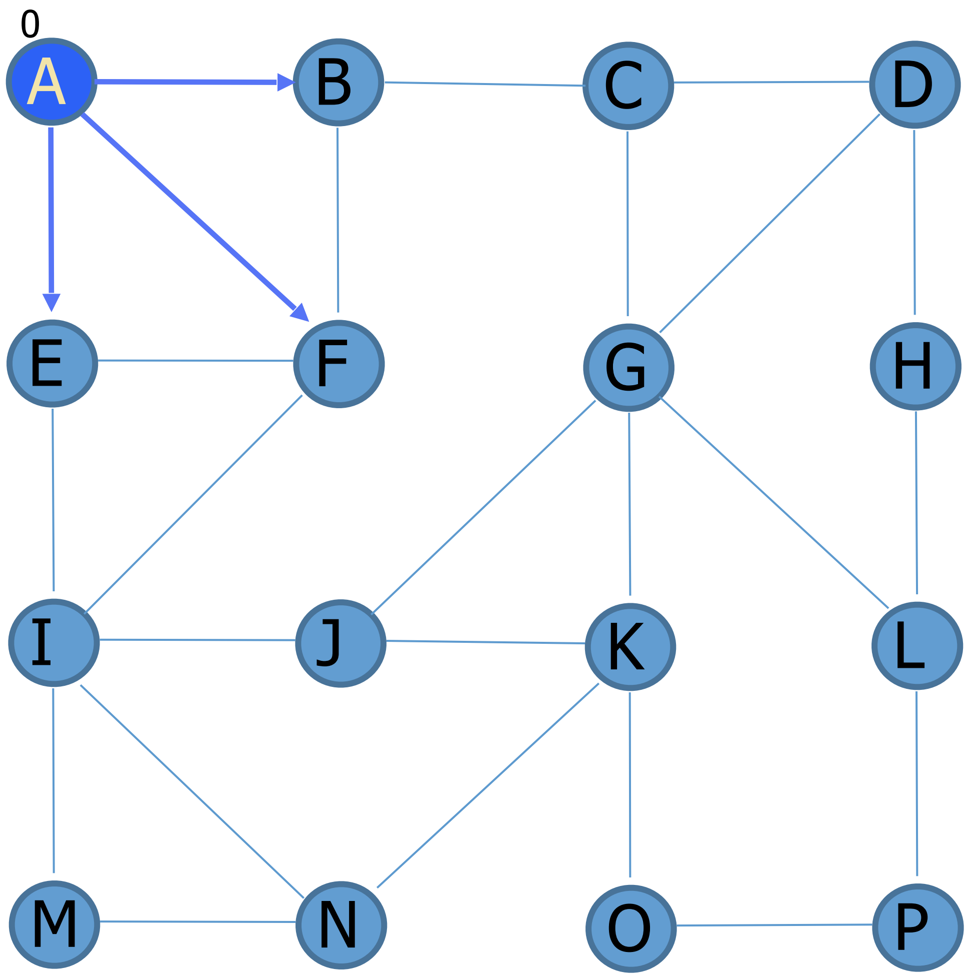
BFS

A visited



BFS

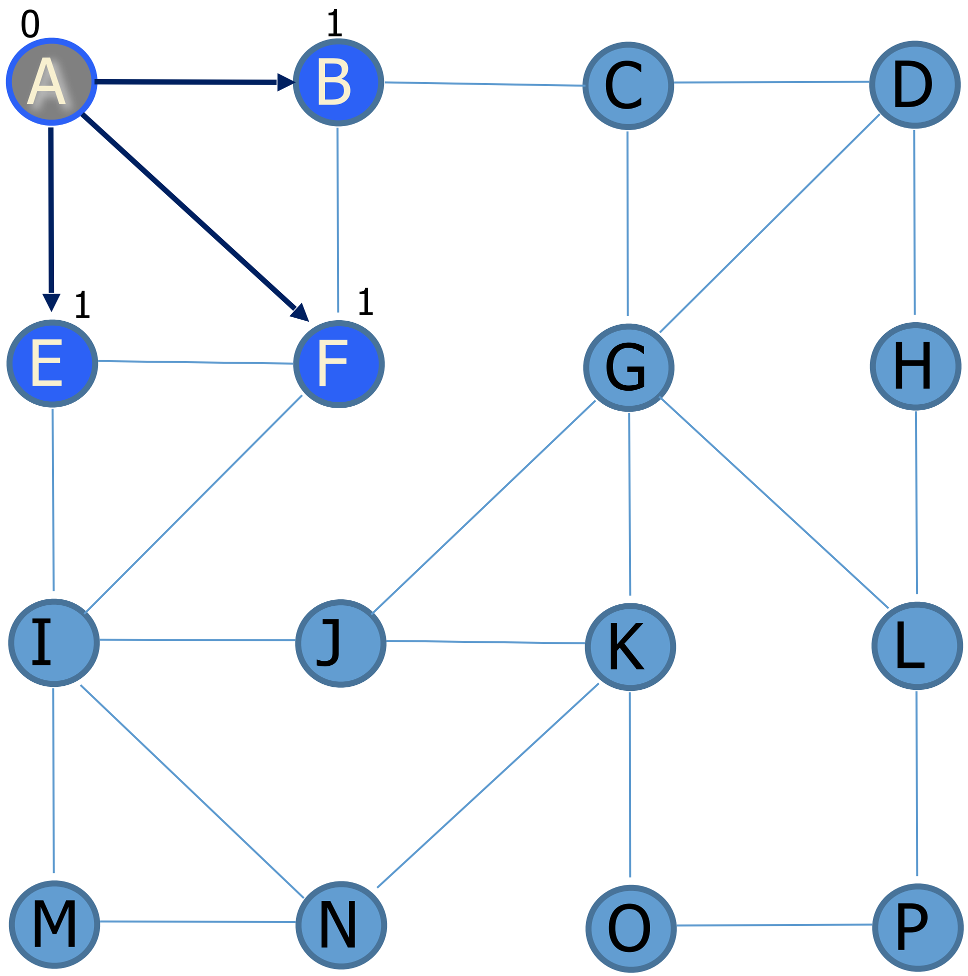
Explore A's neighbors



BFS

B, E, F visited

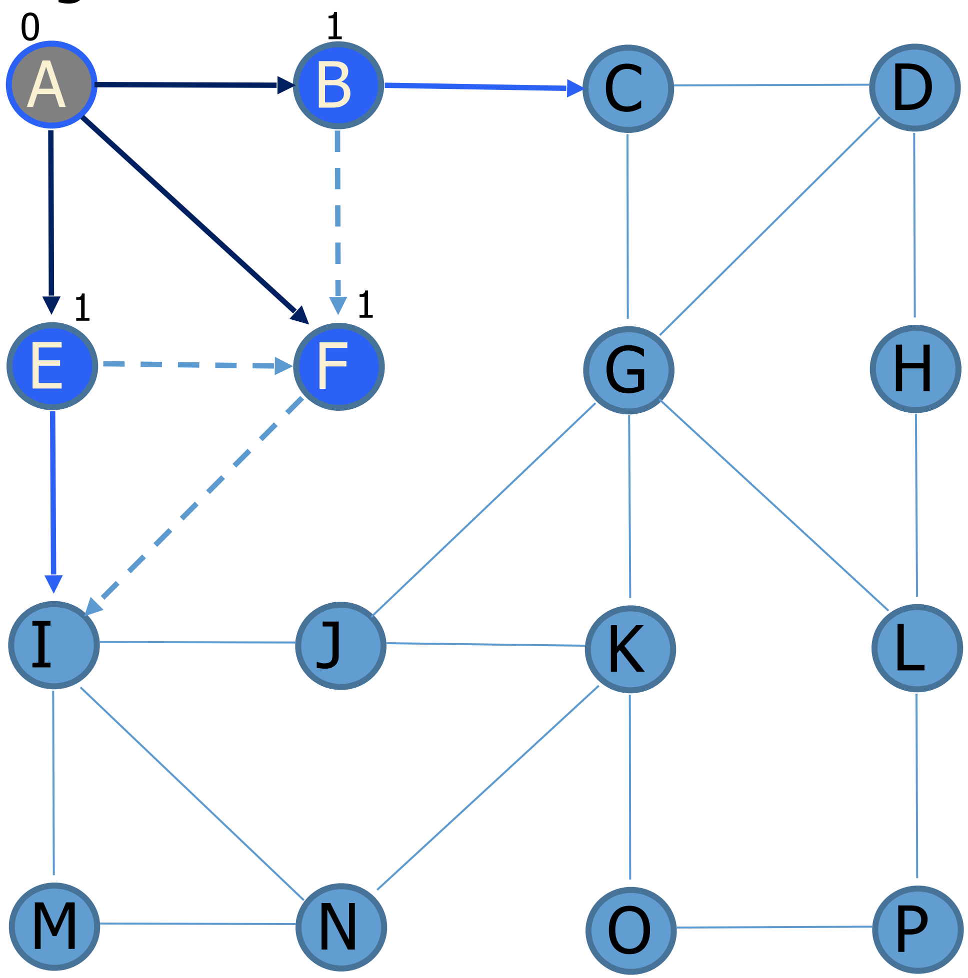
A is processed



BFS

Explore B's neighbors

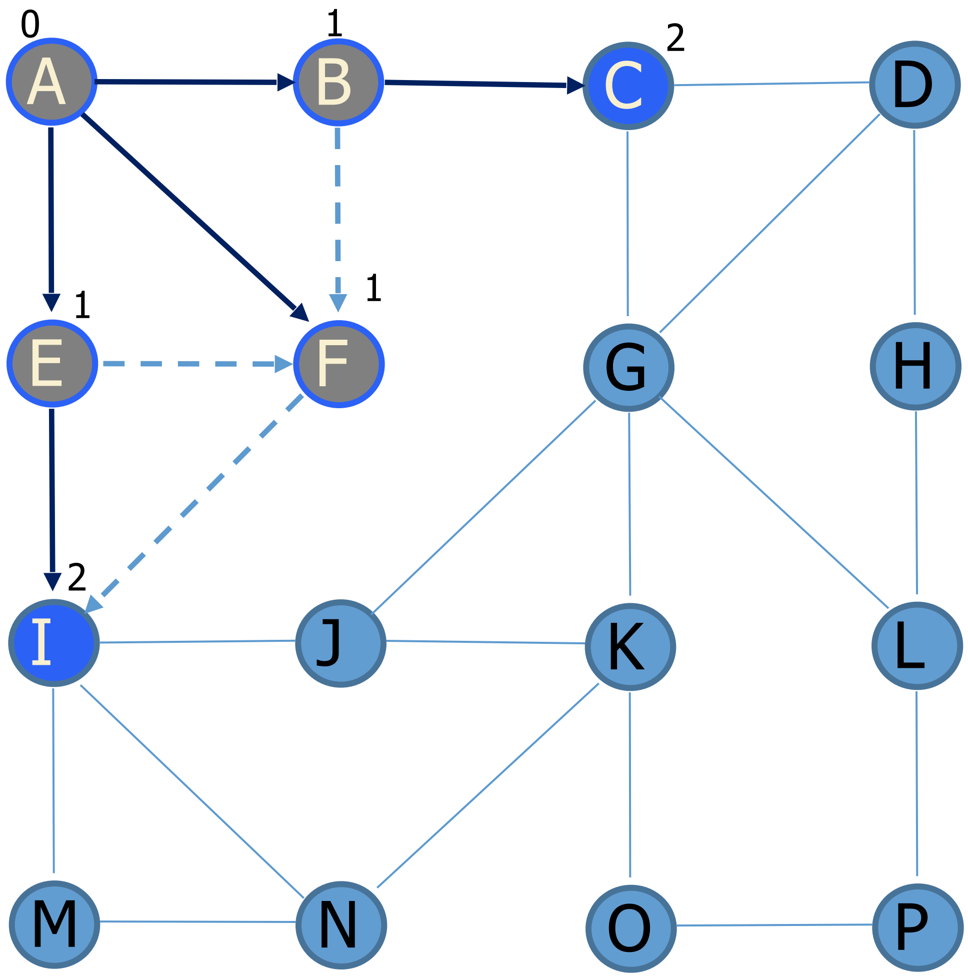
Explore E then F's neighbors



BFS

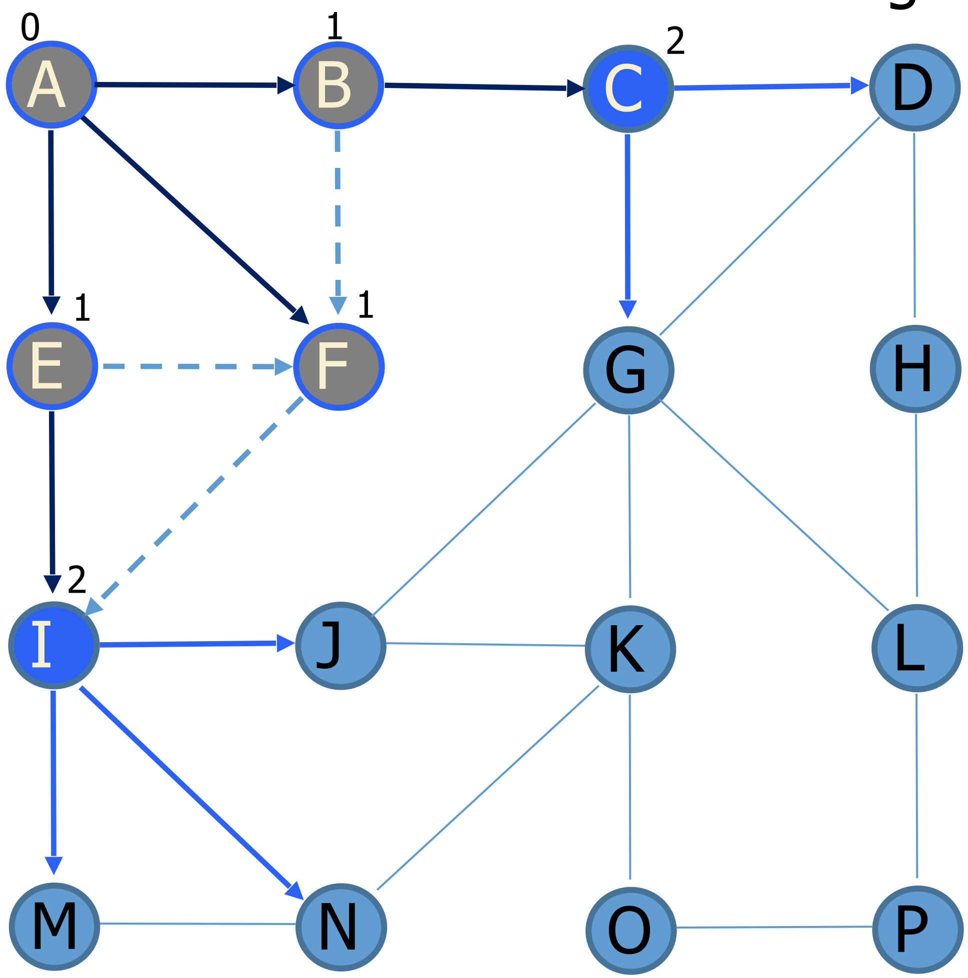
C, I
visited

B,E,F are
processed



BFS

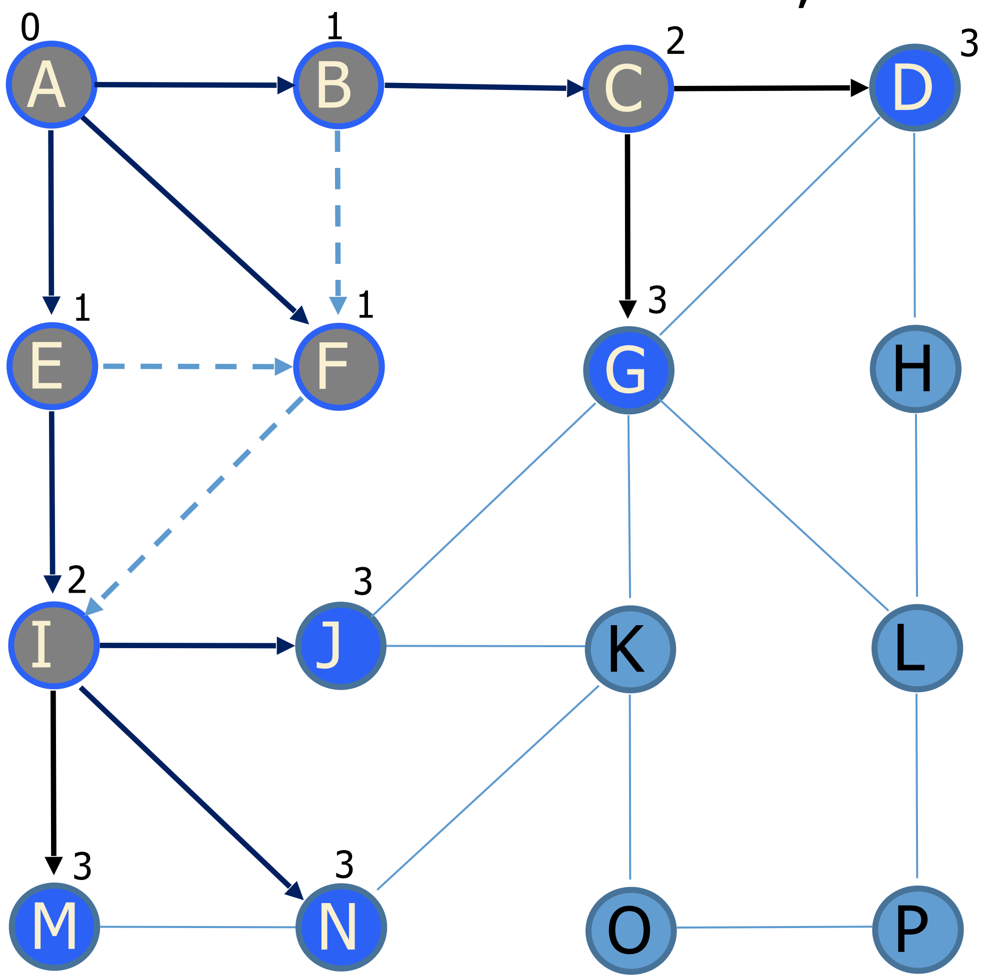
