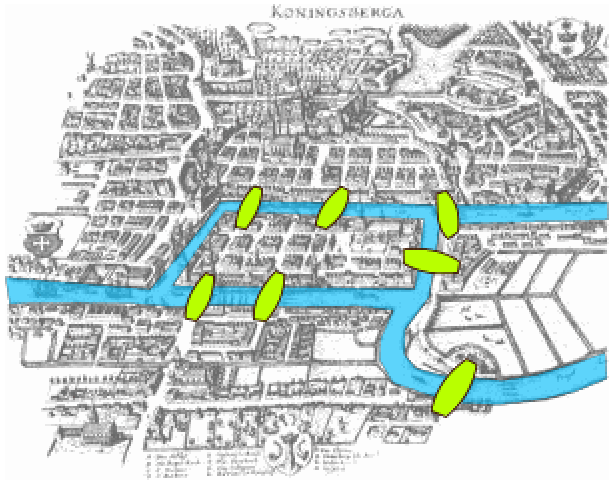
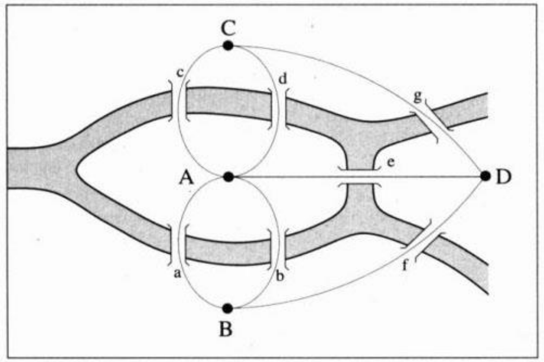
Graph

Industrial AI Lab.
Prof. Seungchul Lee

Graph Theory

- Abstract relations, topology, or connectivity
- Graphs $G = (V, E)$
  - V: a set of vertices (nodes)
  - E: a set of edges (links, relations)
  - weight (edge property)
    - distance in a road network
    - strength of connection in a personal network





Graph Theory

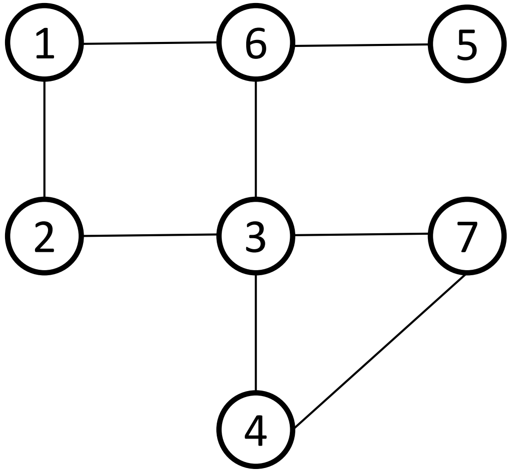
- Graphs can be *directed* or *undirected*

- Graphs model any situation where you have objects and pairwise relationships (symmetric or asymmetric) between the objects

Vertex	**Edge**	
People	like each other	undirected
People	is the boss of	directed
Tasks	cannot be processed at the same time	undirected
Computers	have a direct network connection	undirected
Airports	planes flies between them	directed
City	can travel between them	directed

Adjacent Matrix

- Undirected graph $G = (V, E)$



- Let computers to understand a structure of graph

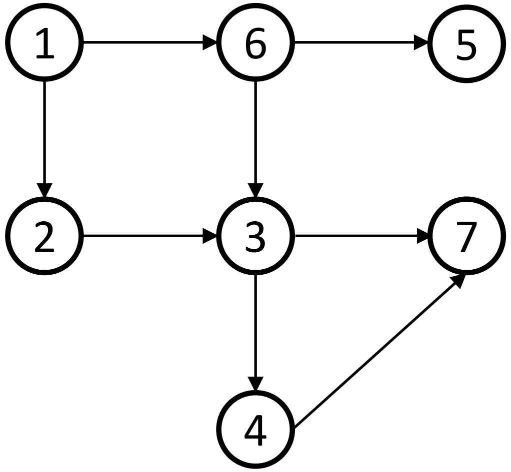
$$V = \{1,2,\cdots,7\}$$

$$E = \{\{1,2\},\{1,6\},\{2,3\},\{3,4\},\{3,6\},\{3,7\},\{4,7\},\{5,6\}\}$$

Adjacency list	Adjacency matrix (symmetric)
$\mathrm{adj}(1) = \{2,6\}$ $\mathrm{adj}(2) = \{1,3\}$ $\mathrm{adj}(3) = \{2,4,6,7\}$ $\mathrm{adj}(4) = \{3,7\}$ $\mathrm{adj}(5) = \{6\}$ $\mathrm{adj}(6) = \{1,3,5\}$ $\mathrm{adj}(7) = \{3,4\}$	$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$

Adjacent Matrix

- Directed graph $G = (V, E)$



- Let computers to understand a structure of graph

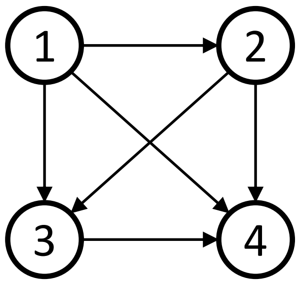
$$V = \{1,2,\cdots,7\}$$

$$E = \left\{\{1,2\},\{1,6\},\{2,3\},\{3,4\},\{3,7\},\{4,7\},\{6,3\},\{6,5\}\right\}$$

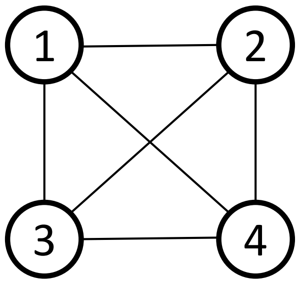
Adjacency list	Adjacency matrix (asymmetric)
$\mathrm{adj}(1) = \{2,6\}$ $\mathrm{adj}(2) = \{3\}$ $\mathrm{adj}(3) = \{4,7\}$ $\mathrm{adj}(4) = \{7\}$ $\mathrm{adj}(5) = \phi$ $\mathrm{adj}(6) = \{3,5\}$ $\mathrm{adj}(7) = \phi$	$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

Quiz 1

- Directed graph

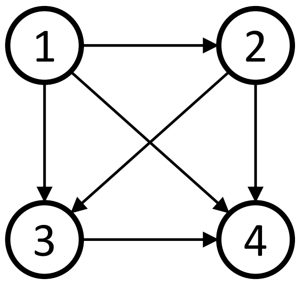


- Undirected graph



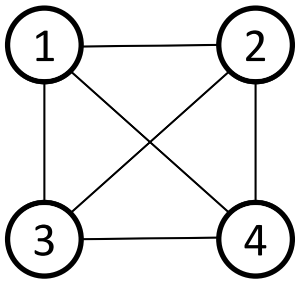
Quiz 1

- Directed graph



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

- Undirected graph



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

Quiz 2

- Directed graph G = (V,E)
  - V = {0,1,2,3,4,5}
  - Adjacency list

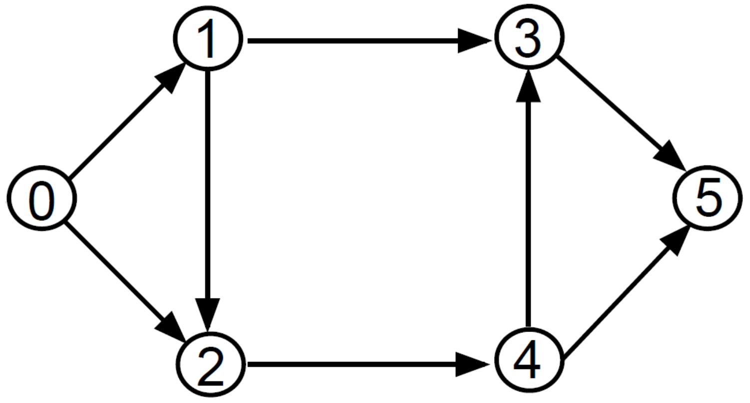
$$
\begin{aligned}
Adj(0) &= \{1,2\} \\
Adj(1) &= \{2,3\} \\
Adj(2) &= \{4\} \\
Adj(3) &= \{5\} \\
Adj(4) &= \{3,5\} \\
Adj(5) &= \emptyset
\end{aligned}
$$

