Unit - 6

Graph

Que- ~~Explain BFS with example and computational complexity.~~

⇒

Que- What is Graph with undirected Graph and pirected Graph and Explain Graph Terminology.

⇒

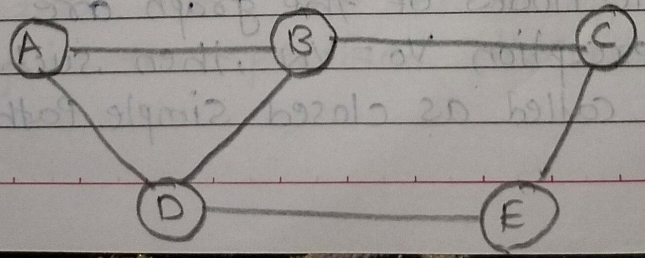
Graph :-

- A graph can be defined as group of vertices and edges that are used to connect these vertices.
- A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having Parent child relationship.

- A graph G can be defined as an ordered set G(V,E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.

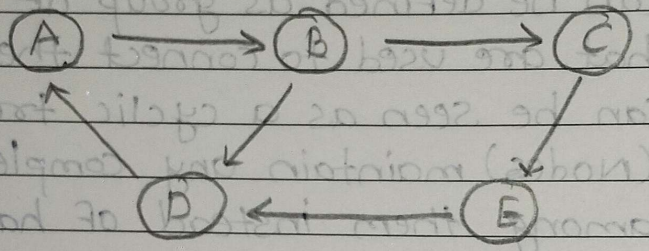
• Undirected Graph :-

- In an undirected graph, edges are not associated with the directions with them.
- An undirected graph its edges are not attached with any of the directions.
- If an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B.



Undirected graph

• Directed graph :-
- In a directed graph, edges from an ordered pair.
- Edges represent a specific path from some vertex A to another vertex B.
- Node A is called initial node while node B is called terminal node.



Directed graph

• Graph Terminology :-

1] Path :-
A path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the initial node U.

2] Closed Path :-
A path will be called as closed path if the initial node is same as terminal node. A path will be closed path if $V_0 = V_N$

3] Simple Path :-
- If all the nodes of the graph are distinct with an exception $V_0 = V_N$, then such that path P is called as closed simple path.

4] Cycle:-
-A cycle can be defined as the path which has no repeated edges or vertices except the first and last vertices.

5] Connected graph:-
A connected graph is the one in which some path exists between every two vertices (u,v) in V. There are no isolated nodes in connected graph.

6] Complete Graph:-
-A complete graph is the one in which every node is connected with all other nodes.
-A complete graph contain n(n-1)/2 edges where n is the number of nodes in the graph.

7] weighted Graph:-
-In a weighted graph, each edge is assigned with some data such as length or weight.
-The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.

8] Digraph:-
A digraph is a directed graph in which each edge of the graph is associated with some direction and the traversing can be done only in the specified direction.

9) loop :-
- An edge that is associated with the similar end points can be called as Loop.

10) Adjacent Nodes :-
- If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbours or adjacent nodes.

11) Degree of the Node :-
- A degree of a node is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.

Traversal technique

Que- Explain Breadth-first search (BFS) with example and computational complexity.

⇒

BFS Algorithm:-

- Breadth-First search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes.
- Then, it selects the nearest node and explores all the unexplored nodes.
- While using BFS for traversal, any node in the graph can be considered as the root node.
- There are many ways to traverse the graph, but among them, BFS is the most commonly used approach.
- It is a recursive algorithm to search all the vertices of a tree or graph data structure.
- BFS puts every vertex of the graph into two categories - visited and non-visited.
- It selects a single node in a graph and after that, visits all the nodes adjacent to the selected node.

