GRAPH Data Structure

Instructor: Javeria Naz

Introduction

- In Tree data structures, one parent node may have no child, or many child nodes.
- Therefore, Tree data structures represent one-to-many relationships.
- In real life, however, we frequently come across problems that can be best described by many-to-many relationships.
- Such problems cannot be solved using trees or other data structures we have studied so far.
- To deal with such problems, graph data structures are used.

Graph

- Graph is a non-linear data structure.
- It consists of a set of nodes which are connected by lines (links).
- The nodes are called vertices. {V_1, V_2, ..., V_n} or points whereas lines that connect the vertices are called edges {e_i, e_z, ..., e_n} or arcs.
- An edge is a pair of vertices, which is represented as:

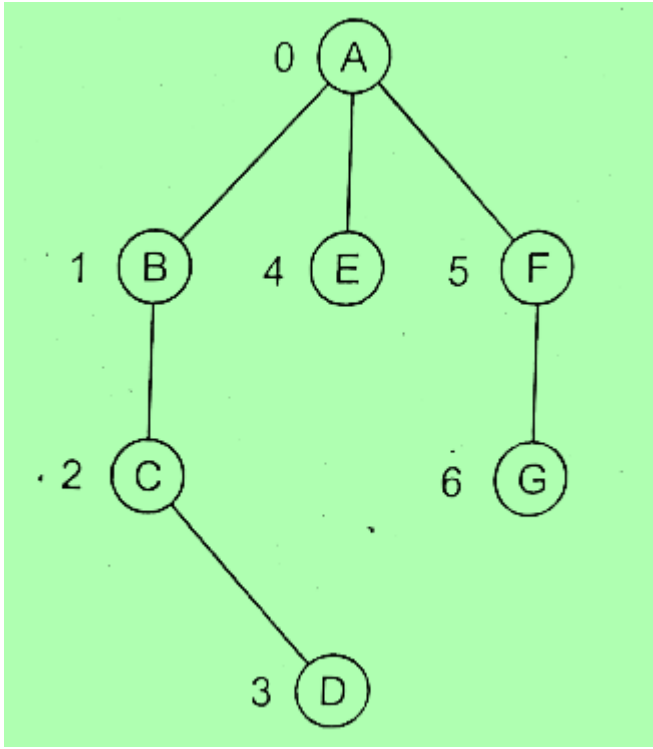
$$e = \{ V_i , V_s \}$$

- Where 'V_i' and 'V_s' denote the start and end vertices/nodes of edge "e", respectively. They are also known as the head and tail nodes of edge 'e'. The two vertices are also called the edge endpoints. Consider the following figure:

Cont.

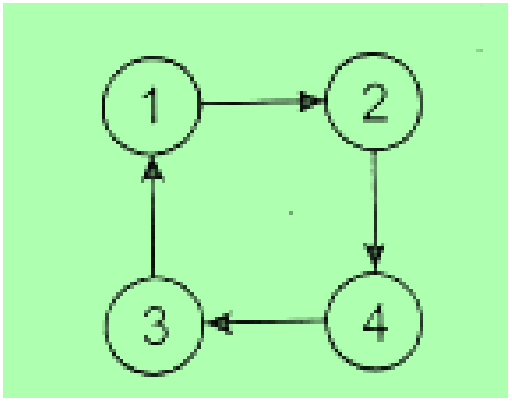
Consider the following figure:

- In this figure, the labeled circles represent vertices.
- Thus, A, B, C, D, E, F and G are vertices, whereas the lines from A to B, B to C, and so on represent edges.



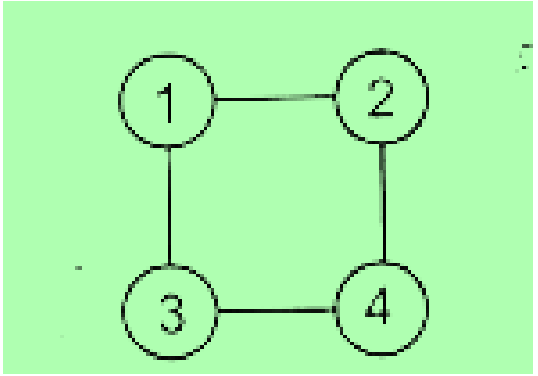
Types of Graph

- **1- Directed Graph**
- A graph with edges directed from one vertex to another is called a directed graph.
- A directed graph is also called a **digraph**.
- The edges of a directed graph are unidirectional i.e. single directional.
- This figure consists of four vertices numbered 1, 2, 3, 4 and four directed edges joining vertices 1 to 2, 2 to 4,4 to 3, and 3 to 1.
- The directed graphs are commonly used to analyze electrical circuits, develop project schedules, find shortest routes, analyze social relationships, and construct models for analysis and solution of many other problems.



