Lecture 34-35

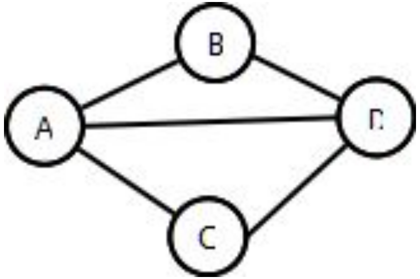
- Introduction to Graphs

IT205: Data Structures (AY 2023/24 Sem II Sec B) — Dr. Arpit Rana

Graph: Definition

A graph G is a non-linear data structure made up of a **set of nodes (vertices, i.e., V)** and a **set of edges (arcs, i.e., E)** that connect them.

- Example, G = (V, E):
  - V = {A, B, C, D}
  - E = {(A,B), (A,C), (A,D), (B,D), (C,D)}



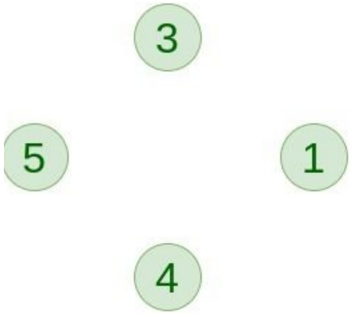
Complexity of Graph-based Algorithms

When we characterize the running time of a graph algorithm on a given graph $G = (V, E)$, we usually measure the size of the input in terms of

- the number of vertices, $|V|$, and
- the number of edges, $|E|$

Hence, the input is denoted using two parameters and not just one.

Types of Graph



Null Graph

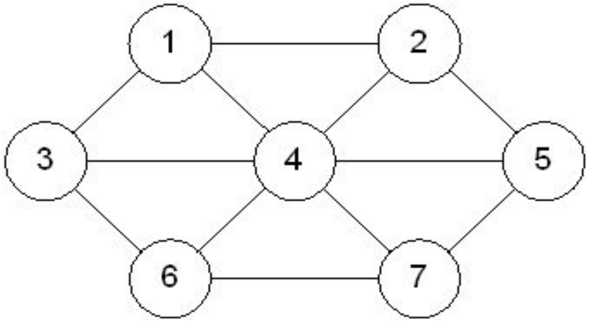
i.e., $E = \phi$

Trivial Graph

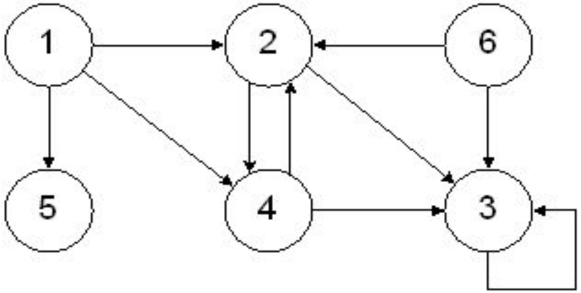
i.e., $E = \phi$ and $| V | = 1$

Types of Graph

- Each edge is specified by a pair of nodes.
- If the pair of nodes that make up the edges are ***ordered pairs***, the graph is said to be a ***directed graph (or digraph)***, otherwise ***undirected graph***.



