DS UNIT 6

[4-marks]

1. **Define graph. Explain its basic components.**

A **graph** is a non-linear data structure used to represent relationships between different objects. It is made up of a collection of nodes connected with each other using links.

A **vertex (node)** is the fundamental unit of a graph that represents an entity or object. For example, in a road network, cities are represented as vertices.

An **edge** is a link that connects two vertices in a graph. It represents the relationship, path, or connection between those vertices.

A graph can be **directed or undirected** based on the nature of edges. In a directed graph, edges have a specific direction, while in an undirected graph, edges allow movement in both directions.

A graph may be **weighted or unweighted**. In a weighted graph, each edge has a value such as cost, distance, or time associated with it.

Graphs are widely used to model real-world systems. Examples include social networks, road maps, computer networks, and communication systems.

2. **What is adjacency matrix representation?**

Adjacency matrix is a method of representing a graph using a two-dimensional array (matrix). Each row and each column of the matrix represent a vertex of the graph.

If there is an **edge between vertex i and vertex j**, the value at position [i][j] in the matrix is **1**. If there is no edge between them, the value stored is **0**.

For a graph having **n vertices**, the adjacency matrix will always be of size **n × n**. This fixed size makes the structure easy to understand and manage.

In **undirected graphs**, the matrix is symmetric because [i][j] = [j][i]. In **directed graphs**, the matrix may not be symmetric since direction matters.

Checking whether an edge exists between two vertices is **very fast**, because it takes constant time. You just need to check the value in the matrix.

However, adjacency matrix uses **more memory**, even if very few edges exist. Therefore, it is not efficient for **sparse graphs** with fewer connections.

3. What is adjacency list representation?

Adjacency list represents a graph using an array of linked lists or arrays. Each index of the array represents a vertex, and the list stores its neighboring vertices.

If a vertex has an edge with another vertex, that connected vertex is added to its list. This clearly shows which vertices are directly connected.

Adjacency list uses less memory because it stores only existing edges. This makes it very suitable for graphs with fewer edges.

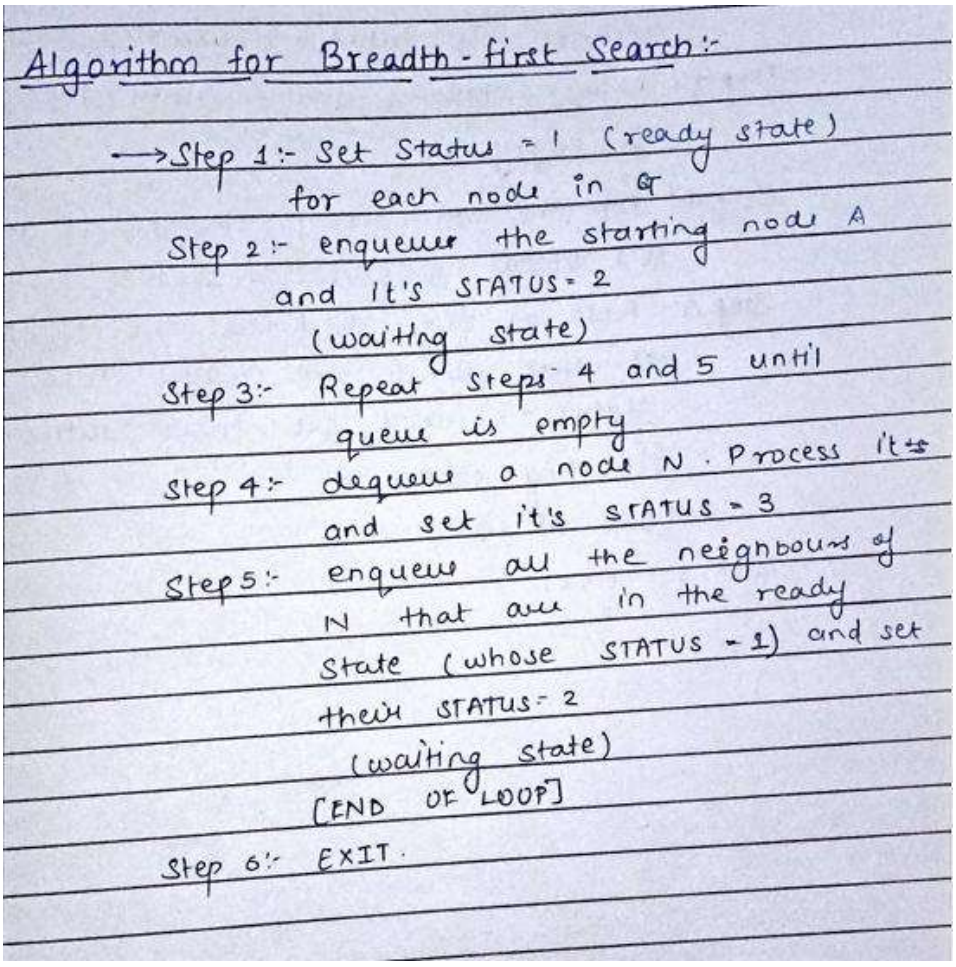
In directed graphs, each list stores only outgoing edges from that vertex. In undirected graphs, each edge is stored in both vertices' lists.