Explore C's neighbors



Explore I's neighbors

BFS

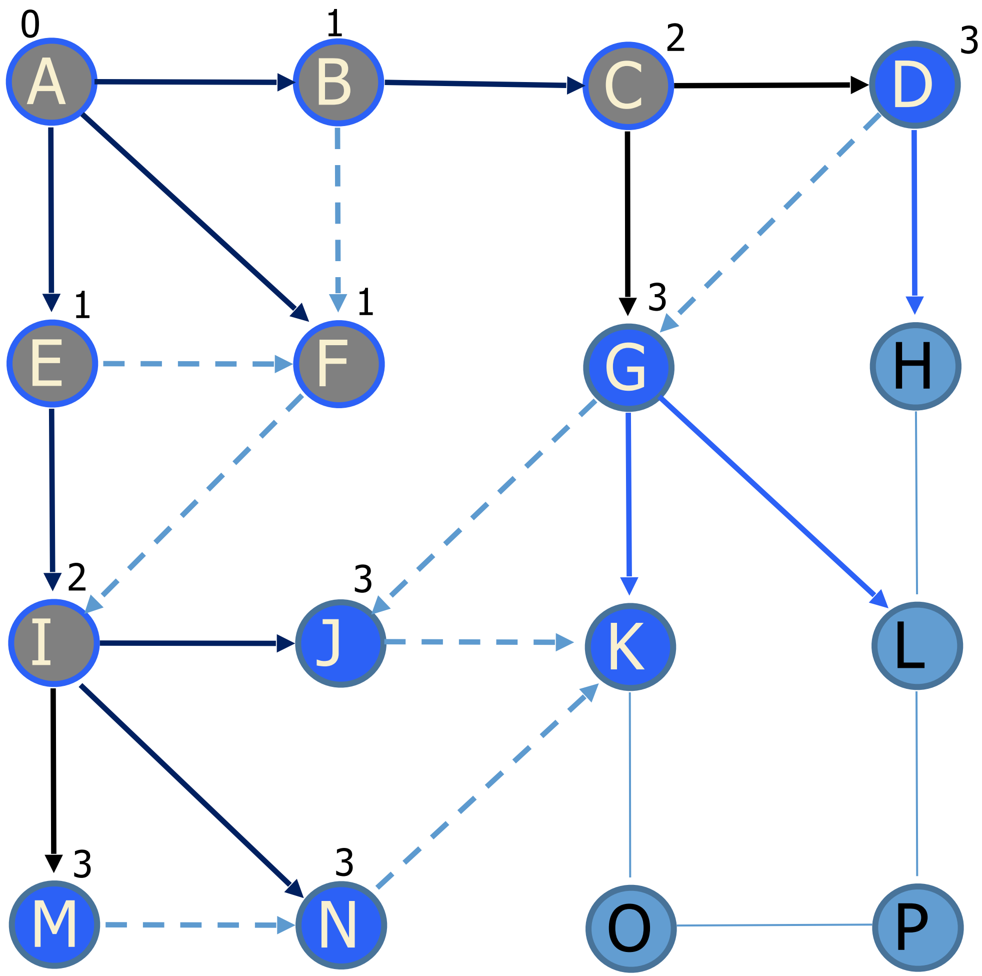
C processed
D,E visited



I processed
J,M,N
visited

BFS

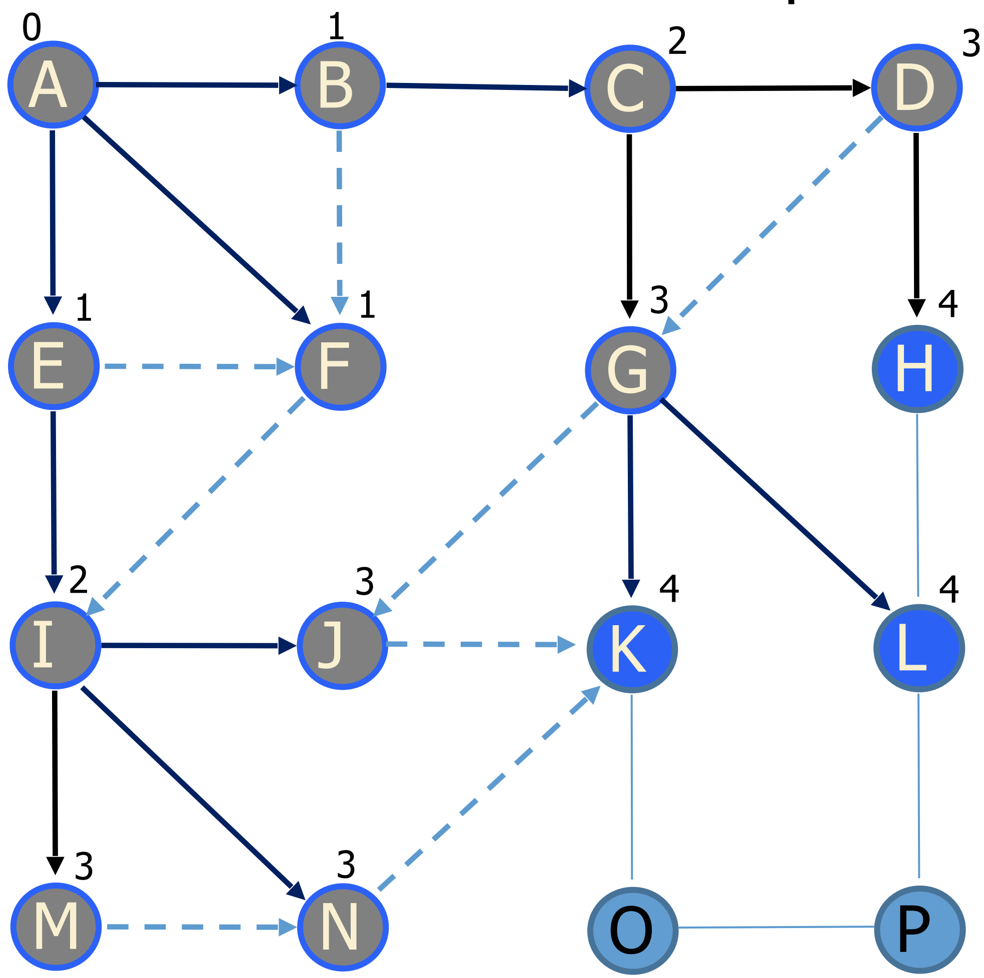
Explore D's,G's neighbors



Explore J's, M's, N's neighbors

BFS

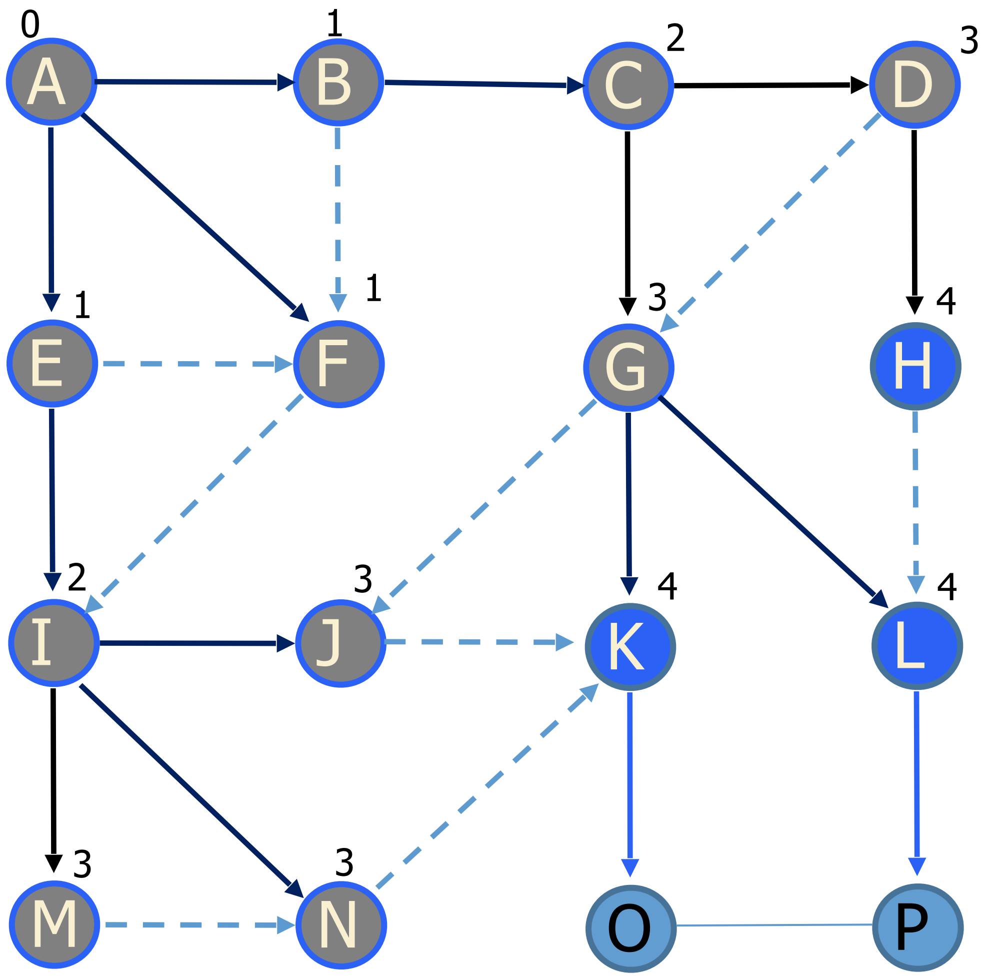
D, G processed



H, K, L visited

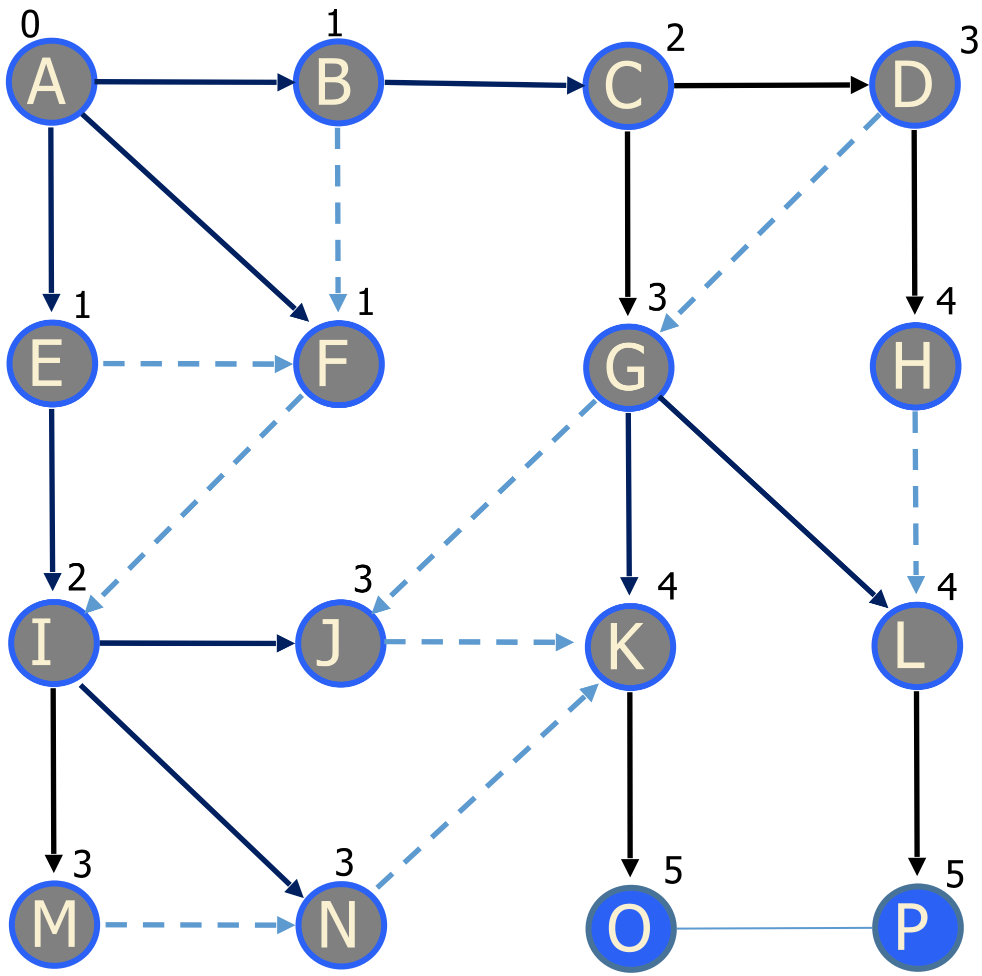
J, M, N processed

BFS



Explore H, K, L neighbors

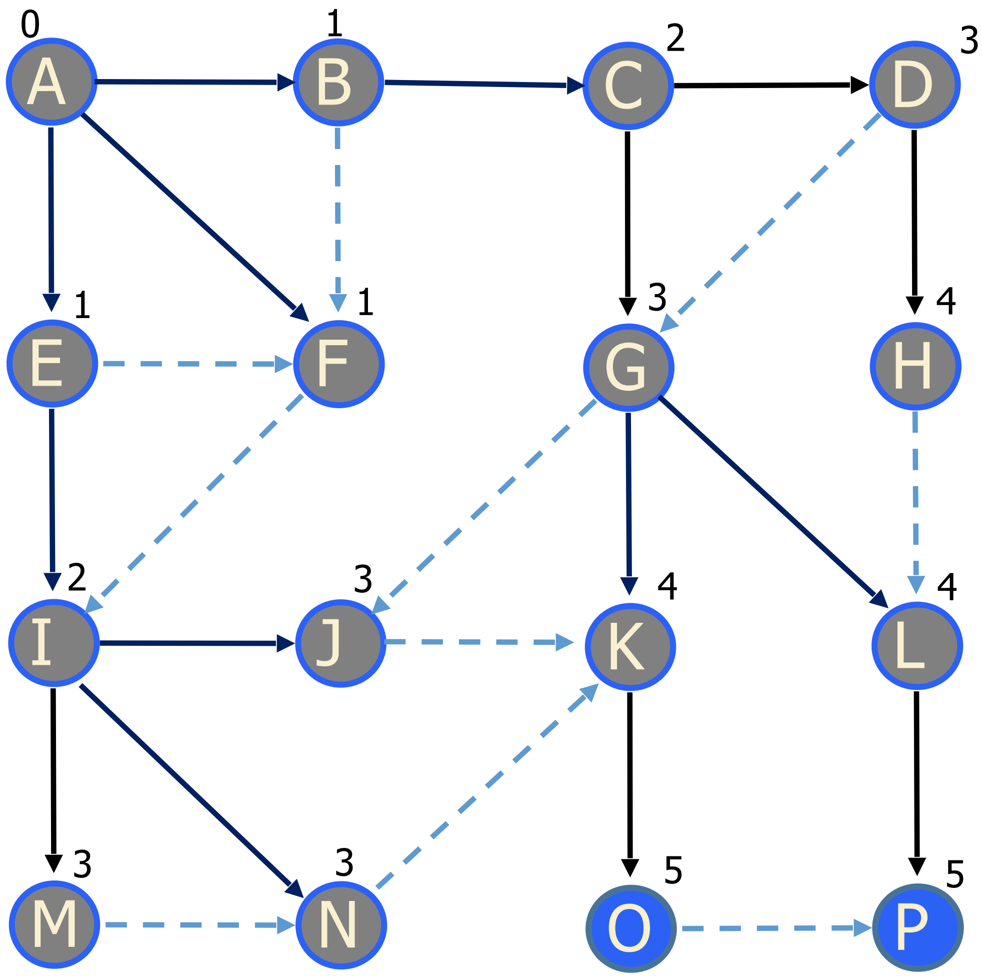
