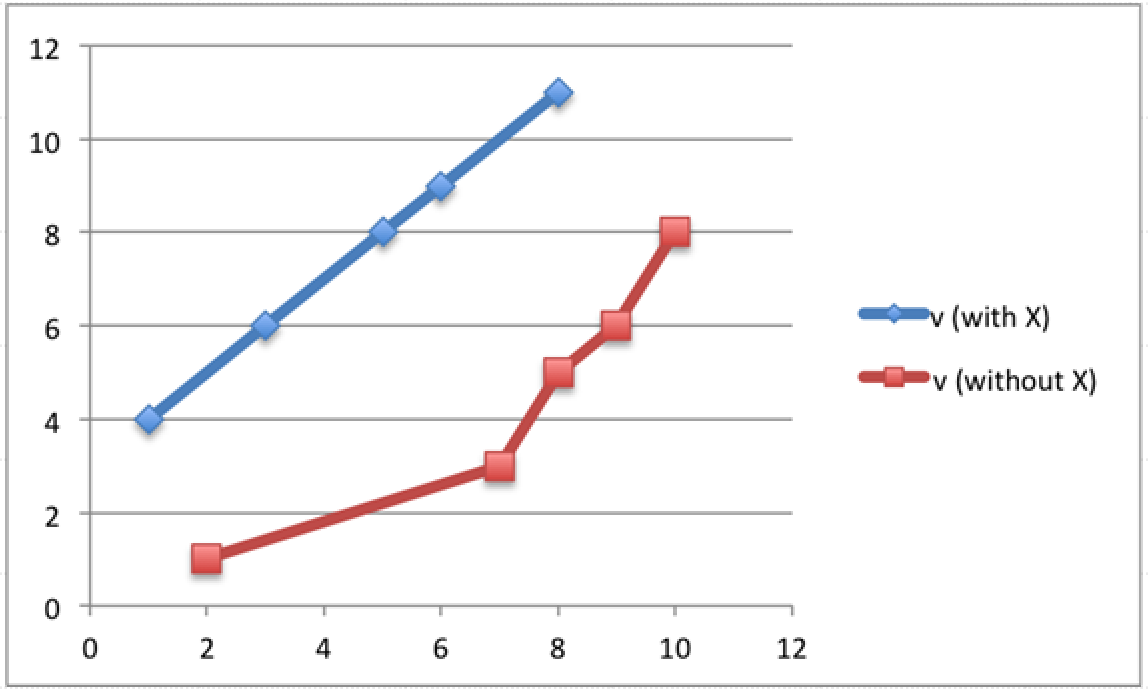
Graph Algorithms

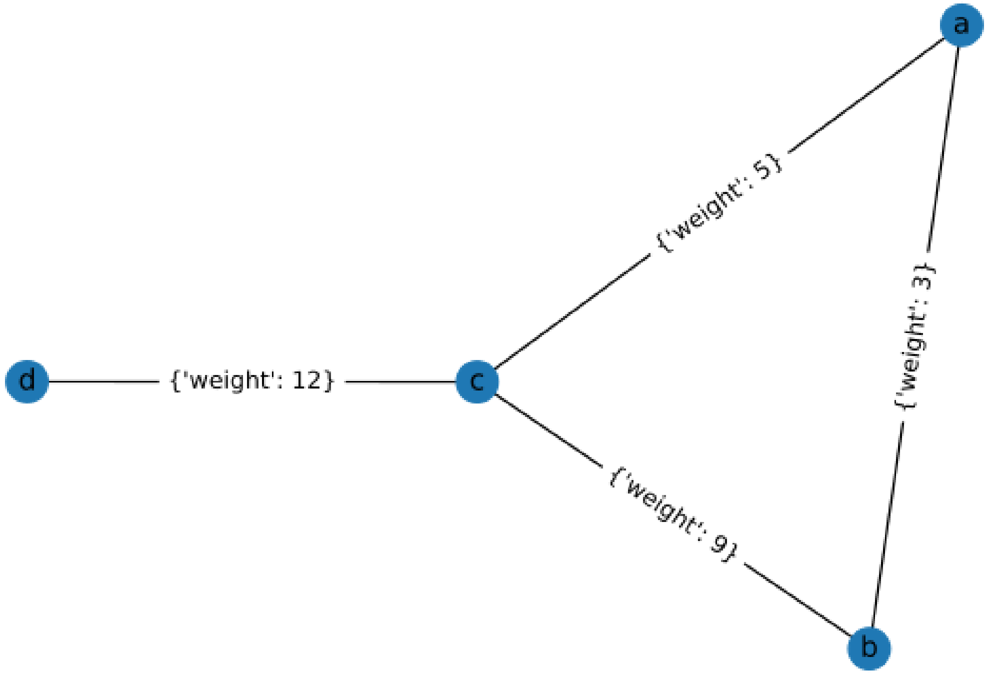
Ahmedur Rahman Shovon

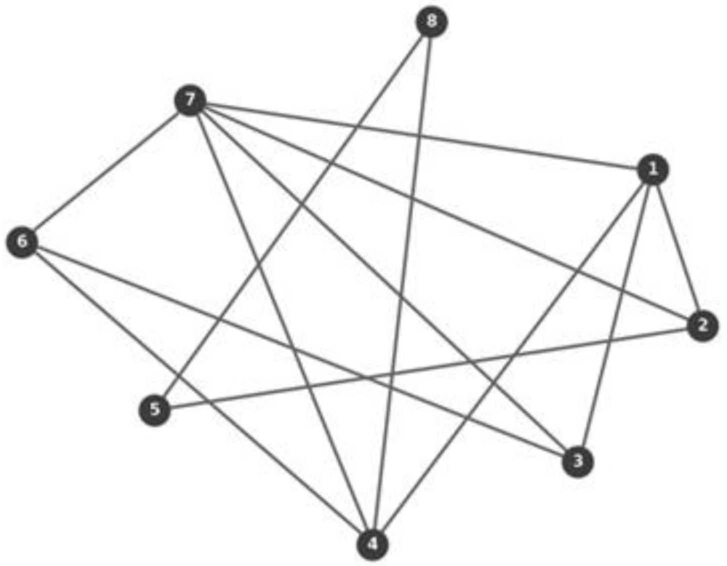
PhD (Student), CS

UAB



Graph in Mathematics or in normal use





What about graph in Algorithm?