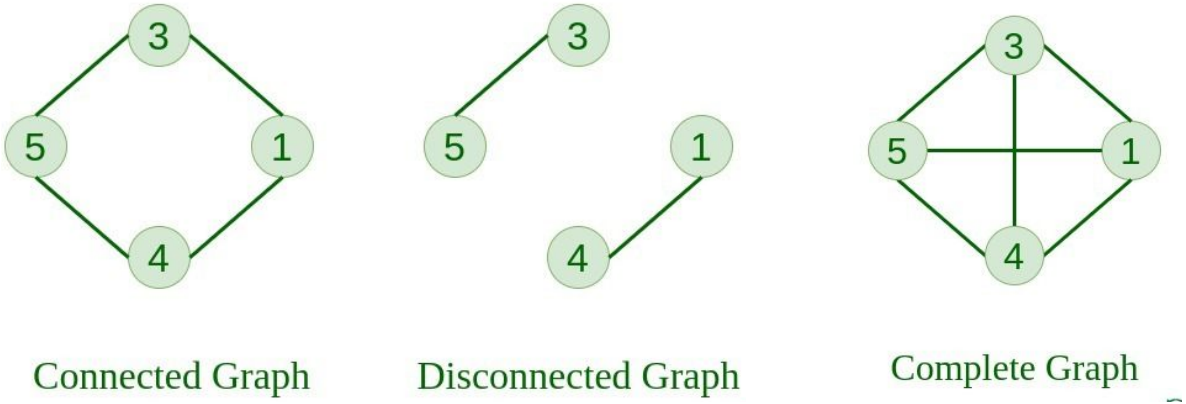
Undirected Graph



Directed Graph

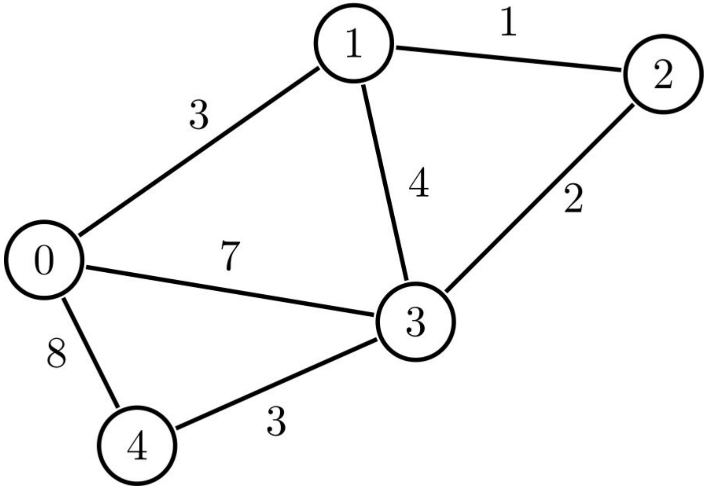
Types of Graph

- The graph in which from one node we can visit any other node in the graph is known as a *connected graph*.
  - The graph in which from each node there is an edge to each other node is known as *fully connected (or complete) graph*..
- The graph in which at least one node is not reachable from a node is known as a *disconnected graph*.



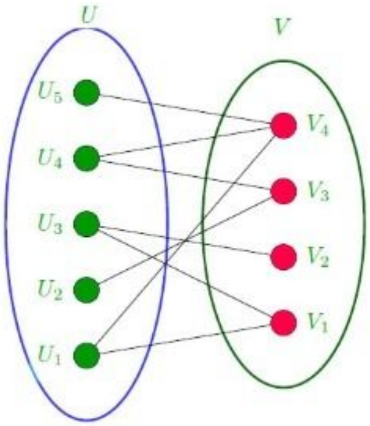
Types of Graph

- A graph in which each edge is associated with a ***weight*** given by a ***weight function*** w: E→R is known as a ***weighted graph***.
    - Weight typically shows ***cost of traversing***, for example, weights are distances between cities



Types of Graph

- A graph in which vertex can be divided into two sets such that vertex in each set does not contain any edge between them is known as ***bipartite graph***.

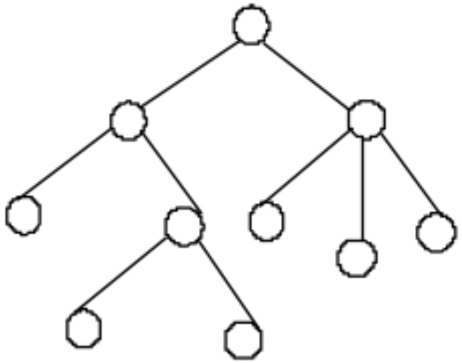


Bipartite Graph

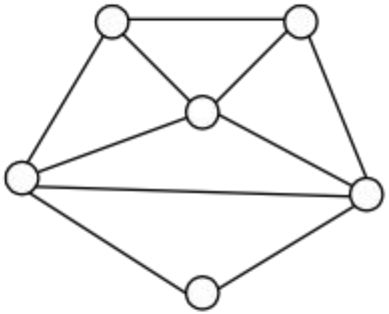
Tree vs. Graphs

Two structures are similar in the sense that they represent connectivity among the nodes

- Both belong to the category of non-linear data structures
- They are different in the sense that tree is ***acyclic*** whereas a graph usably have cycle(s).



Tree

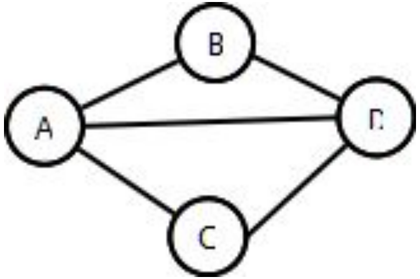


Graph

Degree of a Node in a Graph

The **degree** of a node is the number of edges the node is used to define.

- In the example below:
  - Degree 2: B and C
  - Degree 3: A and D
- A and D have odd degree, and B and C have even degree
- Can also define in-degree and out-degree (defined for digraphs)
  - **In-degree**: Number of edges pointing to a node
  - **Out-degree**: Number of edges pointing from a node
- A node with degree 0 is called as **isolated node**.