- Q: draw the corresponding directed graph

Quiz 2

- Directed graph G = (V,E)
  - V = {0,1,2,3,4,5}
  - Adjacency list

$$
\begin{aligned}
Adj(0) &= \{1, 2\} \\
Adj(1) &= \{2, 3\} \\
Adj(2) &= \{4\} \\
Adj(3) &= \{5\} \\
Adj(4) &= \{3, 5\} \\
Adj(5) &= \emptyset
\end{aligned}
$$



- Q: draw the corresponding directed graph

What Can We Do with a Graph?

- Graph represents abstract relations, topology, or connectivity

- Optimization with a graph
  1) Graph Search Problem
  2) Path Finding Problem
  3) Shortest Path Problem

- There are many engineering applications using such problems
  – Navigation
  – Internet search

1) Graph Search Problem

- Given:
  - a graph $G = (V, E)$ (directed or undirected) and a start node $s \in V$
- Find:
  - a set of nodes $v \in V$ that are *reachable* from node s (i.e., there exist a path from s)
    - Breadth-first search
    - Depth-first search

- Searching a graph
  - Systematically follow the edges of a graph to visit the vertices of the graph

- Used to discover the structure of a graph

Graph Search Problem

- Basic idea
  - Starting with s, move along edges to visit nodes while “marking” the visited nodes to prevent re-visiting. Repeat until there are no unmarked nodes that can be visited by moving along the edges
- Algorithm

Input: graph $G = (V, E)$ and start node $s \in V$.
Output: “mark” on each node reachable from s.

`Graph-Search`(G, s)
    $\triangleright$ $G = (V, E)$ is passed by reference
    $\triangleright$ “marks” all nodes reachable from $s \in V$
  $Q \leftarrow \{s\}$
  while $Q \neq \emptyset$
      do select an element $v \in Q$
          $Q \leftarrow Q \setminus \{v\}$
          mark v
          for each $w \in Adj(v)$
              do if w is not marked
                  then $Q \leftarrow Q \cup \{w\}$
  return

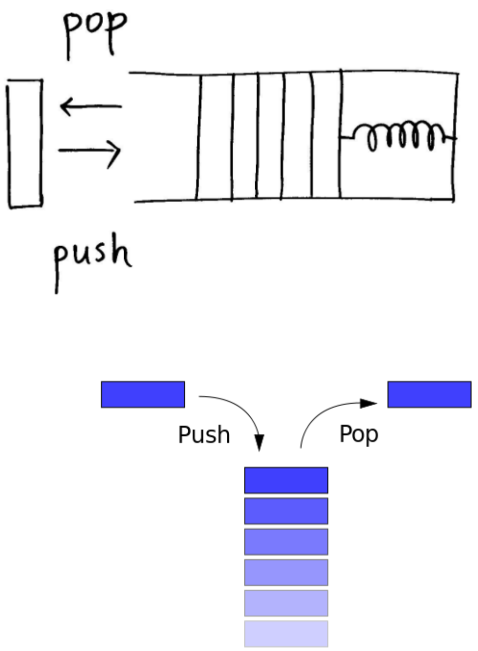
Graph Search Algorithm

- Maintains a set Q of nodes that are "about to be marked." Starts with $Q = \{s\}$ and terminates when $Q = \emptyset$.
- At each iteration, a node v is randomly selected and removed from Q and marked. Then *unmarked* nodes adjacent to v are added to Q.
- Throughout iterations, Q is a *fringe* of a set S of marked nodes,
  - *i.e.*, $Q = \{v | (u, v) \in E, u \in S, v \notin S\}$

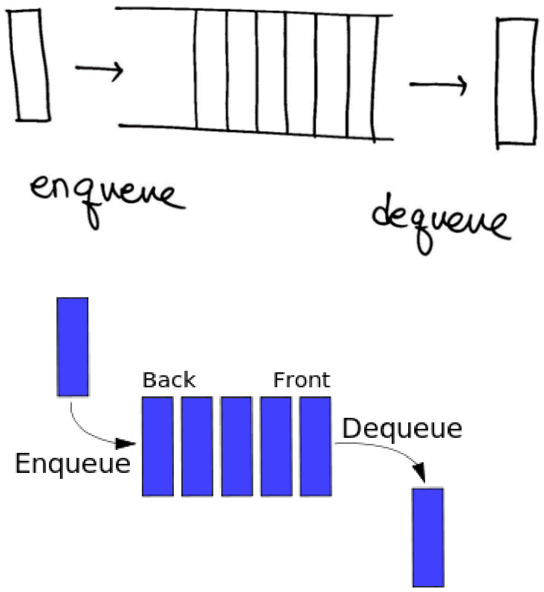
Graph-Search(G, s)
    $\triangleright$ $G = (V, E)$ is passed by reference
    $\triangleright$ "marks" all nodes reachable from $s \in V$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$
    do select an element $v \in Q$
        $Q \leftarrow Q \setminus \{v\}$
        mark v
        for each $w \in Adj(v)$
            do if w is not marked
                then $Q \leftarrow Q \cup \{w\}$
return

Stack and Queue

- $Q \equiv stack$ (LIFO: last-in-first-out) ⇒ Depth-First
- $Q \equiv queue$ (FIFO: first-in-first-out) ⇒ Breadth-First
- $Q \equiv priority\ queue$ (minimum-cost-first-out) ⇒ Dijkstra, A*



Stack: Last-in-First-out (LIFO)



Queue: First-in-First-out (FIFO)

Depth-First Search (DFS)

`Graph-Search` with $Q \equiv$ *stack* (LIFO: last-in-first-out) $\longrightarrow$ a node lastly added to Q is selected first.

Input: graph $G = (V, E)$ and start node $s \in V$.

Output: "mark" on each node reachable from s in *depth-first* order.

`Depth-First`(G, s)

$Q \leftarrow \{s\}$
while $Q \neq \emptyset$
    do select an element $v \in Q$ s.t. Q is a stack
    $\triangleright$ last-in-first-out (LIFO)
        $Q \leftarrow Q \setminus \{v\}$
        mark v
        for each $w \in Adj(v)$
            do if w is not marked
                then $Q \leftarrow Q \cup \{w\}$
return

Breadth-First Search (BFS)

`Graph-Search` with $Q \equiv$ *queue* (FIFO: first-in-first-out) $\longrightarrow$ a node added to Q first is selected first.

Input: graph $G = (V, E)$ and start node $s \in V$.

Output: "mark" on each node reachable from s in *breadth-first* order.