Traversing all neighbors of a vertex is easy and efficient using this representation. Many graph algorithms like BFS and DFS prefer adjacency lists.

Adjacency list is flexible, scalable, and widely used in real-world applications like social networks and routing systems.

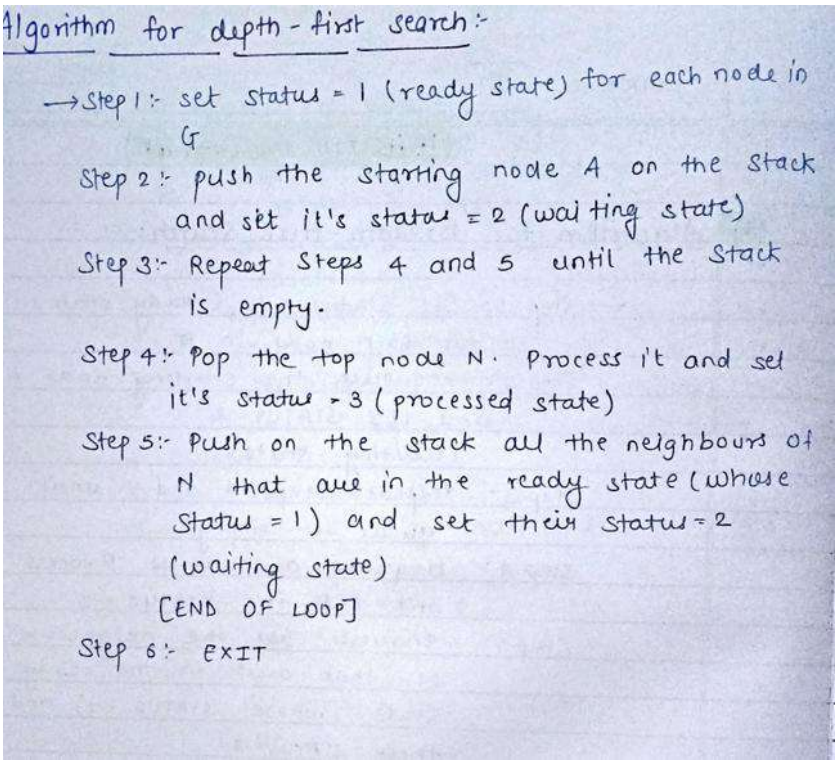
4. Write the algorithm for Breadth First Search (BFS).

Breadth First Search is a graph traversal technique that visits all vertices level by level starting from a given source vertex. It uses a queue to keep track of vertices to be visited next.



5. **Write the algorithm for Depth First Search (DFS).**

Depth First Search is a graph traversal technique that explores as far as possible along one path before backtracking. It uses recursion or a stack to remember previously visited vertices.



Algorithm for depth-first search:-

→ Step 1:- set status = 1 (ready state) for each node in G

Step 2:- push the starting node A on the stack and set it's status = 2 (waiting state)

Step 3:- Repeat Steps 4 and 5 until the Stack is empty.