Representation of Graphs

A graph G (V, E) can be represented in two standard ways

- **Adjacency List (Linked) representation**

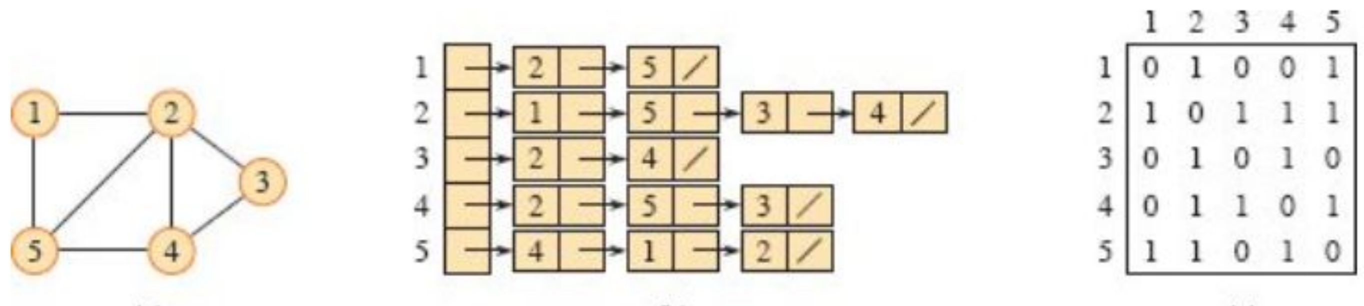
  Used when the graph is sparse, i.e., $| E | << | V |^2$

- **Adjacency Matrix representation.**

  Used when the graph is dense, i.e., $| E | \sim | V |^2$

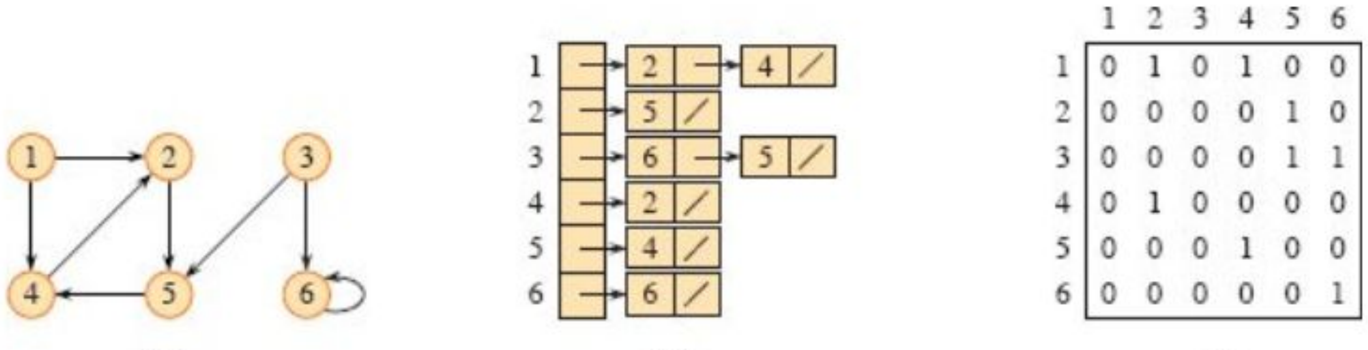
Representation of Undirected Graphs

An undirected graph G (V, E) with | V | = 5, and | E | = 7. The sum of all adjacency lists is 2.| E |.



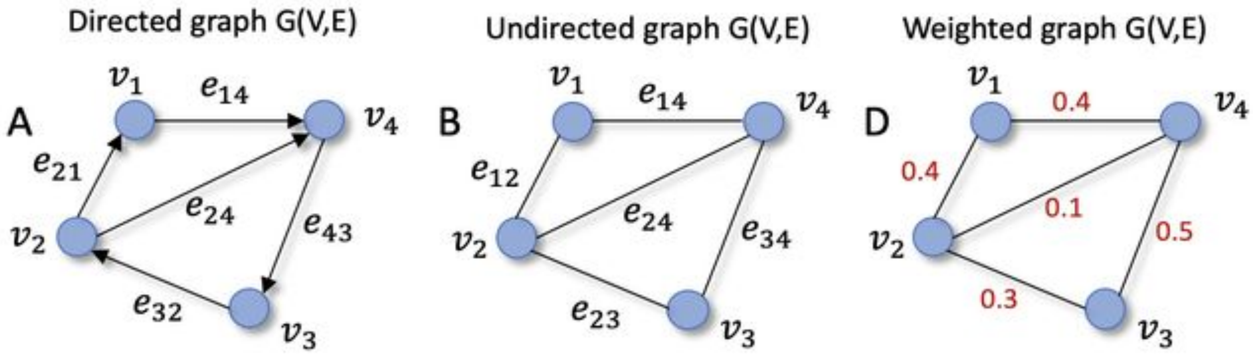
(a) (b) (c)

A directed graph G (V, E) with | V | = 6, and | E | = 8. The sum of all adjacency lists is | E |.



