Graph Algorithms I

Breadth First Search

Graphs

Graph G = (V, E)

V: Set of Nodes (Vertices)
E: Set of Edges

Graphs

Graph $G = (V, E)$

V: Set of Nodes (Vertices)

E: Set of Edges

Let $|V| = n$

Let $|E| = m$

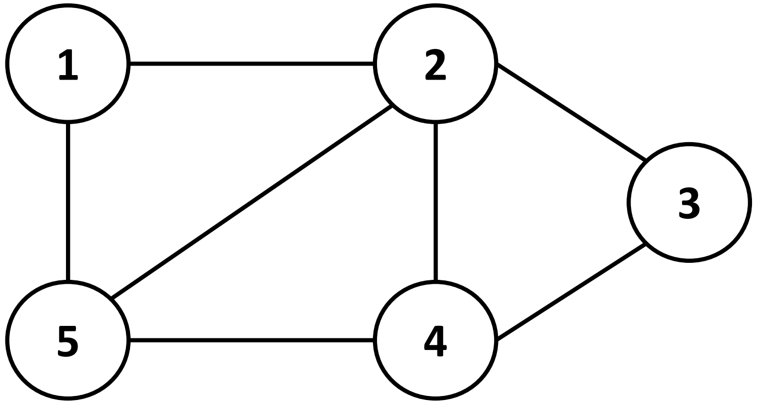
Undirected graph:

Each edge is an **unordered** pair (u,v)
i.e. (u, v) = (v, u)

Undirected graph:

Each edge is an **unordered** pair (u,v)
i.e. (u, v) = (v, u)

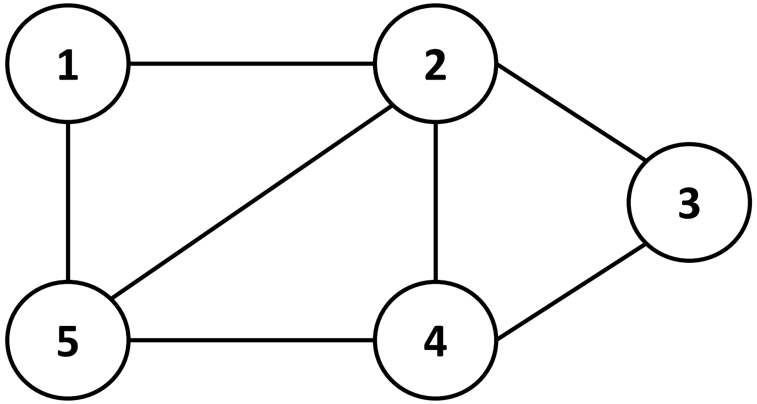
Example:



Undirected graph:

Each edge is an **unordered** pair (u,v)
i.e. (u, v) = (v, u)

Example:

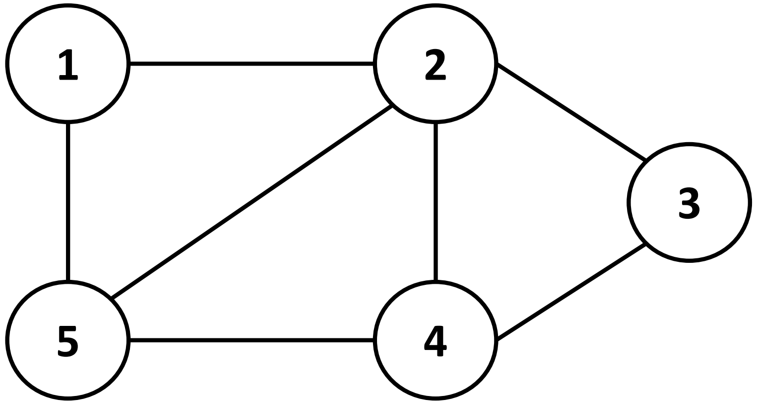


V = {1, 2, 3, 4, 5}

Undirected graph:

Each edge is an **unordered** pair (u,v)
i.e. (u, v) = (v, u)

Example:



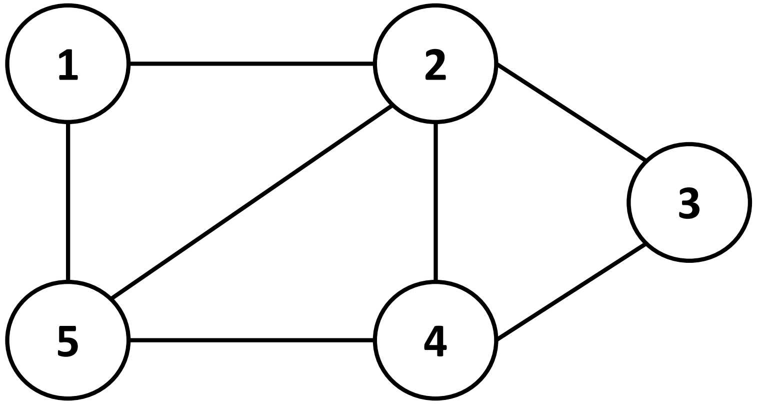
V = {1, 2, 3, 4, 5}

E = { (1, 2), (1, 5), (2, 5), (2, 4),
(4, 5), (3, 2), (3, 4) }

Undirected graph:

Each edge is an **unordered** pair (u,v)
i.e. (u, v) = (v, u)

Example:



V = {1, 2, 3, 4, 5}

E = { (1, 2), (1, 5), (2, 5), (2, 4),
(4, 5), (3, 2), (3, 4) }

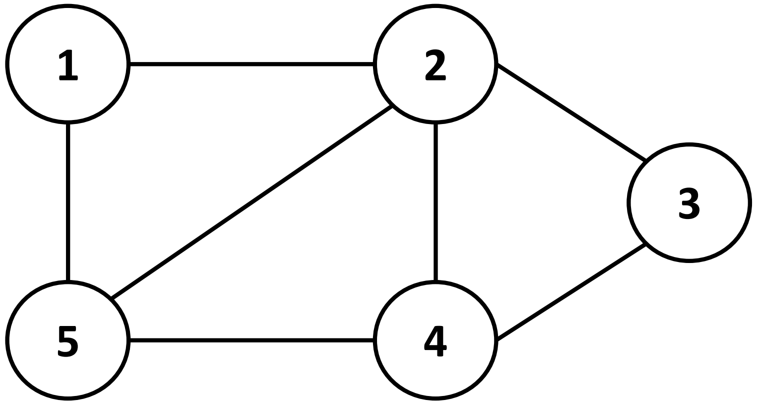
Directed graph:

Each edge is an **ordered** pair (u,v)
i.e. (u, v) $\neq$ (v, u)

Undirected graph:

Each edge is an **unordered** pair (u,v)
i.e. (u, v) = (v, u)

Example:



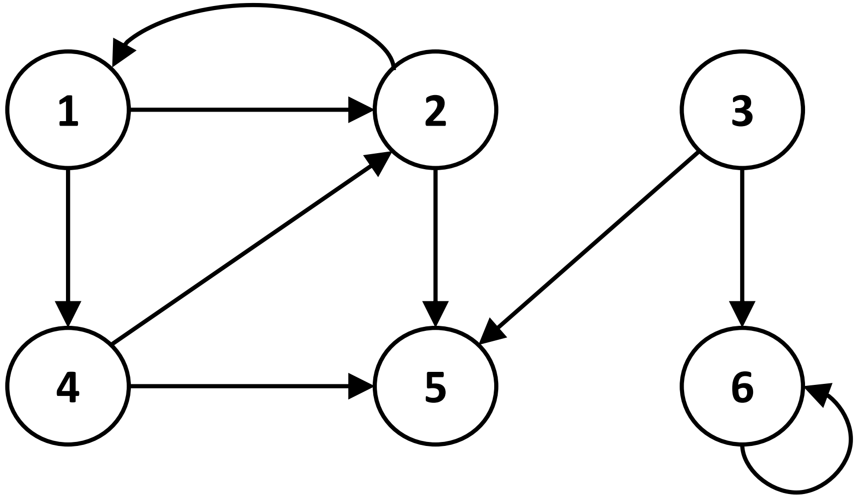
V = {1, 2, 3, 4, 5}

E = { (1, 2), (1, 5), (2, 5), (2, 4), (4, 5), (3, 2), (3, 4) }

Directed graph:

Each edge is an **ordered** pair (u,v)
i.e. (u, v) $\neq$ (v, u)