BFS



H, K, L processed

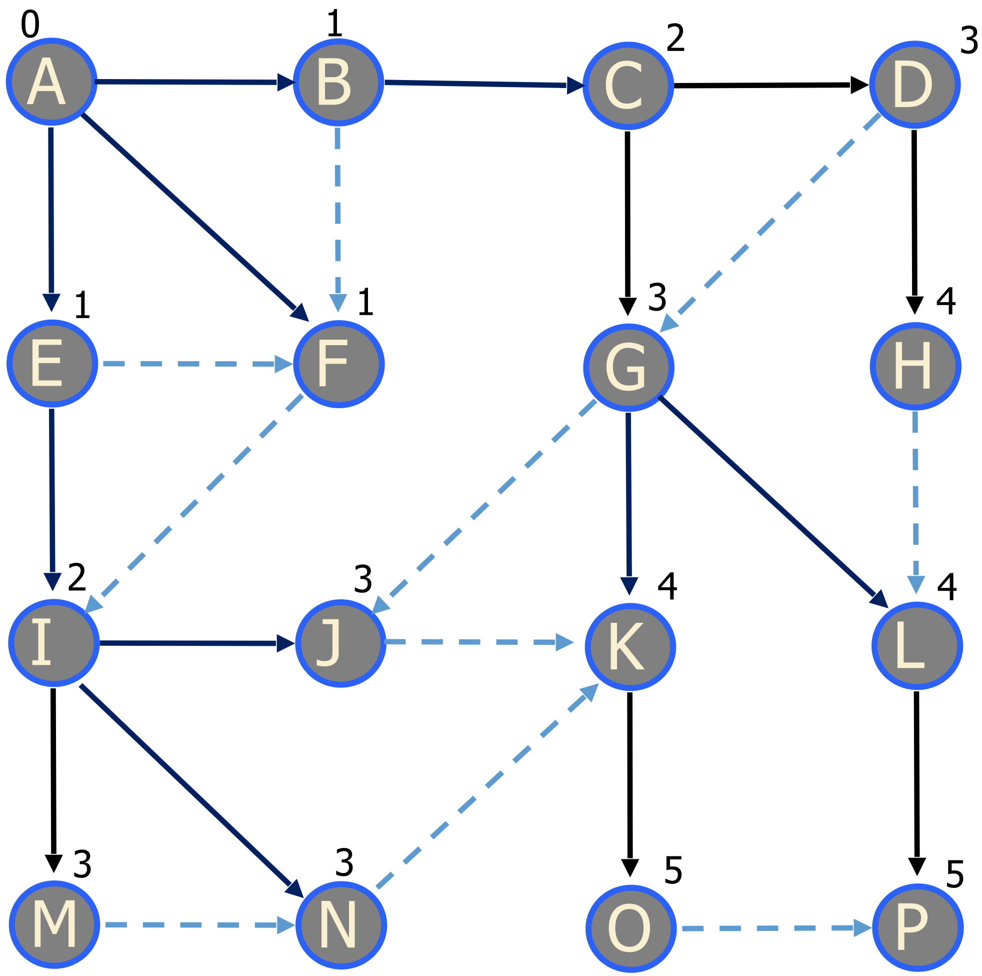
O,P visited

BFS



Explore O's neighbor

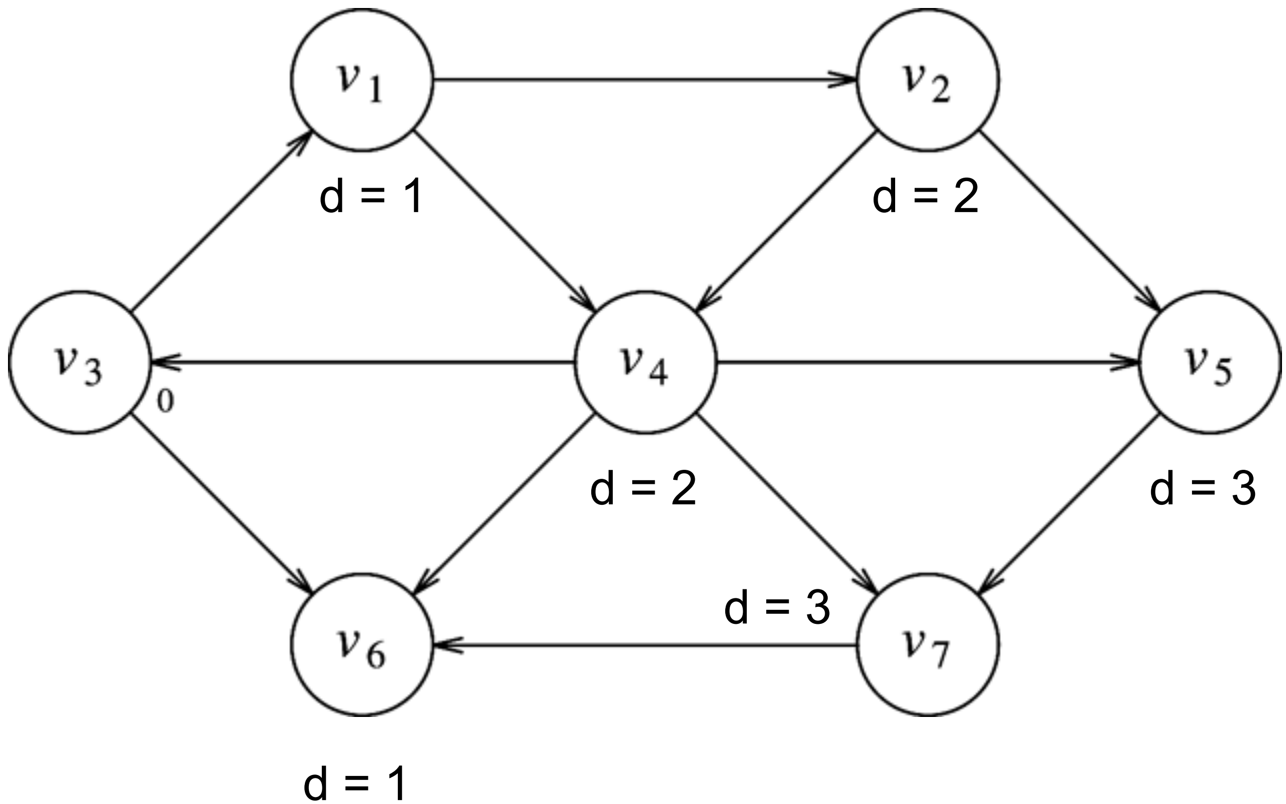
BFS



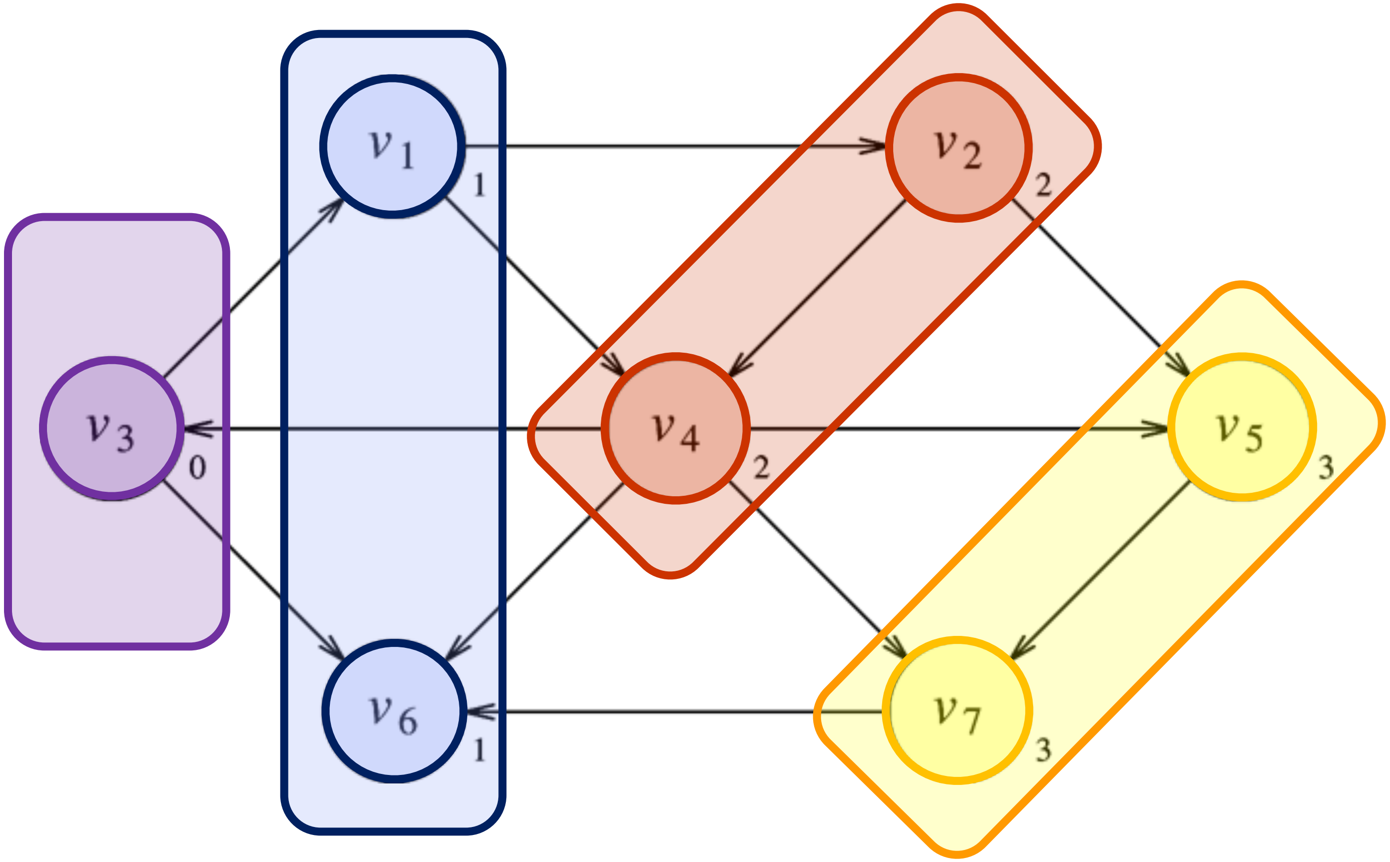
Done

Running Time = $O(|E|+|V|)$

BFS – In Class Activity

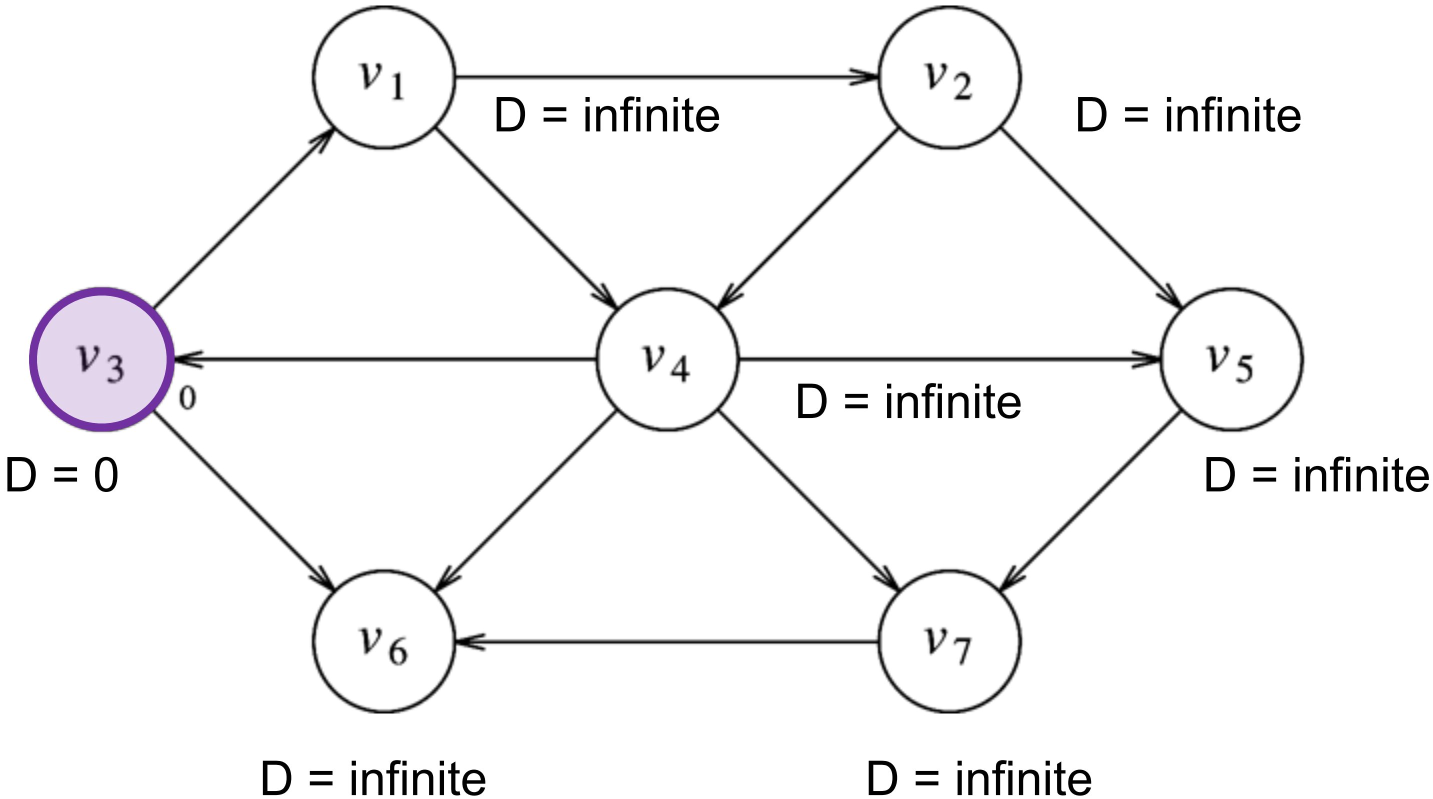


BFS – In Class Activity

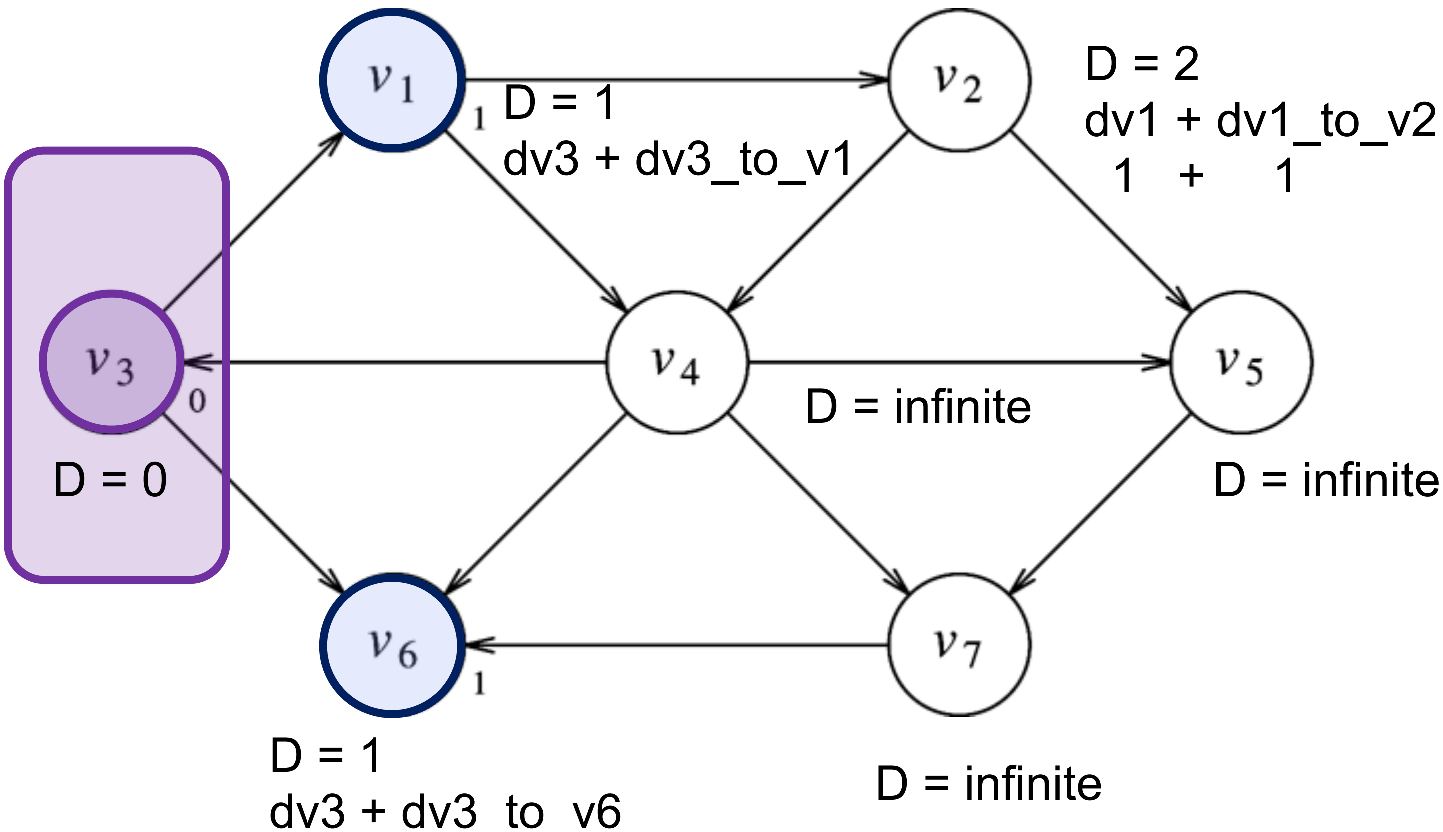


Running Time = $O(|E|+|V|)$, every V is explored, all E in worst case.