`Breadth-First`(G, s)

$Q \leftarrow \{s\}$
while $Q \neq \emptyset$
    do select an element $v \in Q$ s.t. Q is a queue
      $\triangleright$ first-in-first-out (FIFO)
        $Q \leftarrow Q \setminus \{v\}$
        mark v
        for each $w \in Adj(v)$
            do if w is not marked
                then $Q \leftarrow Q \cup \{w\}$
return

DFS Example

- Start from vertex 0 and want to visit all vertices

Since $Q = \{0\} \neq \emptyset$, proceed

Select 0 from Q

$Q = \{0\} \setminus \{0\} = \emptyset$

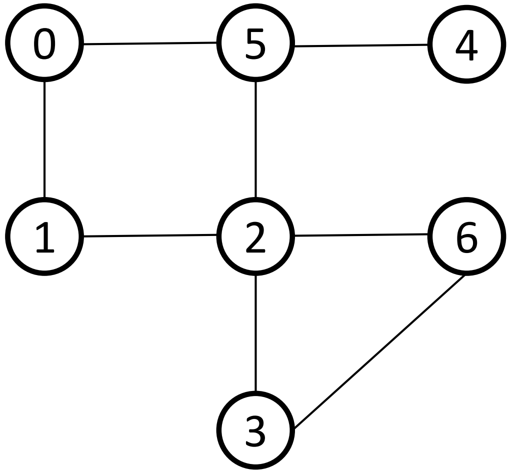
Mark 0

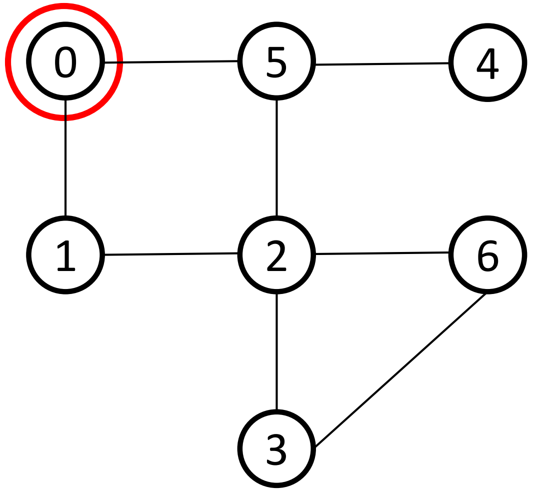
For each node in $Adj(0) = \{1,5\}$

1 is not marked → $Q = \{1\}$

5 is not marked → $Q = \{5, 1\}$

At the end of this iteration, $Q = \{5, 1\}$





DFS Example

Since $Q = \{5, 1\} \neq \emptyset$, proceed

Select 5 from Q

$Q = \{5, 1\} \backslash \{5\} = \{1\}$

Mark 5

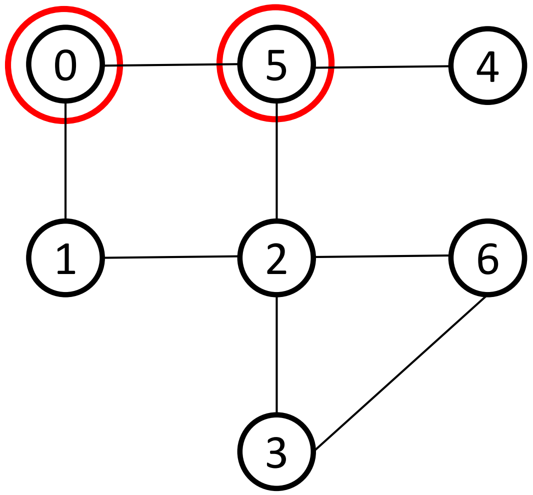
For each node in $Adj(5) = \{0, 2, 4\}$

0 is marked

2 is not marked $\rightarrow$ $Q = \{2, 1\}$

4 is not marked $\rightarrow$ $Q = \{4, 2, 1\}$

At the end of this iteration, $Q = \{4, 2, 1\}$



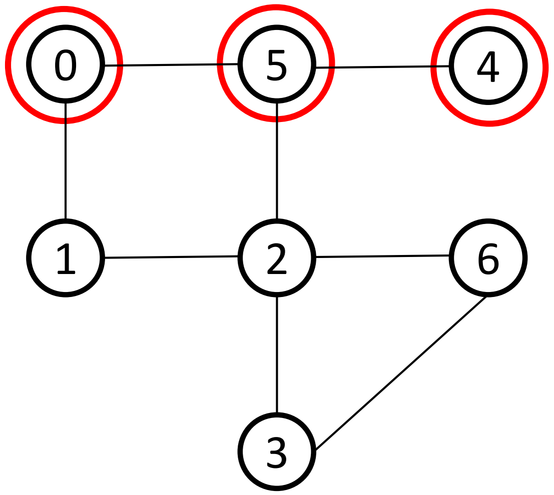
DFS Example

Since $Q = \{4, 2, 1\} \neq \emptyset$, proceed

Select 4 from Q
  $Q = \{4, 2, 1\} \backslash \{4\} = \{2, 1\}$
  Mark 4

For each node in $Adj(4) = \{5\}$
  5 is marked

At the end of this iteration, $Q = \{2, 1\}$



DFS Example

Since $Q = \{2, 1\} \neq \emptyset$, proceed

Select 2 from Q

$Q = \{2, 1\} \setminus \{2\} = \{1\}$

Mark 2

For each node in $Adj(2) = \{1, 3, 5, 6\}$

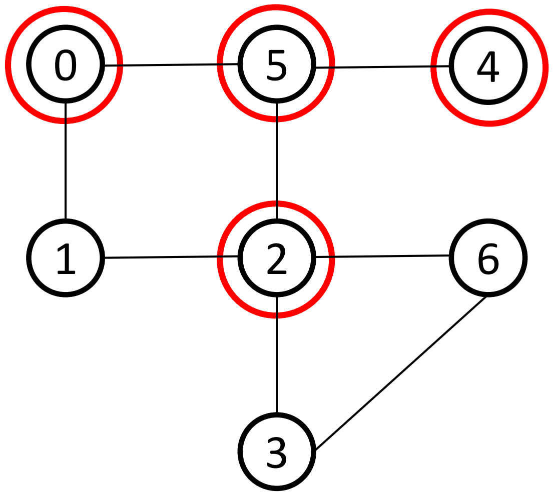
1 is not marked → $Q = Q \cup \{1\} = \{1\}$