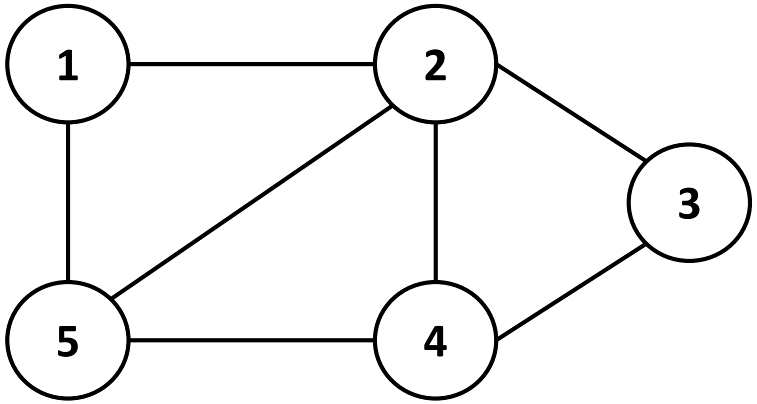
Example:



Undirected graph:

Each edge is an **unordered** pair (u,v)
i.e. $(u, v) = (v, u)$

Example:



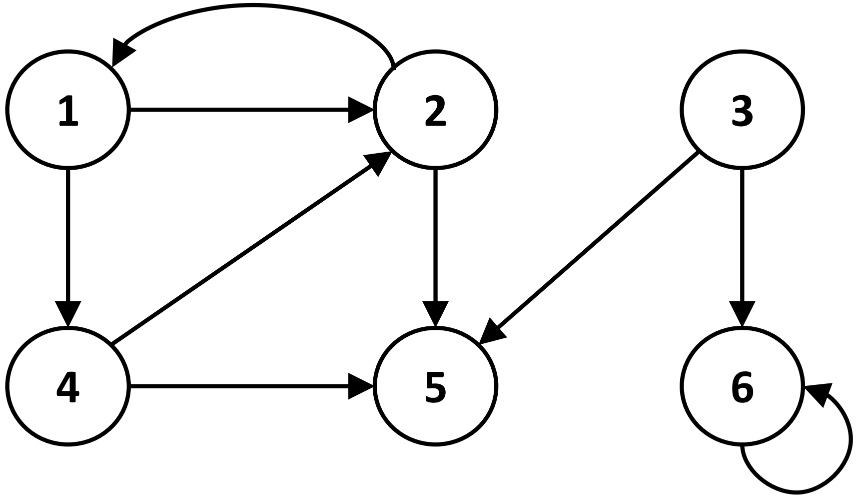
$V = \{1, 2, 3, 4, 5\}$

$E = \{ (1, 2), (1, 5), (2, 5), (2, 4), (4, 5), (3, 2), (3, 4) \}$

Directed graph:

Each edge is an **ordered** pair (u,v)
i.e. $(u, v) \neq (v, u)$

Example:

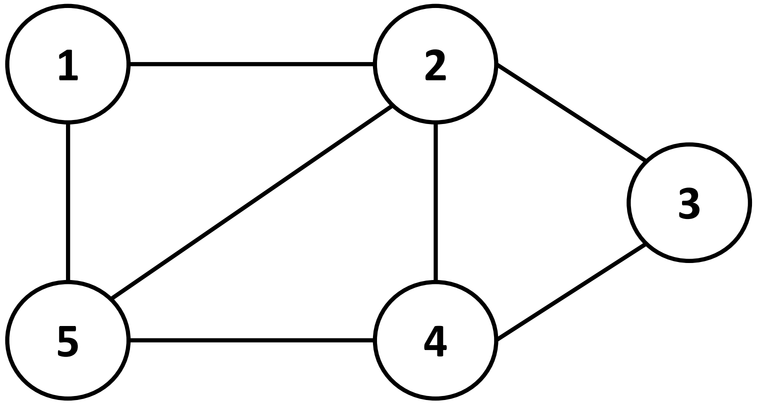


$V = \{1, 2, 3, 4, 5, 6\}$

Undirected graph:

Each edge is an **unordered** pair (u,v)
i.e. (u, v) = (v, u)

Example:



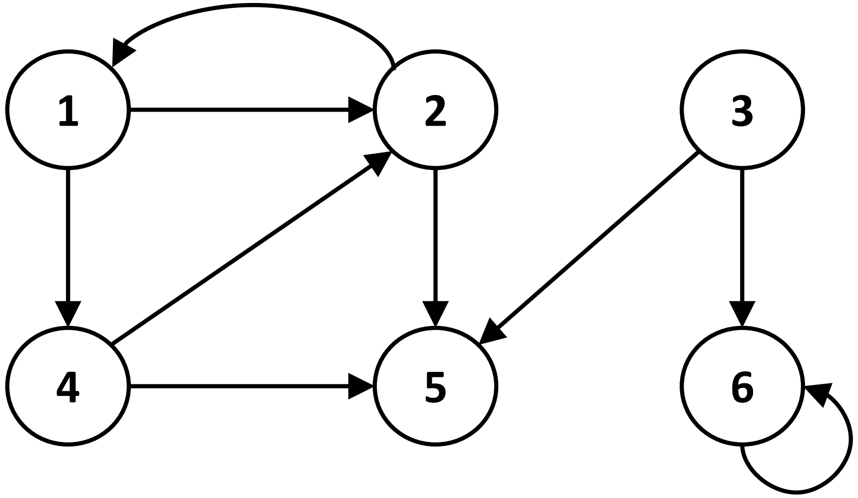
V = {1, 2, 3, 4, 5}

E = { (1, 2), (1, 5), (2, 5), (2, 4), (4, 5), (3, 2), (3, 4) }

Directed graph:

Each edge is an **ordered** pair (u,v)
i.e. (u, v) $\neq$ (v, u)

Example:



V = {1, 2, 3, 4, 5, 6}

E = { (1, 2), (2, 1), (1, 4), (4, 2), (2, 5), (4, 5), (3, 5), (3, 6), (6, 6) }

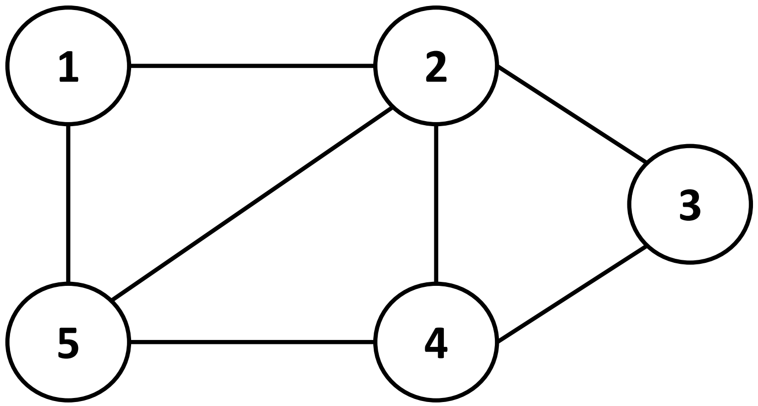
Representing graphs

Representing graphs

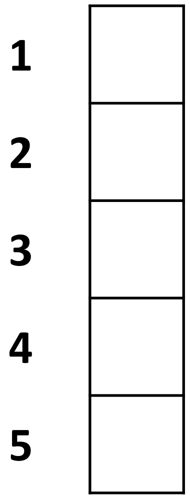
Adjacency List of $G = (V, E)$:

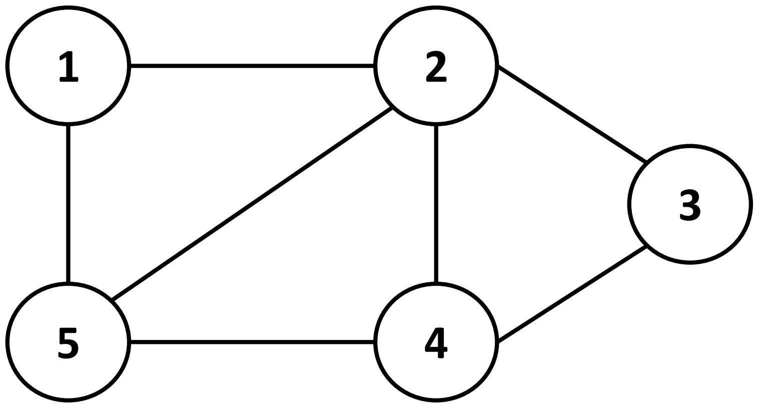
Array **Adj**[1…n] of $n = |V|$ lists, one for each $u \in V$,