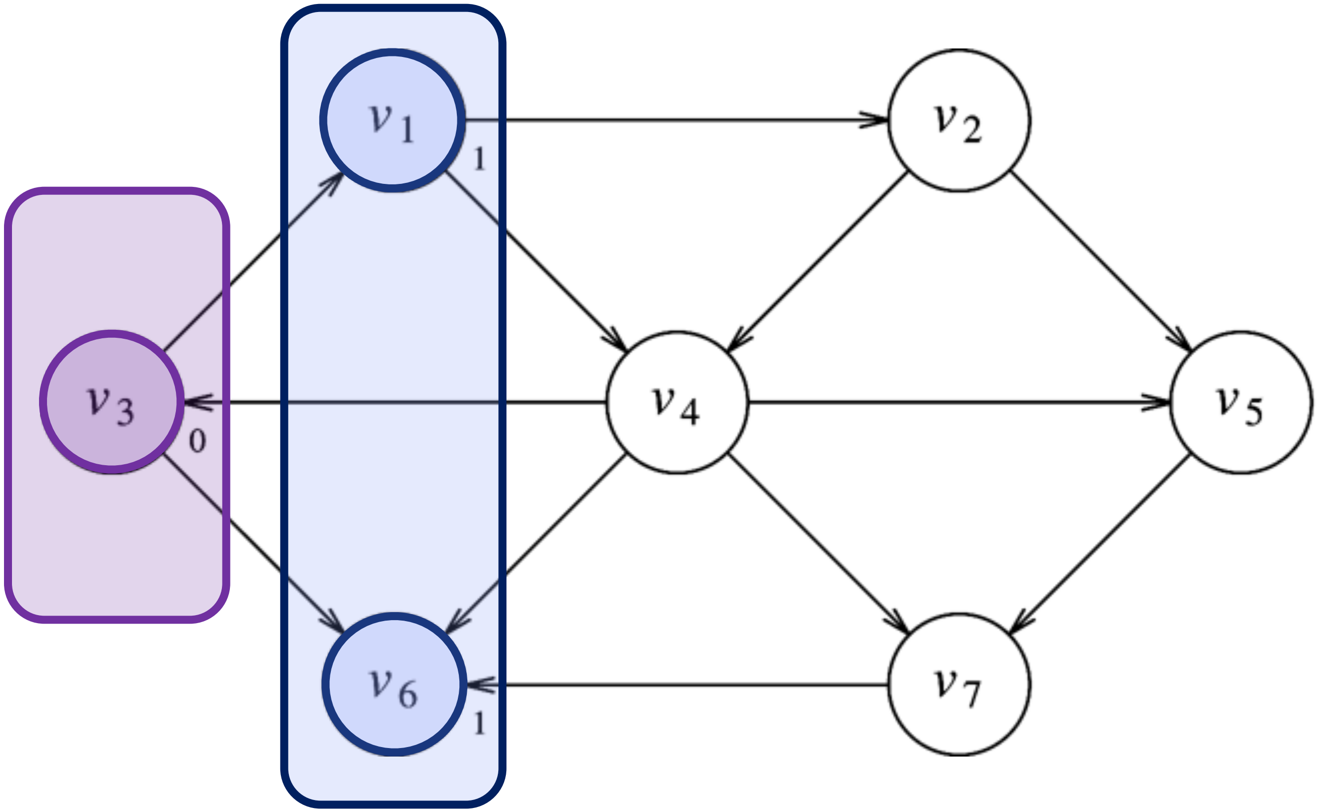
BFS – In Class Activity



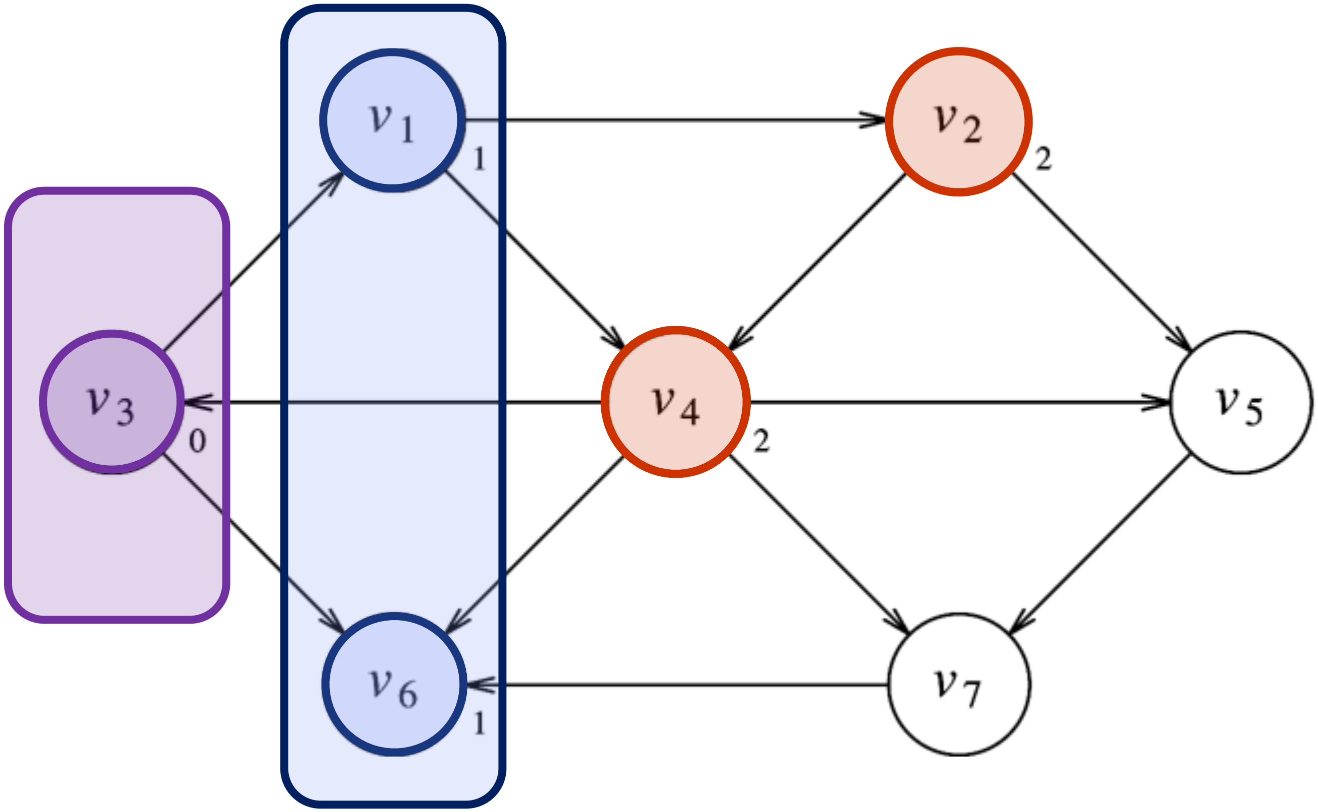
BFS – In Class Activity



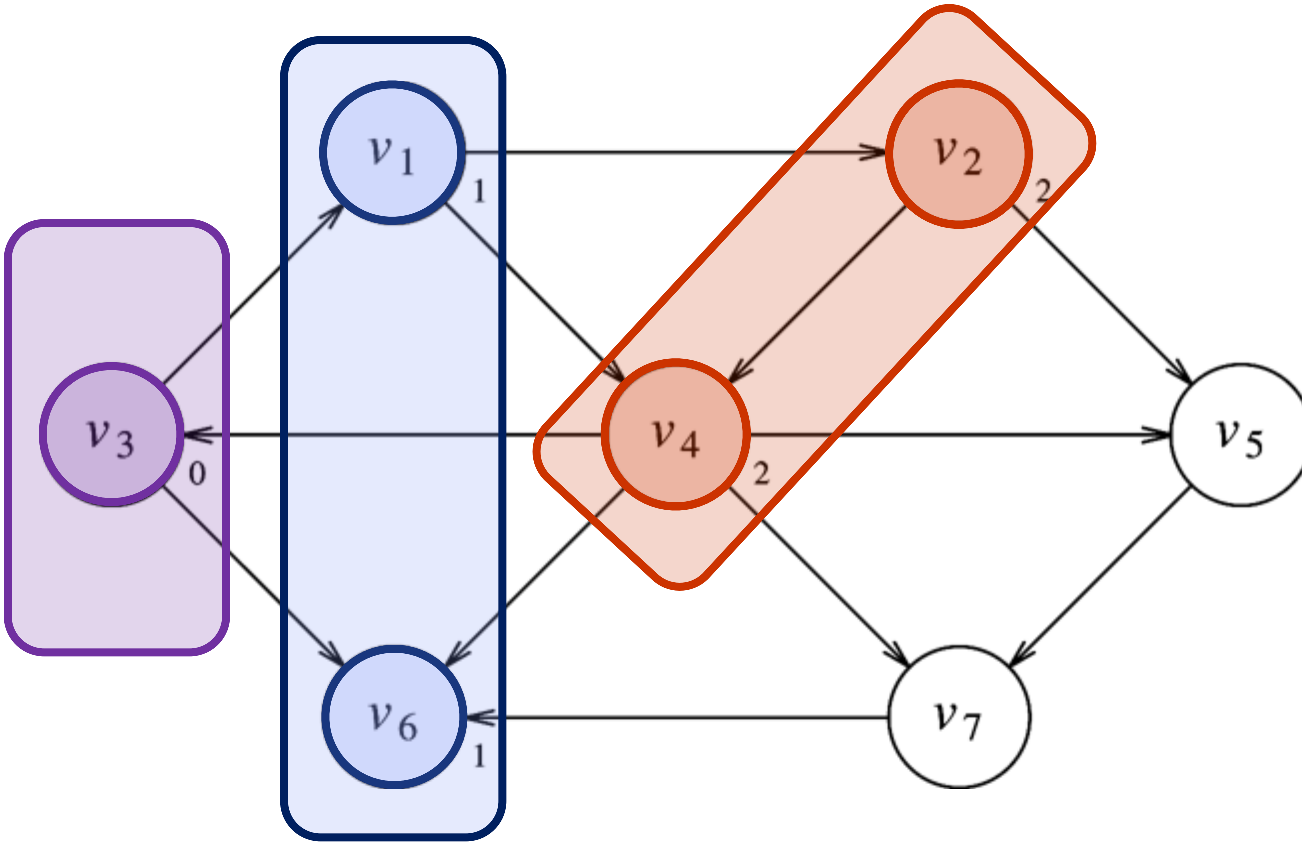
BFS – In Class Activity



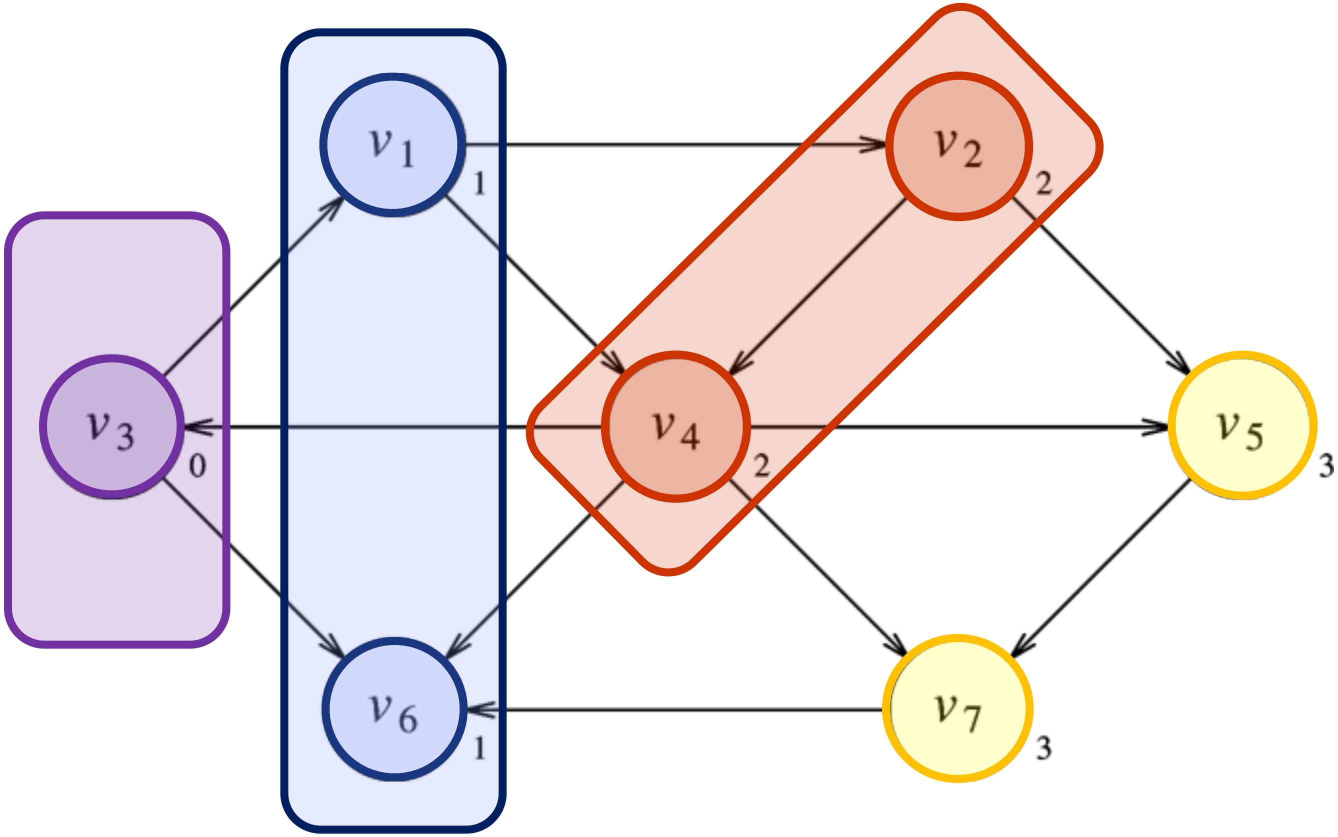
BFS – In Class Activity



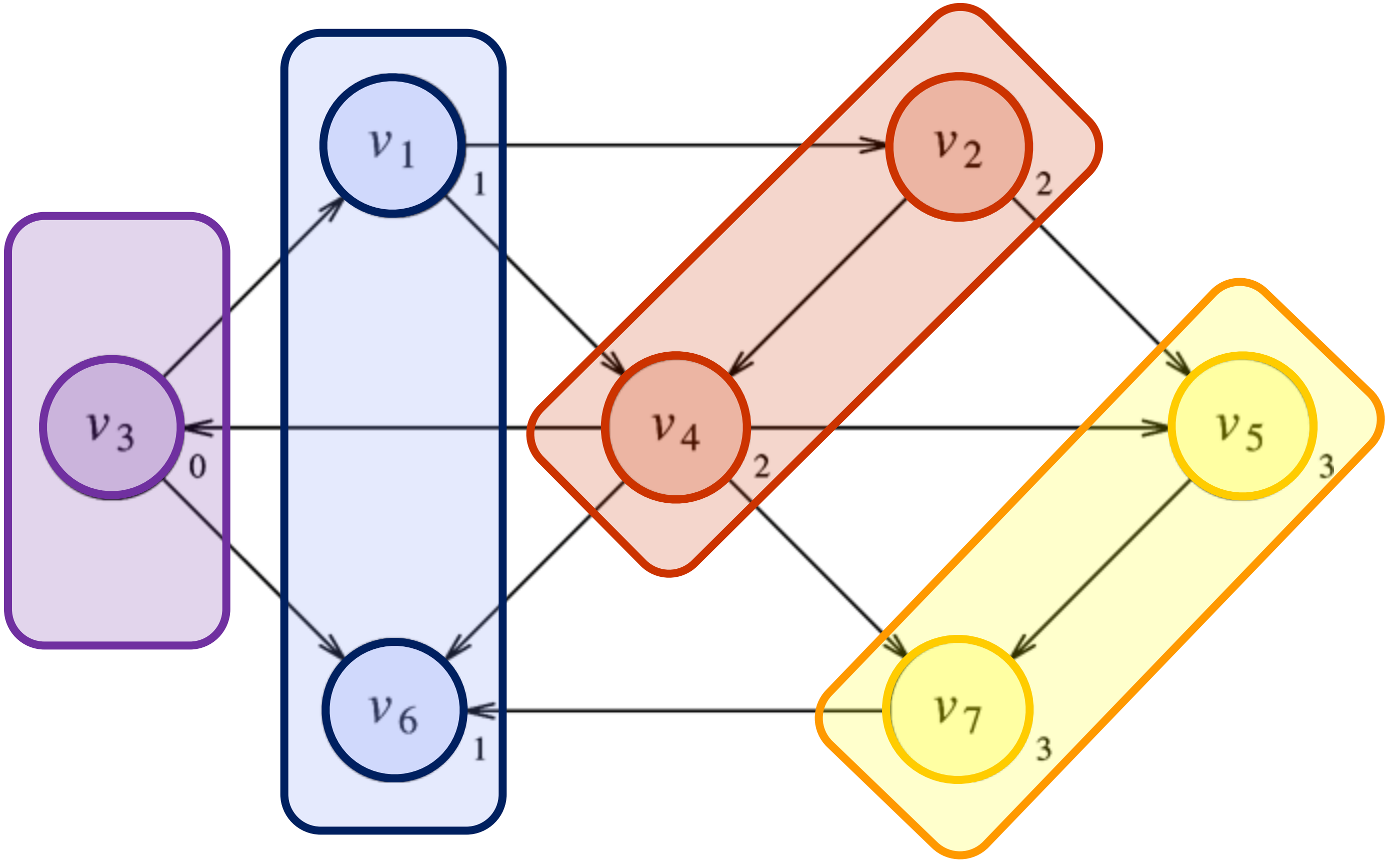
BFS – In Class Activity



BFS – In Class Activity



BFS – In Class Activity



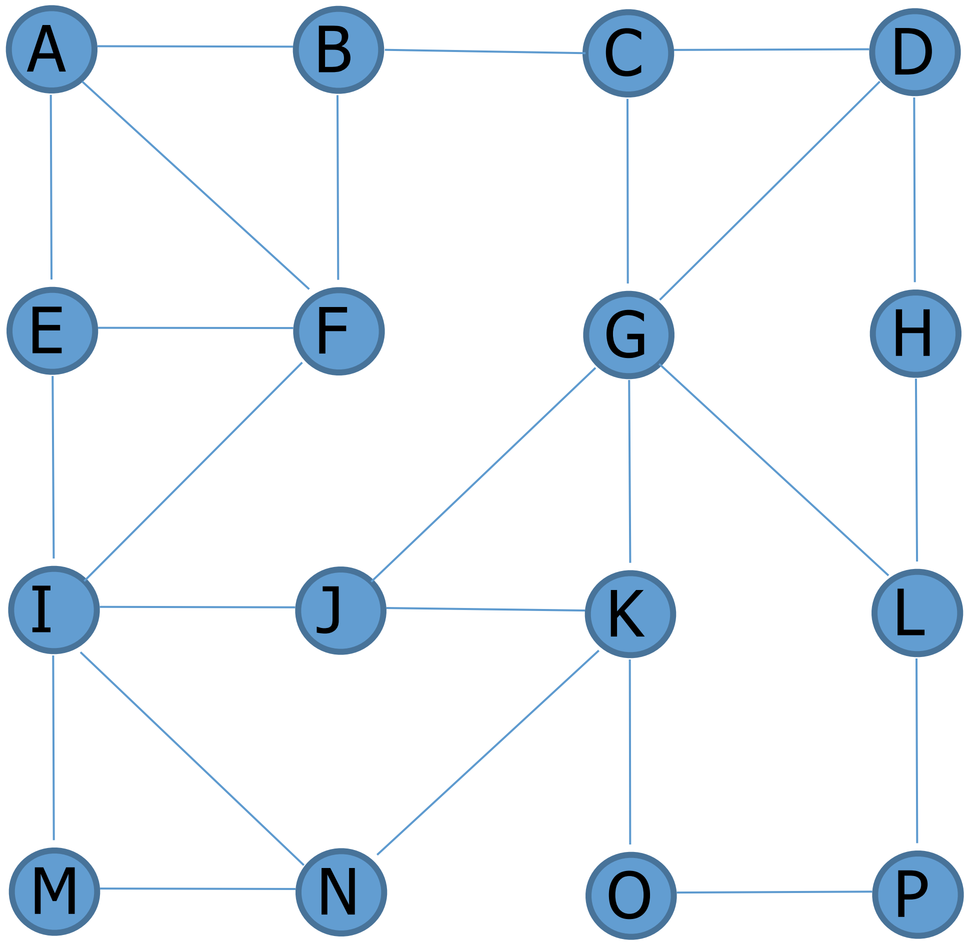
Running Time = $O(|E|+|V|)$

DFS - Depth First Search

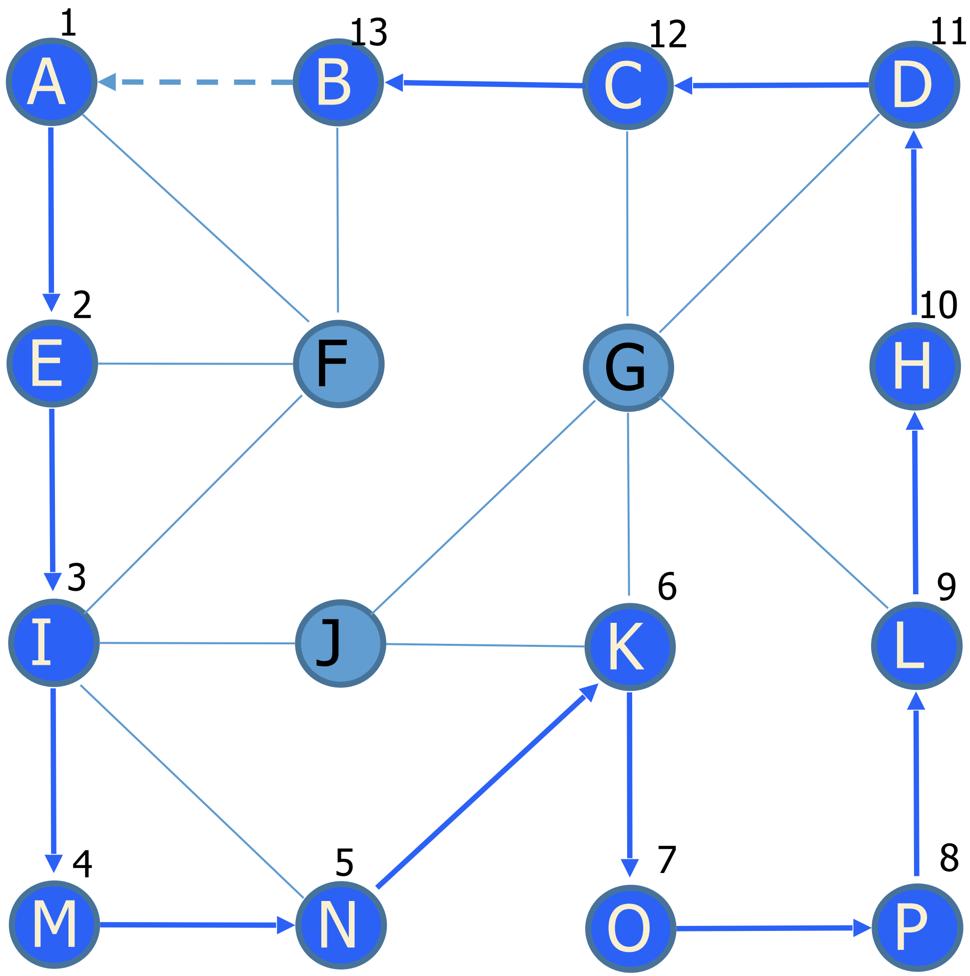
Generalization of a preorder traversal.

1. Start at an arbitrary vertex s.
2. Traverse (visit) as many descendants of s as possible without visiting a previously visited vertex.
3. Once a previously visited vertex is encountered, back out until a vertex v_i with unexplored edges is found.
4. Depth first search on $v_{i.}$
5. Repeat (steps 3-4) until all vertices visited.