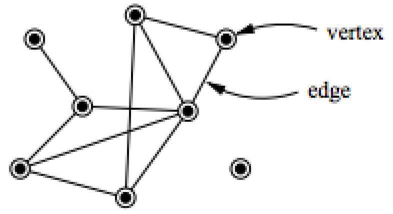
Components of a graph

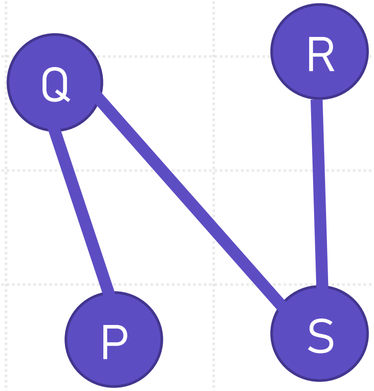
- Vertices / Nodes:
  - Represented as set of vertices

    V = {P, Q, R, S}
- Edges:
  - Represented as set of connection between vertices

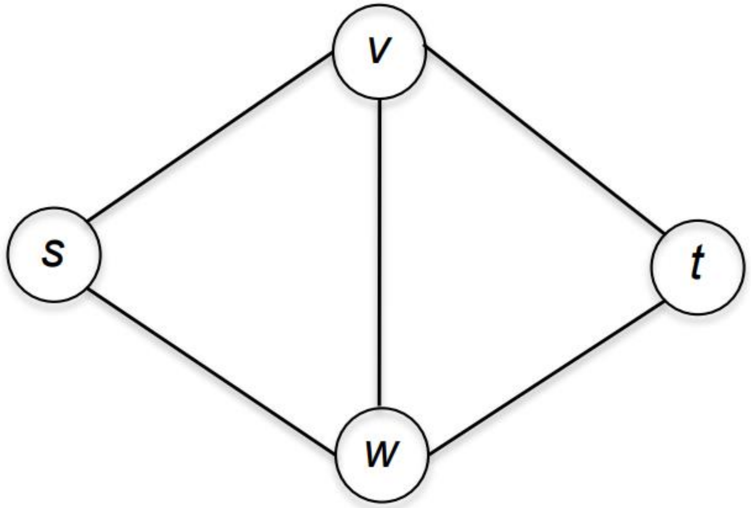
    E = {{P, Q}, {Q, S}, {SR}}
- Graph:
  - Represented as the combination of vertices and edges

    G = (V, E)