Step 4:- Pop the top node N. Process it and set it's status = 3 (processed state)

Step 5:- Push on the stack all the neighbours of N that are in the ready state (whose Status = 1) and set their status = 2 (waiting state)

[END OF LOOP]

Step 6:- EXIT

6. **Differentiate between BFS and DFS.**

Basis	BFS (Breadth First Search)	DFS (Depth First Search)
Traversal method	BFS traverses the graph level by level starting from the source node. It visits all neighbors first.	DFS traverses the graph by going deep into one path before exploring others.
Data structure used	BFS uses a queue to store nodes for traversal. The queue follows FIFO order.	DFS uses a stack or recursion for traversal. It follows LIFO order.
Memory usage	BFS requires more memory because it stores many nodes at the same level.	DFS requires less memory as it stores only the current path.
Shortest path	BFS can find the shortest path in an unweighted graph.	DFS does not guarantee the shortest path.
Applications	BFS is used in shortest path problems and level order traversal.	DFS is used in cycle detection and topological sorting.

7. Define connected graph and complete graph.

Connected Graph

- A connected graph is a graph in which there is a path between every pair of vertices. This means all vertices are reachable from one another.
- In a connected graph, no vertex is isolated from the rest of the graph. You can start from any vertex and reach all other vertices.
- If even one vertex cannot be reached from another vertex, the graph is called disconnected. Connected graphs are common in road networks and communication systems.

Complete Graph

- A complete graph is a graph in which every vertex is directly connected to every other vertex. There is exactly one edge between each pair of vertices.
- In a complete graph with n vertices, each vertex has a degree of (n − 1). This means every vertex is connected to all remaining vertices.
- Complete graphs have the maximum possible number of edges. They are mainly used in theoretical studies and mathematical problems.

[5-marks]

1. **Explain adjacency matrix representation of graph with example.**

Adjacency matrix representation is a method of storing a graph using a **two-dimensional array**. In this representation, the **rows and columns represent the vertices** of the graph. If there is an edge between two vertices, the corresponding cell in the matrix contains **1**; otherwise, it contains **0**.

For a graph having **n vertices**, the adjacency matrix will always be of size **n × n**. This fixed size makes it easy to understand and implement. In an **undirected graph**, the matrix is symmetric because the edge between vertex A and B is the same as between B and A. In a **directed graph**, the matrix may not be symmetric because direction matters.

Example: Consider a graph with vertices A, B, and C. Edges: A–B, B–C

Adjacency Matrix:

```
   A B C
A  0 1 0
B  1 0 1
C  0 1 0
```

Adjacency matrix allows **fast edge checking**, but it consumes more memory, especially for sparse graphs.

2. **Explain adjacency list representation of graph with example.**

Adjacency list representation stores a graph using an **array of lists**. Each vertex has a list that contains all the vertices directly connected to it. This representation stores **only existing edges**, making it memory efficient.

In this method, the size of memory depends on the number of edges rather than the number of vertices. It is very suitable for **sparse graphs** where the number of edges is small. For **directed graphs**, each vertex stores only outgoing edges. For **undirected graphs**, each edge appears in the list of both vertices.

Example:
Consider the same graph with vertices A, B, and C.
Edges: A–B, B–C

Adjacency List:

```
  A → B
B → A, C
  C → B
```

Adjacency list makes it **easy to traverse neighbors** of a vertex and is widely used in algorithms like BFS and DFS.

3. **Explain BFS traversal with example.**

Breadth First Search (BFS) is a graph traversal method in which vertices are visited **level by level** starting from a given source vertex. It explores all neighboring vertices before moving to the next level.

BFS uses a **queue** data structure to store the vertices that need to be visited. The queue follows the First In First Out (FIFO) principle.

The traversal starts from a selected vertex, which is marked as **visited** and inserted into the queue. This ensures that the same vertex is not visited again.

The front vertex is removed from the queue and all its **unvisited adjacent vertices** are visited and added to the queue. This process is repeated until the queue becomes empty.