• Applications of BFS algorithm:-

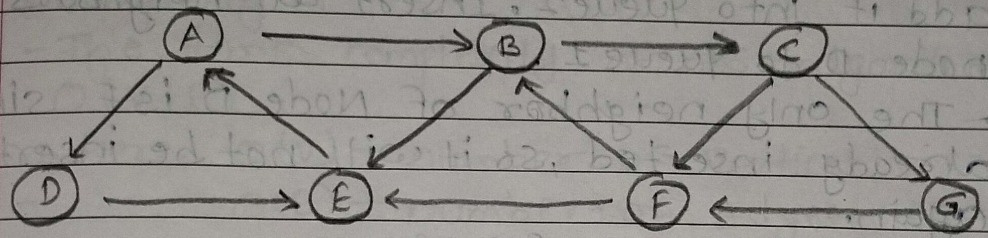
1. BFS can be used to find the neighboring location from a given source location.
2. In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes.

3] BFS can be used in web crawlers to create web page indexes.
4] BFS is used to determine the shortest Path and minimum spanning tree.
5] BFS is also used in cheney's technique to duplicate the garbage collection.
6] It can be used in Ford-Fulkerson method to compute the maximum Flow in a flow network.

• Algorithm :-
The steps involved in the BFS algorithm to explore a graph are given as follows -

Step 1 :- SET STATUS = 1 (ready state) For each node in G
Step 2 :- Enqueue the starting node A and set its STATUS = 2 (waiting state)
Step 3 :- Repeat steps 4 and 5 until QUEUE is empty.
Step 4 :- Dequeue a Node N. Process it and set its STATUS = 3 (Processed state)
Step 5 :- Enqueue 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

Example:-



- Directed graph having 7 vertices.

two queues - QUEUE 1 and QUEUE 2
- QUEUE 1 holds all the nodes that are to be processed
- QUEUE 2 holds all the nodes that are processed and deleted from QUEUE 1.

Adjacency List
A: B, D
B: C, E
C: E, G
G: E
E: B, F
F: A
D: F

Step 1:- First, add A to queue 1 and NULL to queue 2
∴ QUEUE 1 = {A}
QUEUE 2 = {NULL}

Step 2:- Now, delete node A from queue 1 and add it into queue 2. Insert all neighbors of node A to queue 1.
∴ QUEUE 1 = {B, D}
QUEUE 2 = {A}

Step 3:- Now, delete node B from queue 1 and add it into queue 2. Insert all neighbors of node B to queue 1.
∴ QUEUE 1 = {D, C, F}
QUEUE 2 = {A, B}

Step4:- Now, delete node D from queue 1 and add it into queue2. Insert all neighbors of node D to queue 1.
- The only neighbor of Node D is F since it already inserted, so it will not be inserted again.

∴ QUEUE 1 = {C, F}
QUEUE 2 = {A, B, D}

Step 5:- Delete node C from queue 1 and add it into queue 2. Insert all neighbors of node C to queue 1.

∴ QUEUE1 = {F, E, G}
QUEUE2 = {A, B, D, C}

Step 6:- Delete node F from queue 1 and add it into queue 2. Insert all neighbors of node F to queue 1.
- since all the neighbors of node F are already present, we will not insert them again.

∴ QUEUE 1 = {E, G}
QUEUE2 = {A, B, D, C, F}

Step7:- Delete node E from queue1. Since all of its neighbors have already been added, so we will not insert them again.
- Now all the nodes are visited and the target node E is encountered into queue 2.

∴ QUEUE 1 = {G}
∴ QUEUE 2 = {A, B, D, C, F, E}

- Complexity :-

- The time complexity of BFS algorithm is $O(V+E)$

where, $O(V)$ - number of vertices
$O(E)$ - number of Edges

- Space complexity of BFS can be expressed as $O(V)$, where V is the number of vertices.

Que- Explain Depth-first search (DFS) with example and computational complexity.-

⇒

DFS Algorithm:-

- It is a recursive algorithm to search all the vertices of a tree data structure or a graph.
- The depth-First search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.
- Because of the recursive nature, stack data structure can be used to implement the DFS algorithm.
- The process of implementing the DFS is similar to the BFS algorithm.

- The step by step process to implement the DFS traversal is given as follows-

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and Push that vertex into the stack.
3. After that, Push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex From the stack.
6. Repeat steps 2, 3 and 4 until the stack is empty.