Cont.

- **2- Undirected Graph**
- A graph whose edges are not directed is called an undirected graph.
- It means that edges have no direction.
- Following Figure shows undirected graph.

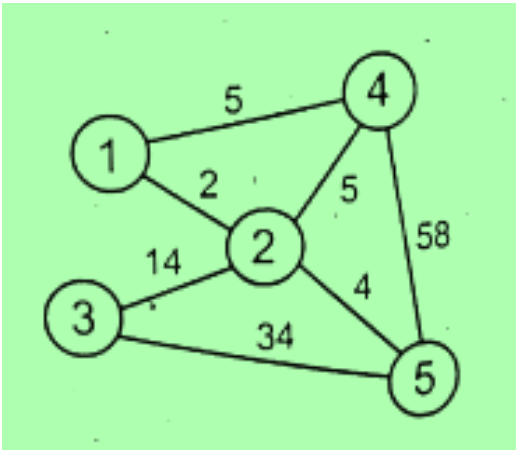


Cont.

- **3. Weighted Graph**
- A graph which has a weight, number or any value associated with each edge (called edge weight) is called weighted graph.
- Weight of an edge is sometimes also called its cost.
- The weight of an edge usually represents certain conditions or situations associated with the edge.
- For example, in a road network, weight of each road may denote the distances between two cities.
- An edge is a weighted graph is represented as:

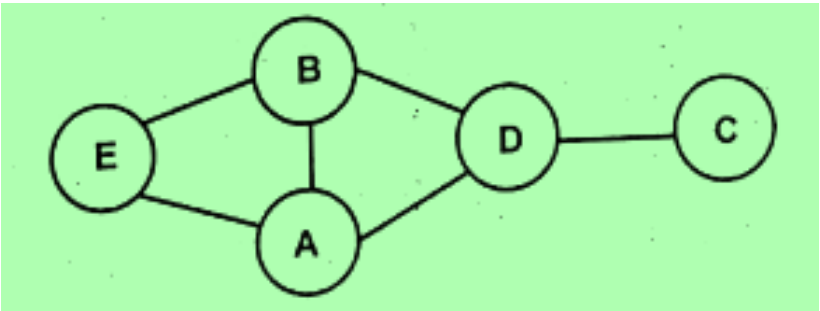
$$e = \{ V_i , V_s , W \}$$

- Where ‘V_i' and ‘Vs' denote the start and end vertices/nodes of edge “e”, respectively, W is the weight of edge.



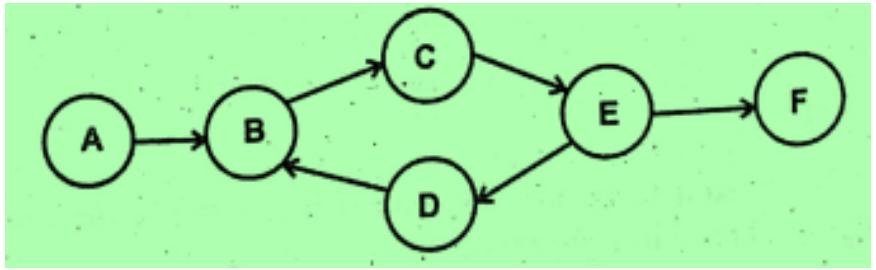
Cont.

- **4- Degree of Node**
- The number of edges that a node contains is called degree of the node.
- In the diagram shown below, nodes A, B & D have degree of 3, node E has a degree of 2 and graph shown below node C has a degree of 1.
- A node that has degree 0 is called isolated node. A graph that has only one isolated node is called null graph.



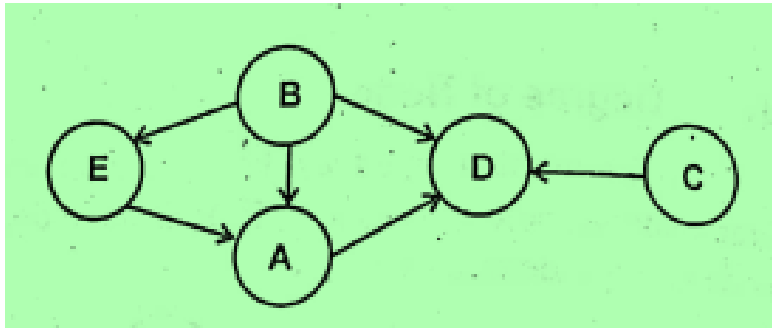
Cont.