(a) (b) (c)

Representation of Various Types of Graphs as Adjacency Matrix



E

	v_1	v_2	v_3	v_4
v_1	0	0	0	1
v_2	1	0	0	1
v_3	0	1	0	0
v_4	0	0	1	0

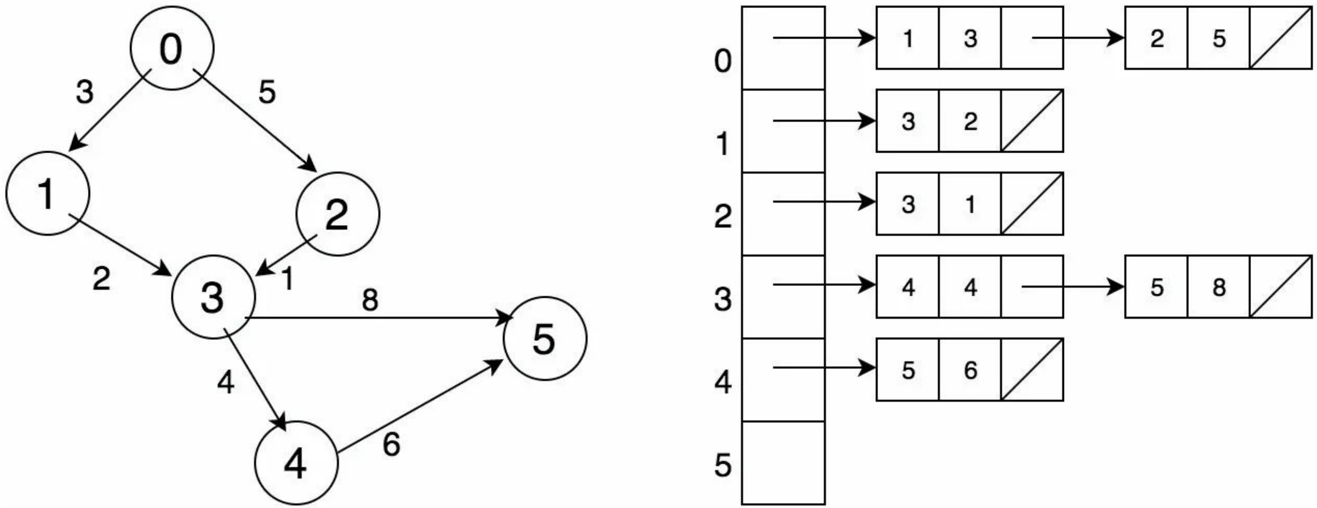
F

	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	1
v_3	0	1	0	1
v_4	1	1	1	0

H

	v_1	v_2	v_3	v_4
v_1	0	0.4	0	0.4
v_2	0.4	0	0.3	0.1
v_3	0	0.3	0	0.5
v_4	0.4	0.1	0.5	0

Representation of Weighted Digraphs as Adjacency List



Adjacency List vs. Adjacency Matrix

- A potential disadvantage of adjacency list is that it provides no quicker way to find whether an edge (u, v) is present in a graph G. (it take Θ (|V| + |E|) time).
  - Some improvements in search are still possible.
- Adjacency matrix solves this problem on the cost of Θ ($|V|^2$) space.
- In case of unweighted graphs, adjacency matrix require just one bit per entry (0 or 1) which makes it more space efficient for smaller graphs.