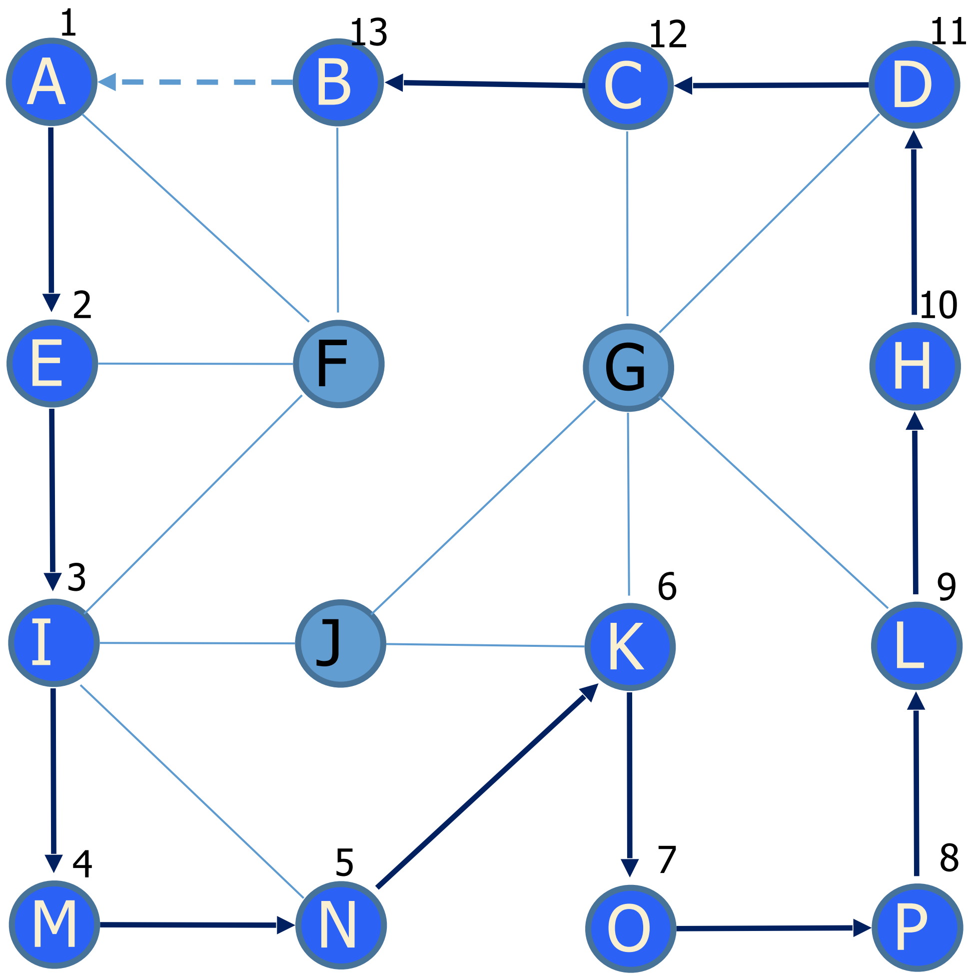
DFS - Depth First Search



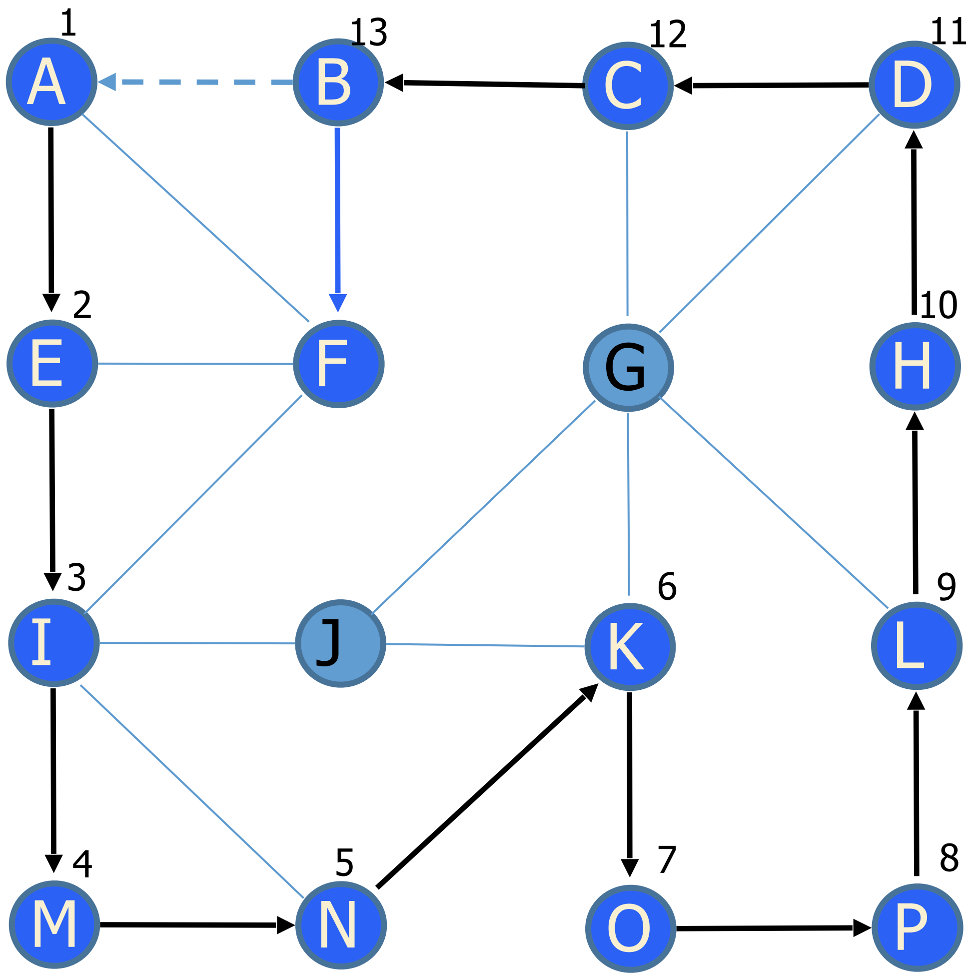
DFS - Depth First Search



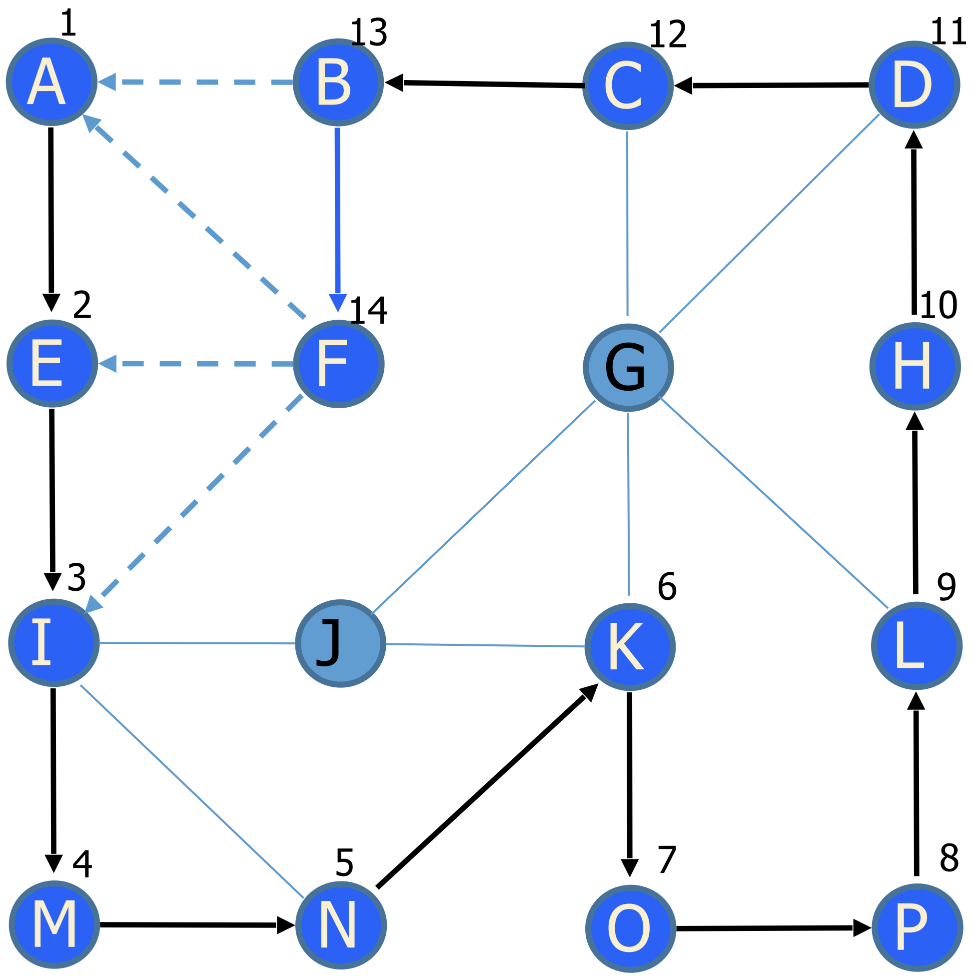
DFS - Depth First Search



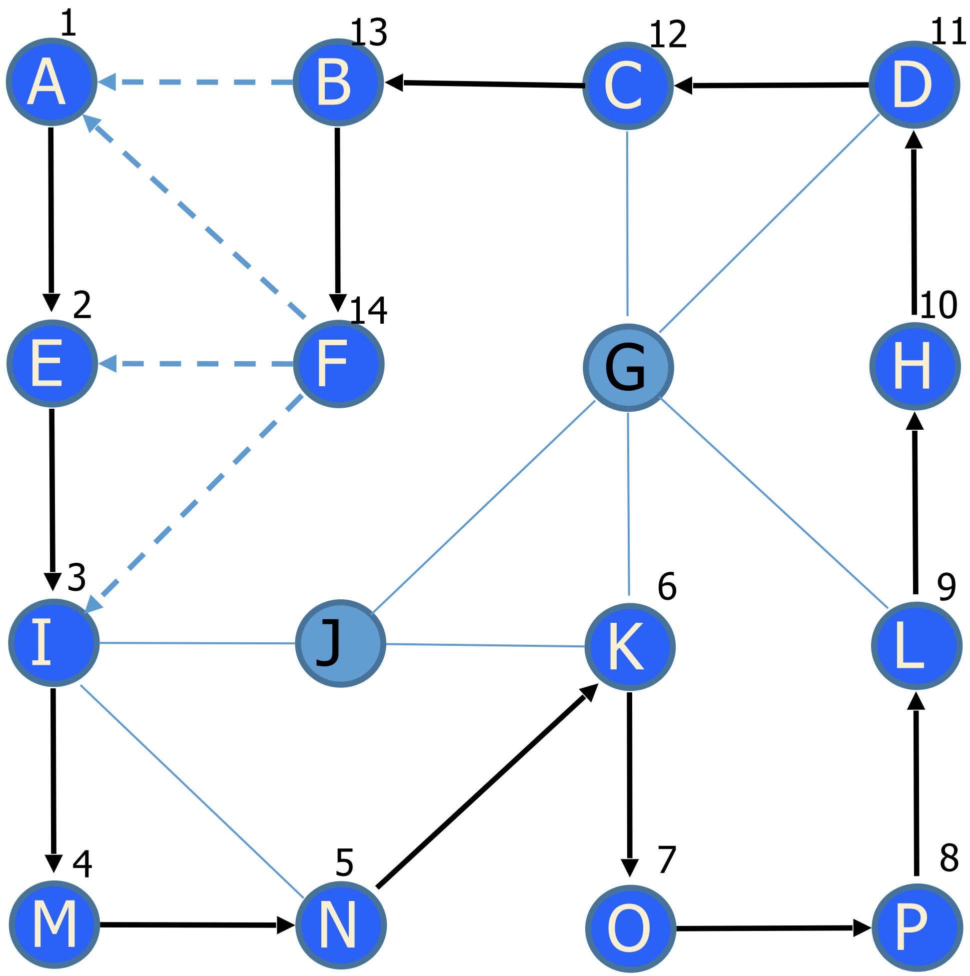
DFS - Depth First Search



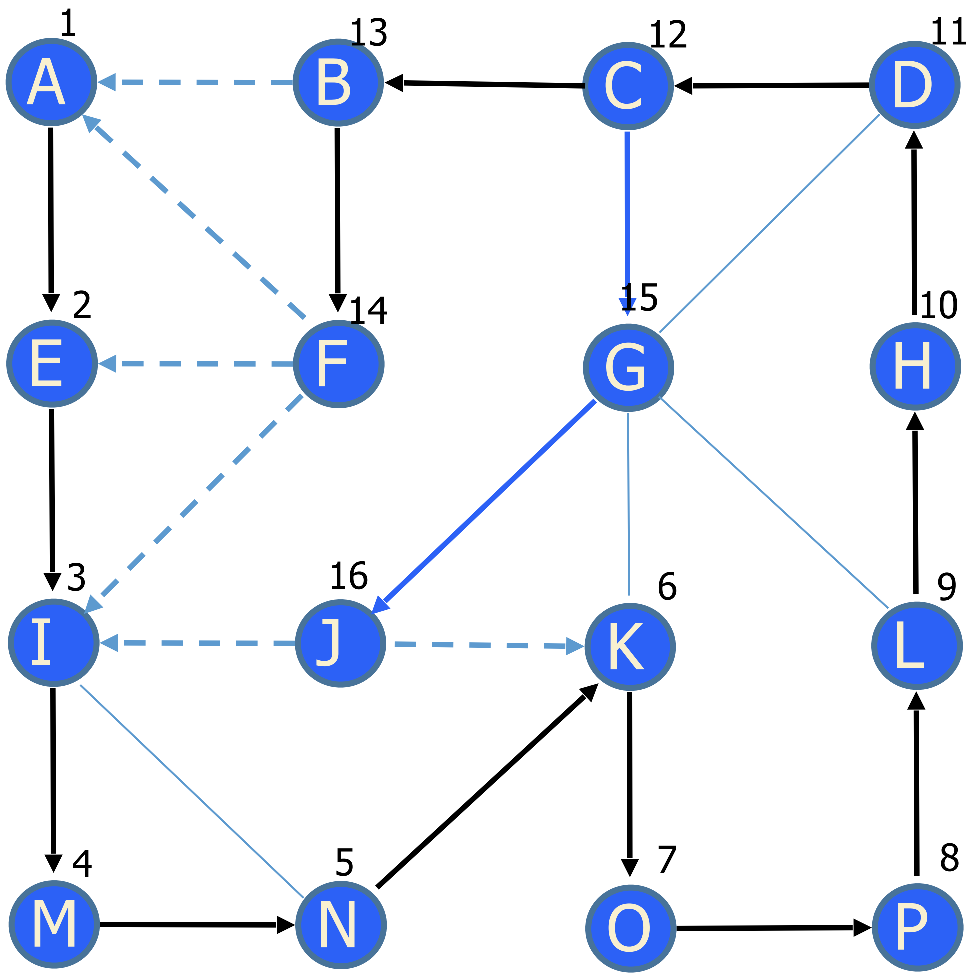
DFS - Depth First Search



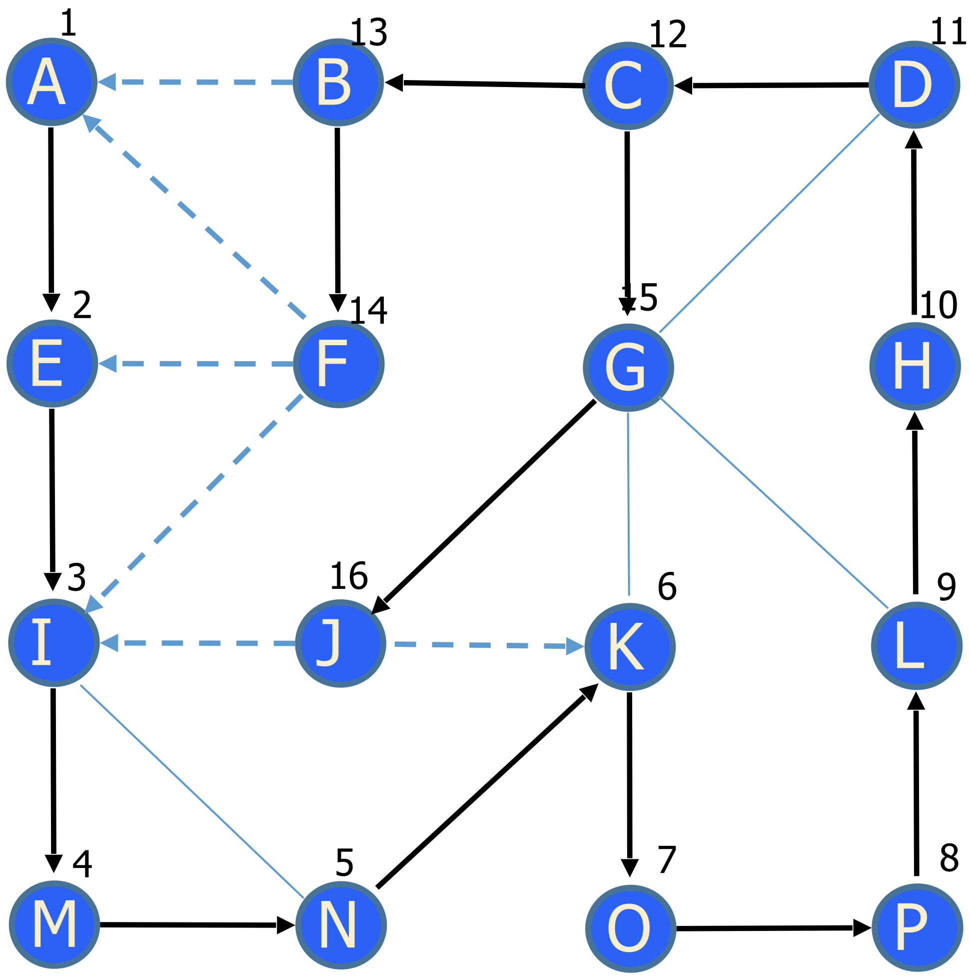
DFS - Depth First Search



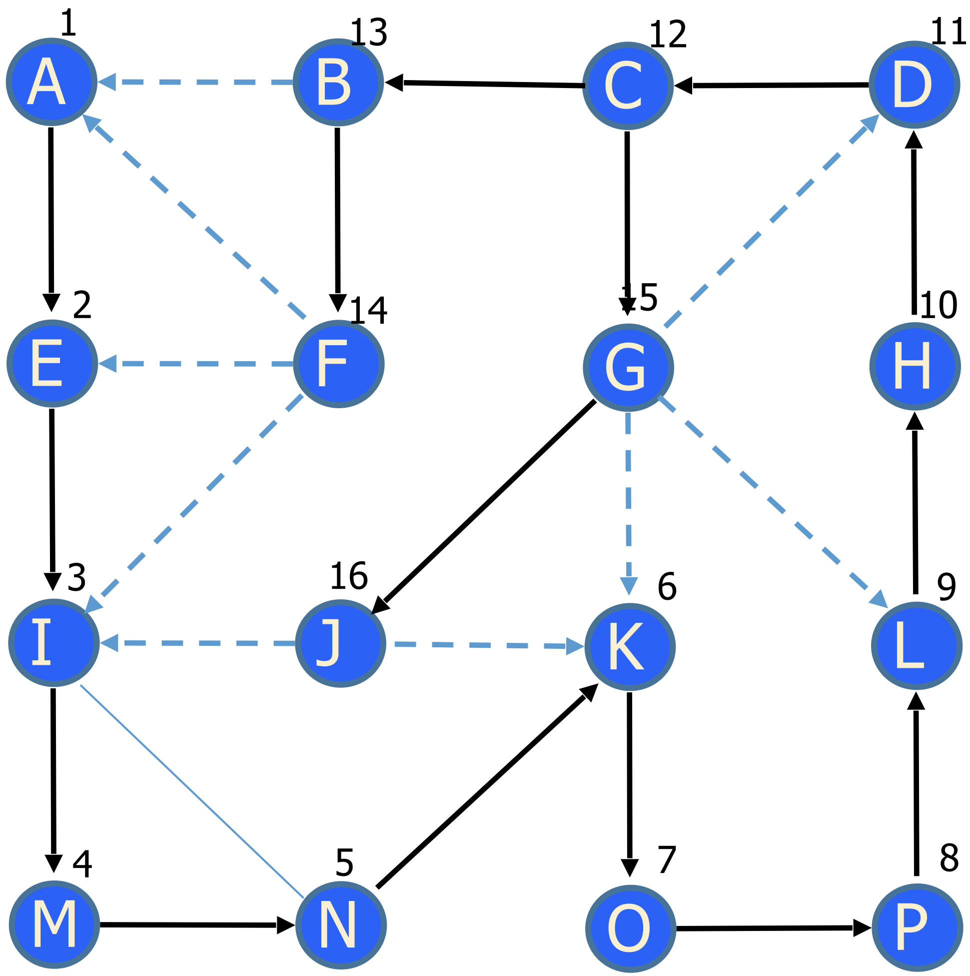
DFS - Depth First Search



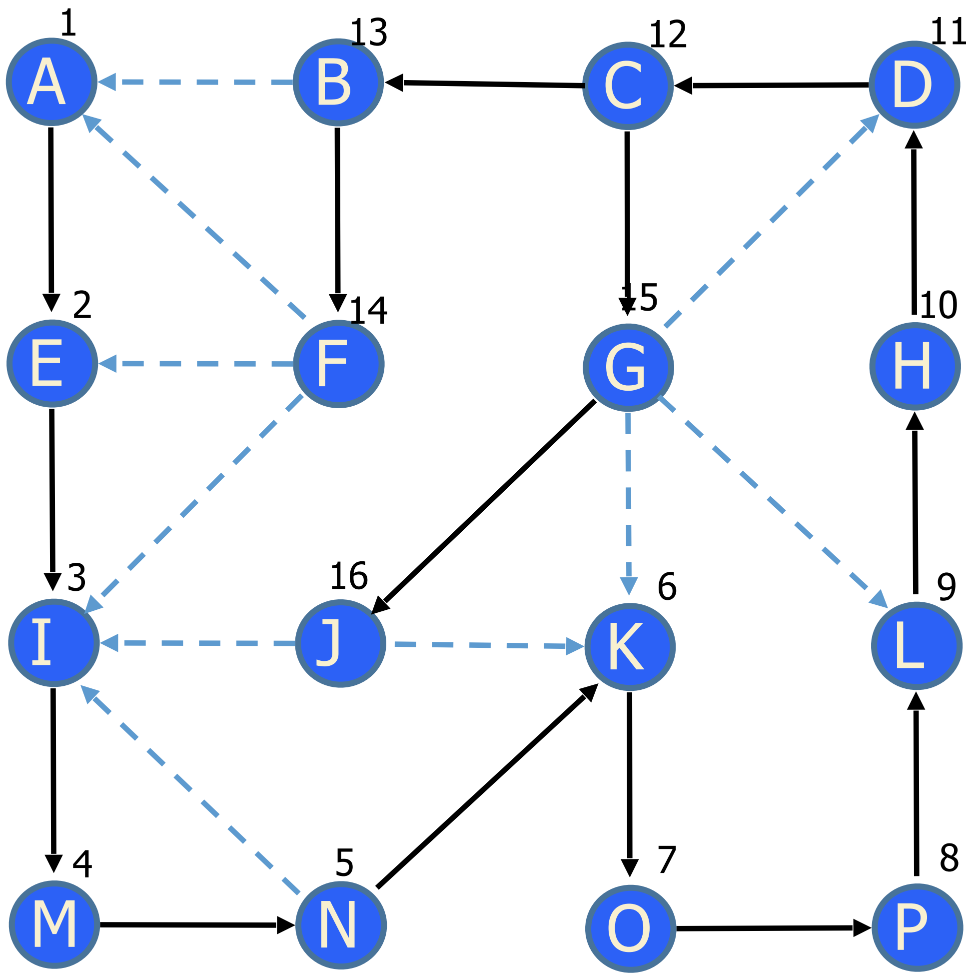
DFS - Depth First Search



DFS - Depth First Search

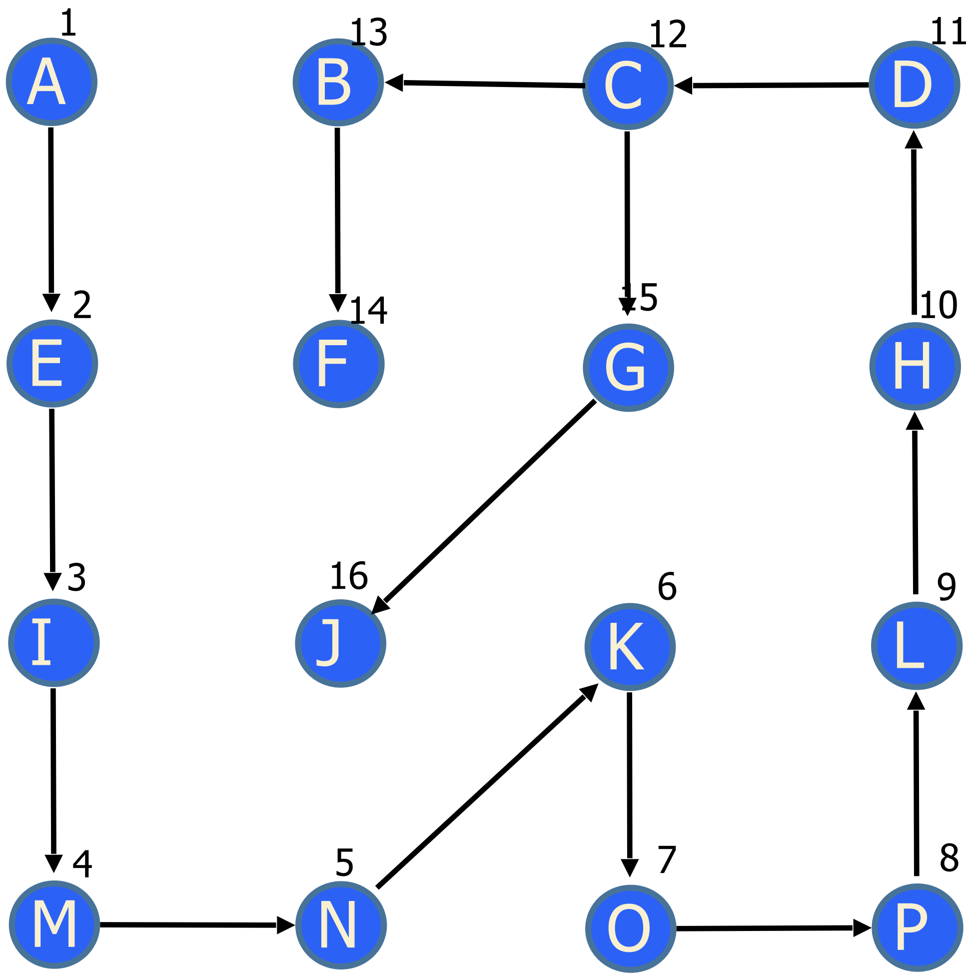


DFS - Depth First Search

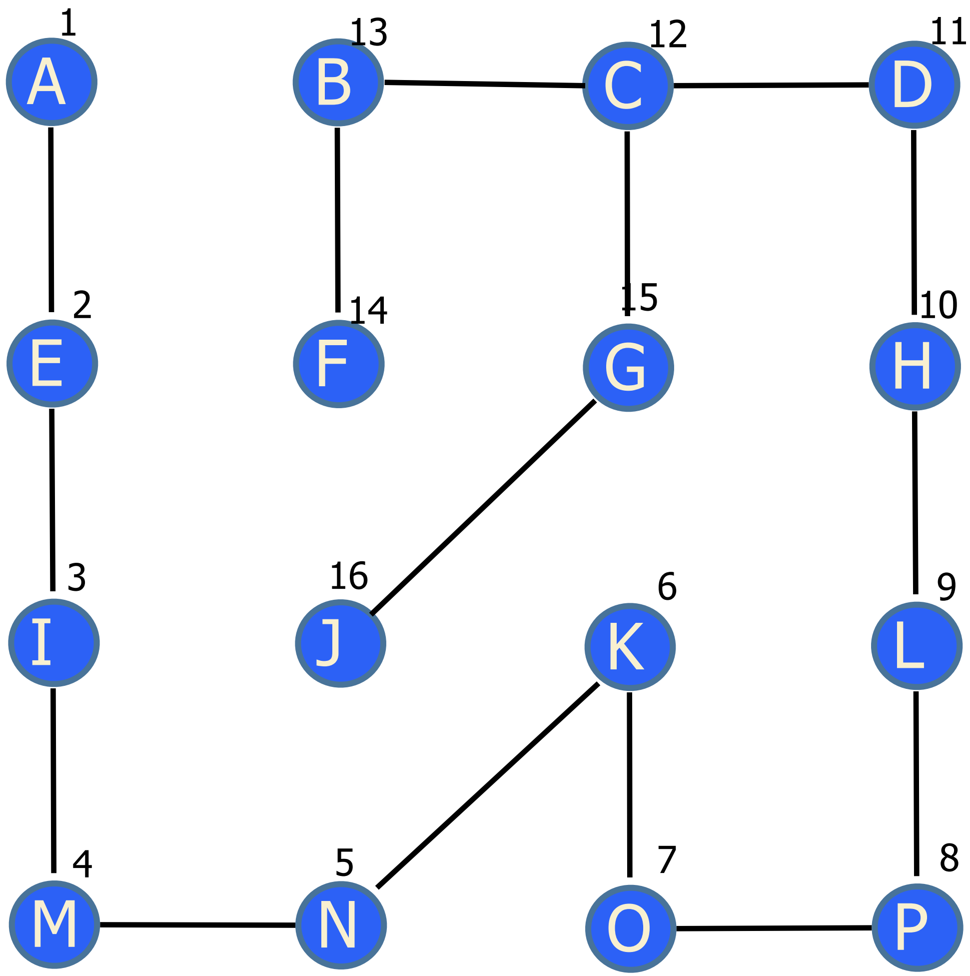


Running Time = $O(|E|+|V|)$

Spanning Tree



Spanning Tree



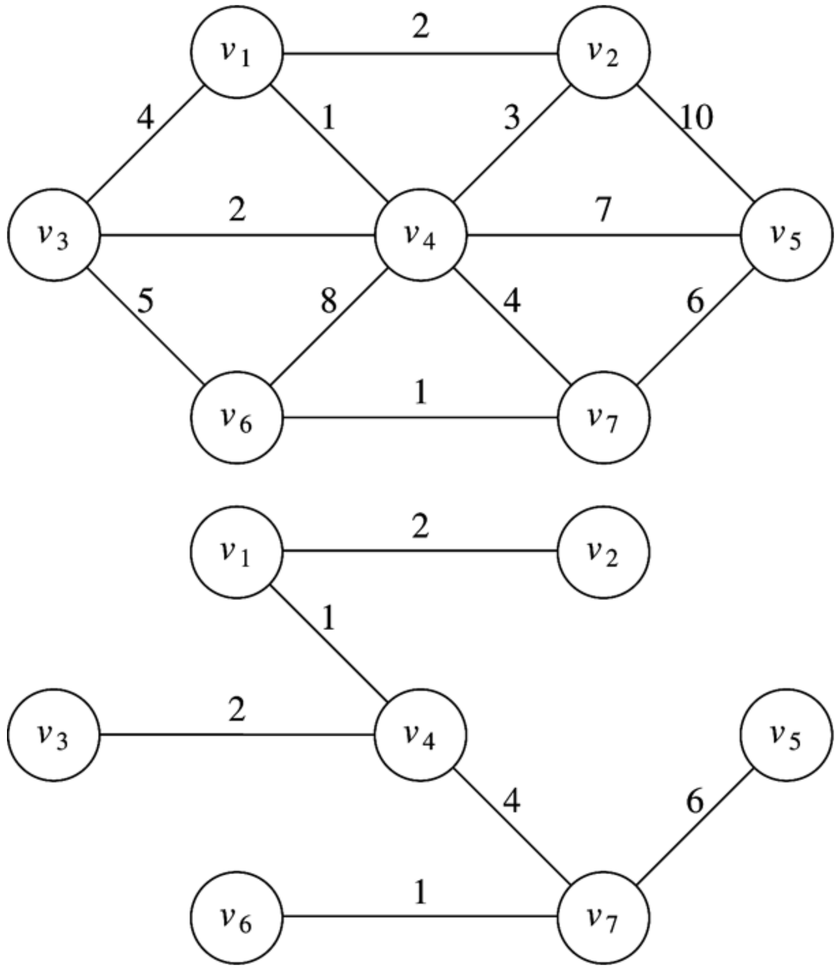