- **5. In-Degree & Out-Degree**
- In a directed graph, the number of edges that point to a given vertex is called its in-degree.
- In the directed graph shown in figure, the in-degree of node A is 0 and node B has an in-degree of 2.
- The number of edges that point from a given vertex is called its out-degree.
- In graph shown in figure the out-degree of nodes A, B, C & D is "1". Similarly, node E has an out-degree of 2.
- The node that has an out-degree of "0", called terminal node or leaf node and other nodes are called the branch nodes. In the figure, F is terminal node.
- 10 The sum of the out-degree & in-degree is the total degree of the node. The degree of a loop node is 2 and that of an isolated node is 0.



Cont.

- **6- Source & Sink Nodes**
- The node that has a positive out-degree but zero in-degree is called source node.
- In the graph shown in figure, node B is the source node. This node has a positive out-degree of 3 and its in-degree is zero.
- The node that has zero out-degree but a positive in-degree is called sink node.
- In the graph shown in figure, node D is the sink node.
- This node "D" has a positive in-degree of 3 and its out-degree is zero.

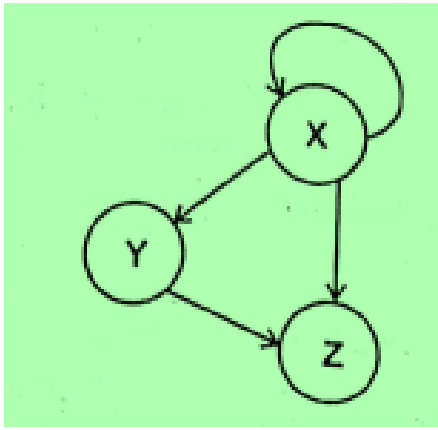


Cont.

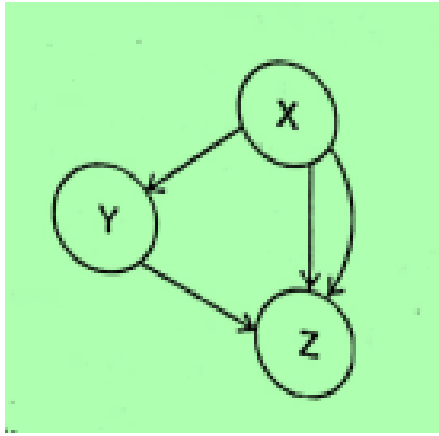
- **7. Path & Length of Graph**
- A path in a graph is a sequence of vertices such that there exists an edge or arc between consecutive vertices.
- It is written as a sequence of vertices / nodes {V_{11}, V_{i2}, ... V_{ix}).
- A path which does not repeat any node is known as simple path.
- A path is usually assumed to be a simple path, unless otherwise defined.
- The number of edges in a path of a graph is called length of the graph.
- For a path consisting of 'n' nodes, the path length is 'n-1!

Cont.

- **Loop Edge**
- An Edge is said to be loop edge if Head & Tail points to the same node.

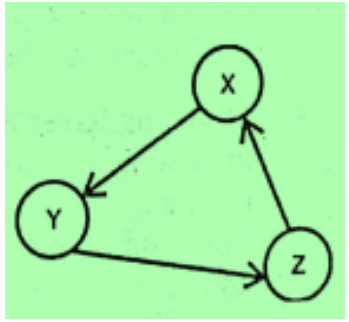


- **Multiple Edges**
- If there exists more than one edges between the same two nodes.



Cont.

- **Cycle & Acycle**
- A path which originates or begins and ends at the same node is called cycle.
- In other words, a path from a particular node up to itself is called a cycle.
- The length of a cycle must be at least 1.
- The directed graph that has no cycles is called acyclic graph.
- A directed acyclic graph is also known as DAG.

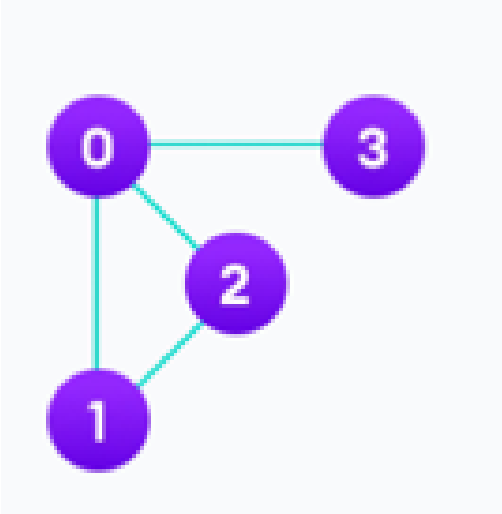


Graph Representation

Graphs are commonly represented in two ways:

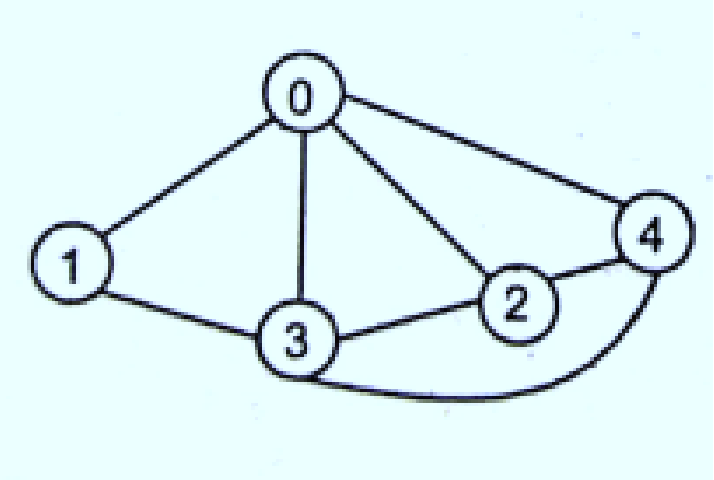
1. Adjacency Matrix

- An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex.
- If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.



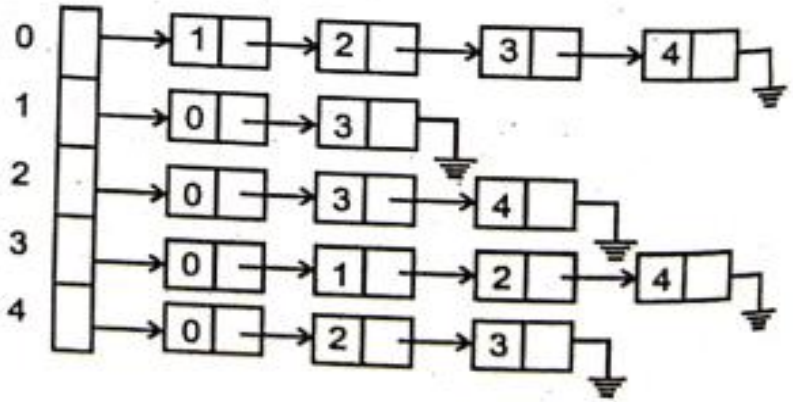
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

Graph Representation



- **2. Adjacency List**

- An adjacency list represents a graph as an array of linked lists.
- The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.
- The adjacency list for the graph we made in the first example is as follows:




```
0 - [1, 2, 3, 4]
1 - [0, 3]
2 - [0, 3, 4]
3 - [0, 1, 2, 4]
4 - [0, 2, 3]
```

Graph traversal

- Graph traversal is the technique used for searching a vertex in a graph.
- The graph traversal is also used to decide the order of vertices to be visited in a search process.
- A graph traversal finds the edges to be used in the search process without creating loops.
- It means that using graph traversal, we visit all the vertices of a graph without getting into looping path.
- There are two standard graph traversal techniques. These are as follows:

  i) Breadth-First Search (BFS)

  ii) Depth First Search (DFS)

Breadth-First Search (BFS)

- Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighboring nodes.
- Then, it selects the nearest node and explore all the unexplored nodes.
- The algorithm follows the same process for each of the nearest node until it finds the goal.
- Breadth First Search (BFS) algorithm traverses a graph in a level-order traversal and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration

BFS Algorithm

- **Write an algorithm to explain the Breadth-First Search.**
  - START
  - Define a Queue of size total number of vertices of the graph.
  - Select any vertex as starting point for traversal (If start is not mentioned)
  - Visit that vertex and insert it into the queue.
  - Visit all the adjacent vertices of the vertex which is at front of the Queue.
  - Insert the vertices into the queue which are not visited.
  - When there is no new vertex to be visited from the vertex at front of the Queue then delete the front vertex from the queue.
  - Repeat step 4 and 5 until queue becomes empty.
  - EXIT

DFS - Depth-first search

- Depth-first search is an algorithm for traversing or searching tree or graph data structures.
- The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.
- So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node.
- Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

DFS Algorithm

- **Write an algorithm to explain the Depth-First Search.**
  - START
  - Define a Stack of size total number of vertices in the graph.
  - Select any vertex as starting point for traversal. Visit that vertex and push it onto the Stack.
  - Visit all the adjacent vertices of the vertex which is at the top of the stack.
  - Push the vertices onto the stack which
  - Repeat step 4 until there is no new vertex to be visit from the vertex on the top of the stack.
  - When there is no new vertex to be visit then use back tracking and pop one vertex from the top of stack.
  - Repeat steps 4 to 6 until stack becomes Empty
  - EXIT

Practice Task

- Examples as solved in class.
- Use the following graph, traverse using BFS and DFS.
- Also explain each step theoratically.