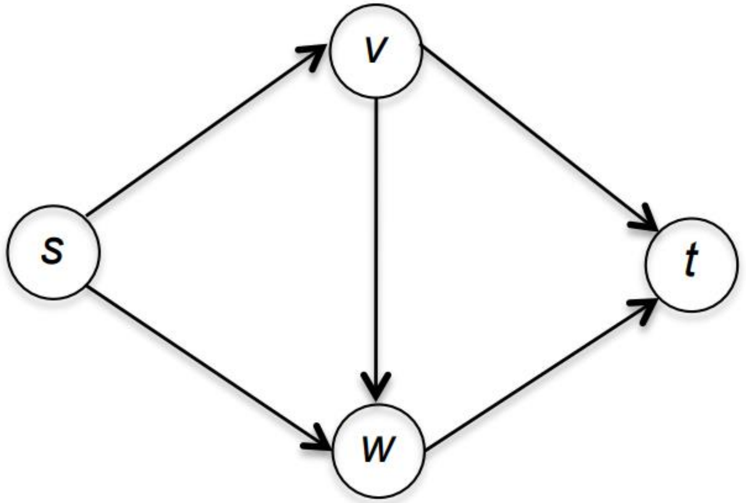




Types of Graphs



(a) An undirected graph



(b) A directed graph

V = {s, v, t, w}
E = {{s, v}, {v, t}, {t, w}, {v, w}, {w, s}}

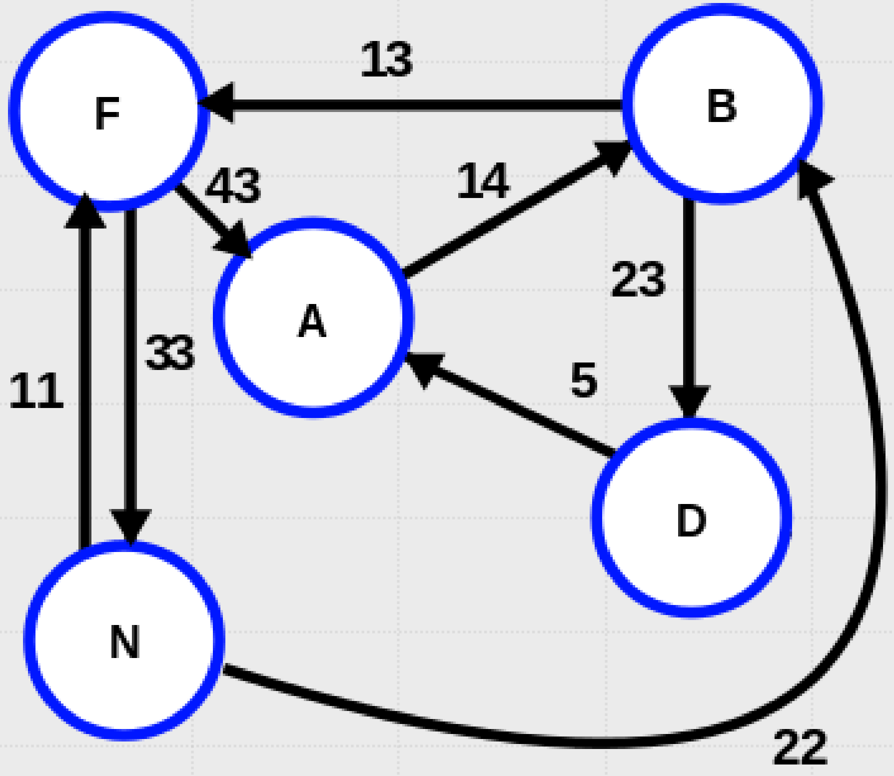
E contains Unsorted order of vertices

V = {s, v, t, w}
E = {(s, v), (s, w), (v, w), (v, t), (w, t)}

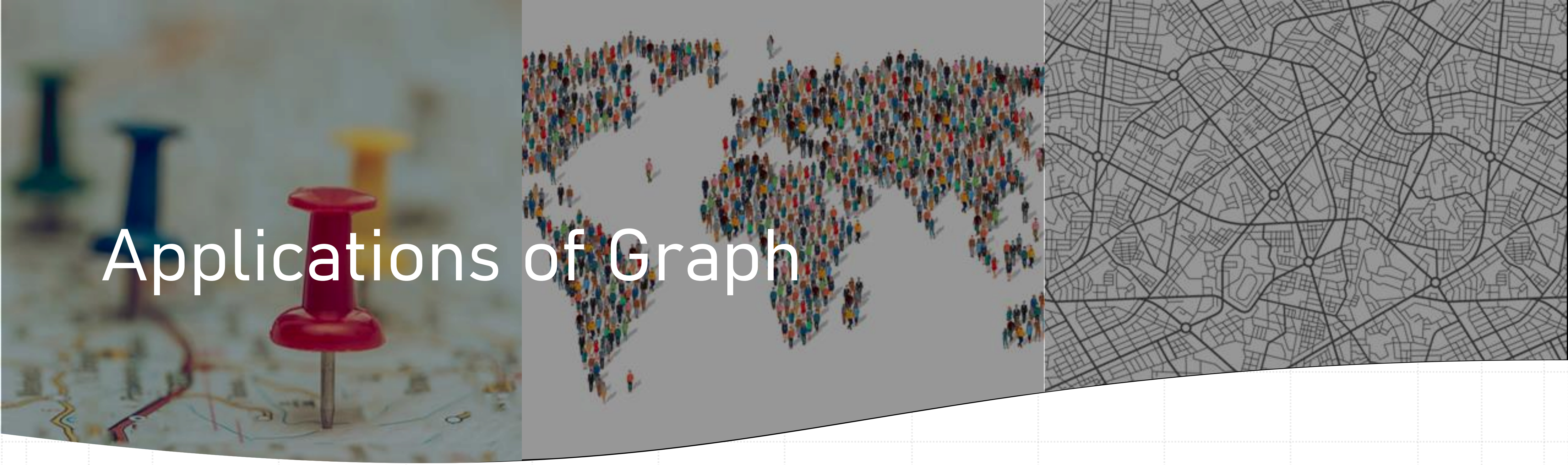
E contains Sorted order of vertices

Weighted Graph

- Directed and undirected graph can have weight between vertices
- The weight is also known as distance



Applications of Graph

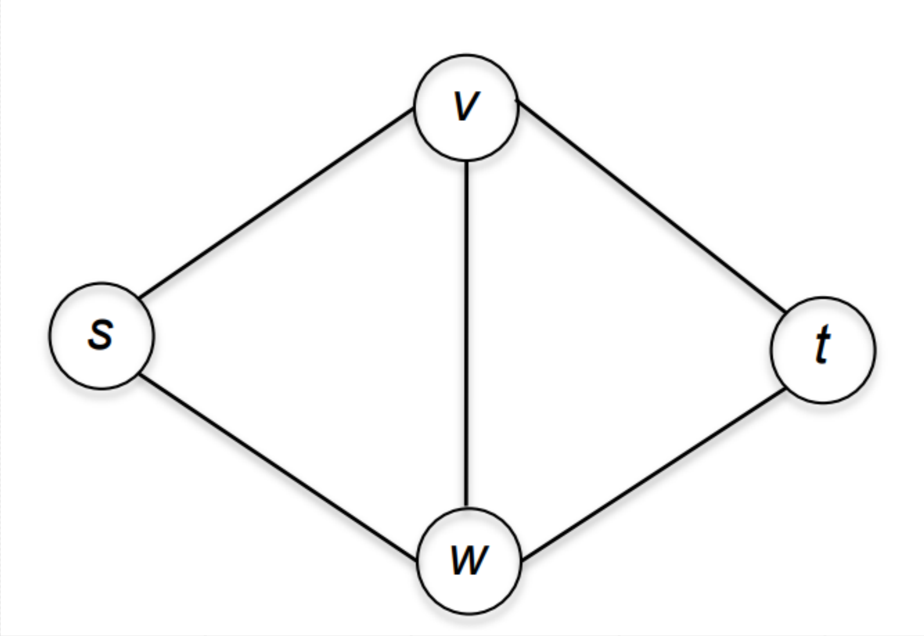


- Road networks, navigation
- The World Wide Web
- Social Networks
- Puzzle solving