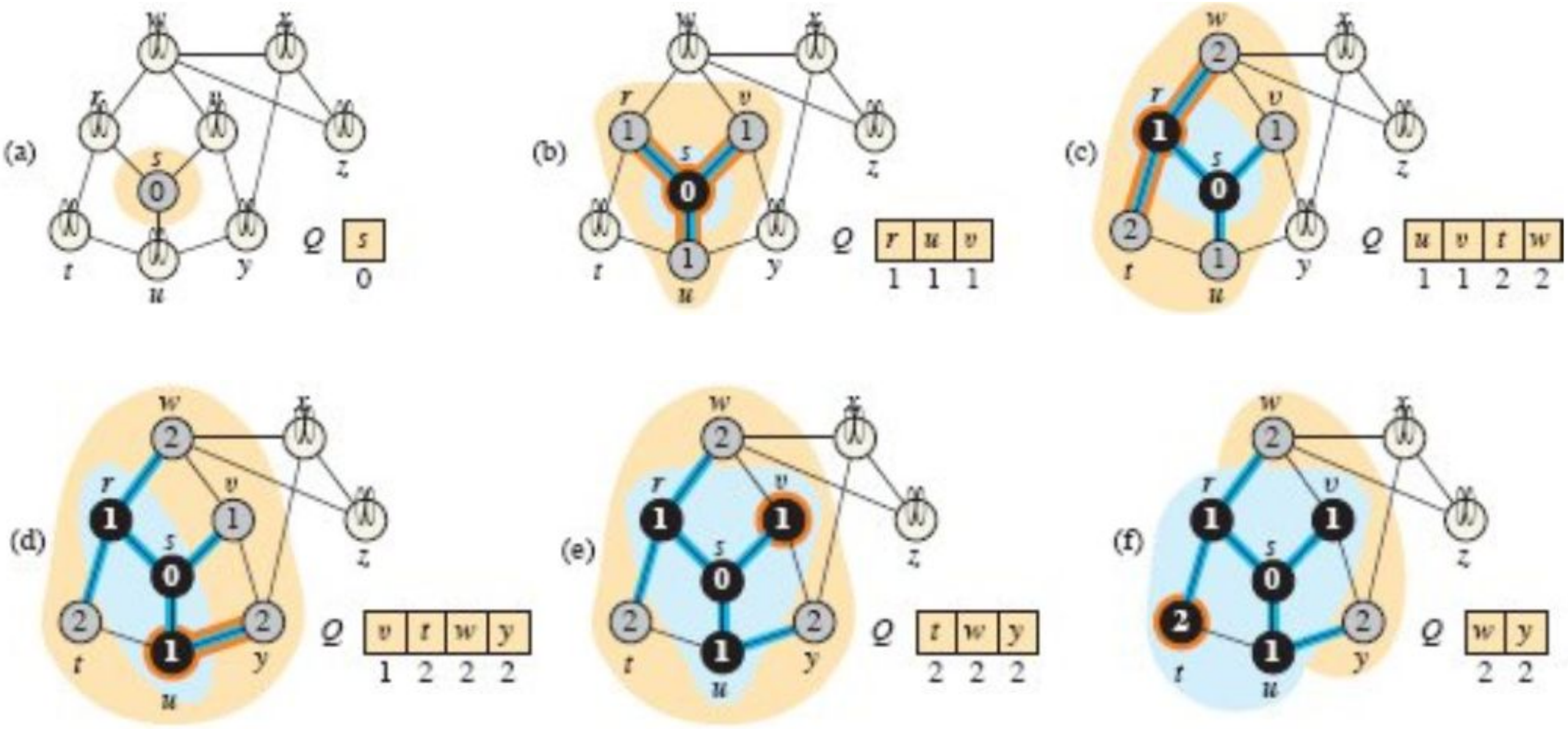
Graph Traversals: Breadth-First Search (BFS)

- Given a graph G = (V, E) and a source vertex s, BFS explores the edges of G to “discover” every vertex that is reachable from s.
    - It computes the **distance** from s to each reachable vertex v: the smallest number of edges needed to go from s to v.
    - Starting from s, the algorithm first discovers all neighbors of s which have distance 1, then discovers vertices with distance 2, and so on, until it has discovered every vertex reachable from s.