D D M M Y Y Y Y

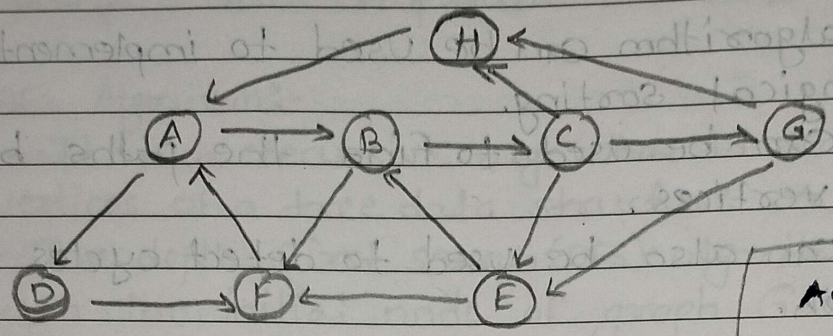
• Applications of DFS algorithm :-

1) DFS algorithm can be used to implement the topological sorting.
2) It can be used to find the paths between two vertices.
3) It can also be used to detect cycles in the graph.
4) DFS algorithm is also used for one solution puzzles.
5) DFS is used to determine if a graph is biparitite or not.

• Algorithm :-

Step 1 :- SET STATUS = 1 (ready state) for each node in G.
Step 2 :- Push the starting node A on the stack and set its STATUS = 2 (waiting state)
Step 3 :- Repeat steps 4 and 5 until STACK is empty
Step 4 :- Pop the top Node N. Process it and set its 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

• Example:-



Adjacency List

A : B, D
B : C, F
C : E, G, H
G : E, H
E : B, F
F : A
D : F
H : A

- Directed graph having 7 vertices.

Step1:- First, Push H onto the stack.

∴ STACK = H

Step2: Pop the top element from the stack i.e. H, and Print it. Now, PUSH all the neighbors of H onto the stack that are in ready state.

∴ Print: H

∴ STACK : A

Step3:- Pop the top element from the stack i.e. A and Print it. Now, PUSH all the neighbors OF A onto the stack that are in ready state.

∴ Print: A

∴ STACK: B, D

Step4:- Pop the top element from the stack i.e. D and Print it. Now PUSH all the

neighbors of D onto the stack that are in ready state.

∴ Print: D

STACK: B, F

Step 5:- Pop the top element from the stack i.e. F and Print it. Now, PUSH all the neighbors of F onto the stack that are in ready state.

∴ Print: F

STACK: B

Step 6:- Pop the top element from the stack i.e. B and Print it. Now, PUSH all the neighbors of B onto the stack that are in ready state.

∴ Print: B

STACK: C

Step 7:- Pop the top element from the stack, i.e. C and Print it. Now, PUSH all the neighbors of C onto the stack that are in ready state.

∴ Print: C

STACK: E, G

Step 8:- Pop the top element from the stack. i.e. G and PUSH all the neighbors of G onto the stack that are in ready state.

∴ Print: G

STACK: E

Step 9:- Pop the top element from the stack i.e. E and PUSH all the neighbors of E onto the stack that are in ready state.

∴ Print : E

STACK : empty.

- Complexity of DFS :-

- The time complexity of the DFS algorithm is O(V+E)

  where, V = number of vertices

  E = number of edges

- The space complexity of DFS algorithm is O(V).

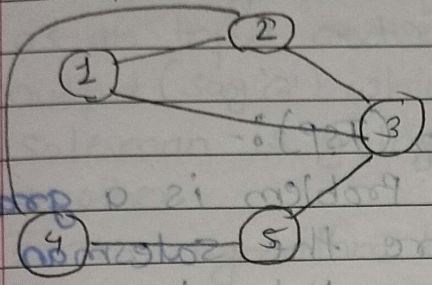
* Que- Difference between Adjacency matrix and Adjacency List.