Adj[u] is the list of all nodes v such that $(u,v) \in E$

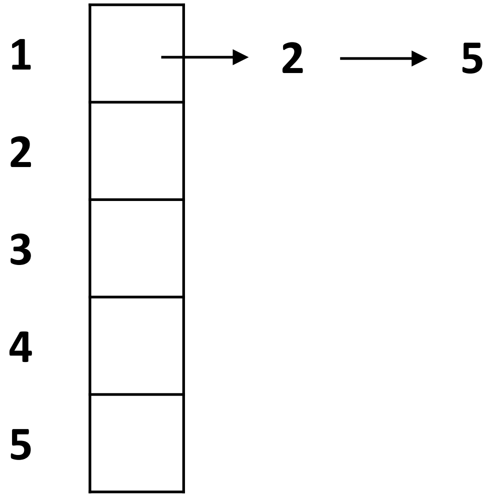


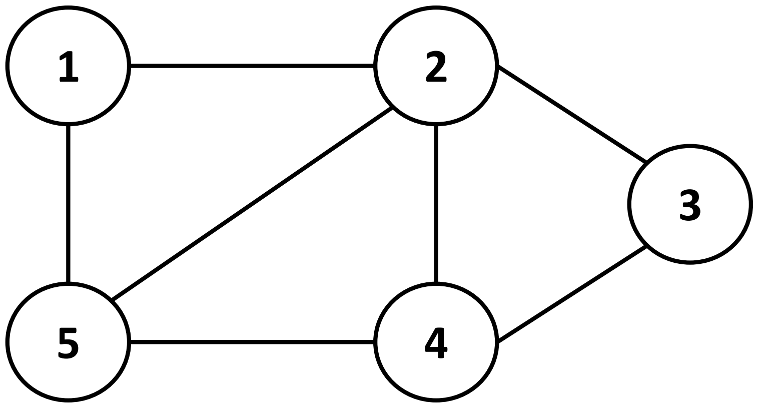
Adjacency list representation



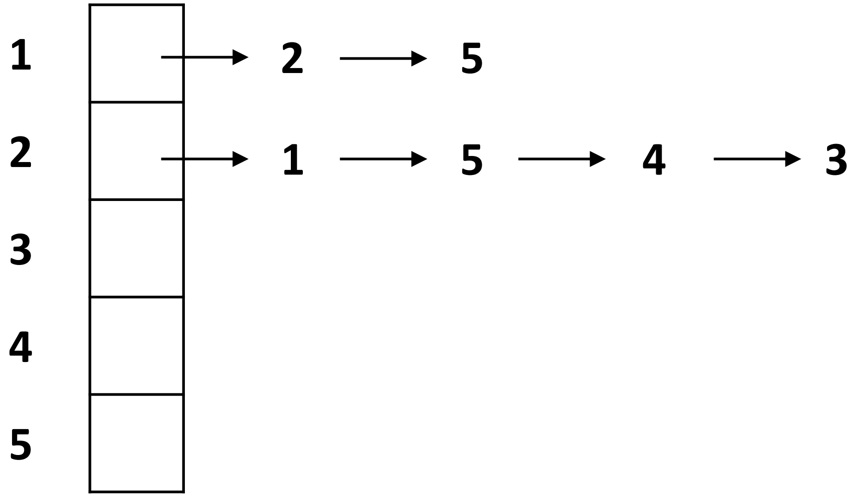


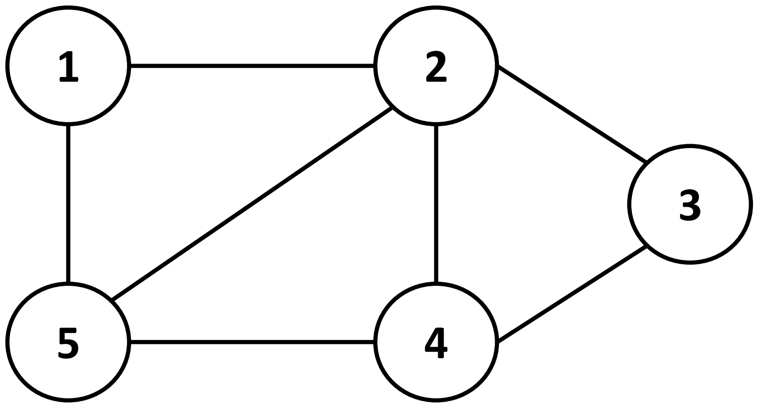
Adjacency list representation



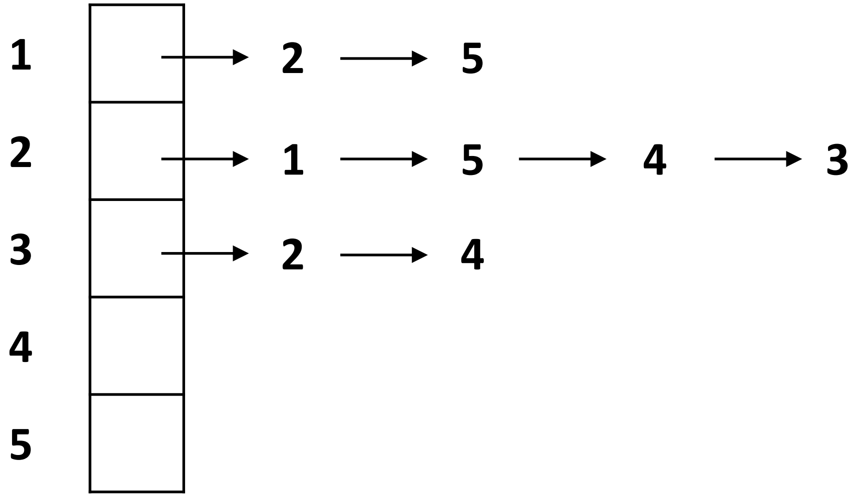


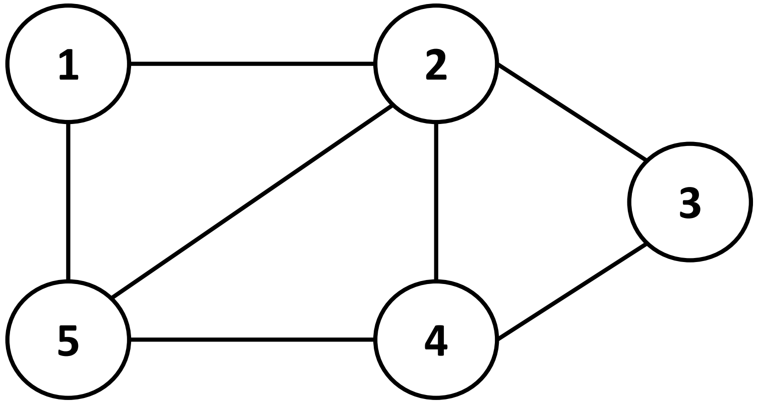
Adjacency list representation





Adjacency list representation





Adjacency list representation

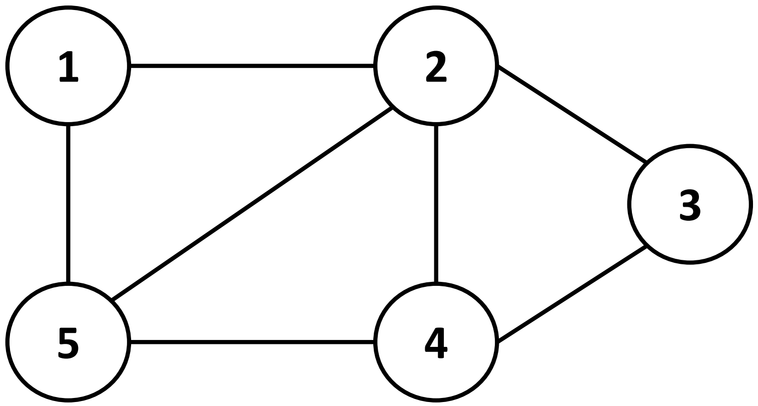
1 → 2 → 5

2 → 1 → 5 → 4 → 3

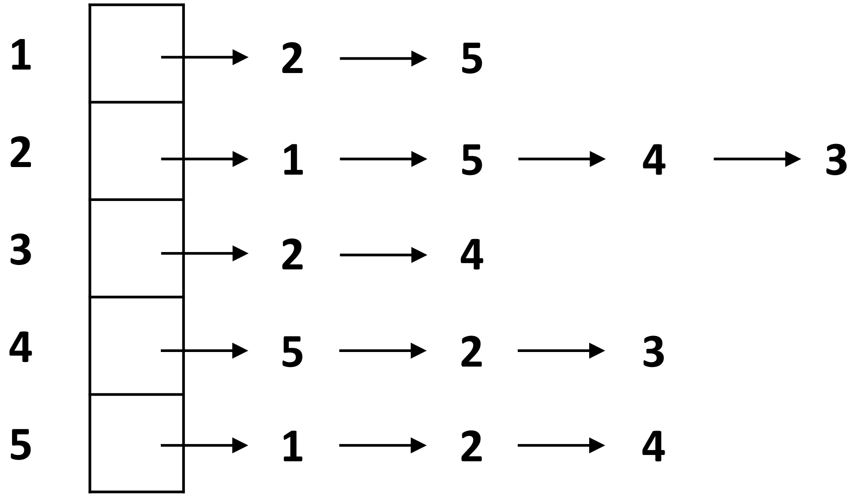
3 → 2 → 4

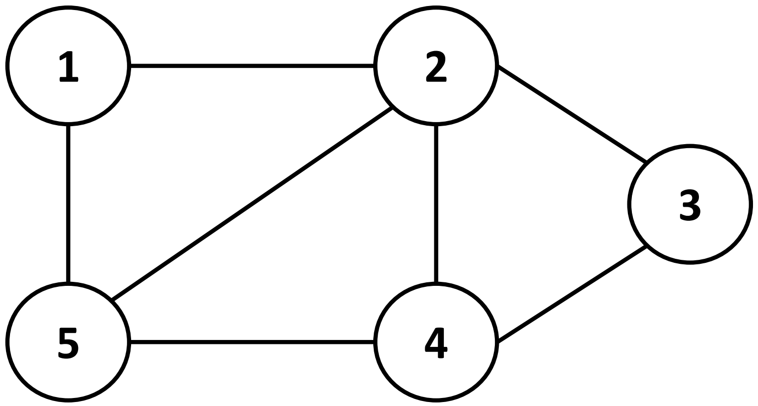
4 → 5 → 2 → 3

5



Adjacency list representation



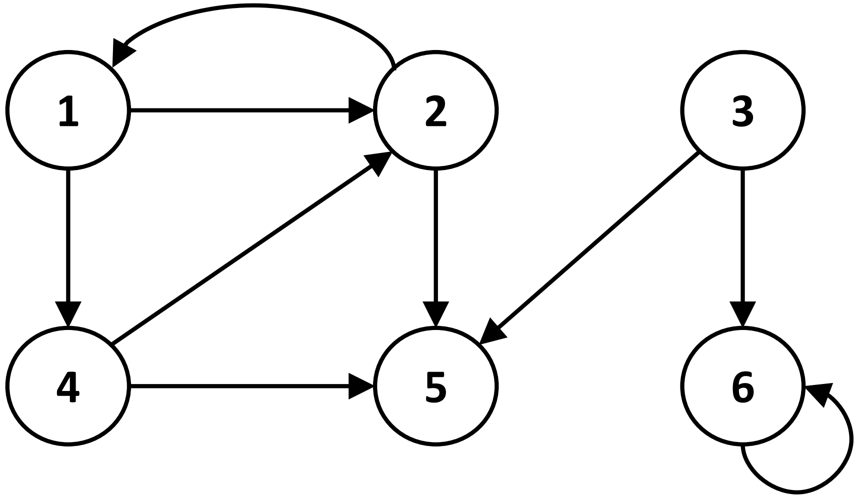


Adjacency list representation

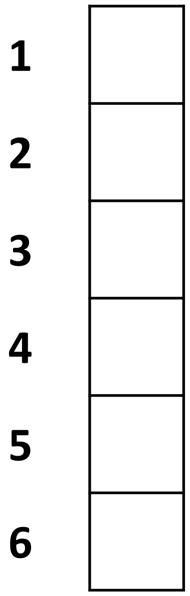
1 → 2 → 5
2 → 1 → 5 → 4 → 3
3 → 2 → 4
4 → 5 → 2 → 3
5 → 1 → 2 → 4