Notation for Graphs

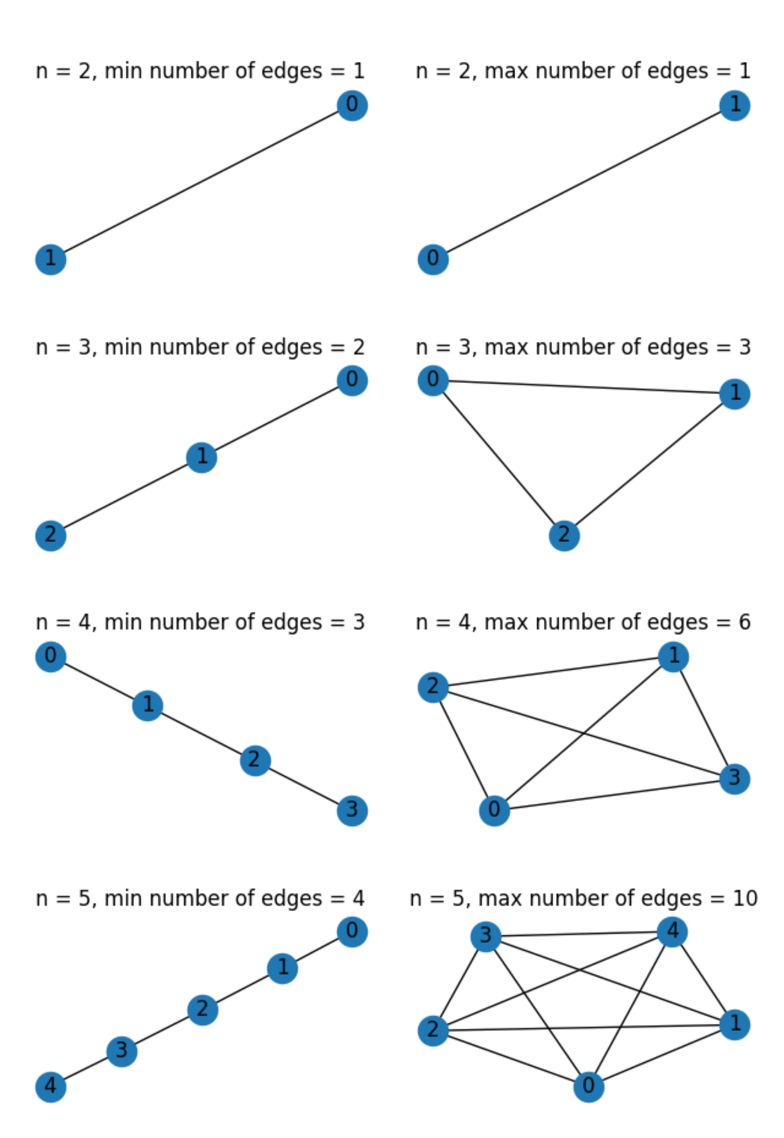
- For a graph G = (V, E) with vertex set V and edge set E:
  - n = |V| denotes the number of vertices.
  - m = |E| denotes the number of edges.

- In the figure,
  - n = number of vertices = 4
  - m = number of edges = 5



Minimum & Maximum number of Edges

- Assume the following properties:
  1. The graph is undirected
  2. The graph is connected
  3. There are no parallel edges between vertices



Quick Quiz

- For an undirected connected graph with n vertices and no parallel edge, what are the minimum and maximum numbers of edges?
  a) Minimum number of edges = $n - 1$, maximum number of edges = $(n * (n - 1)) / 2$
  b) Minimum number of edges = $n - 1$, maximum number of edges = n^2
  c) Minimum number of edges = n, maximum number of edges = 2^n
  d) Minimum number of edges = $n - 1$, maximum number of edges = n^n

Quiz Solution

- For an undirected connected graph with *n* vertices and no parallel edge, what are the minimum and maximum numbers of edges?
  a) Minimum number of edges = $n - 1$, maximum number of edges = $(n * (n - 1)) / 2$

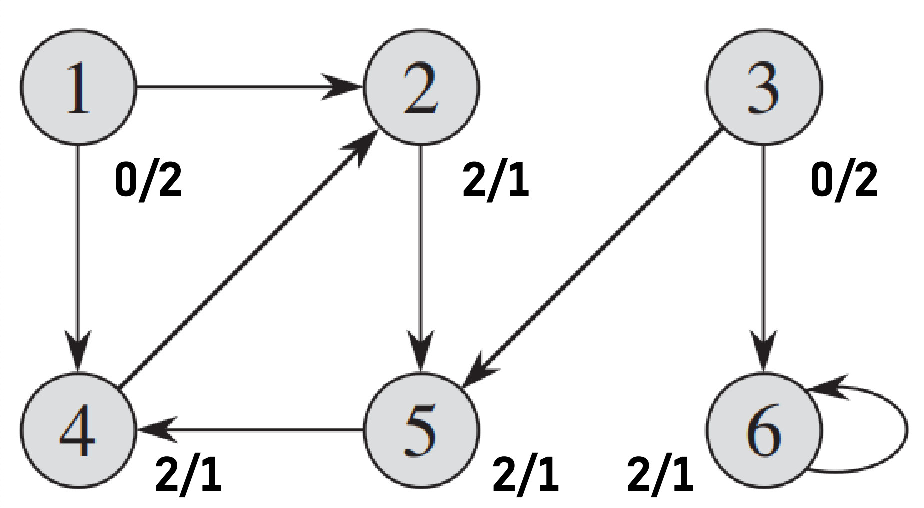
- A connected graph with the minimum number of edges (n-1 edges) is called a **Tree**.
- A connected graph with maximum number of edges ((n * (n - 1)) / 2 edges) is known as **Complete Graph**

Sparse and Dense Graphs

- A graph is **Sparse** if the number of edges is relatively close to linear in the number of vertices.
  - $| E | \cong n$
- A graph is **Dense** if the number of edges is relatively close to quadratic in the number of vertices.
  - $| E | \cong n^2$
- Graph representation may vary for Sparse and Dense graphs