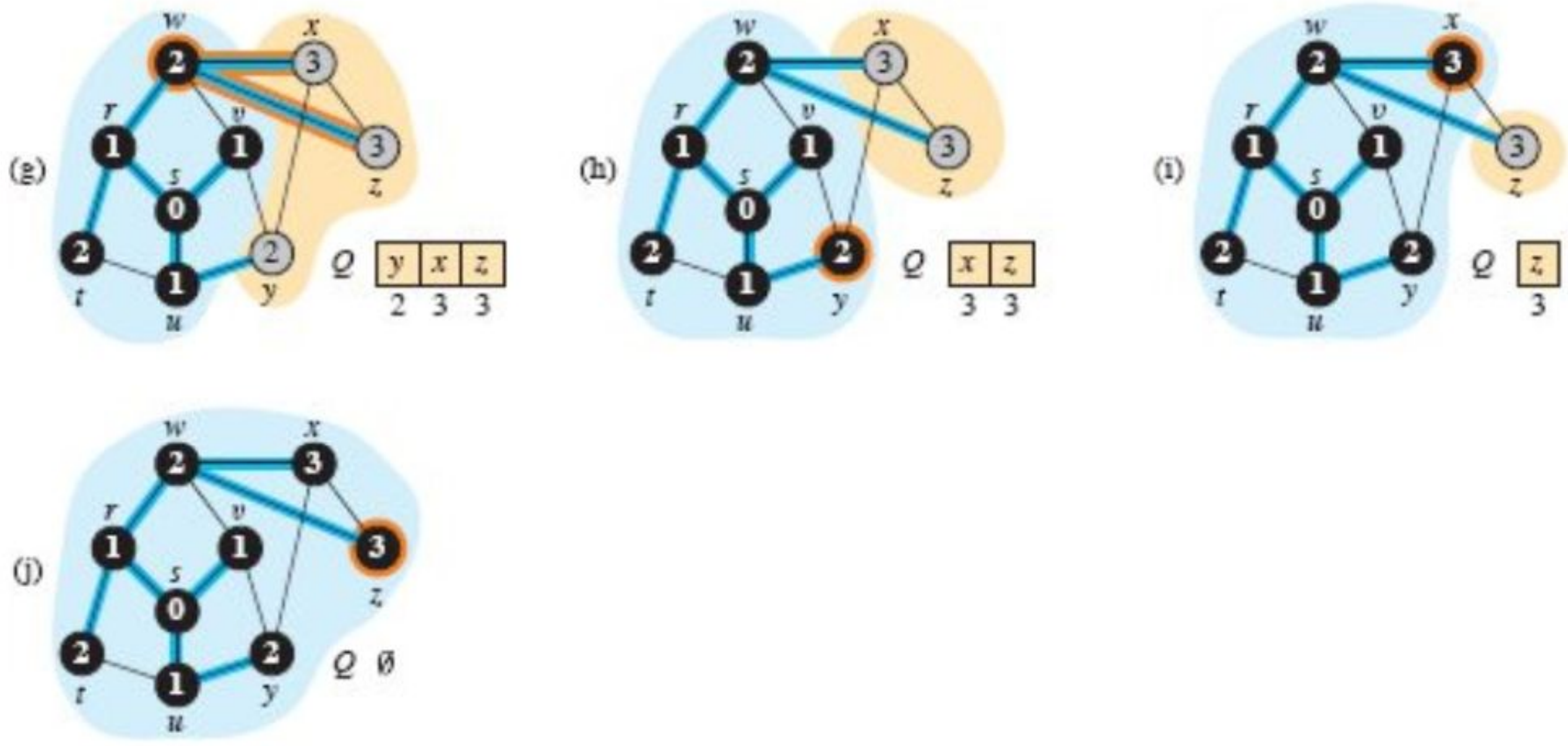
Graph Traversals: Breadth-First Search (BFS)

- It uses a FIFO queue containing some vertices at a distance k, possibly followed by some vertices at distance k+1.
- To keep track of progress, BFS colors each vertex white (initialized), gray (discovered, i.e., added to the queue), and black (explored, i.e., all vertex's edges have been explored).
- BFS constructs a breadth-first tree, initially containing only its root, which is the source vertex s.
  - Whenever the search discovers a white vertex v in the course of scanning the adjacency list of a gray vertex u, the vertex v and the edge (u, v) are added in the tree.
  - We say that u is the predecessor or parent of v in the breadth-first tree.

Graph Traversals: Breadth-First Search (BFS)



Graph Traversals: Breadth-First Search (BFS)



Analysis of Breadth-First Search

- A vertex is enqueued and dequeued at most once which takes O (1) time. So, the total time devoted to queue operations is O (|V|).

- The procedure scans the adjacency list of each vertex when the vertex is dequeued, so, at most once.

    - The sum of the lengths of all | V | adjacency lists is Θ (|E|), total time spent in scanning adjacency lists is O (|V| + |E|).