MST – Minimum Spanning Tree

- **Spanning Tree** – Given an undirected, connected graph G, a spanning tree T is a connected acyclic subgraph (tree) that contains all vertices of the graph.
- **Minimum Spanning Tree** – the spanning tree of the smallest weight, where the weight is the sum of the weights on all T's edges.
- **MST Problem** – Find the minimum spanning tree for a given weighted connected graph.
- Not always a unique MST. There can be more than 1 MST with a different set of edges, perhaps some shared, in 1 or more MSTs.

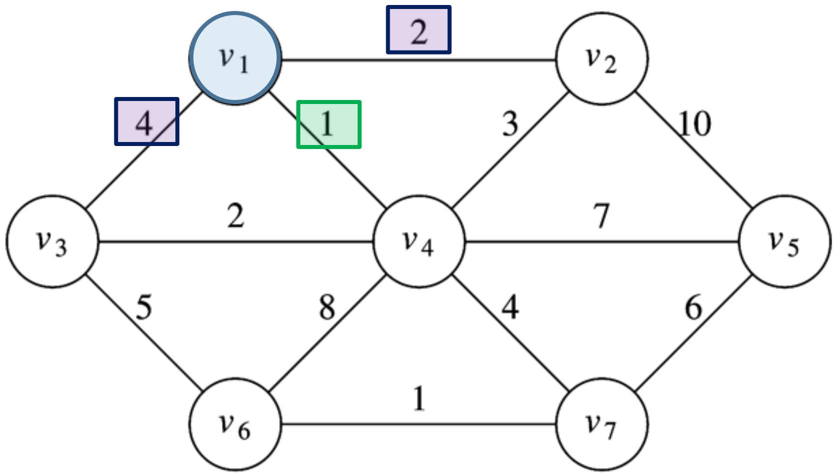
MST

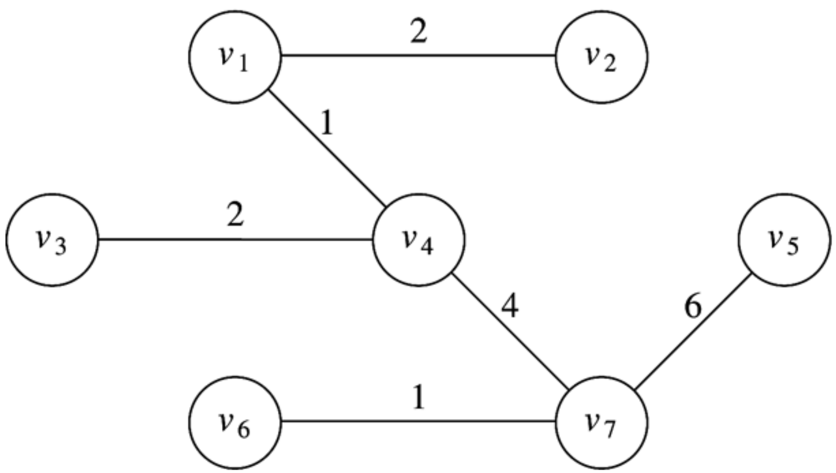
- useful for minimizing the amount of wiring needed for:
  - phone lines
  - cable lines
- used in wireless sensor networks
- used in network routing algorithms