3 is not marked → $Q = \{3, 1\}$

5 is marked

6 is not marked → $Q = \{6, 3, 1\}$

At the end of this iteration, $Q = \{6, 3, 1\}$



DFS Example

Since $Q = \{6, 3, 1\} \neq \emptyset$, proceed

Select 6 from Q

$Q = \{6, 3, 1\} \backslash \{6\} = \{3, 1\}$

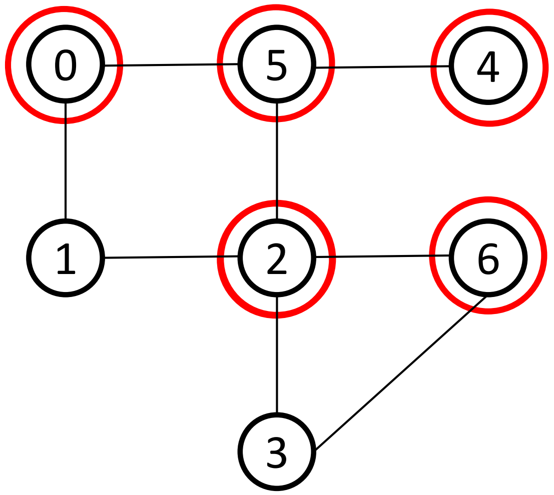
Mark 6

For each node in $Adj(6) = \{2, 3\}$

2 is marked

3 is not marked $\rightarrow$ $Q = \{3, 1\}$

At the end of this iteration, $Q = \{3, 1\}$



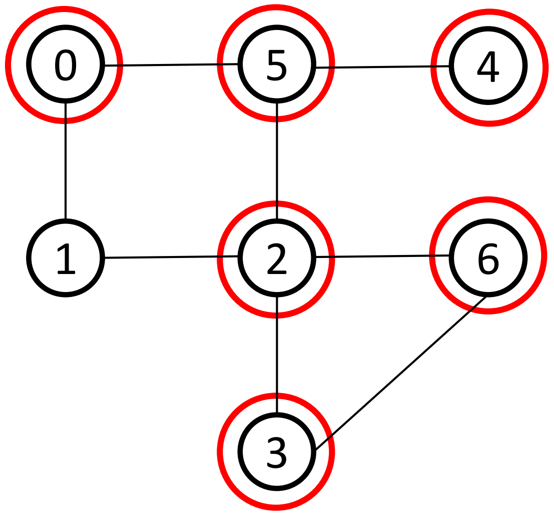
DFS Example

Since $Q = \{3, 1\} \neq \emptyset$, proceed

Select 3 from Q
  $Q = \{3, 1\} \backslash \{3\} = \{1\}$
  Mark 3

For each node in $Adj(3) = \{2, 6\}$
  2 is marked
  6 is marked

At the end of this iteration, $Q = \{1\}$



DFS Example

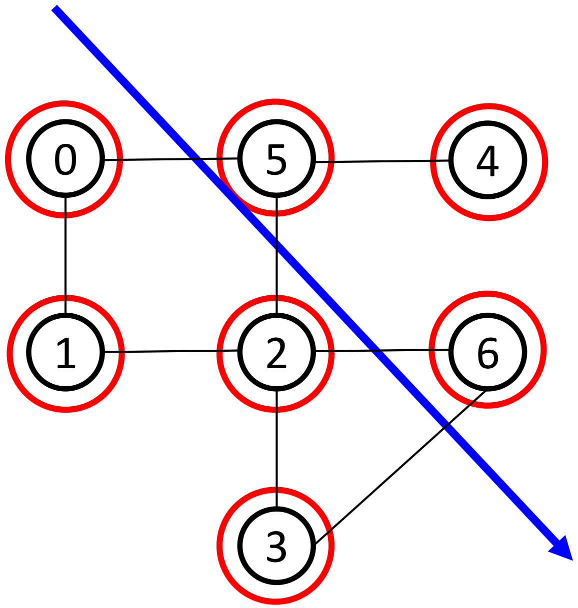
- Hope you to understand why it is **depth**-first search
- Tends to go deeper

Since $Q = \{1\} \neq \emptyset$, proceed

Select 1 from Q
  $Q = \{1\} \setminus \{1\} = \emptyset$
  Mark 1

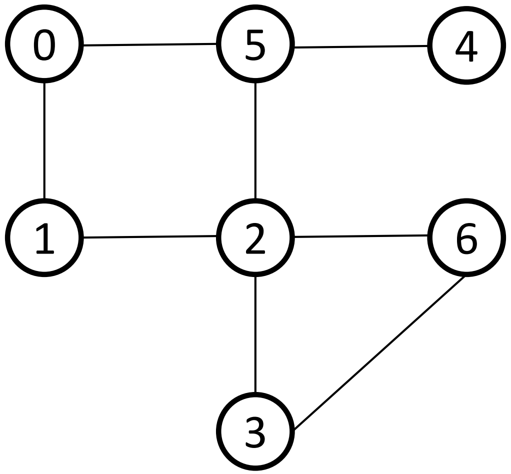
For each node in $Adj(1) = \{0, 2\}$
  0 is marked
  2 is marked

At the end of this iteration, $Q = \emptyset$
done



Quiz

- Start from vertex 1 and want to visit all vertices
- Use queue for Breadth-first search



2) Path Finding Problem

- Given:
  - a graph $G = (V, E)$ (directed or undirected) and a start node $s \in V$
- Find:
  - path *p* from *s* to each node $v \in V$

- *Graph-Search* with additional termination conditions
  - terminates as soon as a path is found from some $s \in S$ to some $t \in T$

Path Finding Algorithm

Input: graph $G = (V, E)$, start node set $S \subseteq V$, and goal node set $T \subseteq V$.

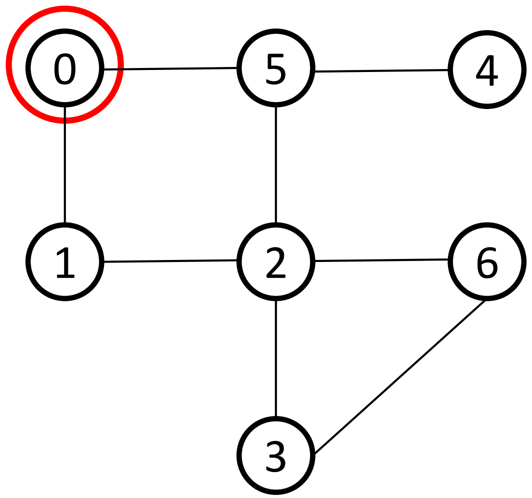
Output: path p from node $s \in S$ to node $t \in T$.

Find-Path(G, S, T)

for each $v \in V[G]$
  $label[v] \leftarrow nil$
for each $v \in S$
  if $v \in T$
    then return $\langle v \rangle$
$Q \leftarrow S$
while $Q \neq \emptyset$
  do select an element $v \in Q$
    $Q \leftarrow Q \setminus \{v\}$
    mark v
    if $v \in T$
      then return path(v)
    for each $w \in Adj(v)$
      do if w is not marked
          then $label[w] \leftarrow v$
            $Q \leftarrow Q \cup \{w\}$
return FALSE $\triangleright$ no path from $s \in S$ to $t \in T$

Path Finding Example

- Find a path from Start node 0 to Goal node 3



S = 0
T = 3

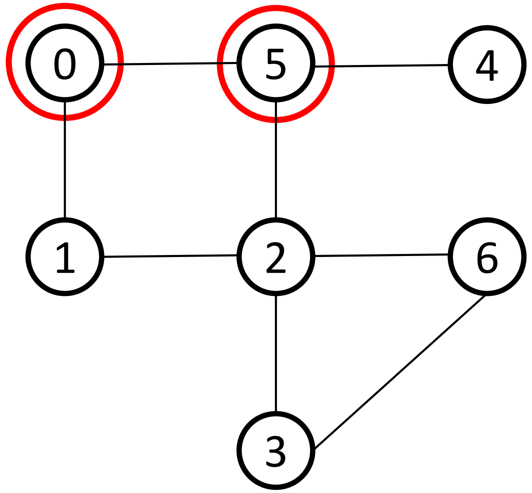
Since $Q = \{0\} \neq \emptyset$, proceed

Select 0
  $Q = \{0\} \backslash \{0\} = \emptyset$
  Mark 0

$0 \neq 3\ (= T)$

For each node in $Adj(0) = \{1, 5\}$
  1 is not marked → $\text{label}[1] = 0,\ Q = \{1\}$
  5 is not marked → $\text{label}[5] = 0,\ Q = \{5, 1\}$