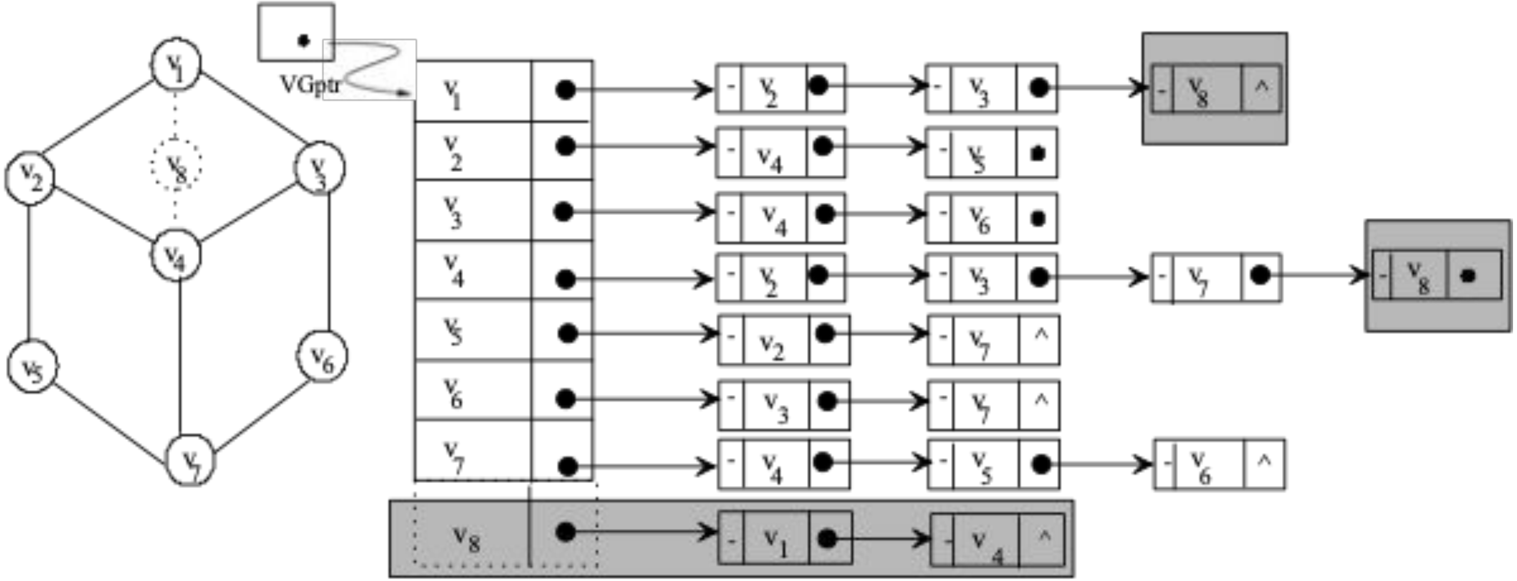
- Thus the total running time of BFS procedure is O (|V| + |E|).

Alternatively, a graph can be traversed using Depth-First Search (similar to in-order traversal of the tree). However, we will not discuss that procedure in this course.

Other Operations on a Graph

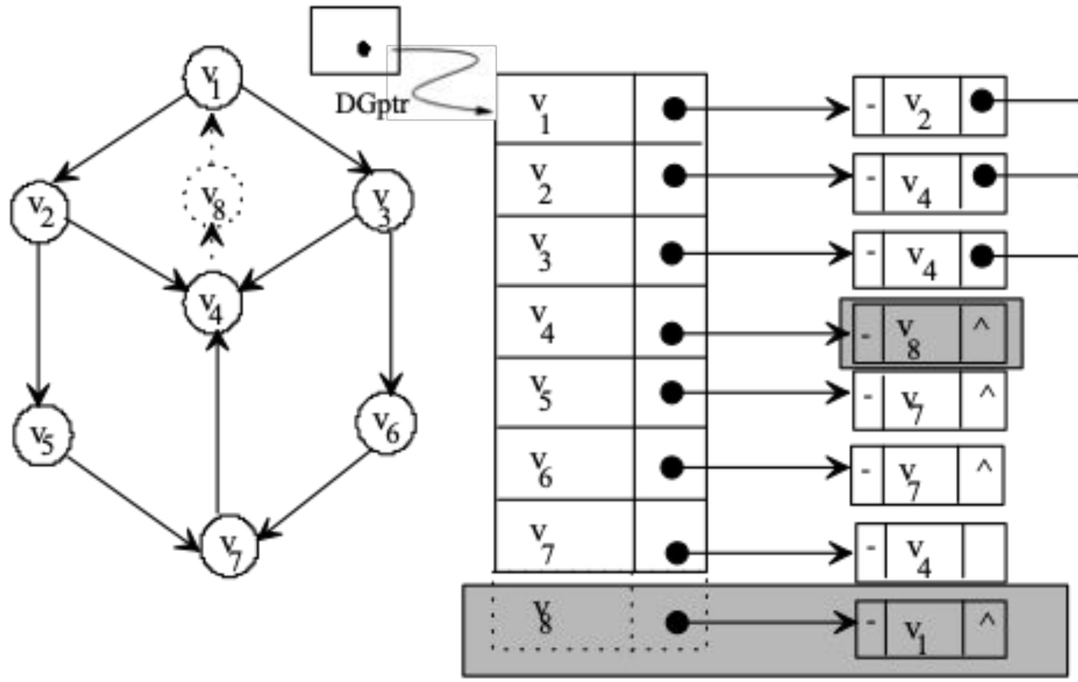
- Insertion
  - To insert a vertex and hence establishing connectivity with other vertices in the existing graph.
  - To insert an edge between two vertices in the graph.
- Deletion
  - To delete a vertex from the graph.
  - To delete an edge from the graph.
- Merging
  - To merge two graphs G1 and G2 into a single graph.