Prim's Algorithm

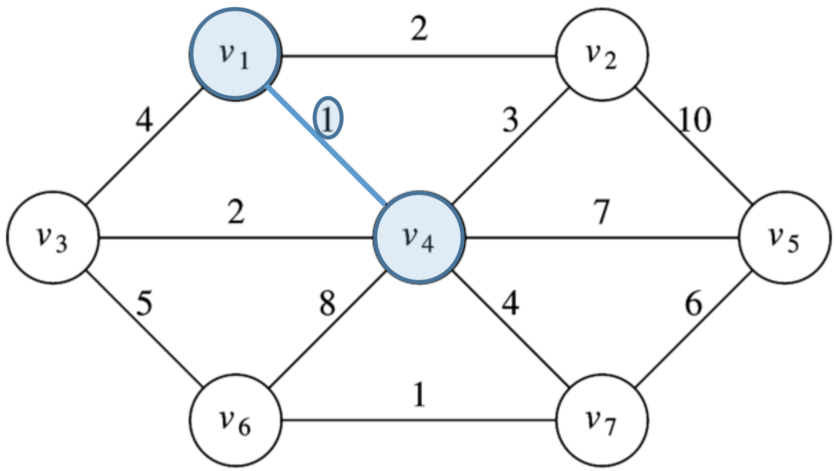


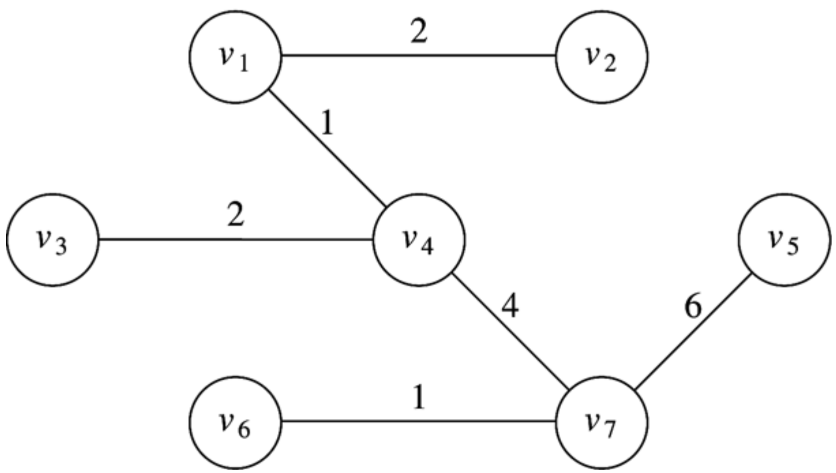
Prim's Algorithm



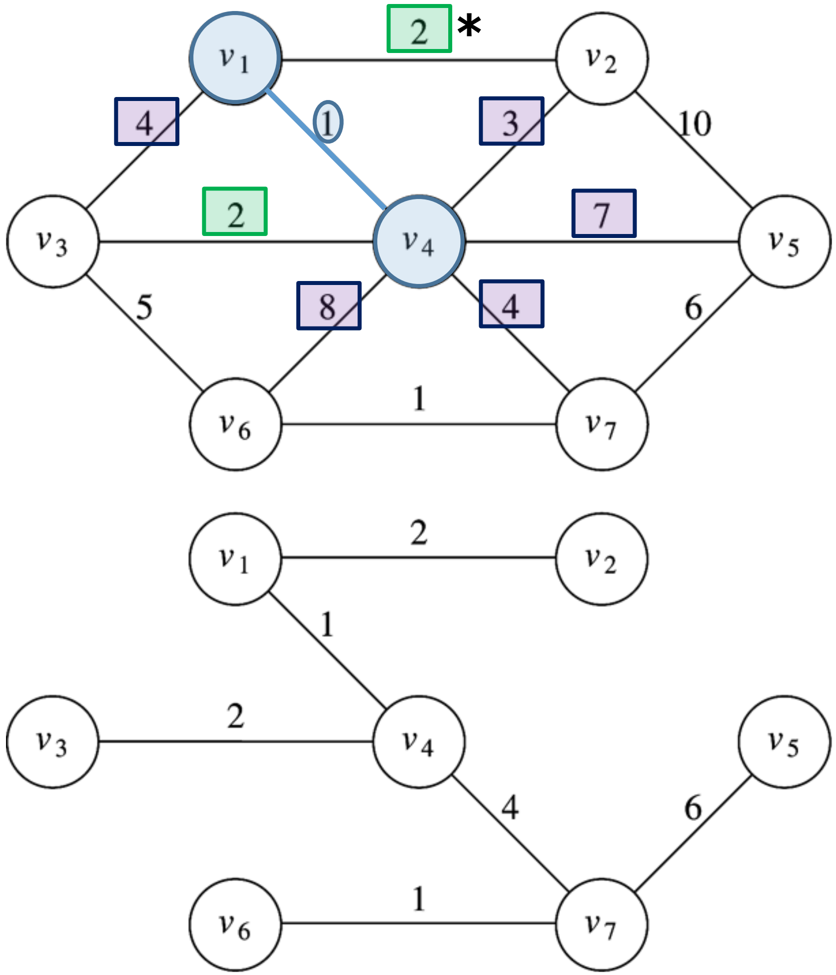


Prim’s Algorithm

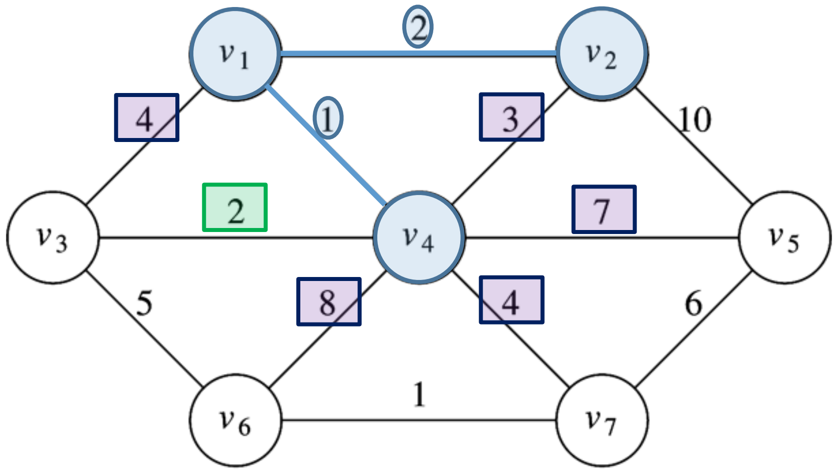


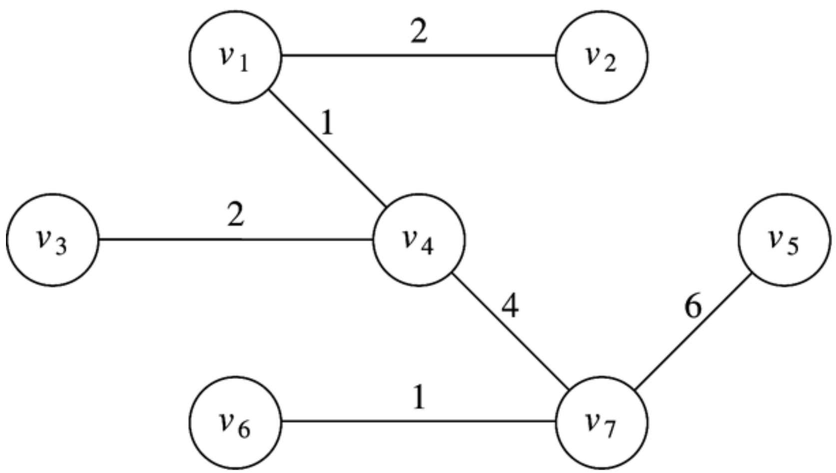


Prim's Algorithm

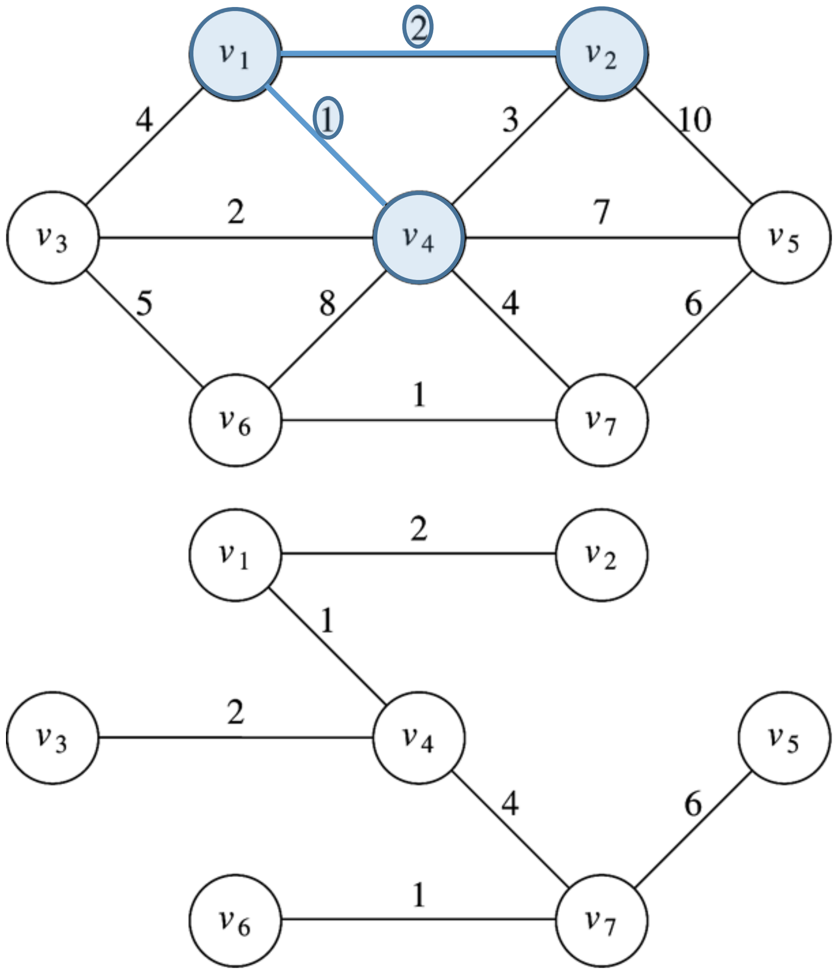


Prim's Algorithm

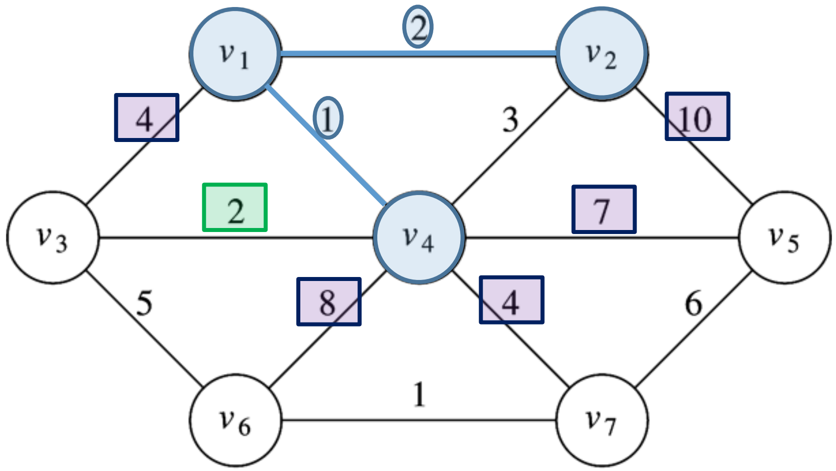


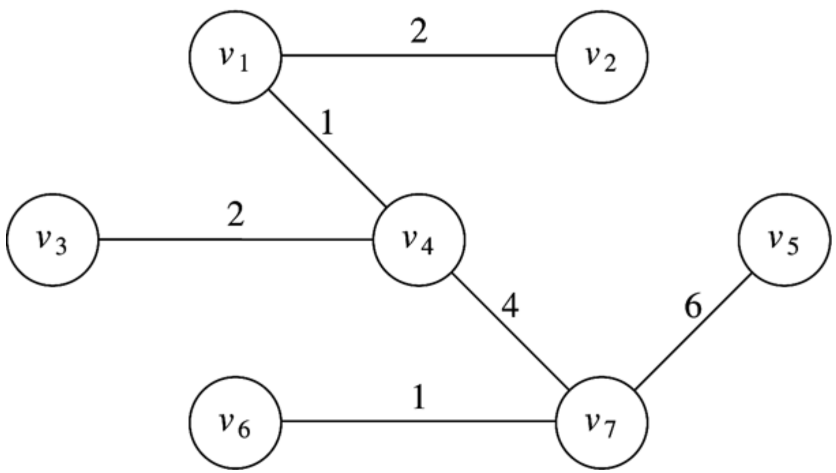


Prim's Algorithm

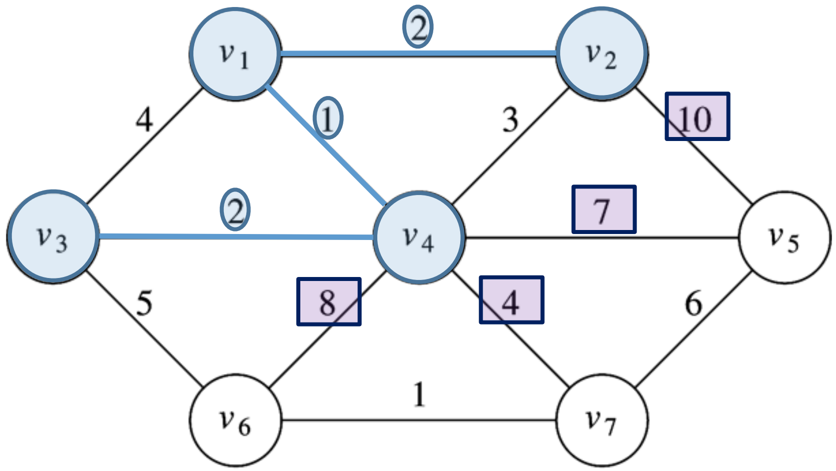


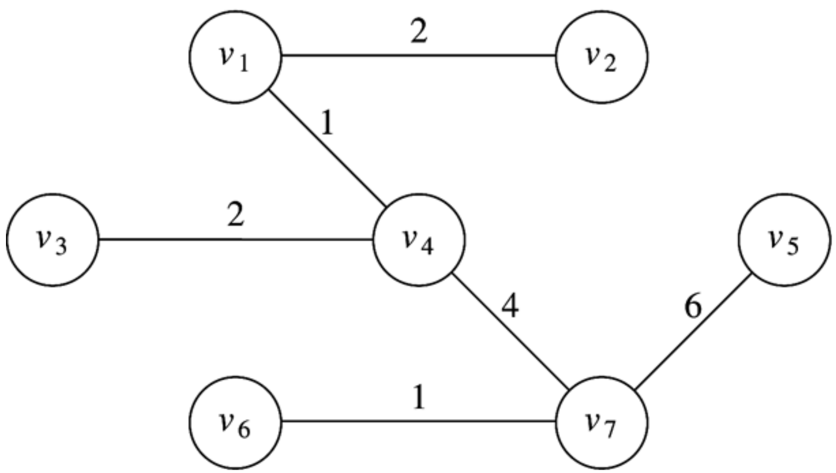
Prim's Algorithm



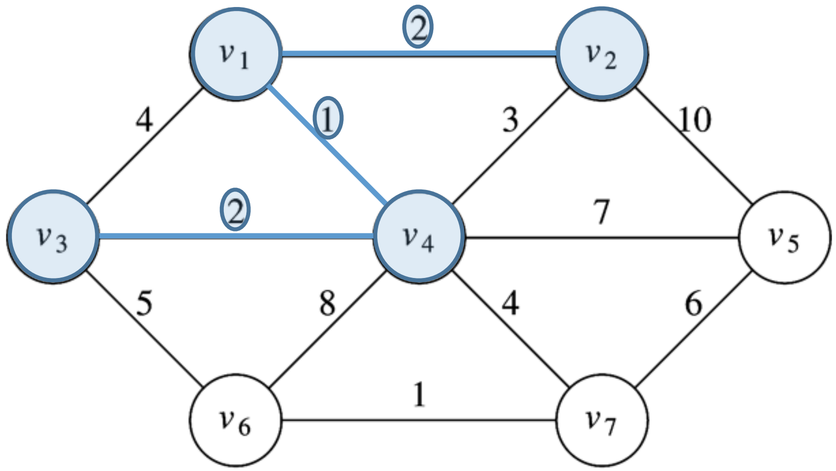


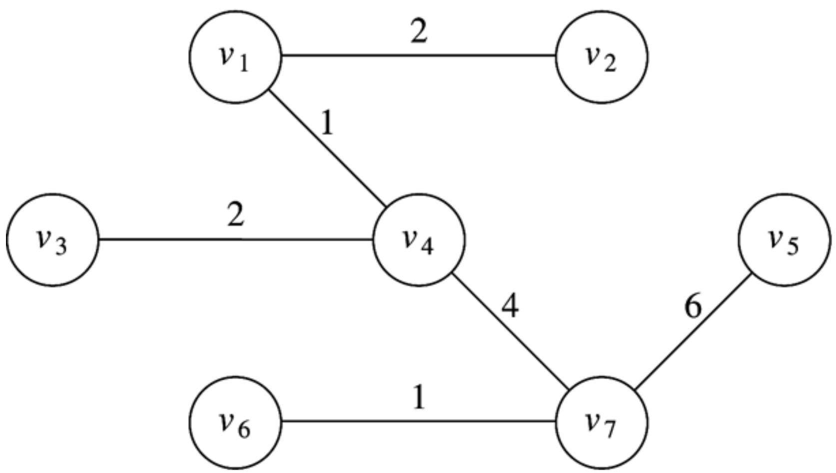
Prim's Algorithm



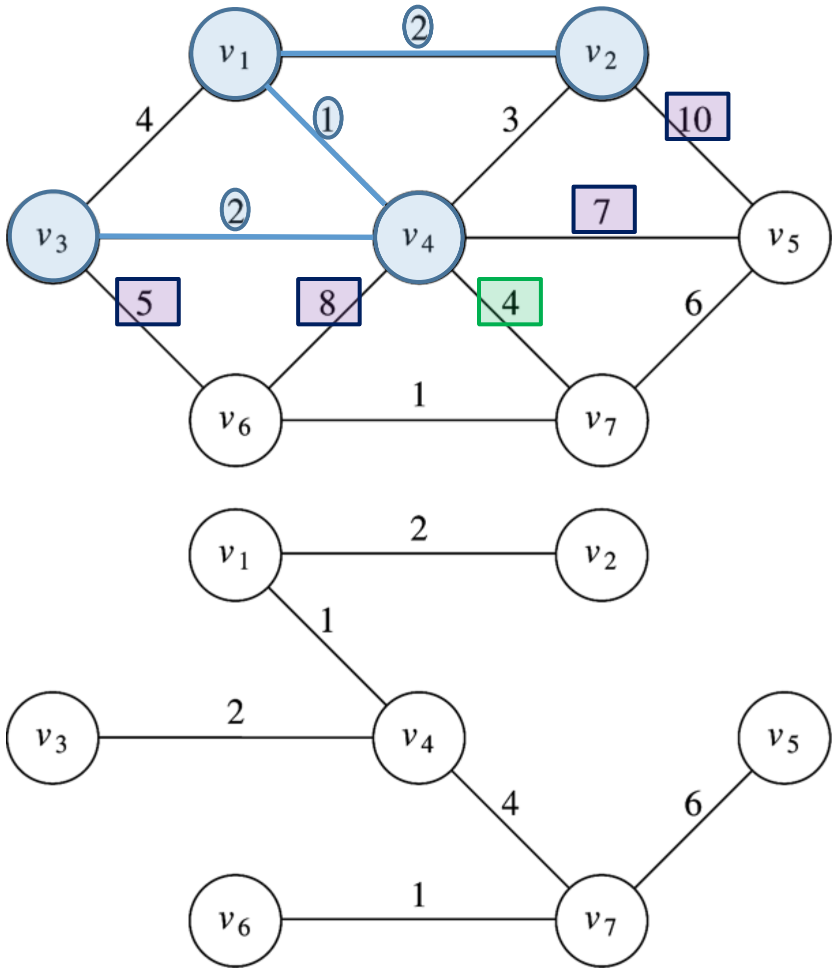


Prim's Algorithm

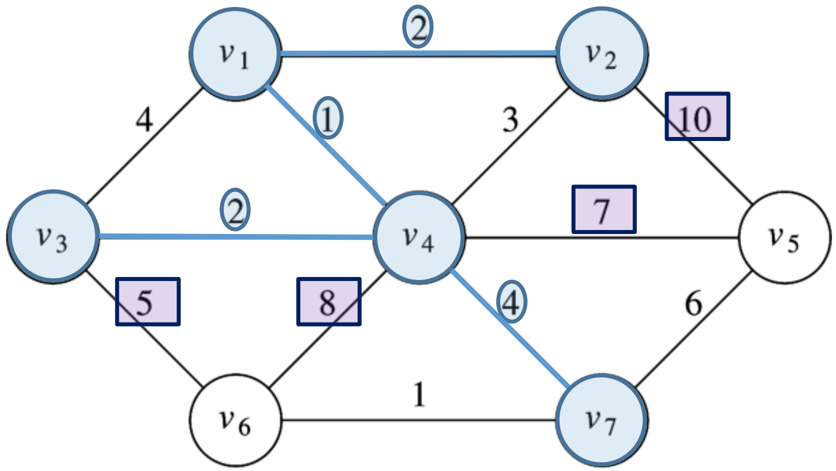


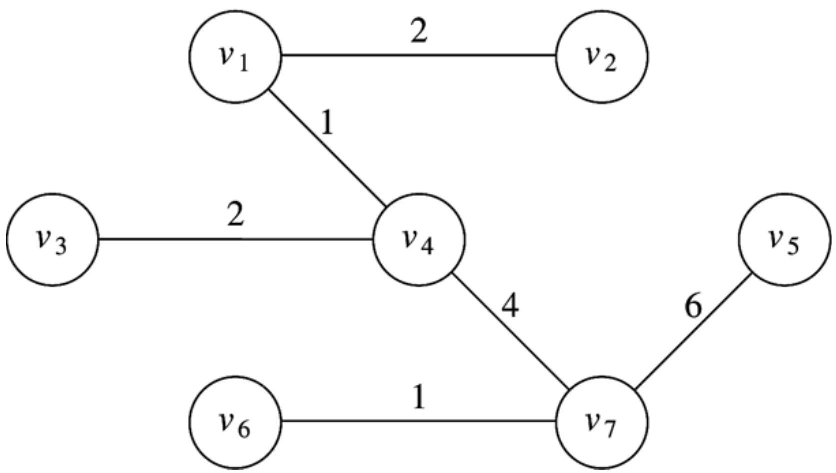


Prim’s Algorithm

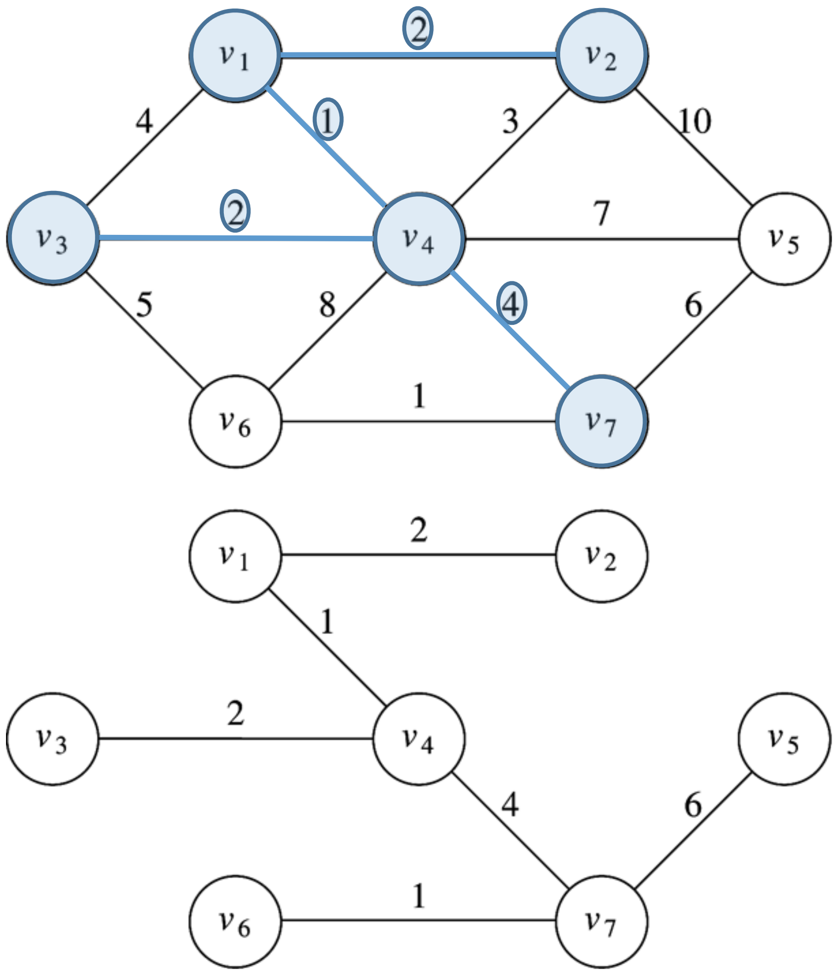