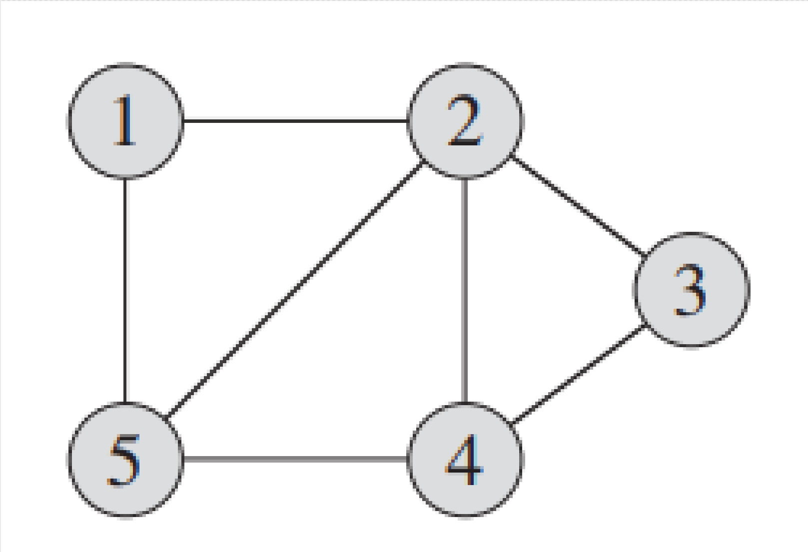
In-degree and out-degree

- In a directed graph,
  - The **in-degree** of a vertex is the total number of incoming edges
  - The **out-degree** of a vertex is the total number of outgoing edges
- In an undirected graph,
  - The **degree** of a vertex is the total number of adjacent nodes
- The sum of degrees of all nodes is the double of number of edges of the graph

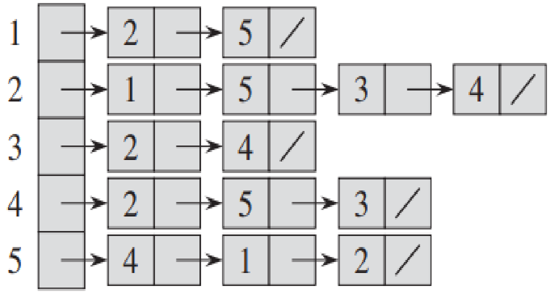


Graph Representation

- Two standard ways to represent a graph G = (V, E) are:
  a) Adjacency-list representation
  b) Adjacency-matrix representation



Graph G = (V, E)



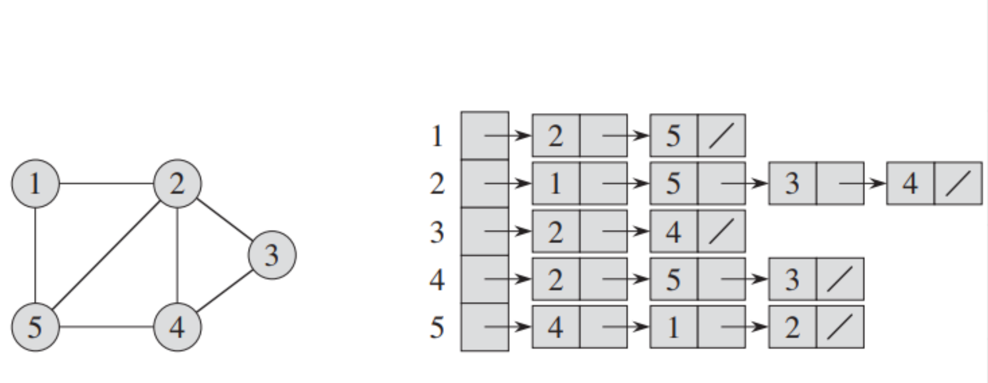
Adjacency-list representation of G

	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

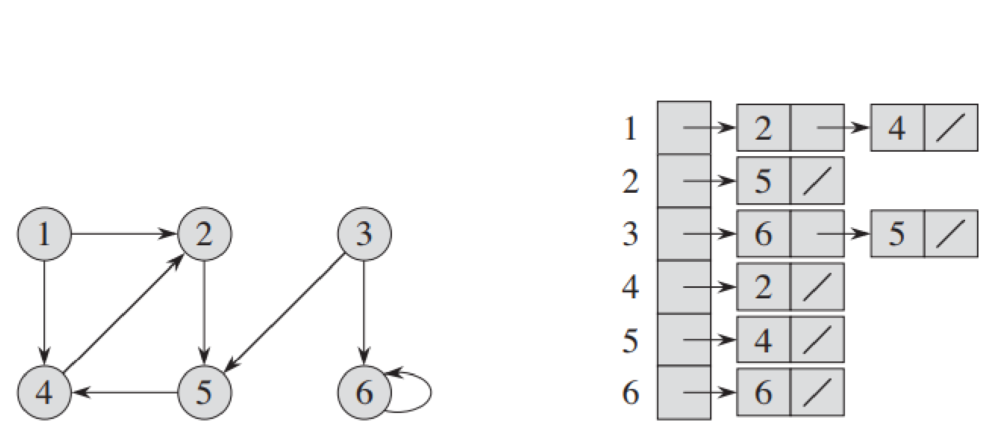
Adjacency-matrix representation of G

Adjacency-list representation

- Adjacency-list representation of graph G = (V, E) consists of:
  a) An array for each vertex, *Adj of |V|*
  b) Each array index consists of a list of vertices which are connected to the vertex, *Adj[u]* contains the list of vertices which are connected to vertex *u*
- Suitable for Sparse graphs