Insertion Into an Undirected Graph



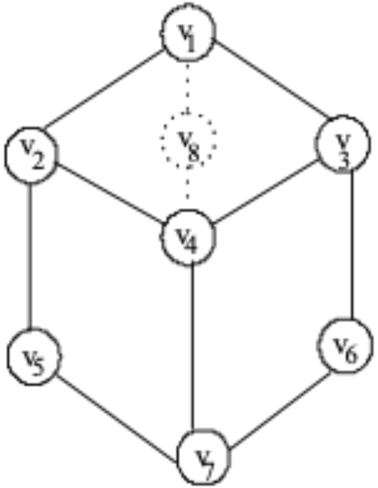
(a)

Insertion Into a Digraph

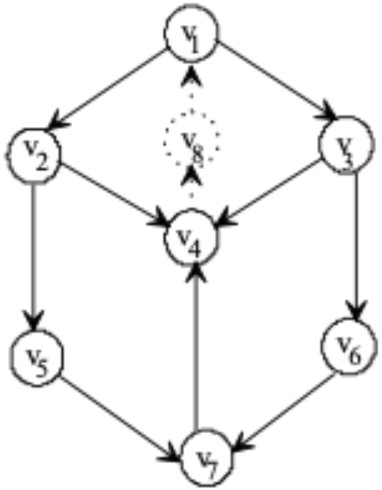


(b)

Insertion Into a Graph: Matrix Representation

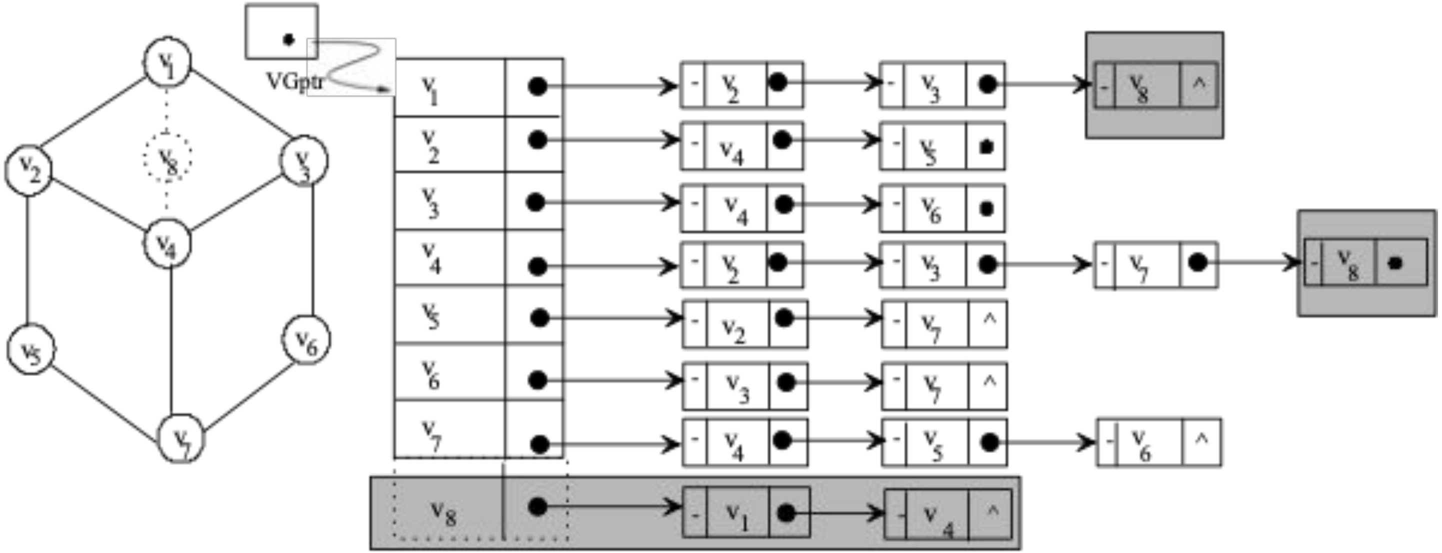


	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	1
2	1	0	0	1	1	0	0	0
3	1	0	0	1	0	1	0	0
4	0	1	1	0	0	0	1	1
5	0	1	0	0	0	0	1	0
6	0	0	1	0	0	0	1	0
7	0	0	0	1	1	1	0	0
8	1	0	0	1	0	0	0	0



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	0	0	0	1	1	0	0	0
3	0	0	0	1	0	1	0	0
4	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	1	0
7	0	0	0	1	0	0	0	0
8	1	0	0	0	0	0	0	0

Deletion from a Graph



(a)

End of the Course

- Keep Learning!!