Path Finding Example



Since $Q = \{5, 1\} \neq \emptyset$, proceed

Select 5

$Q = \{5, 1\} \backslash \{5\} = \{1\}$

Mark 5

$5 \neq 3\ (= T)$

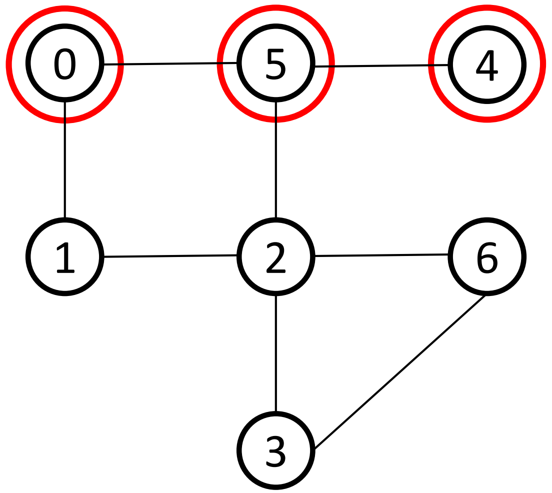
For each node in $Adj(5) = \{0, 2, 4\}$

0 is marked

2 is not marked → $\mathrm{label}[2] = 5\ \ Q = \{2, 1\}$

4 is not marked → $\mathrm{label}[4] = 5,\ \ Q = \{4, 2, 1\}$

Path Finding Example



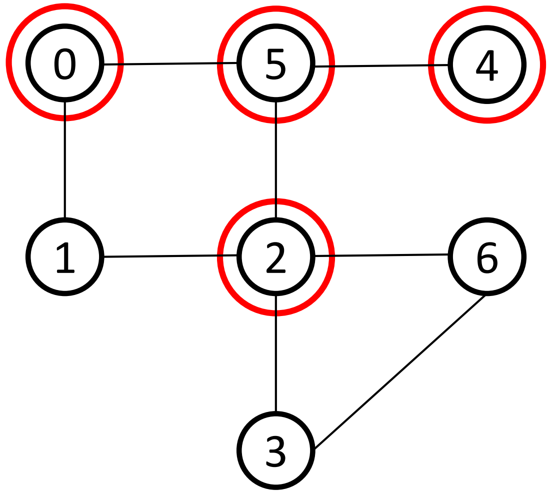
Since $Q = \{4, 2, 1\} \neq \emptyset$, proceed

Select 4
  $Q = \{4, 2, 1\} \setminus \{4\} = \{2, 1\}$
  Mark 4

$4 \neq 3 \ (= T)$

For each node in $Adj(4) = \{5\}$
  5 is marked

Path Finding Example



Since $Q = \{2, 1\} \neq \emptyset$

Select 2

$Q = \{2, 1\} \setminus \{2\} = \{1\}$

Mark 2

$2 \neq 3\ (= T)$

For each node in $Adj(2) = \{1, 3, 5, 6\}$

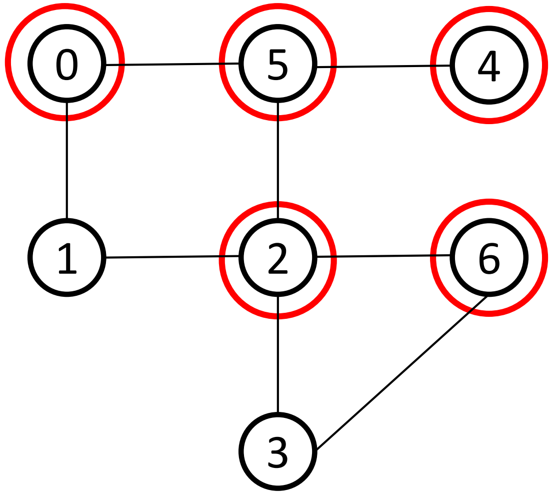
1 is not marked → $\text{label}[1] = 2,\ Q = Q \cup \{1\} = \{1\}$

3 is not marked → $\text{label}[3] = 2,\ Q = \{3, 1\}$

5 is marked

6 is not marked → $\text{label}[6] = 2,\ Q = \{6, 3, 1\}$

Path Finding Example



Since $Q = \{6, 3, 1\} \neq \emptyset$, proceed

Select 6
  $Q = \{6, 3, 1\} \backslash \{6\} = \{3, 1\}$
  Mark 6

$6 \neq 3 \ (= T)$

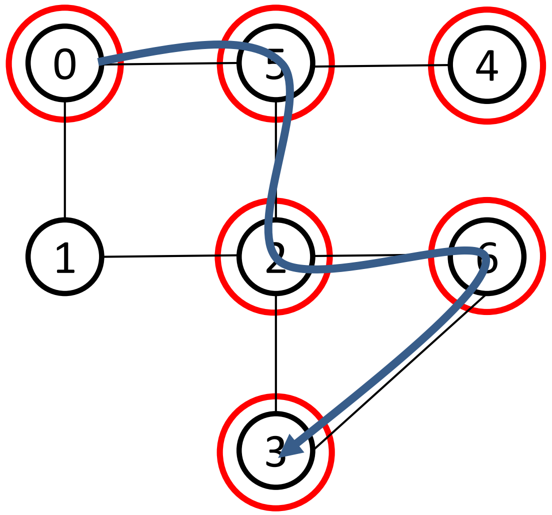
For each node in $Adj(6) = \{2, 3\}$
  2 is marked
  3 is not marked → $\text{label}[3] = 6, \ Q = \{3, 1\}$

Path Finding Example

Since $Q = \{3, 1\} \neq \emptyset$, proceed

Select 3
$Q = \{3, 1\} \backslash \{3\} = \{1\}$
Mark 3

$3 = 3\ (= T)$ → Return Path(3)

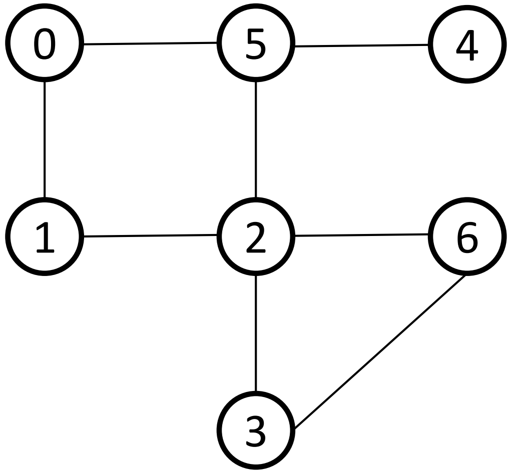


$$4 \to 7 \to 3 \to 6 \to 1$$

Recursively backtracking

Quiz

- Use *queue* for path finding from node 0 to node 6



3) Shortest Path Problem

- Given:
  - a graph $G = (V, E)$ (directed or undirected) with edge weights $c_{uv} \in \mathbb{R}$ and a start node $s \in V$

- Find:
  - shortest path length $\delta(s, v)$ from s to node $v \in V$

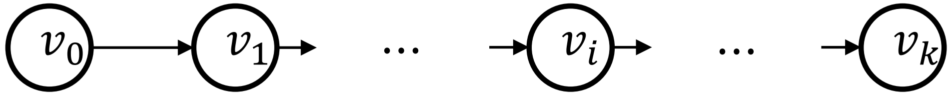
$$\delta(s, v) = \min_{p \in P_{sv}} c(p)$$

  - where $P_{sv} = \{s \to v\}$ is a set of paths from s to v, and $c(p)$ is length (cost) of path p
  - Often interested in shortest path p^* itself

  - Dynamic programming, Dijkstra's algorithm, Bellman-Ford algorithm, A* algorithm

Path

- $p = \langle v_0, \cdots, v_k \rangle$ is a path from v_0 to v_k

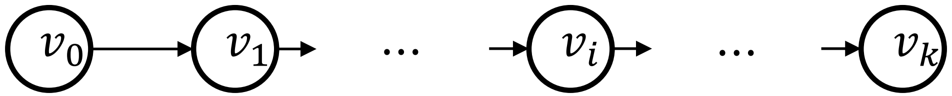


$$p = \langle v_0, v_1, \cdots, v_i, \cdots, v_k \rangle$$

- Length of path: $|p| \equiv$ number of $edges$ in p
  - $p \circ q \equiv$ "concatenation of two paths p and q."