Adjacency-list for undirected graph

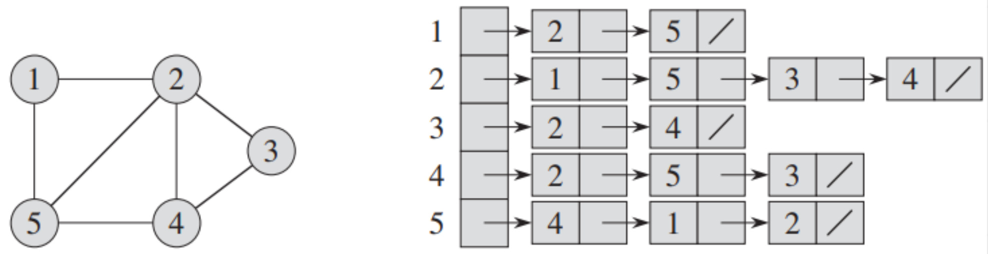


Adjacency-list for directed graph

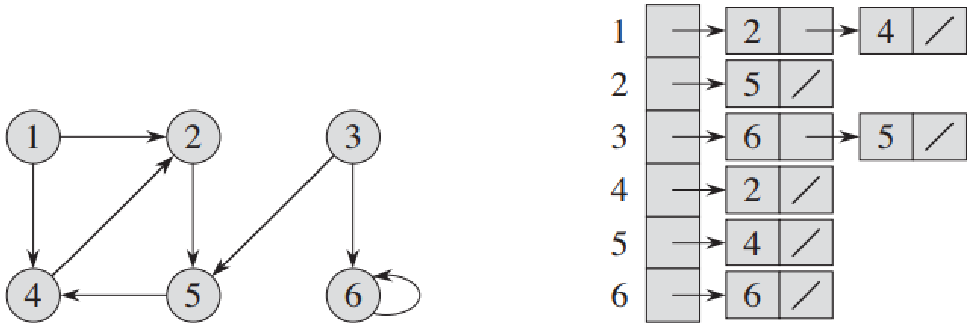
Adjacency-list space requirement

- Given a graph G with *n* vertices and *m* edges.
  What is the space requirement of adjacency-list representation of graph G?
  a) $\Theta(n)$
  b) $\Theta(m)$
  c) $\Theta(n + m)$
  d) $\Theta(n^2)$

Space requirements **$\Theta(n + m)$**



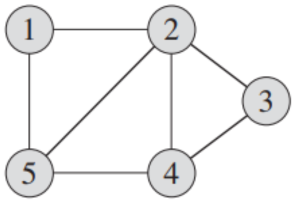
Adjacency-list for undirected graph



Adjacency-list for directed graph

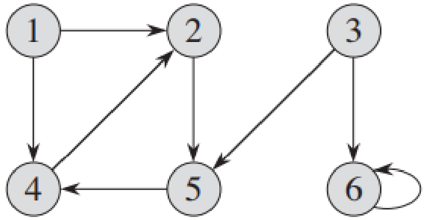
Adjacency-matrix representation

- Adjacency-matrix representation of graph G = (V, E), n = |v| consists of:
  a) A square matrix A of size $n \times n$
  b) Each entry A_{ij} is defined as:
     $A_{ij} = 1$, if edge$(i, j) \in E$
     $A_{ij} = 0$, otherwise
- Suitable for Dense graphs



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{array}\right] \end{array}$$

Adjacency-matrix for undirected graph



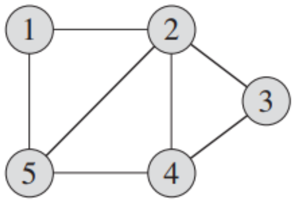
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \left[\begin{array}{cccccc} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

Adjacency-matrix for directed graph

Adjacency-matrix space requirement

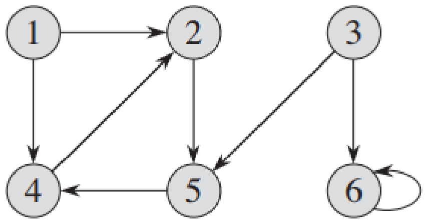
- Given a graph G with n vertices and m edges. What is the space requirement of adjacency-matrix representation of graph G?
  a) $\Theta(n)$
  b) $\Theta(m)$
  c) $\Theta(n + m)$
  d) $\Theta(n^2)$

Space requirements **$\Theta(n^2)$**



	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

Adjacency-matrix for undirected graph

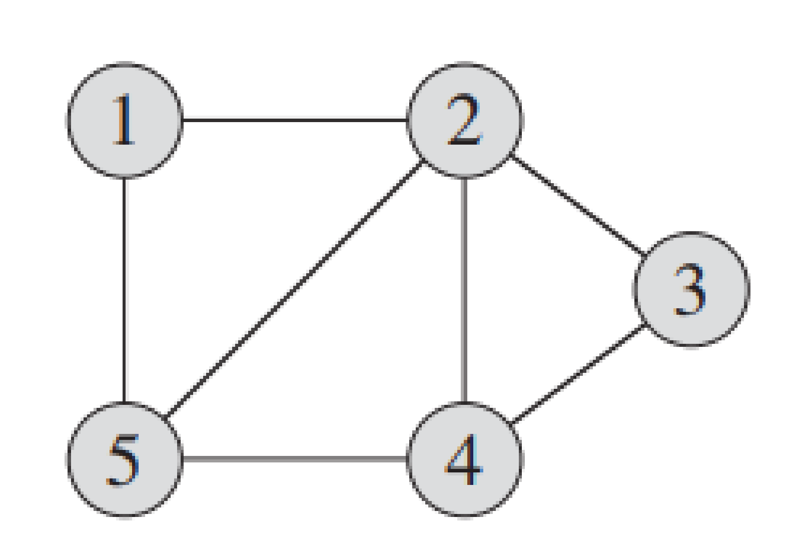


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

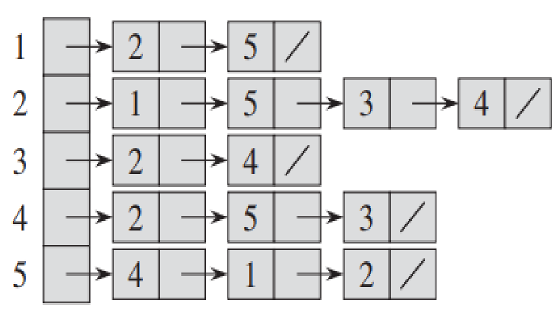
Adjacency-matrix for directed graph

Comparing the representations

- For Sparse graphs (| E | ≅ n) adjacency-list representation takes less space
- For Dense graphs (| E | ≅ n^2) adjacency-matrix representation is faster
- The representation should be selected based on the requirement