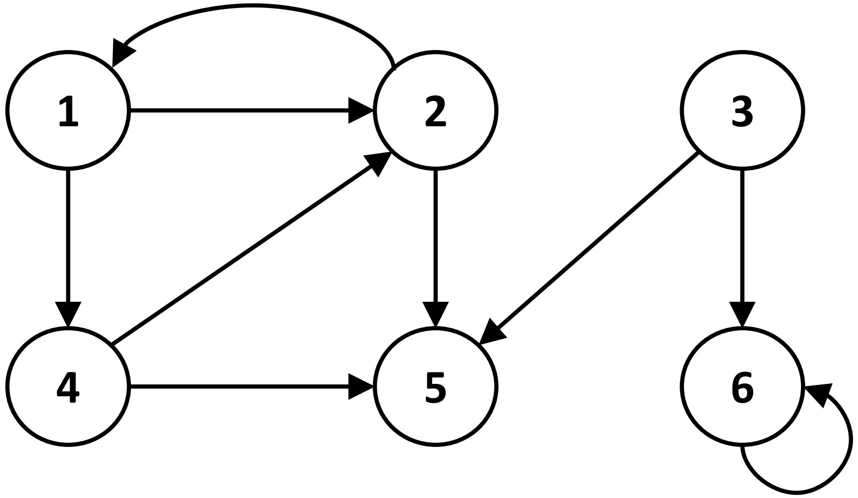
Size: $\Theta\ (n + m)$

Note that $m \leq n^2$

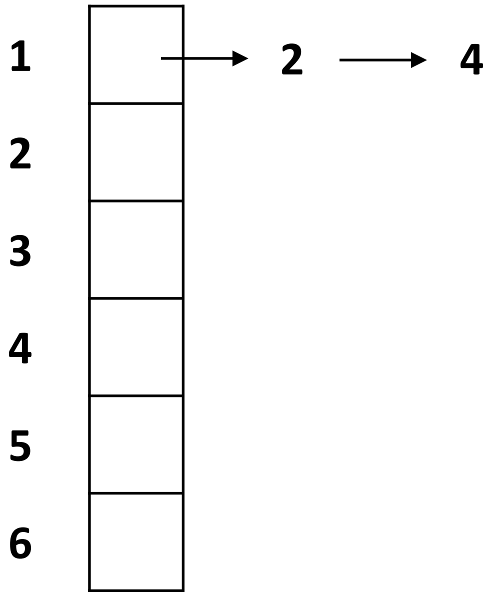


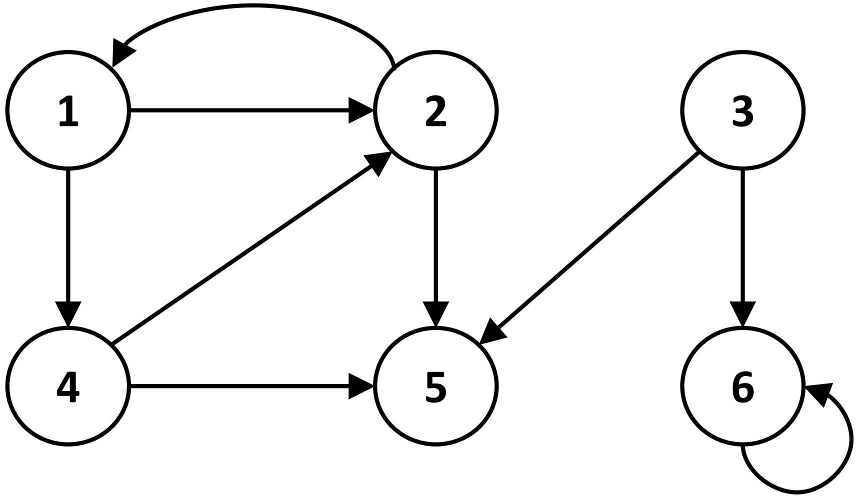
Adjacency list representation



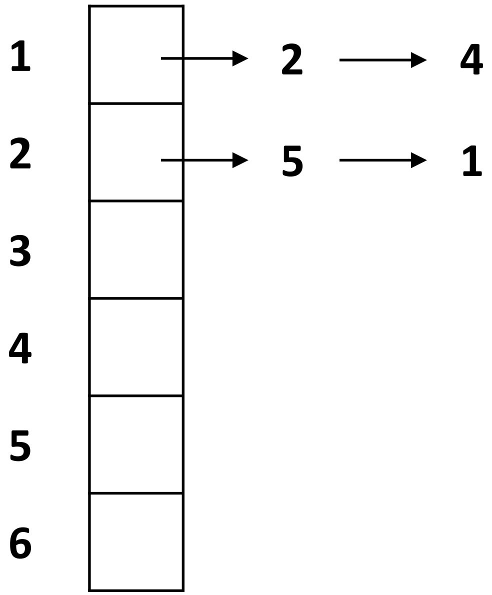


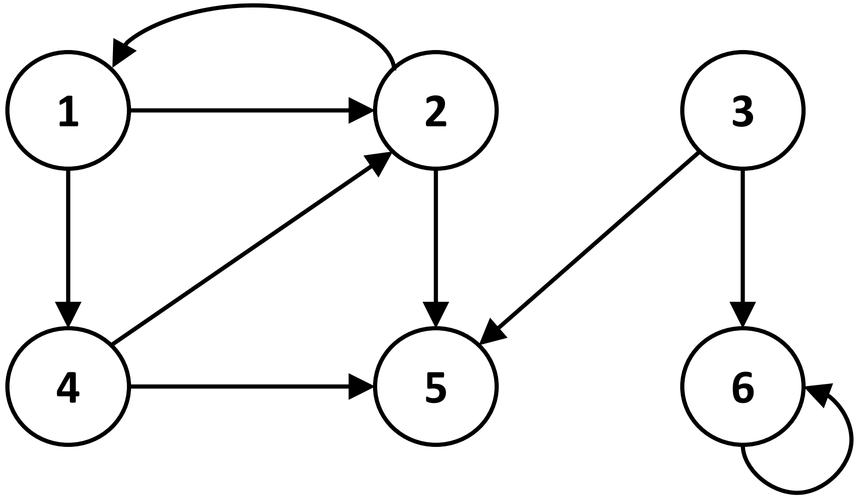
Adjacency list representation





Adjacency list representation





Adjacency list representation

1 → 2 → 4

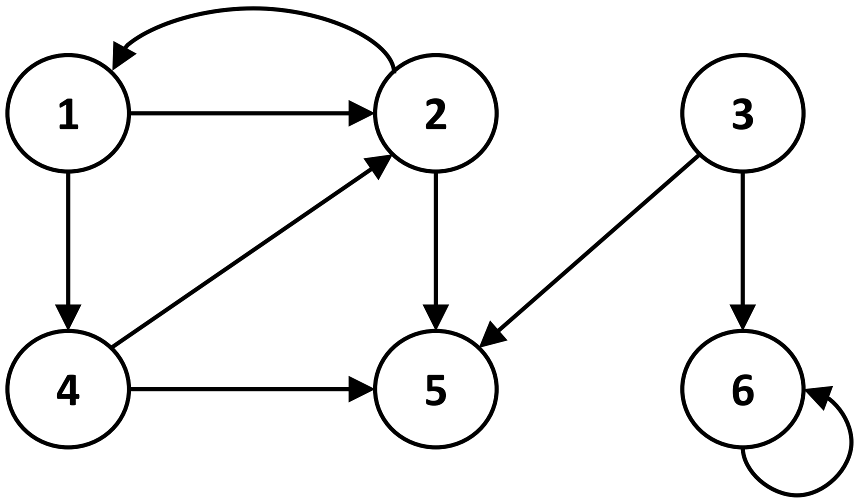
2 → 5 → 1

3 → 5 → 6

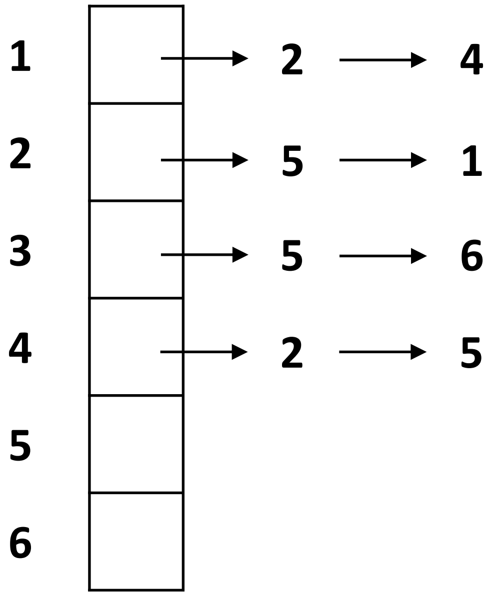