Prim's Algorithm

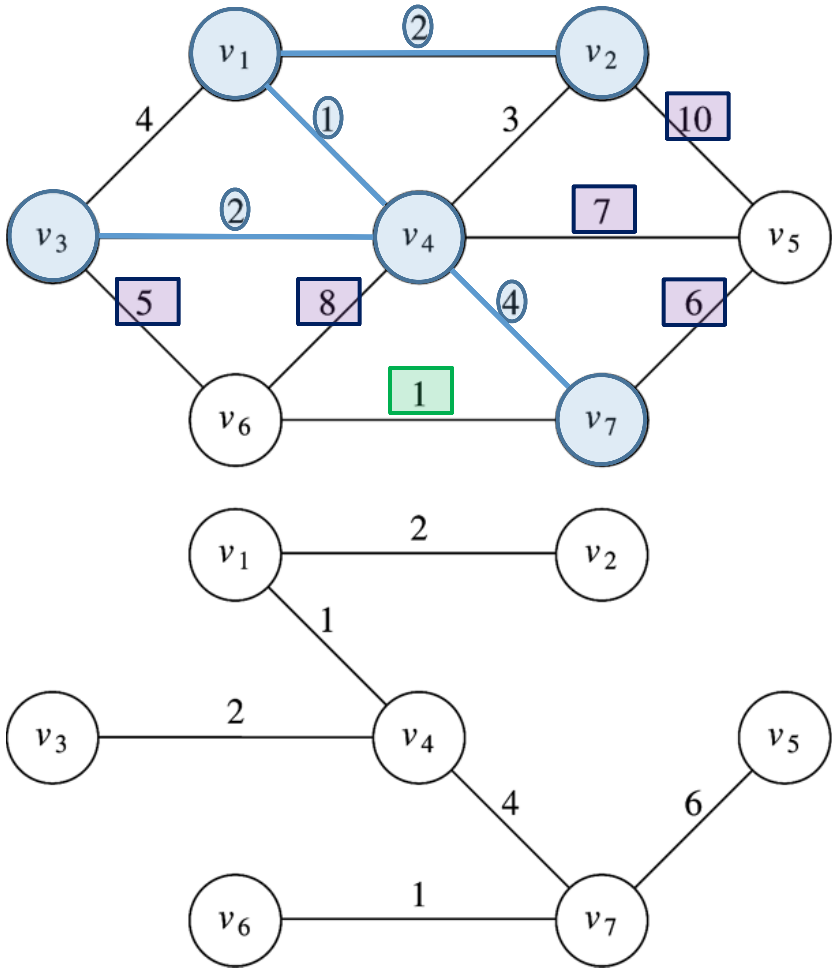




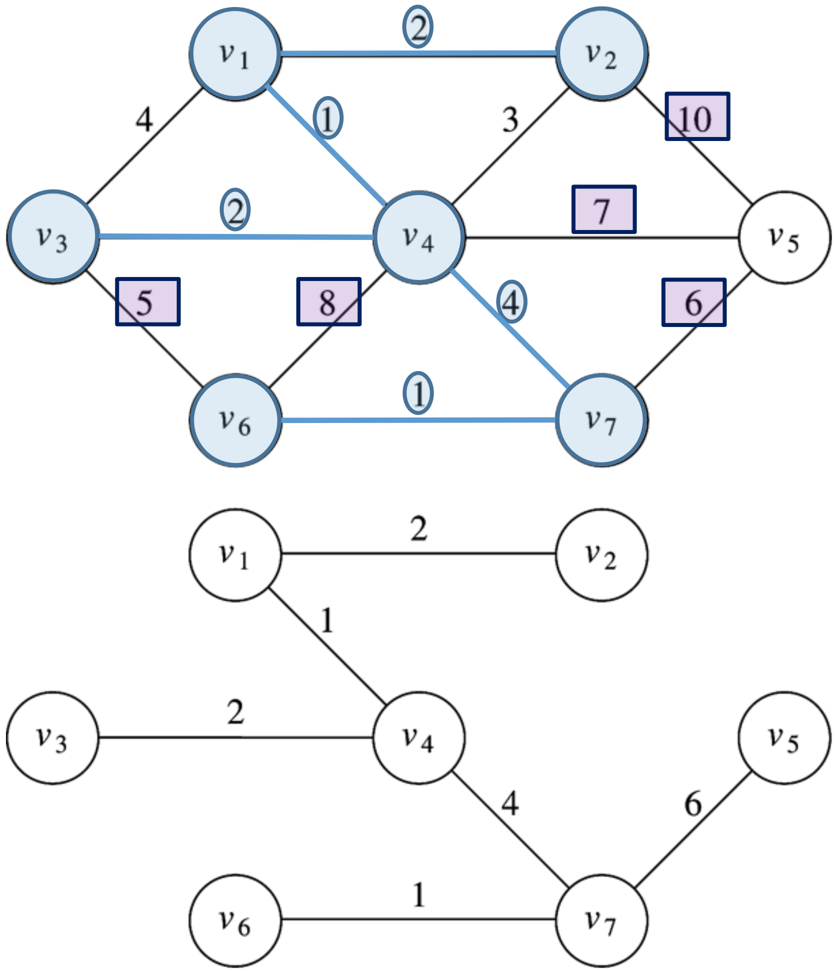
Prim's Algorithm



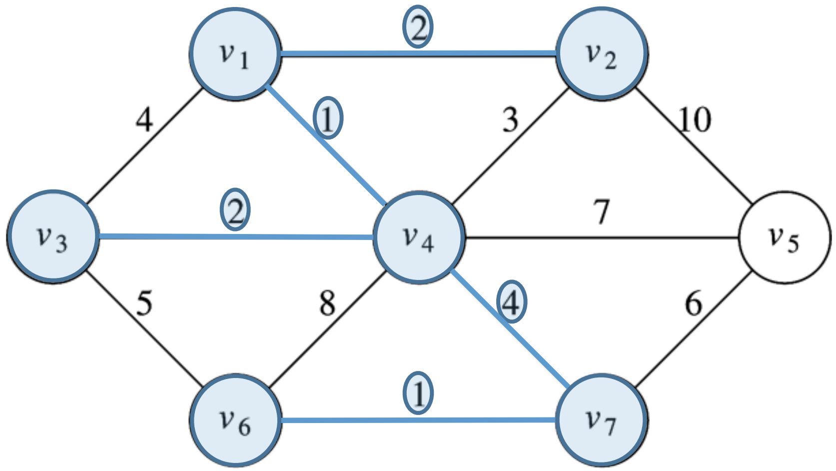
Prim's Algorithm

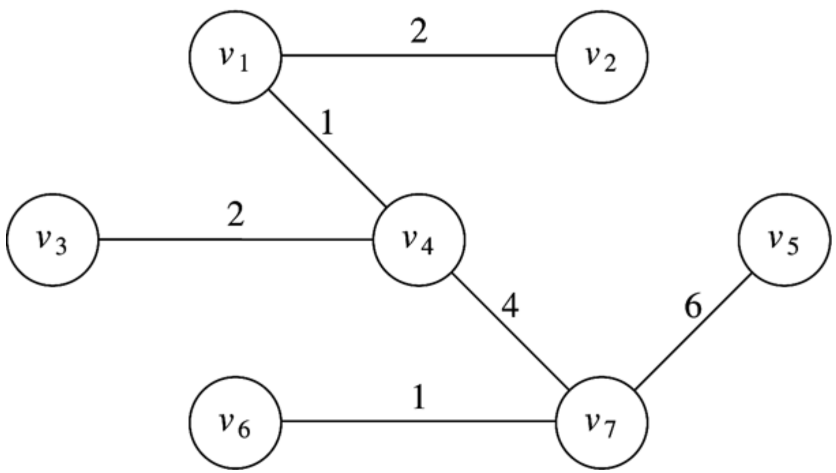


Prim's Algorithm

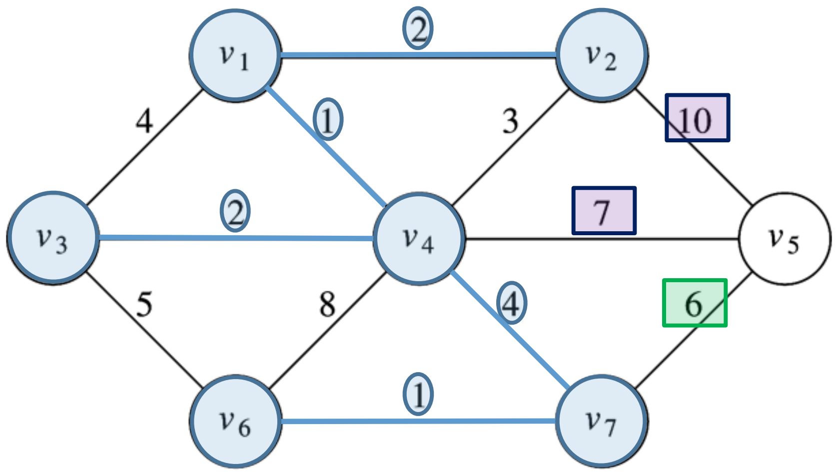


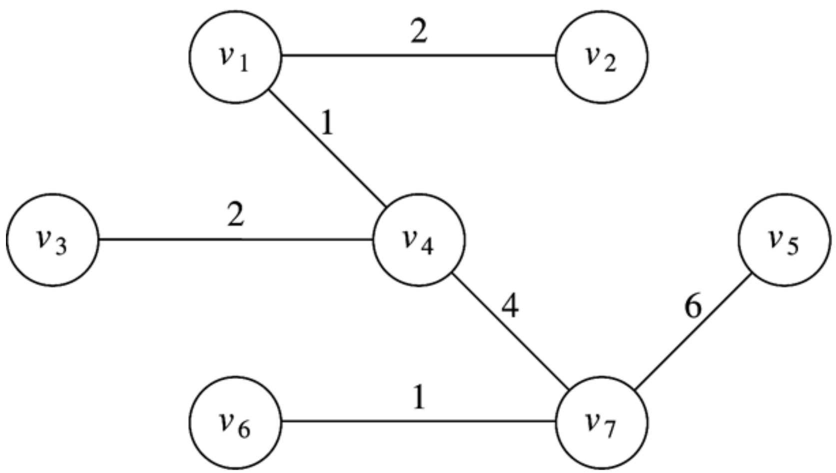
Prim's Algorithm



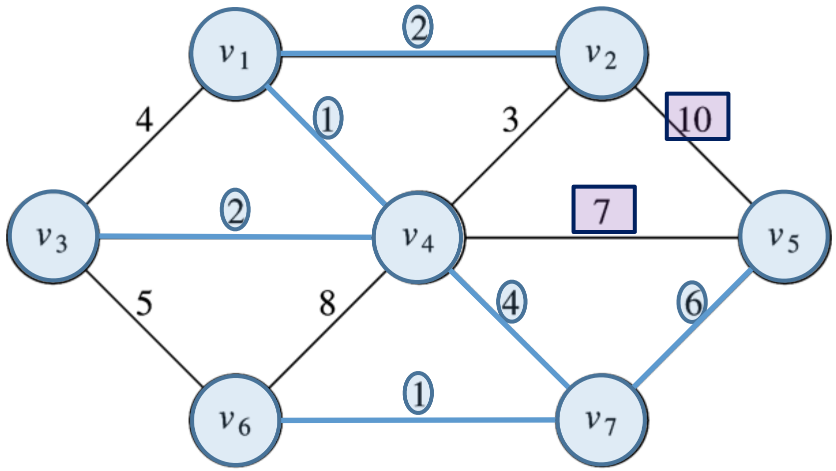


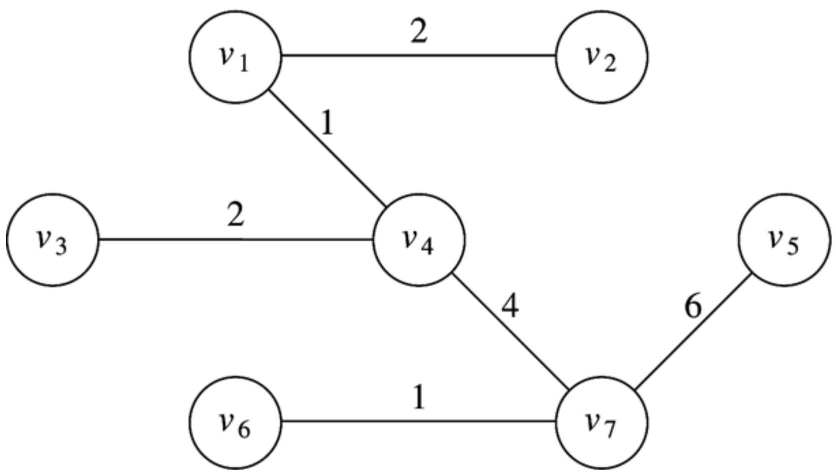
Prim's Algorithm



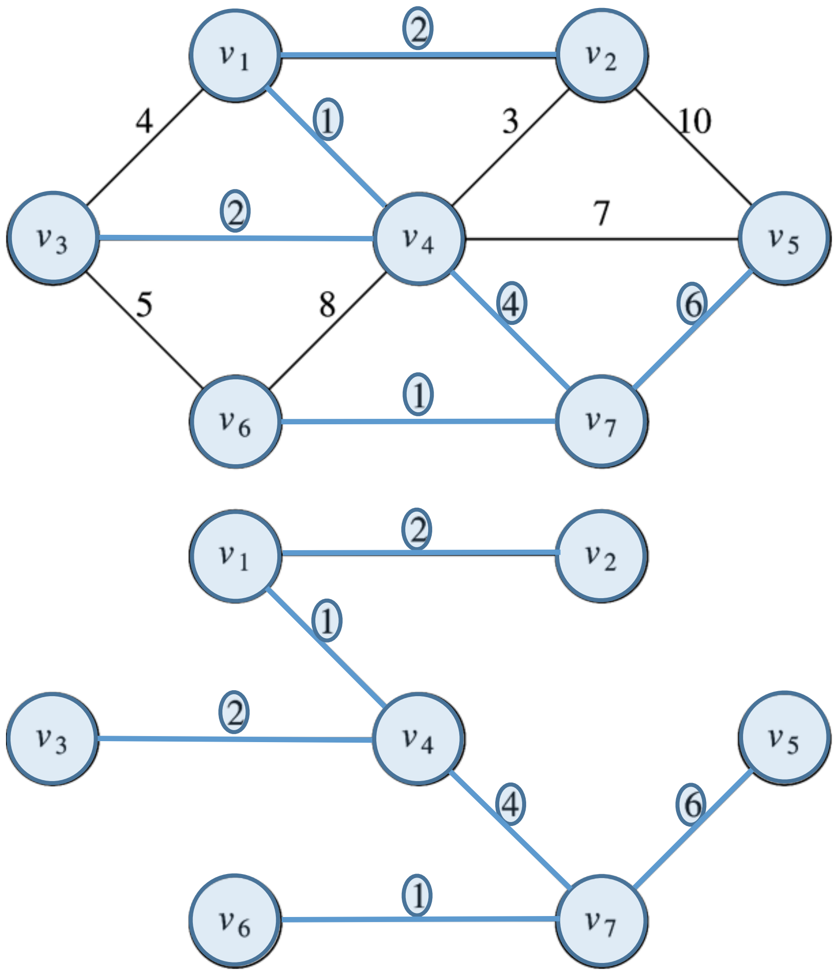


Prim’s Algorithm



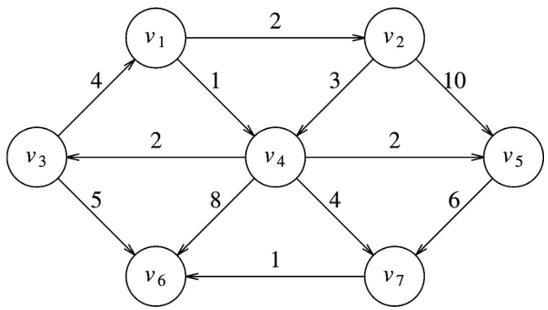


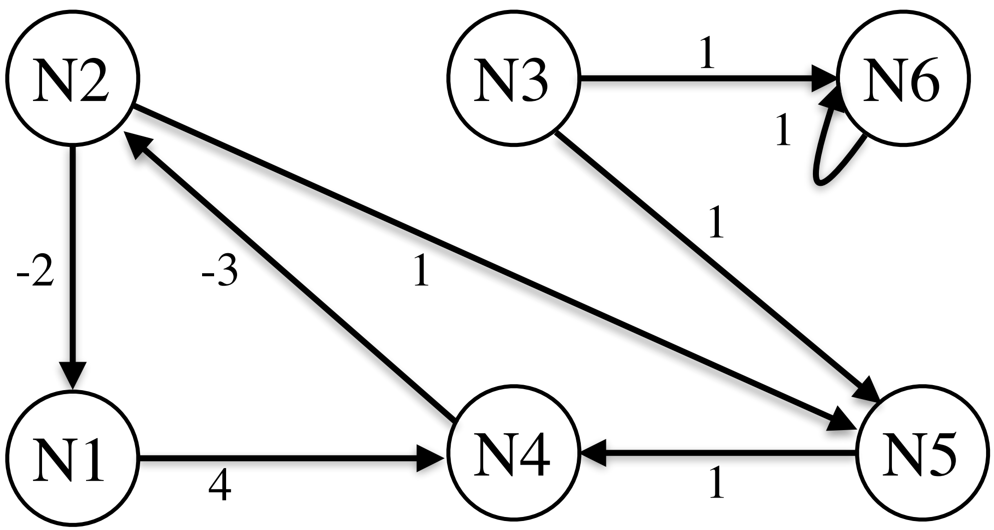
Prim's Algorithm

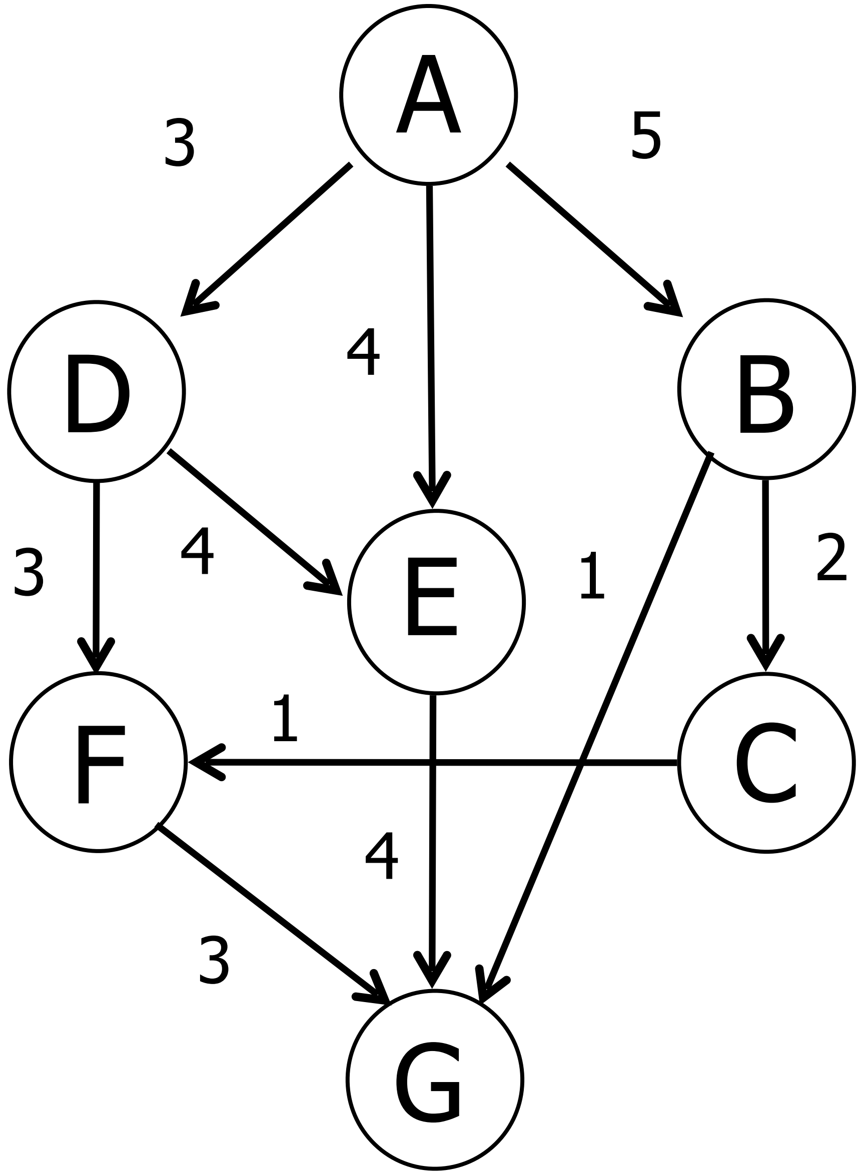


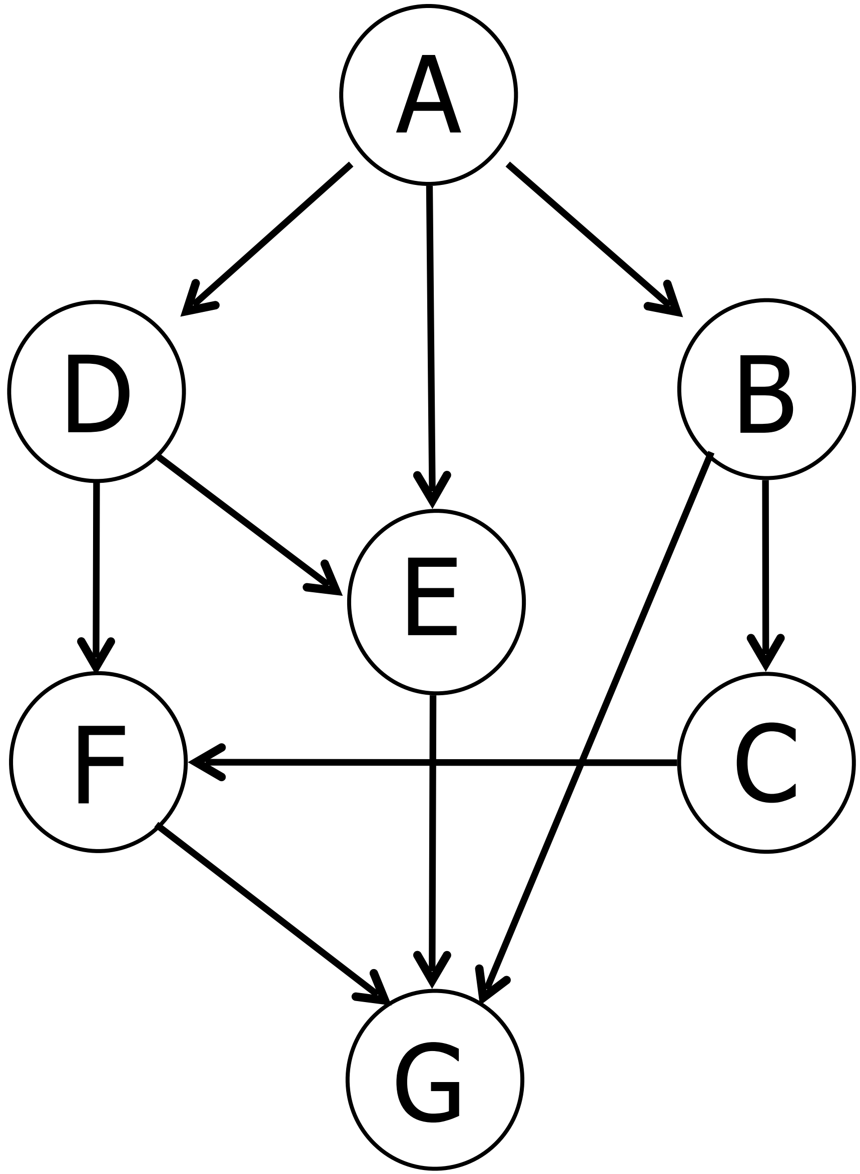
- $O(|V|^2)$ Linear scan to find min
- $O(|E|\log|V|)$ Binary heaps

Directed Graph - Weighted






A
B
C
D
E
F
G
3
5
4
4
3
1
2
1
4
3


A
D
E
B
F
C
G