Graph G = (V, E)



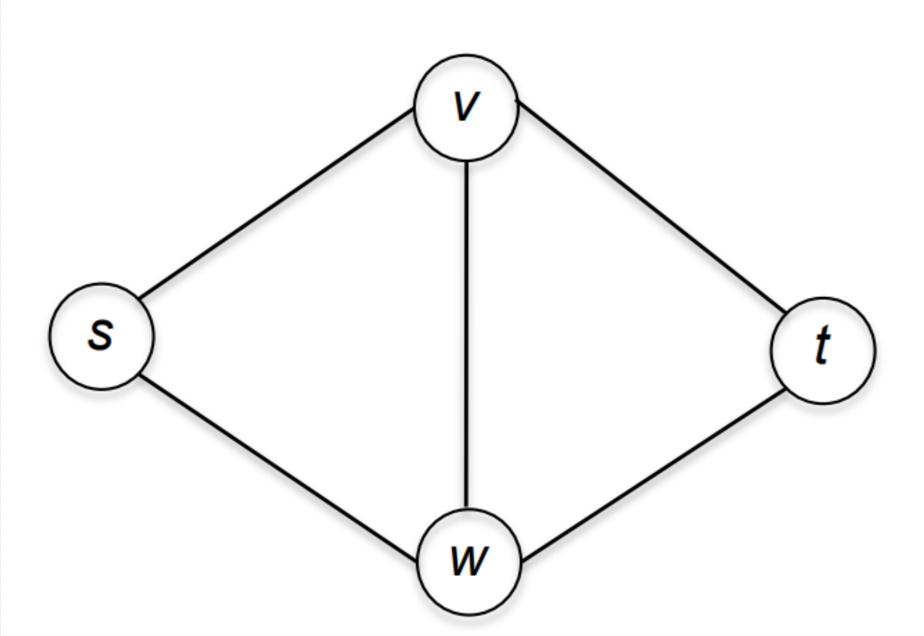
Adjacency-list representation of G

	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

Adjacency-matrix representation of G

Graph Search

- For a graph G = (V, E) with vertex set V and edge set E, graph search refers to the process of visiting each vertex in a graph
- Traversals are classified by the order in which the vertices are visited
- Also known as **Graph Traversal**