4

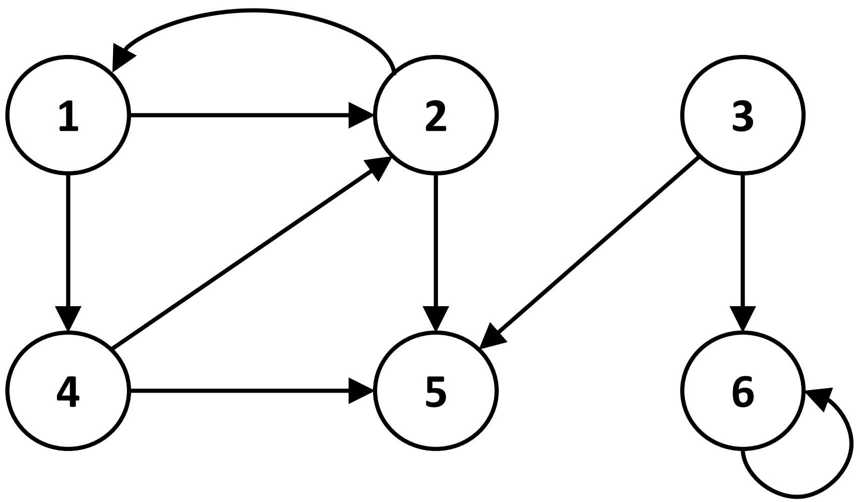
5

6

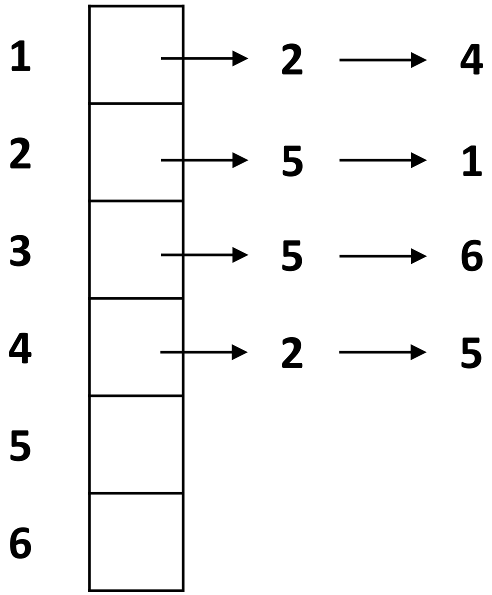


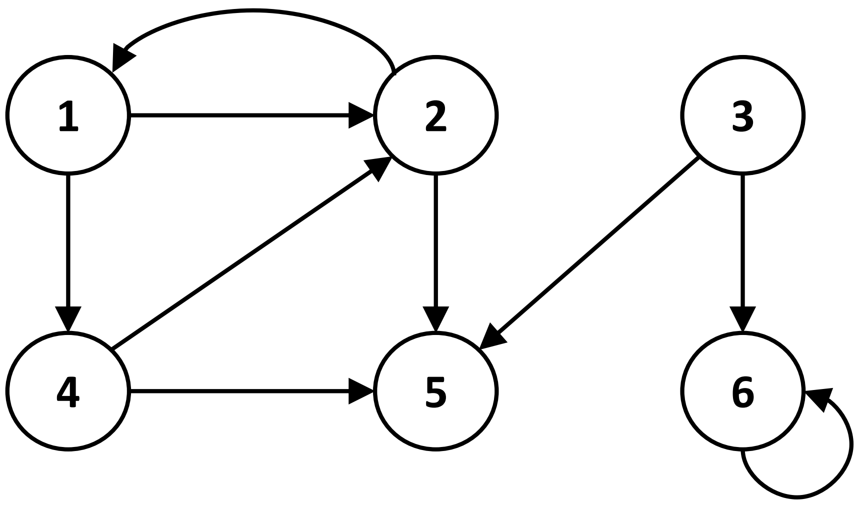
Adjacency list representation





Adjacency list representation





Adjacency list representation

1 → 2 → 4

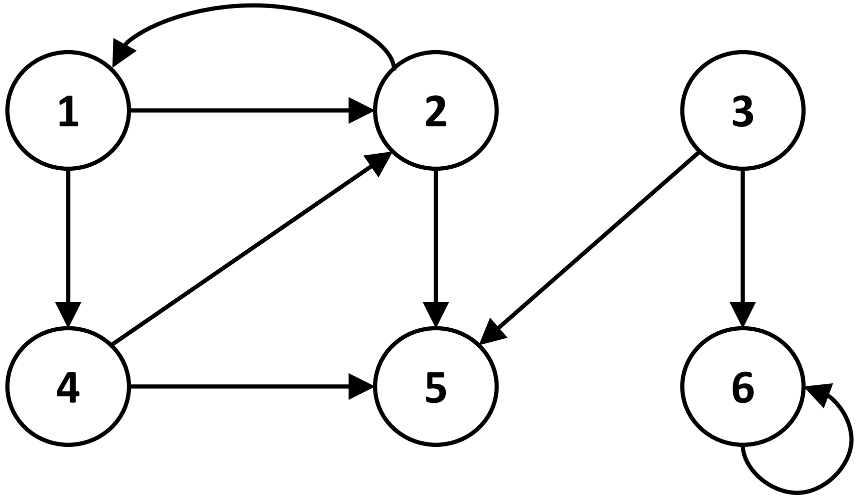
2 → 5 → 1

3 → 5 → 6

4 → 2 → 5

5

6 → 6



Adjacency list representation

1 → 2 → 4

2 → 5 → 1

3 → 5 → 6

4 → 2 → 5

5

6 → 6

Size: $\Theta\ (n + m)$

Representing graphs

Adjacency List of $G = (V, E)$:

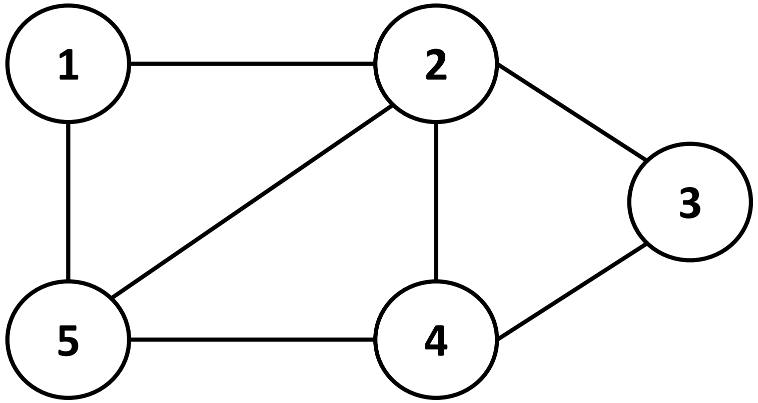
Array **Adj**[1…n] of $n = |V|$ lists, one for each $u \in V$,

Adj[u] is the list of all nodes v such that $(u,v) \in E$

Adjacency Matrix of $G = (V, E)$:

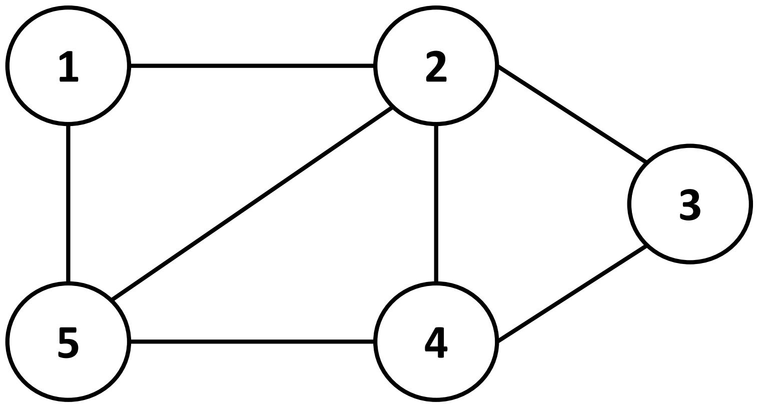
n x n matrix A such that for $1 \leq i,j \leq n$,

$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$



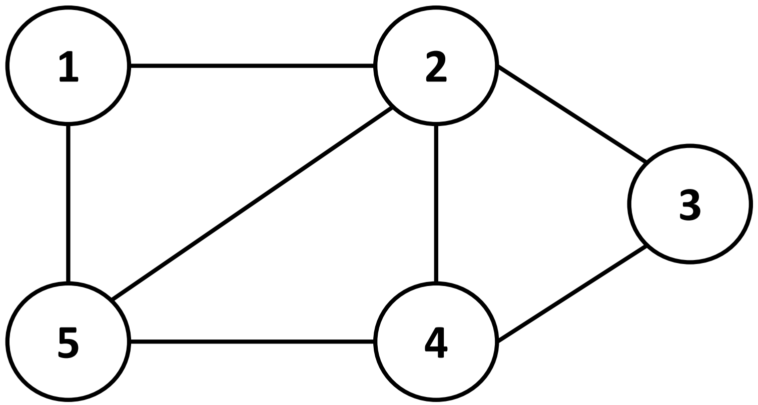
Adjacency matrix representation

$$
\begin{array}{c c}
 & \begin{array}{ccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\
\begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{array} & \left(\begin{array}{ccccc} \; & \; & \; & \; & \; \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{array}\right)
\end{array}
$$



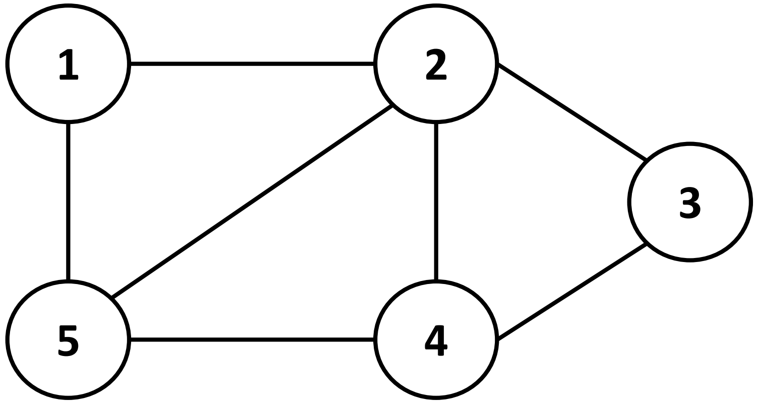
Adjacency matrix representation

$$\begin{array}{cc} & \begin{matrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{matrix} \\ \begin{matrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{pmatrix} \end{array}$$



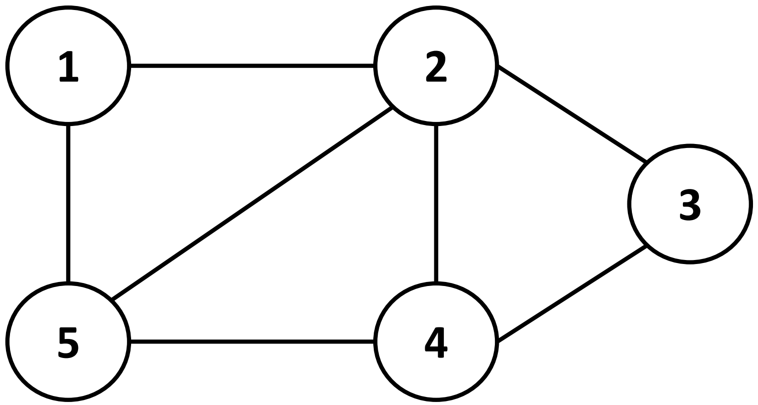
Adjacency matrix representation

$$\begin{array}{c c}
 & \begin{matrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{matrix} \\
\begin{matrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ & & & & \\ & & & & \\ & & & & \end{pmatrix}
\end{array}$$



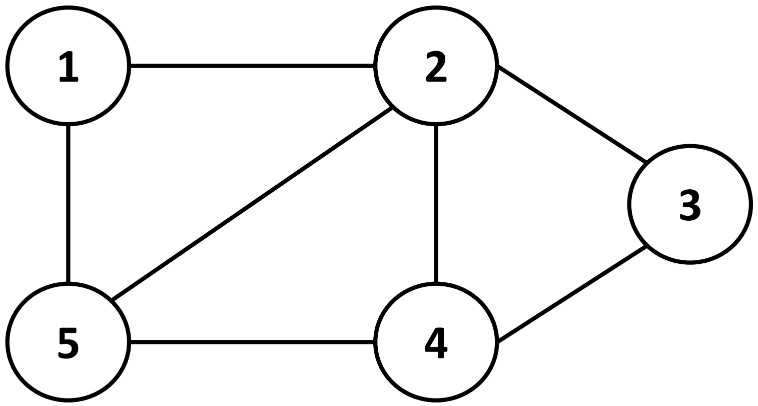
Adjacency matrix representation

$$\begin{array}{c c}
 & \begin{array}{ccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\
\begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ & & & & \\ & & & & \end{pmatrix}
\end{array}$$



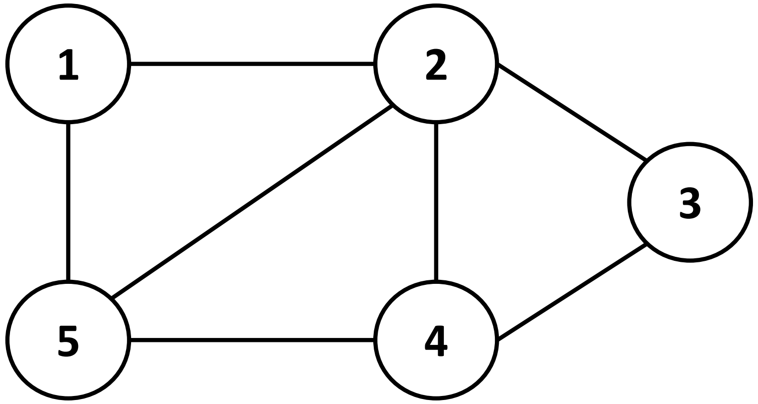
Adjacency matrix representation

$$\begin{array}{cc} & \begin{array}{ccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\ \begin{array}{c} \mathbf{1}\\ \mathbf{2}\\ \mathbf{3}\\ \mathbf{4}\\ \mathbf{5} \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ & & & & \end{pmatrix} \end{array}$$



Adjacency matrix representation

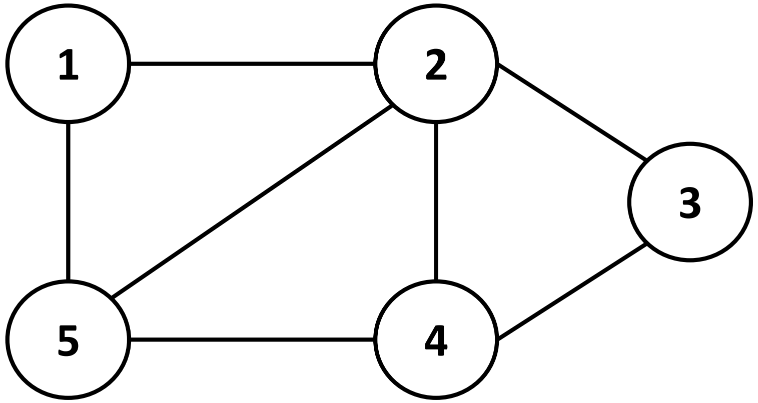
$$\begin{array}{c c}
 & \begin{array}{ccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\
\begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}
\end{array}$$