Cost of Path

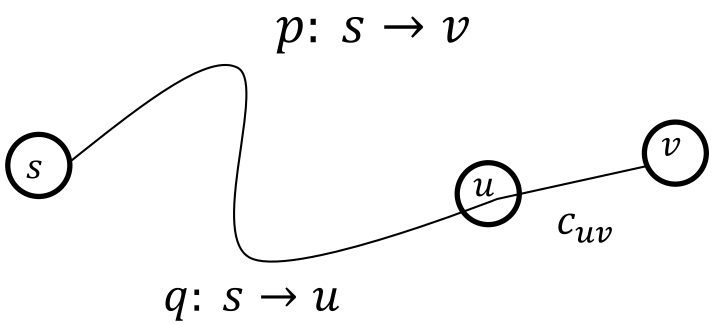
$$c(p) = \sum_{(u,v)\in E_p} c_{uv}$$



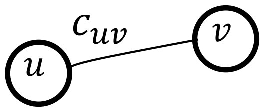
"Length" (cost) of path p (recursive definition):

$$c(p) = \begin{cases} 0 & \text{if } |p| = 0 \\ c(q) + c_{uv} & \text{otherwise, where } p = q \circ \langle u, v \rangle \end{cases}$$

$$C(p) = C(q) + C_{uv} \qquad \text{where } p = q \circ \langle u, v \rangle$$



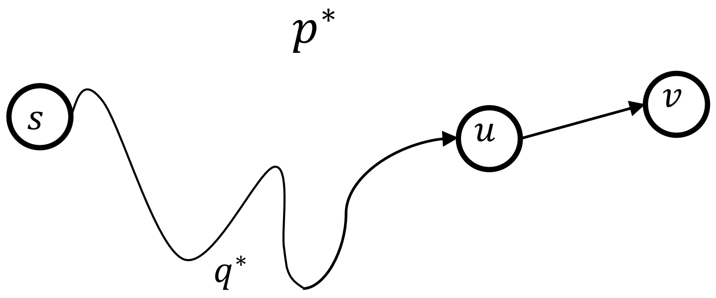
- "Shortest path" = minimum cost path



Optimal Structure of Shortest Path

Theorem: Let p^* to be a shortest path from s to v and $p^* = q^* \circ \langle u, v \rangle$. Then q^* is a shortest path from s to u.

Interpretation: subpaths of a shortest path are also shortest.



$$p^* = q^* \circ \langle u, v \rangle$$

Optimal Structure of Shortest Path with DP

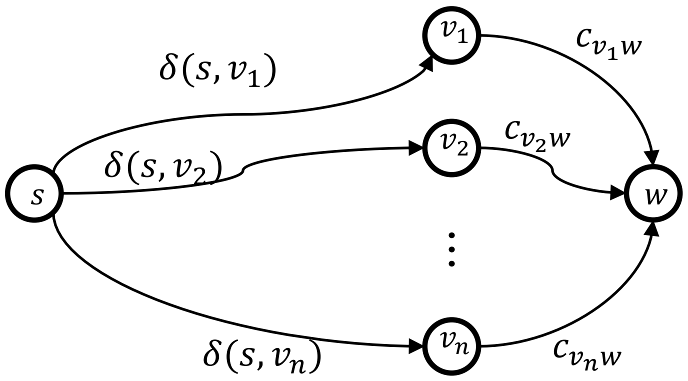
- Revisit DP
  - Memorize (remember) & re-use solutions to subproblems that helps solve the problem

$$\text{key ideas} = \text{original problem} \rightarrow \begin{cases} \text{subproblem}\rightarrow & \begin{cases} \text{subproblem}\rightarrow \\ \text{subproblem}\rightarrow \end{cases} \\ \text{subproblem}\rightarrow & \begin{cases} \text{subproblem}\rightarrow \\ \text{subproblem}\rightarrow \end{cases} \end{cases}$$

- Shortest path optimization

$$\delta(s,v) = \min_{p \in P_{sv}} c(p)$$

$$\delta(s,w) = \min_{v \in \{u \mid w \in Adj(u)\}} \{\delta(s,v) + c_{vw}\}$$



Dijkstra's Algorithm

- Dijkstra's Algorithm = Dynamic programming on graph
- Graph-Search with $Q \equiv priority\ queue$ on $\rho[v]$
  - a node v with minimum $\rho[v]$ is selected first
  - minimum-cost-first-out ⇒ Dijkstra

Basic idea: initially $\rho[v] = \infty$ for all nodes. Starting with s, mark nodes as `Graph-Search`. When a node v is newly marked, for each node $w \in Q$ update $\rho[w]$ with

$$\rho[w] \longleftarrow \min\{\rho[w],\ \rho[v] + c_{vw}\}$$

When all nodes reachable from s are marked, $\forall v \in V, \rho[v] = \delta(s, v)$ ($\rho[v] = \infty$ if v is not reachable from s).

Dijkstra's Algorithm

Dijkstra(G, s)

$\triangleright$ when returns, $\rho[v] = \delta(s, v)$

for each $v \in V[G] \setminus \{s\}$

  do $\rho[v] \leftarrow \infty$

$\rho[s] \leftarrow 0$

$Q \leftarrow \{s\}$

while $Q \neq \emptyset$

  do select an element $v \in Q$ s.t. $\rho[v] = \min_{u \in Q} \rho[u]$

    $Q \leftarrow Q \setminus \{v\}$

    mark v

    for each $w \in Adj(v)$

      do if w is not marked

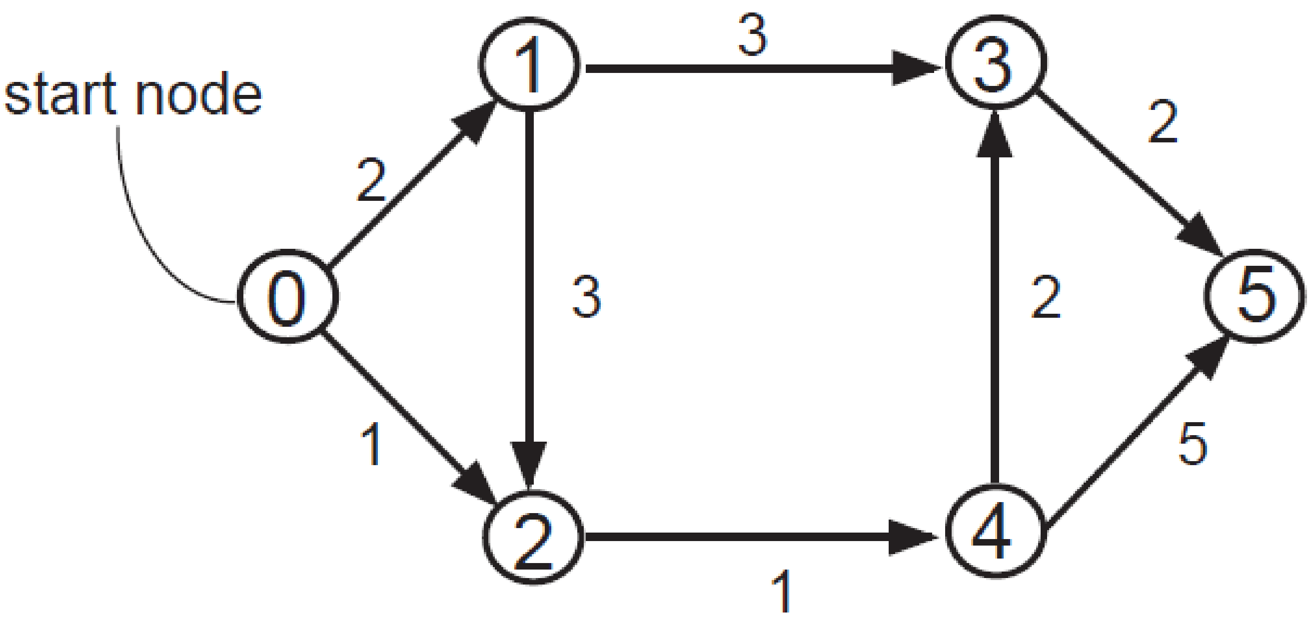
        then $\rho[w] \leftarrow \min\{\rho[w], \rho[v] + c_{vw}\}$