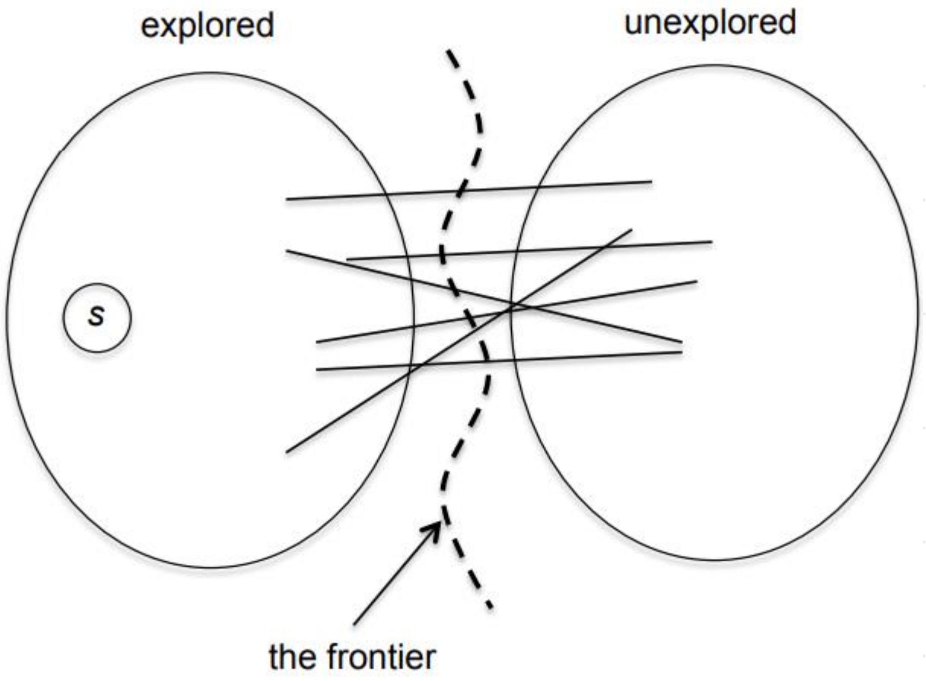


Applications of Graph Search

- Checking connectivity
- Shortest paths
- Planning
- Connected components

Breadth-first Search

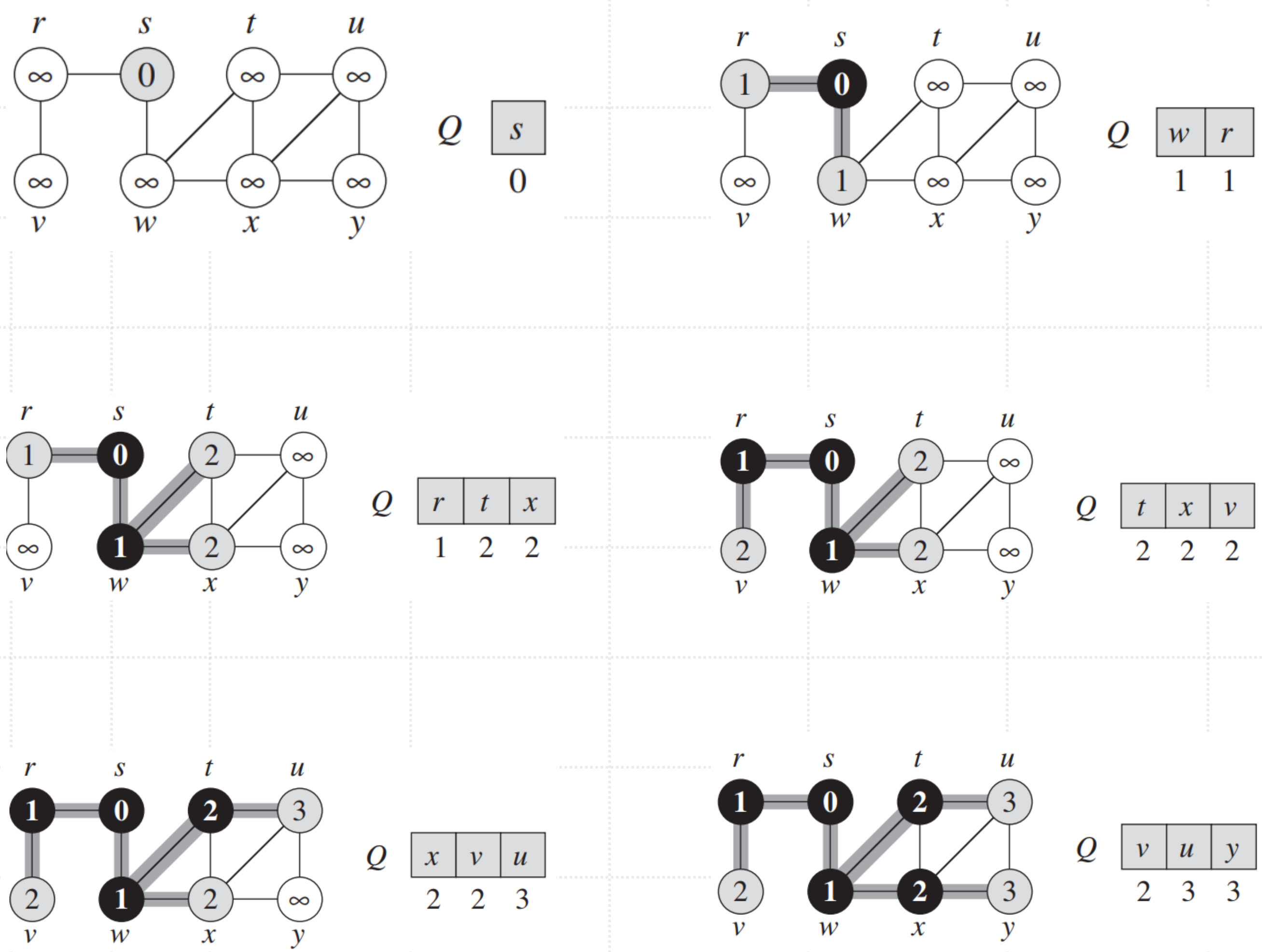
- Breadth-first search (BFS) is one of the simplest algorithms for graph traversal
- BFS calculates the shortest path between two nodes
- For a given graph, $G = (V, E)$ and a source vertex $s \in V$ BFS explores the edges of G to discover every vertex that is reachable from s
- BFS works on both directed and undirected graphs
- It expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier, so it is called Breadth-first search
- It discovers all vertices at distance k from s before discovering any vertices at distance $k + 1$



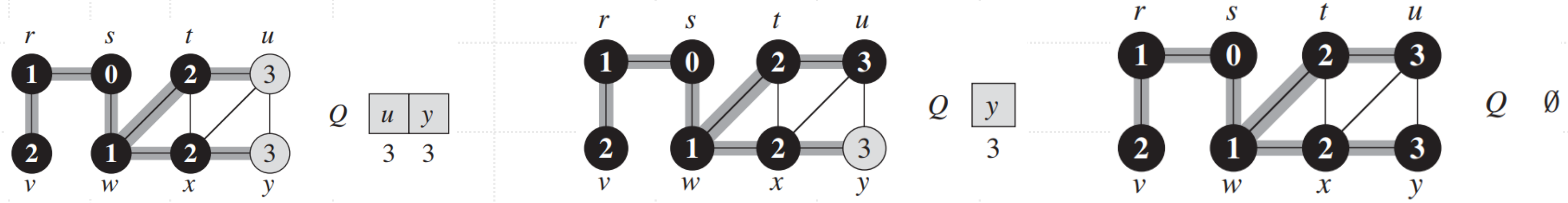
BFS Algorithm

- Initially each of the vertex is configured as:
  - Color to WHITE
  - depth to infinity
  - parent as NIL
- A queue is used to keep track of the traversal
- If a vertex with WHITE color is discovered, it painted GRAY and added to the queue
- If a vertex completes traversal, it painted BLACK
- The traversal continues until the queue is empty
- Running time: O(V + E)
- BFS ensures to visit each node once only

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = Ø
ENQUEUE(Q, s)
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```




```
BFS(G, s)
    for each vertex u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
    s.color = GRAY
    s.d = 0
    s.π = NIL
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
            if v.color == WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.color = BLACK
```



Challenge Solving Session Live

1. Challenge: https://practice.geeksforgeeks.org/problems/bfs-traversal-of-graph/1
2. Solution: https://gist.github.com/arsho/5a0e8670b328909b22b94069e157de5d

References

1. Rivest, R. L., Leiserson, C. E., Stein, C., Cormen, T. H. (2009). Introduction to Algorithms. United Kingdom: MIT Press.
2. Roughgarden, T. (2018). Algorithms Illuminated: Graph algorithms and data structures. Part 2. United States: Soundlikeyourself Publishing LLC.

THANK YOU