Adjacency matrix representation

$$
\begin{array}{c c}
 & \begin{array}{ccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\
\begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{array} &
\begin{pmatrix}
0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0
\end{pmatrix}
\end{array}
$$

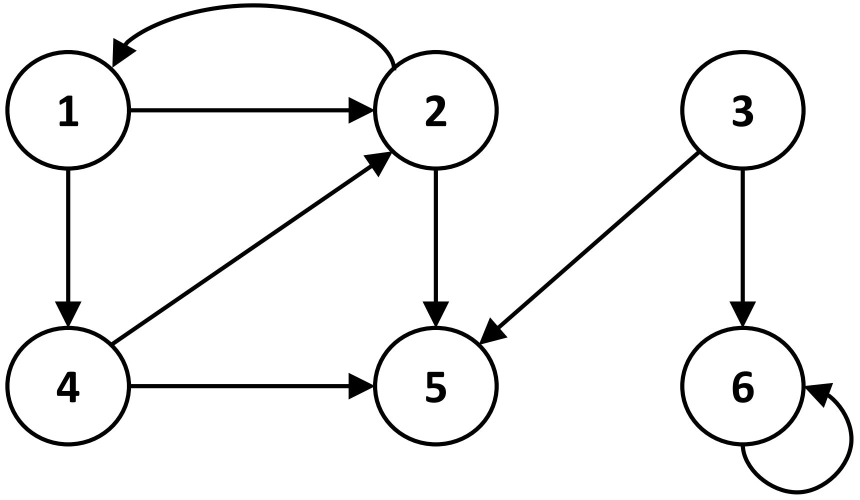
Adjacency matrix of an undirected graph is symmetric about its diagonal



Adjacency matrix representation

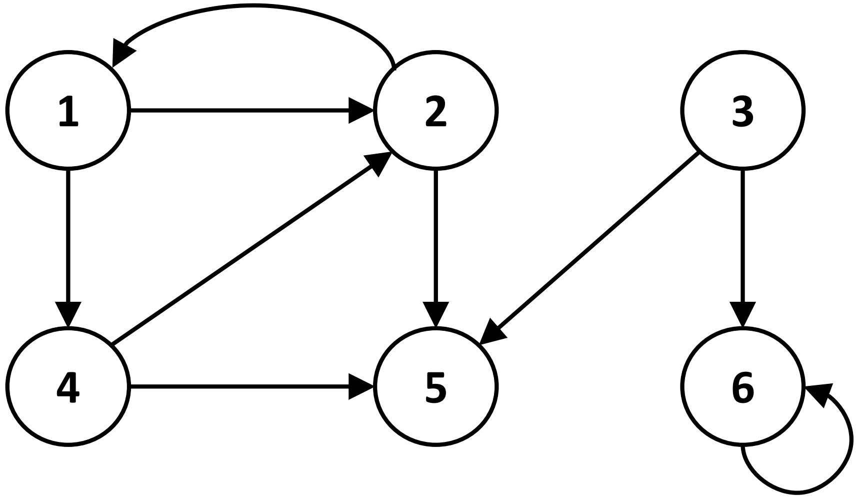
$$
\begin{array}{c|c}
 & \begin{matrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{matrix} \\
\begin{matrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}
\end{array}
$$

Size: $\Theta\ (n^2)$



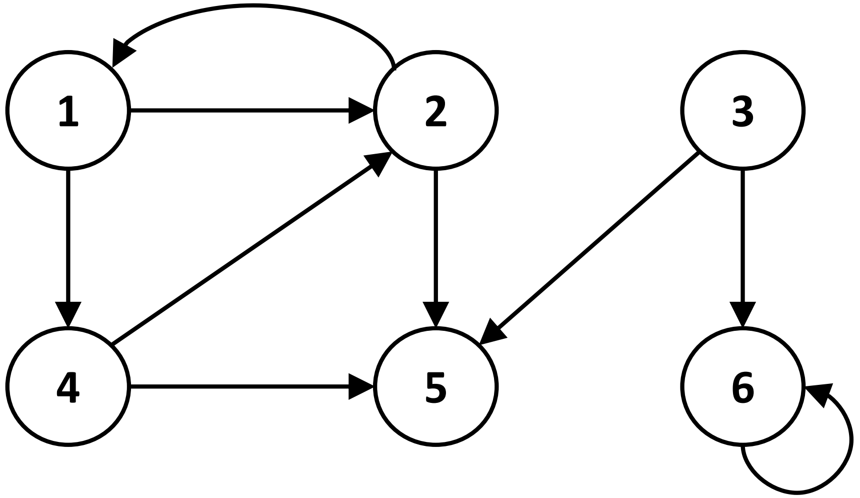
Adjacency matrix representation

$$
\begin{array}{c c}
 & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \left(\begin{array}{cccccc} \; & \; & \; & \; & \; & \; \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{array}\right)
\end{array}
$$



Adjacency matrix representation

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

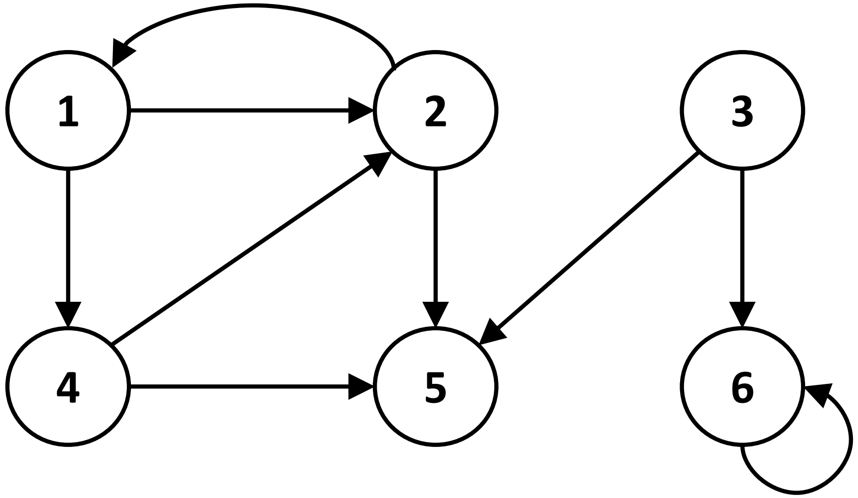


Adjacency matrix representation

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

$A[2, 4] \neq A[4, 2]$

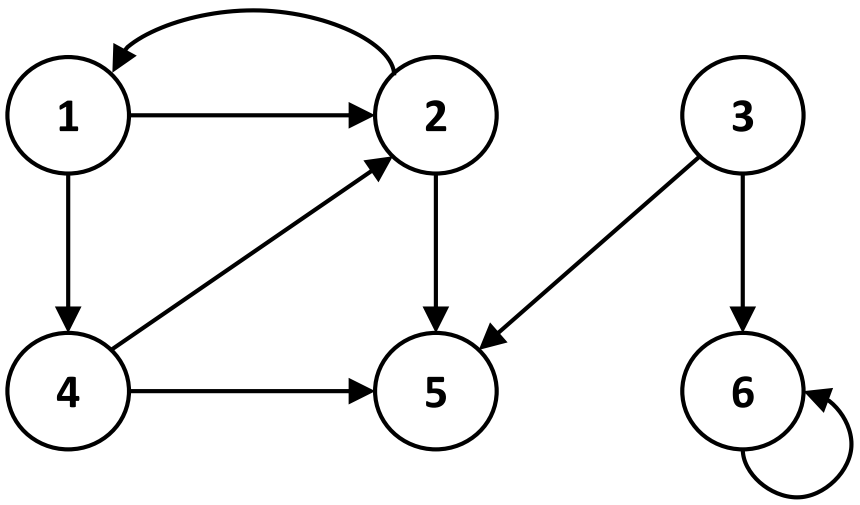
Adjacency matrix of a directed graph is not necessarily symmetric about its diagonal



Adjacency matrix representation

$$
\begin{array}{c|c}
 & \begin{matrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \end{matrix} \\
\begin{matrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}
\end{array}
$$

Size: $\Theta(n^2)$



Adjacency list representation

1 → **2** → **4**

2 → **5** → **1**

3 → **5** → **6**

4 → **2** → **5**

5

6 → **6**

Adjacency matrix representation

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

When to use one representation over the other?