          $Q \leftarrow Q \cup \{w\}$

return

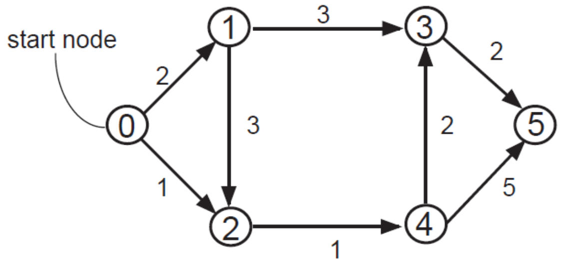
if $\quad label[w] \leftarrow v$

Example



Example

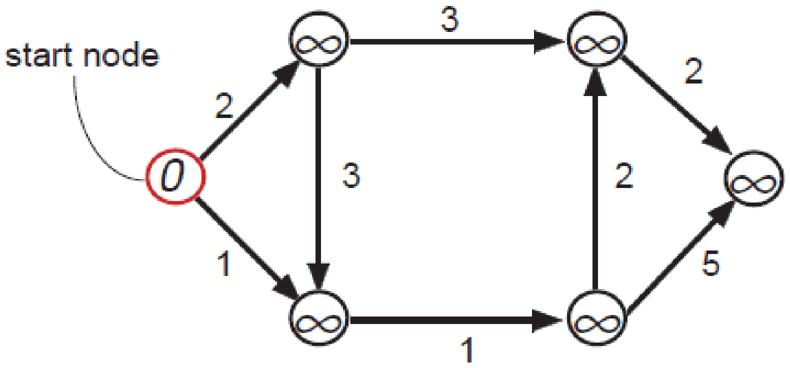
$\rho[0] = 0$
Since $Q = \{0\} \neq \emptyset$, proceed

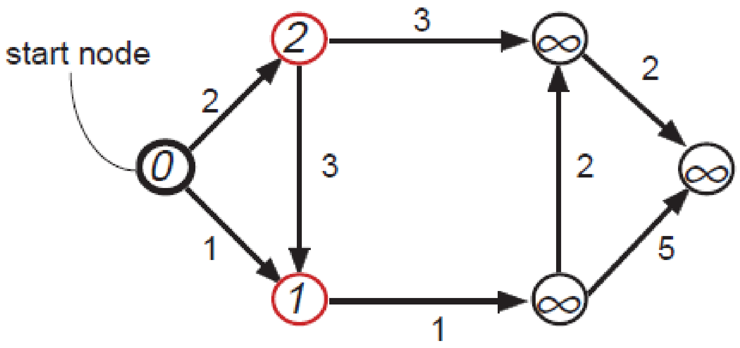


Select 0 from Q
  $Q = \{0\} \backslash \{0\} = \emptyset$
  Mark 0

For each node in $Adj(0) = \{1,2\}$
  1 is not marked → $\rho[1] = \min\{\infty, \rho[0] + 2\} = 2,\ \text{label}[1] = 0,\ Q = \{1\}$
  2 is not marked → $\rho[2] = \min\{\infty, \rho[0] + 1\} = 1,\ \text{label}[2] = 0,\ Q = \{2, 1\}$

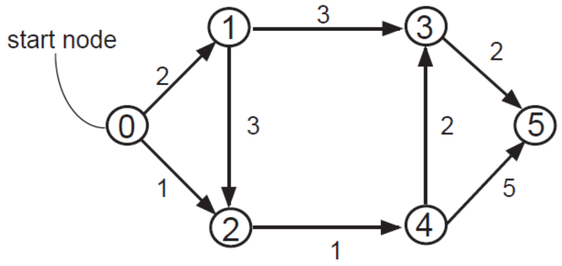
At the end of this iteration, $Q = \{2, 1\}$ and $S = \{0\}$





Example

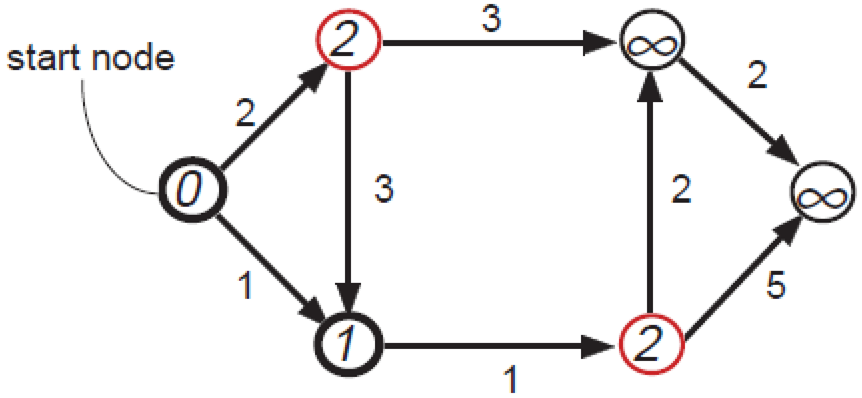
Since $Q = \{2,1\} \neq \emptyset$, proceed



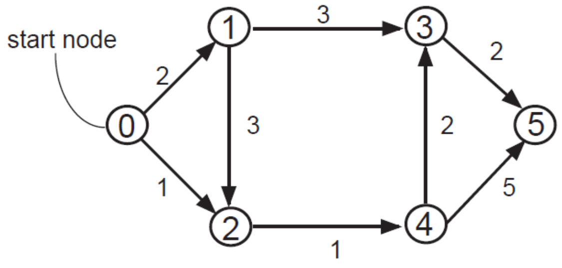
Select 2 from Q (why?)
  $Q = \{2,1\} \backslash \{2\} = \{1\}$
  Mark 2

For each node in $Adj(2) = \{4\}$
  4 is not marked → $\rho[4] = \min\{\infty, \rho[2] + 1\} = 2,\ \text{label}[4] = 2,\ Q = \{4, 1\}$

At the end of this iteration, $Q = \{4, 1\}$ and $S = \{2, 0\}$



Example



Since $Q = \{4,1\} \neq \emptyset$, proceed

Select 1 from Q

$Q = \{4,1\} \backslash \{1\} = \{4\}$

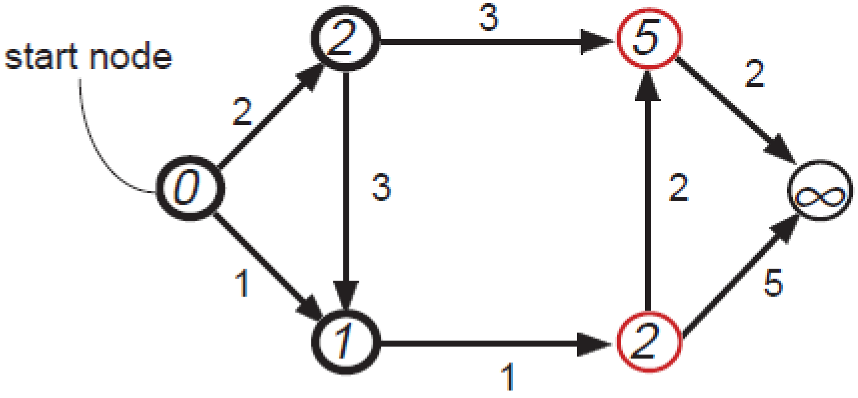
Mark 1

For each node in $Adj(1) = \{2,3\}$

2 is marked

3 is not marked → $\rho[3] = \min\{\infty, \rho[1] + 3\} = 4,\ \text{label}[3] = 1,\ Q = \{3, 4\}$