Example:
Consider a graph with vertices A, B, C, D, and E.
Edges: A–B, A–C, B–D, C–E.
Starting BFS from A, the traversal order is:
A → B → C → D → E

BFS is commonly used to find the **shortest path** in unweighted graphs and in level-order traversal

4. **Explain DFS traversal with example.**

Depth First Search (DFS) is a graph traversal technique that explores a graph by going **as deep as possible** along one path before backtracking. It focuses on depth rather than levels.

DFS uses a **stack or recursion** to remember the previously visited vertices. This helps the algorithm return to earlier vertices when needed.

The traversal begins at a chosen vertex, which is marked as **visited** and processed. The algorithm then moves to an unvisited adjacent vertex.

DFS continues visiting vertices deeply until no unvisited adjacent vertex is left. After that, it **backtracks** to explore other paths.

Example:
Using the same graph with vertices A, B, C, D, and E,
Starting DFS from A, one possible traversal order is:
A → B → D → C → E

DFS is widely used in **cycle detection**, **topological sorting**, and path-finding problems.

5. Compare adjacency matrix and adjacency list.

Basis	Adjacency Matrix	Adjacency List
Representation	Uses a 2-D array to store graph data. Rows and columns represent vertices.	Uses an array of lists where each list stores neighbors of a vertex.
Memory usage	Requires more memory because space is allocated for all vertex pairs.	Uses less memory because only existing edges are stored.
Edge checking	Checking an edge is very fast using matrix indexing.	Edge checking is slower because the list must be searched.
Best suited for	Suitable for dense graphs with many edges.	Suitable for sparse graphs with fewer edges.
Implementation	Simple to understand and implement.	Slightly complex but more efficient.

6] Explain applications of graph in real-world problems.

Graphs are used in **social networks** to represent users as vertices and friendships as edges. This helps in finding mutual friends and recommendations.

In **road maps and navigation systems**, cities are vertices and roads are edges. Graphs help in finding the shortest path between locations.

Graphs are widely used in **computer networks** to represent devices and connections. They help in routing data efficiently.

In **project management**, graphs are used to represent tasks and their dependencies. This helps in scheduling and planning.

Graphs are also used in **search engines** to rank web pages. Links between pages are treated as edges.

7] Explain directed and undirected graphs.

Directed Graph

- A directed graph is a graph in which edges have a specific direction. Each edge goes from one vertex to another.
- The direction shows one-way relationships, such as one-way roads or following someone on social media.
- In directed graphs, an edge from A to B does not mean there is an edge from B to A.

Undirected Graph

- An undirected graph is a graph in which edges have no direction. The connection works in both directions.
- If there is an edge between A and B, then both A can reach B and B can reach A.
- Undirected graphs are used to represent mutual relationships, such as friendship networks or two-way roads.

[10-marks]

1. **Explain graph data structure. Describe its representation techniques with examples.**

- A graph is a non-linear data structure used to represent relationships between different elements. It is made up of vertices (nodes) and edges (connections) that show how the vertices are related to each other.
- Vertices represent objects or entities such as cities, users, or computers. Each vertex stores data and acts as a point in the graph where connections begin or end.
- Edges represent the relationship or link between two vertices. An edge may show a road between cities, a friendship between users, or a cable between computers.
- Graphs can be directed or undirected depending on the direction of edges. In a directed graph, edges have a direction, while in an undirected graph, edges allow movement in both directions.
- Graphs can also be weighted or unweighted. In weighted graphs, each edge has a value like cost, distance, or time, which is useful in real-world problems.

Representation Techniques of Graph

Adjacency Matrix Representation stores a graph using a two-dimensional array. Rows and columns represent vertices, and the value stored shows whether an edge exists or not.

- If there is an edge between vertex *i* and vertex *j*, the matrix value is 1, otherwise it is 0. For a graph with *n* vertices, the matrix size is always $n \times n$.
- Example:
  Vertices: A, B, C
  Edges: A–B, B–C
  The adjacency matrix clearly shows these connections using rows and columns.
- Adjacency matrix allows fast edge checking, but it uses more memory. It is not suitable for graphs with fewer edges.

