DATA STRUCTURES

Topic Name
INTRODUCTION OF GRAPH DATA STRUCTURE
Instructor:
Zainab Malik

Content

- Introduction to Graph Data Structure
- Graphical Representation using adjacency matrix and adjacency list.
- Graph Traversal (Breadth-First Search and depth First Search)

## Introduction to Graph Data Structure.

- Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs).

- Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook.

- **For example,** in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, etc.

Applications Of Graph

- Google Maps, Apple Maps, Waze use graphs to treat all cities and places as nodes and routes between them as edges.

- Webgraph describes a directed graph between pages of the WWW. Each page is a vertex and the hyperlinks are edges. This is the basic idea behind the Google Page Ranking Algorithm.

- Uber, Ola, Lyft uses a graph to find the shortest and cheapest path for a car from one city to another.

- The graph also used in a database for representing Entity-Relationship.

- Graph theory also used to study molecules in chemistry and physics.

Example of Graph Data Structure

- The following is a graph with 5 vertices and 7 edges.
- This graph G can be defined as G = (V , E)
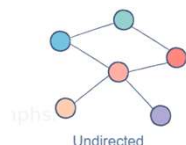Where V = {A,B,C,D,E} and E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.
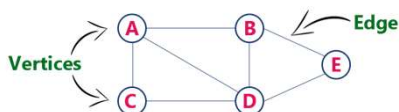
Types of graphs

- Undirected Graphs
- Directed Graphs

Undirected Graph:

In an undirected graph, nodes are connected by edges that are all bidirectional. For example if an edge connects node 1 and 2, we can traverse from node 1 to node 2, and from node 2 to 1.

Undirected

Directed Graph

• In a directed graph, nodes are connected by directed edges – they only go in one direction. For example, if an edge connects node 1 and 2, but the arrow head points towards 2, we can only traverse from node 1 to node 2 – not in the opposite direction.

Directed
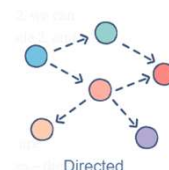
Graph Representations

• Graph data structure is represented using following representations.
1. Adjacency Matrix
2. Adjacency List

Types of Graph Representations:

• Adjacency Matrix:
• An Adjacency Matrix is a 2D array of size V x V where V is the number of nodes in a graph. A slot matrix[i][j] = 1 indicates that there is an edge from node i to node j.

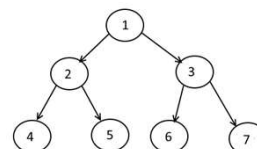|	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0
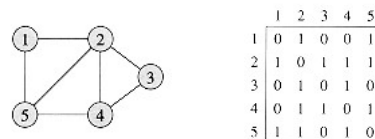
Adjacency List

• To create an Adjacency list, an array of lists is used. The size of the array is equal to the number of nodes.
• A single index, array[i] represents the list of nodes adjacent to the ith node.

Example:

Tree is a type of a graph.
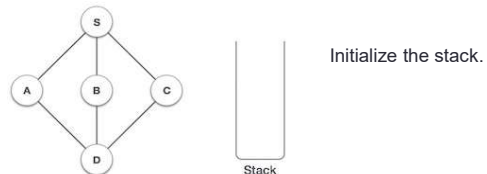For practice create its Adjacency Matrix and Adjacency List

---

13

Depth First Search (DFS) algorithm

- Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember the next vertex to continue search, when a dead end occurs in any iteration.

- *Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.*
- *Rule 2 – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)*
- *Rule 3 – Repeat Rule 1 and Rule 2 until the stack is empty.*
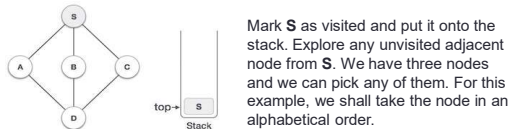

14

Example

- In this Example we are discussing about the Depth First Search(DFS) Algorithm Step by Step.
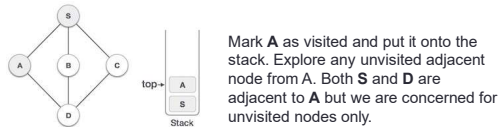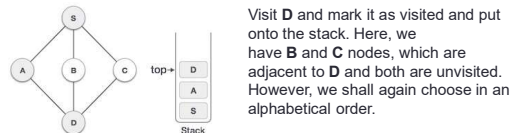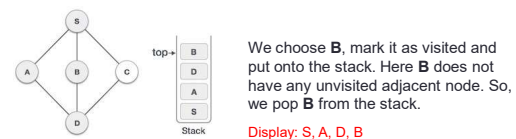- In the Depth First Search(DFS) algorithm we are using a Stack Data Structure.

Step 1:

Initialize the stack.

15

- **Step 2:**

Mark **S** as visited and put it onto the stack. Explore any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.

Display: S

Step 3:

Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only.

Display: S, A

16

- **Step 4:**

Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order.

Display: S, A, D

Step 5:

We choose **B**, mark it as visited and put onto the stack. Here **B** does not have any unvisited adjacent node. So, we pop **B** from the stack.

Display: S, A, D, B

17

- **Step 6:**

We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find **D** to be on the top of the stack.

Display: S, A, D, B

Step 7:

Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

Display: S, A, D, B, C

18

- **Step 8:**
- Repeat the step no 5 and pop up the all alphabets one by one from the stack.

- As **C** does not have any unvisited adjacent node so we keep popping the stack until we find a node that has an unvisited adjacent node. In this case, there's none and we keep popping until the stack is empty.

Display: S, A, D, B, C

19

Breadth First Search (BFS) algorithm

- Breadth First Search (BFS) algorithm traverses a graph in a *breadthward* motion and uses a queue to remember the next vertex to continue search, when a dead end occurs in any iteration.

- *Rule 1* – *Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.*
- *Rule 2* – *If no adjacent vertex is found, remove the first vertex from the queue.*
- *Rule 3* – *Repeat Rule 1 and Rule 2 until the queue is empty.*
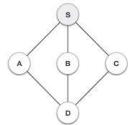

20

Example

- In this Example we are discussing about the Breadth First Search (BFS) Algorithm Step by Step.
- In the Breadth First Search (BFS) algorithm we are using a Queue Data Structure.

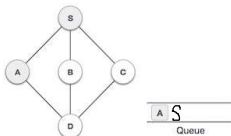- **Step 01:**
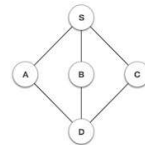
Initialize the queue.

21

- **Step 02:**

We start from visiting **S** (starting node), and mark it as visited.
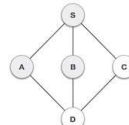
Display: S

Step 03:

We then see an unvisited adjacent node from **S**. In this example, we have three nodes but alphabetically we choose **A**, mark it as visited and enqueue it.
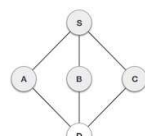
Display: S, A

22

- **Step 04:**

Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it.

Display: S, A, B

Step 05:

Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it.
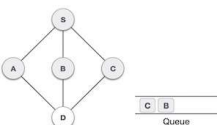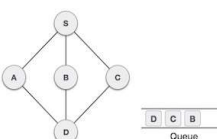
Display: S, A, B, C

23

- **Step 06:**

Now, **S** is left with no unvisited adjacent nodes. So, we dequeue and find **A**.

Display: S, A, B, C

Step 07:

From **A** we have **D** as unvisited adjacent node. We mark it as visited and enqueue it.

Display: S, A, B, C, D

24

- At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

Display: S, A, B, C, D

25

Thank You