At the end of this iteration, $Q = \{3, 4\}$ and $S = \{1, 2, 0\}$



Example

Since $Q = \{3, 4\} \neq \emptyset$, proceed

Select 4

$Q = \{3,4\} \backslash \{4\} = \{3\}$

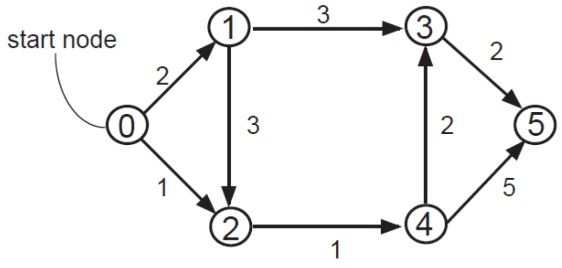
Mark 4

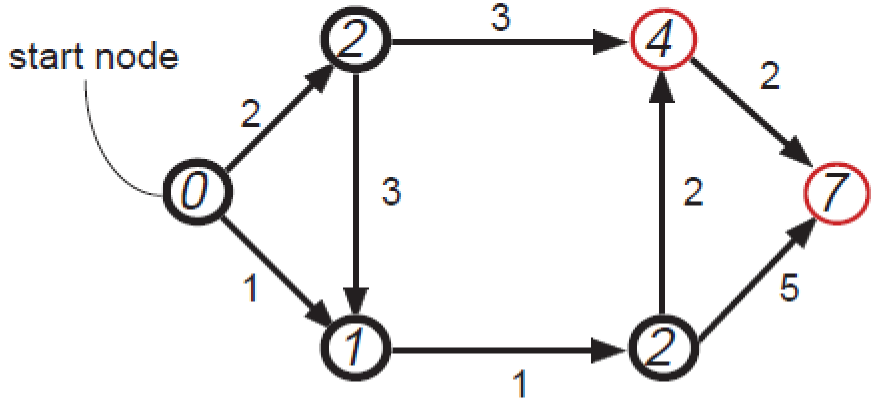
For each node in $Adj(4) = \{3,5\}$

3 is not marked → $\rho[3] = \min\{\infty, \rho[4] + 2\} = 4,\ \text{label}[3] = 4,\ Q = \{3\}$

5 is not marked → $\rho[5] = \min\{\infty, \rho[4] + 5\} = 7,\ \text{label}[5] = 4,\ Q = \{5, 3\}$

At the end of this iteration, $Q = \{5, 3\}$ and $S = \{4, 1, 2, 0\}$





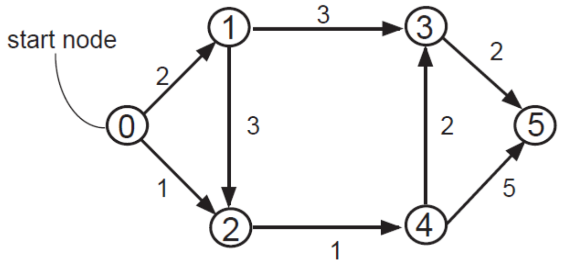
Example

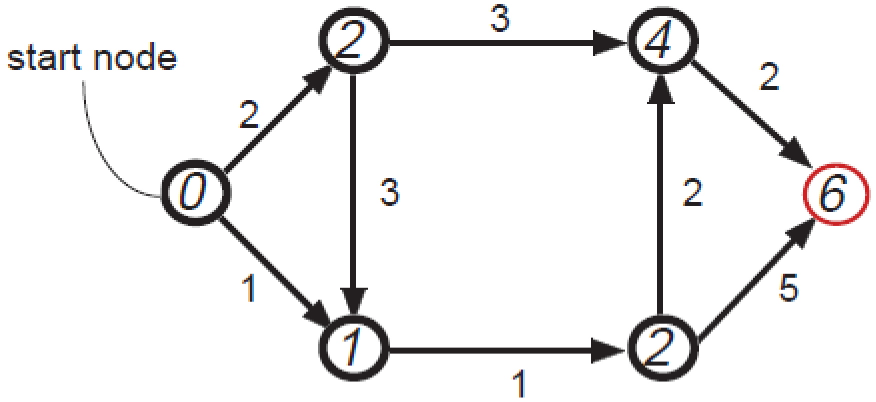
Since $Q = \{5, 3\} \neq \emptyset$, proceed

Select 3
$Q = \{5,3\} \backslash \{3\} = \{5\}$
Mark 3

For each node in $Adj(3) = \{5\}$
5 is not marked → $\rho[5] = \min\{7, \rho[3] + 2\} = 6$, $\text{label}[5] = 3$, $Q = \{5\}$

At the end of this iteration, $Q = \{5\}$ and $S = \{5, 4, 1, 2, 0\}$





Example

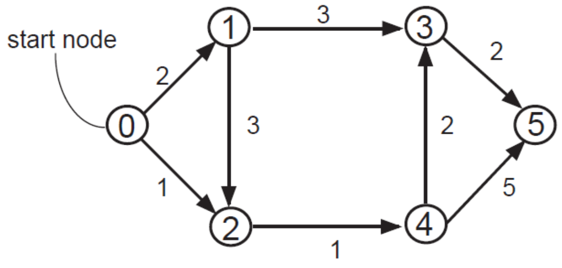
Since $Q = \{5\} \neq \emptyset$, proceed

Select 5

$Q = \{5\} \backslash \{5\} = \emptyset$

Mark 5

No Adj(5)



$\rho[0] = 0$ label$[0] =$

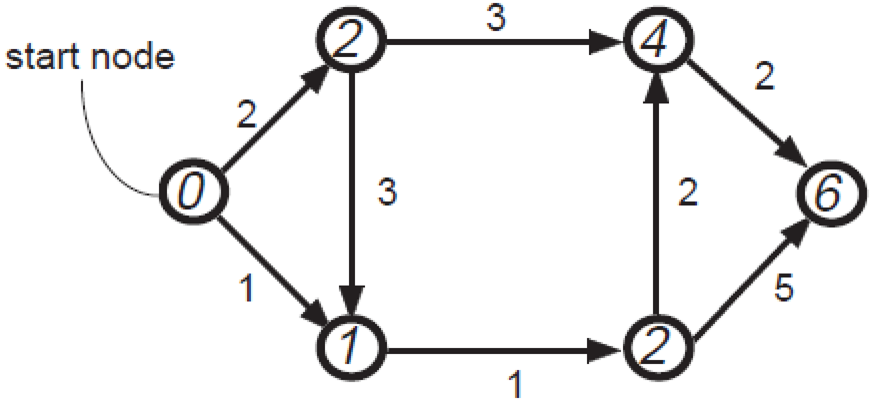
$\rho[1] = 2$ label$[1] = 0$

$\rho[2] = 1$ label$[2] = 0$

$\rho[3] = 4$ label$[3] = 4$

$\rho[4] = 2$ label$[4] = 2$

$\rho[5] = 6$ label$[5] = 3$



Probabilistic Road Maps (PRM)

- **For robot path planning**