Adjacency List Representation stores a graph using an array of lists. Each vertex has a list of vertices that are directly connected to it.

- This method stores only existing edges, which saves memory. It is very efficient for sparse graphs with fewer connections.
- Example:
  A → B
  B → A, C
  C → B
  This representation clearly shows neighbors of each vertex.
- Adjacency list is commonly used in graph traversal algorithms like BFS and DFS. It is flexible and efficient for real-world applications.

2. **Explain Breadth First Search (BFS) and Depth First Search (DFS) algorithms with examples.**

Breadth First Search (BFS)

- Breadth First Search (BFS) is a graph traversal algorithm that visits vertices level by level starting from a given source vertex. It first visits all immediate neighbors before moving to the next level of vertices.
- BFS uses a queue data structure to store vertices that are to be visited next. This ensures that vertices are processed in the order in which they are discovered.
- In BFS, the starting vertex is marked as visited and inserted into the queue. This prevents the same vertex from being visited multiple times.
- The algorithm removes a vertex from the front of the queue and visits it. Then, all its unvisited adjacent vertices are marked as visited and added to the queue.
- This process continues until the queue becomes empty. BFS ensures that vertices closest to the starting vertex are visited first.
- Example:
  Consider a graph with vertices A, B, C, D, and E.
  Edges: A–B, A–C, B–D, C–E.
  Starting BFS from A, the traversal order will be:
  A → B → C → D → E
- BFS is commonly used to find the shortest path in an unweighted graph and in level-order traversal.

Depth First Search (DFS)

- Depth First Search (DFS) is a graph traversal algorithm that explores a graph by going as deep as possible along one path before backtracking. It focuses on depth rather than breadth.
- DFS uses a stack or recursion to remember previously visited vertices. This helps the algorithm return to earlier vertices when needed.
- The traversal starts from a chosen vertex, which is marked as visited and processed. Then, an unvisited adjacent vertex is selected.
- DFS continues visiting vertices deeply until no unvisited adjacent vertex is left. After that, the algorithm backtracks to explore other paths.
- This process repeats until all vertices of the graph are visited. DFS is useful for exploring all possible paths.
- Example:
  Using the same graph with vertices A, B, C, D, and E,
  One possible DFS traversal starting from A is:
  A → B → D → C → E
- DFS is widely used in cycle detection, topological sorting, and solving puzzles like mazes.

3. **Compare BFS and DFS based on traversal, data structures used, and applications.**

Basis	BFS (Breadth First Search)	DFS (Depth First Search)
Traversal	BFS traverses the graph level by level, visiting all adjacent vertices before moving to the next level. It explores nodes closer to the source first.	DFS traverses the graph by going deep along one path before backtracking. It explores one branch completely before moving to another.
Data structure used	BFS uses a queue to store vertices during traversal. The queue follows the FIFO (First In First Out) principle.	DFS uses a stack or recursion to store vertices during traversal. It follows the LIFO (Last In First Out) principle.
Applications	BFS is used to find the shortest path in an unweighted graph and in level-order traversal. It is also used in network broadcasting.	DFS is used in cycle detection, topological sorting, and solving problems like mazes. It is useful where deep exploration is required.

4. **Explain graph traversal techniques and their applications in computer networks and social networks.**

Explain Graph Traversal Techniques and Their Applications in Computer Networks and Social Networks

- Graph traversal is the process of visiting all the vertices of a graph in a systematic way. It helps in exploring and processing all nodes and edges of a graph.
- The two main graph traversal techniques are Breadth First Search (BFS) and Depth First Search (DFS). Both techniques ensure that all reachable vertices of a graph are visited.

Breadth First Search (BFS)

- BFS traversal visits vertices level by level, starting from a given source vertex. It first visits all immediate neighbors before moving to the next level.
- BFS uses a queue data structure to store vertices that need to be visited next. This ensures that vertices are visited in the order they are discovered.
- In computer networks, BFS is used for routing and broadcasting. It helps in finding the shortest path between two computers or routers in a network.
- In social networks, BFS is used to find shortest connections such as mutual friends or degrees of separation between users. It is also used in friend suggestion systems.