When to use one representation over the other?

- For sparse graphs ($m << n^2$), Adjacency List is more space-efficient.

When to use one representation over the other?

- For sparse graphs ($m << n^2$), Adjacency List is more space-efficient.

- Finding whether a certain $(u, v) \in E$:
  - Is $O(1)$ with Adjacency Matrix
  - Is $O(n)$ with Adjacency List

Graph Search

Systematic exploration of graph:

Graph Search

Systematic exploration of graph:

- Start at some node and explore it

Graph Search

Systematic exploration of graph:

- Start at some node and explore it

  i.e., follow its edges to discover new nodes

Graph Search

Systematic exploration of graph:

- Start at some node and explore it

  i.e., follow its edges to discover new nodes

- Explore newly discovered nodes, i.e., follow edges from those nodes etc…

Graph Search

Systematic exploration of graph:

- Start at some node and explore it

    i.e., follow its edges to discover new nodes

- Explore newly discovered nodes, i.e., follow edges from those nodes etc…

Graph searches reveal structural properties of G:

Graph Search

Systematic exploration of graph:

- Start at some node and explore it

  i.e., follow its edges to discover new nodes

- Explore newly discovered nodes, i.e., follow edges from those nodes etc…

Graph searches reveal structural properties of G:

Is G connected?

Graph Search

Systematic exploration of graph:

- Start at some node and explore it

  i.e., follow its edges to discover new nodes
- Explore newly discovered nodes, i.e., follow edges from those nodes etc…

Graph searches reveal structural properties of G:

Is G connected?

Does G have a cycle?

Graph Search

Systematic exploration of graph:

- Start at some node and explore it

    i.e., follow its edges to discover new nodes

- Explore newly discovered nodes, i.e., follow edges from those nodes etc…

Graph searches reveal structural properties of G:

Is G connected?

Does G have a cycle?

Shortest path info,

etc…

Graph Search

Two basic types of graph searches:

Graph Search

Two basic types of graph searches:

- Breadth First Search (BFS)
- Depth First Search (DFS)

Breadth First Search

Breadth First Search

BFS started at a node s

Breadth First Search

BFS started at a node s

(1) Explore s (i.e. follow all the edges out of s to discover new nodes)

Breadth First Search

BFS started at a node s

(1) Explore s (i.e. follow all the edges out of s to discover new nodes)

(2) Explore discovered nodes in the **order of their discovery**

Breadth First Search

BFS started at a node s

(1) Explore s (i.e. follow all the edges out of s to discover new nodes)

(2) Explore discovered nodes in the **order of their discovery**

[“First Discovered-First Explored” policy]

Breadth First Search

BFS started at a node s

(1) Explore s (i.e. follow all the edges out of s to discover new nodes)

(2) Explore discovered nodes in the **order of their discovery**

[“First Discovered-First Explored” policy]

To do so, put them in a Queue and explore them in FIFO order

Breadth First Search

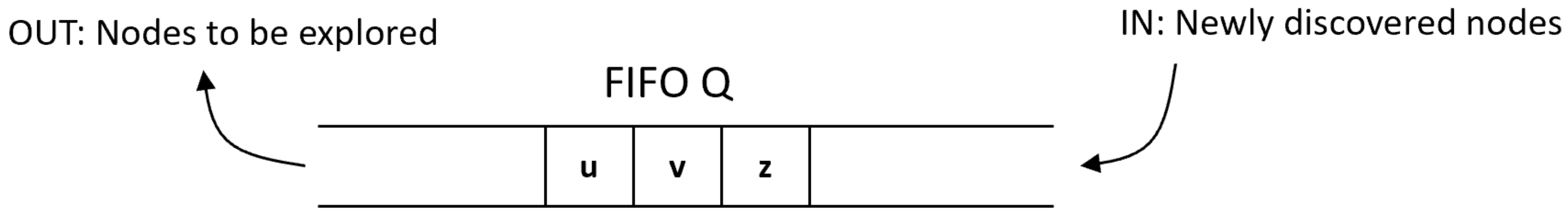
BFS started at a node s

(1) Explore s (i.e. follow all the edges out of s to discover new nodes)

(2) Explore discovered nodes in the **order of their discovery**

[“First Discovered-First Explored” policy]

To do so, put them in a Queue and explore them in FIFO order



Breadth First Search

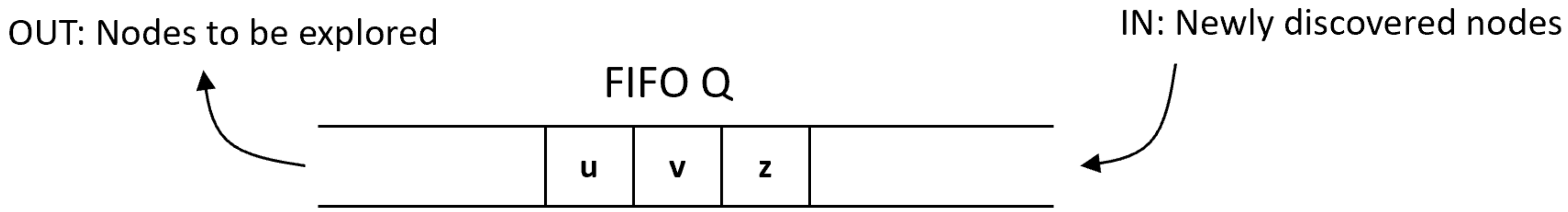
BFS started at a node s

(1) Explore s (i.e. follow all the edges out of s to discover new nodes)

(2) Explore discovered nodes in the **order of their discovery**

[“First Discovered-First Explored” policy]

To do so, put them in a Queue and explore them in FIFO order



Do (2) until all discovered nodes are explored

BFS algorithm maintains the following information for each node v:

BFS algorithm maintains the following information for each node v:

1. color[v] =

BFS algorithm maintains the following information for each node v:

1. color[v] = { white : v is undiscovered

BFS algorithm maintains the following information for each node v:

$$1.\ \text{color}[v] = \begin{cases} \text{white} & : \quad v \text{ is undiscovered} \\ \text{grey} & : \quad v \text{ was discovered but not yet explored} \\ & \end{cases}$$

BFS algorithm maintains the following information for each node v:

1. color[v] =

white	:	v is undiscovered
grey	:	v was discovered but not yet explored
black	:	v was discovered and explored

BFS algorithm maintains the following information for each node v:

2. p[v]

BFS algorithm maintains the following information for each node v:

2. p[v] = u : node v was discovered while exploring u

BFS algorithm maintains the following information for each node v:

2. p[v] = u : node v was discovered while exploring u
: i.e. u discovered v

BFS algorithm maintains the following information for each node v:

2. p[v] = u : node v was discovered while exploring u
: i.e. u discovered v

3. d[v]

BFS algorithm maintains the following information for each node v:

2. p[v] = u : node v was discovered while exploring u
: i.e. u discovered v

3. d[v] : Length of discovery path from s

BFS algorithm maintains the following information for each node v:

2. p[v] = u : node v was discovered while exploring u
: i.e. u discovered v

3. d[v] : Length of discovery path from s

$$s \rightarrow u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u \rightarrow v$$

BFS algorithm maintains the following information for each node v:

2. p[v] = u : node v was discovered while exploring u
: i.e. u discovered v

3. d[v] : Length of discovery path from s

$$s \underbrace{\rightarrow u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u \rightarrow}_{d[v]} v$$

BFS algorithm maintains the following information for each node v:

2. p[v] = u : node v was discovered while exploring u
: i.e. u discovered v

3. d[v] : Length of discovery path from s

$$s \underbrace{\overbrace{\to u_1 \to u_2 \to \ldots \to}^{d[u]} u \to}_{d[v]} v$$

BFS algorithm maintains the following information for each node v:

2. p[v] = u : node v was discovered while exploring u
: i.e. u discovered v

3. d[v] : Length of discovery path from s

$$s \underbrace{\overbrace{\rightarrow u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u}^{d[u]} \rightarrow v}_{d[v]}$$

Clearly, d[v] = d[u] + 1

```
BFS(G, s)                                              /*  G = (V, E) and s ∈ V  */
    color[s] ← grey ;  d[s] ← 0 ;  p[s] ← NIL
    For each v ∈ V – {s}   do
            color[v] ← white
            d[v] ← ∞
            p[v] ← NIL

    Q ← empty ; ENQ(Q, s)                 /*  Q: nodes that are discovered but not yet explored  */
    While Q is not empty  do
            u ← DEQ(Q)                                  /*  Explore u  */
            For each (u, v) ∈ E  do                     /*  Explore edge (u,v)  */
                If color[v] = white  then do            /*  If v is first discovered  */
                    color[v] ← grey
                    d[v] ← d[u] + 1
                    p[v] ← u
                    ENQ(Q, v)
                End If
            End For
            color[u] ← black                            /*  Done exploring u  */
    End While
End BFS
```