Depth First Search (DFS)

- DFS traversal explores a graph by going deep into one path before backtracking. It visits a vertex and then continues to an unvisited adjacent vertex.
- DFS uses a stack or recursion to keep track of visited vertices. This helps in backtracking when no further unvisited vertices are available.
- In computer networks, DFS is used for network topology discovery and cycle detection. It helps in checking whether loops exist in network connections.
- In social networks, DFS is used to explore communities or groups. It is useful in detecting connected components and analyzing user clusters.

5. **Explain adjacency matrix and adjacency list representations with advantages and disadvantages.**

Adjacency Matrix Representation

- Adjacency matrix is a method of representing a graph using a two-dimensional array. Rows and columns of the matrix represent the vertices of the graph.
- If there is an edge between two vertices, the corresponding cell value is 1, otherwise it is 0. In weighted graphs, the cell may store the weight instead of 1.
- For a graph with n vertices, the size of the adjacency matrix is always n × n. This makes the structure simple and easy to understand.

Advantages of Adjacency Matrix

- It allows very fast checking of whether an edge exists between two vertices. Accessing a matrix cell takes constant time.
- The representation is simple to implement and understand. It is suitable for small and dense graphs.

Disadvantages of Adjacency Matrix

- It requires large memory space, even if very few edges are present. This makes it inefficient for sparse graphs.
- Traversing all neighbors of a vertex takes more time because the entire row must be scanned.

Adjacency List Representation

- Adjacency list represents a graph using an array of lists. Each vertex stores a list of vertices directly connected to it.
- This method stores only existing edges, which makes it more memory efficient. It is widely used in real-world applications.
- In undirected graphs, each edge is stored in the lists of both connected vertices. In directed graphs, only outgoing edges are stored.

Advantages of Adjacency List

- It uses less memory compared to adjacency matrix. It is ideal for sparse graphs.
- Traversing adjacent vertices is efficient and fast. It works well with graph traversal algorithms like BFS and DFS.

Disadvantages of Adjacency List

- Checking whether a specific edge exists takes more time. The list of neighbors must be searched.
- Implementation is slightly more complex than adjacency matrix.

6. **Define graph. Explain Graph storage structure. Perform depth first search (DFS) and Breadth First Search for any graph.**

Definition of Graph

- A graph is a non-linear data structure used to represent relationships between different entities. It consists of a set of vertices (nodes) and edges (links) that connect these vertices.
- Vertices represent objects such as cities, users, or computers, while edges represent the relationship or connection between them. Graphs are widely used in computer networks, social networks, maps, and scheduling problems.
- A graph can be directed or undirected depending on whether edges have direction. It can also be weighted or unweighted based on whether edges have associated values.

Graph Storage Structures

1. Adjacency Matrix

- Adjacency matrix represents a graph using a two-dimensional array. Rows and columns represent vertices of the graph.
- If there is an edge between vertex *i* and vertex *j*, the matrix value is 1, otherwise it is 0. In weighted graphs, the value represents the weight.
- This method is easy to understand and allows fast edge checking. However, it consumes more memory and is not suitable for sparse graphs.

2. Adjacency List

- Adjacency list represents a graph using an array of lists. Each vertex stores a list of vertices connected to it.
- This method stores only existing edges, so it uses less memory. It is efficient for sparse graphs and commonly used in graph algorithms.

Example Graph (for DFS and BFS)

Vertices: A, B, C, D, E
Edges: A–B, A–C, B–D, C–E

Depth First Search (DFS)

- DFS is a traversal technique that explores the graph by going as deep as possible before backtracking.
- It uses a stack or recursion to keep track of visited vertices.
- DFS Traversal (starting from A):
  A → B → D → C → E
- DFS is useful in cycle detection, topological sorting, and exploring connected components.

Breadth First Search (BFS)

- BFS is a traversal technique that visits vertices level by level, starting from a source vertex.
- It uses a queue to store vertices to be visited next.
- BFS Traversal (starting from A):
  A → B → C → D → E
- BFS is useful for finding the shortest path in unweighted graphs and network broadcasting.