⇒

Adjacency matrix	Adjacency List
1) This representation makes use of V x V matrix, so space required in worst case is $O(\|V\|^2)$	1) In this representation, For every vertex we store its neighbours. - overall space complexity is $O(\|V\|+\|E\|)$
2) In order to add a new vertex to V×V matrix the storage must be increases to $(\|V\|+1)^2$. To achieve this we need to copy the whole matrix. Therefore the complexity is $O(\|V\|^2)$.	2) There are two Pointers in adjacency list First Points to the front node and the other one Points to the rear node. Thus insertion of a vertex can be done directly in $O(1)$ time.
3) To add an edge say from i to j, matrix [i][j]=1 which requires $O(1)$ time.	3) similar to insertion of vertex here also two Pointers are used Pointing to the rear and Front of the list. Thus, an edge can be inserted in $O(1)$ time.
4) In order to remove a vertex from V * V matrix	4) In order to remove a vertex, we need to search

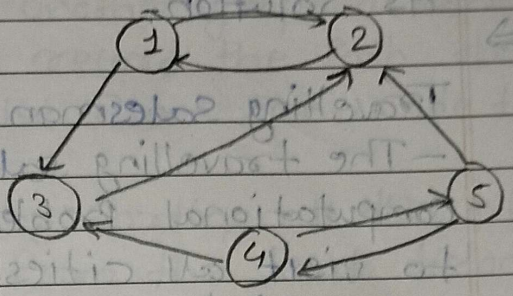
the storage must be decreased to $\|V\|^2$ from $(\|V\|+1)^2$. To achieve this we need to copy the whole matrix. Therefore the complexity is $O(\|V\|^2)$.	for the vertex which will require $O(\|V\|)$ time in worst case, after this we need to traverse the edges and in worst case it will require $O(\|E\|)$ time. Hence, total time complexity is $O(\|V\|+\|E\|)$.
5] To remove an edge say from i to j, matrix [i][j] = 0 which requires $O(1)$ time.	5] To remove an edge traversing through the edges is required and in worst case we need to traverse through all the edges. Thus, the time complexity is $O(\|E\|)$.
6] In order to find for an existing edge the content of matrix needs to be checked. Given two vertices say i and j matrix [i][j] can be checked in $O(1)$ time.	6] In an adjacency list every vertex is associated with a list of adjacent vertices. For a given graph, in order to check for an edge we need to check for vertices adjacent to given vertex. A vertex can have at most $O(\|V\|)$ neighbours and in worst can we would have to check for every adjacent vertex. Therefore, time complexity is $O(\|V\|)$.

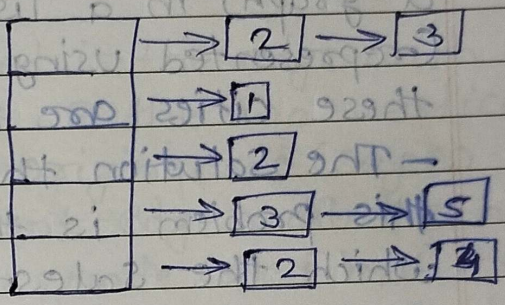
7) Example:-



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

7) Example:-





Que- Explain TSP (Travelling salesman Problem) and its solution.

⇒

Travelling Salesman Problem (TSP):-

- The travelling salesman Problem is a graph computational Problem where the salesman needs to visit all cities (represented using nodes in a graph) in a list just once and the distances (represented using edges in the graph) between all these cities are known.
- The solution that is needed to be found for this problem is the shortest possible route in which the salesman visits all the cities and returns to the origin city.
- There are various approaches to find the solution to the travelling salesman Problem:- naive approach, greedy approach, dynamic programming approach etc.

• Travelling Salesman Problem using greedy approach:-

- The inputs taken by the algorithm are the graph G{V, E}, where V is the set of vertices and E is the set of edges.
- The shortest Path of graph G starting from one vertex returning to the same vertex is obtained as the output.

- Algorithm:-

1] Travelling salesman problem takes a graph G{V,E} as an input and declare another graph as the output (say 'G') which will record the path the salesman is going to take from one node to another.

2] The algorithm begins by sorting all the edges in the input graph G from the least distance to the largest distance.

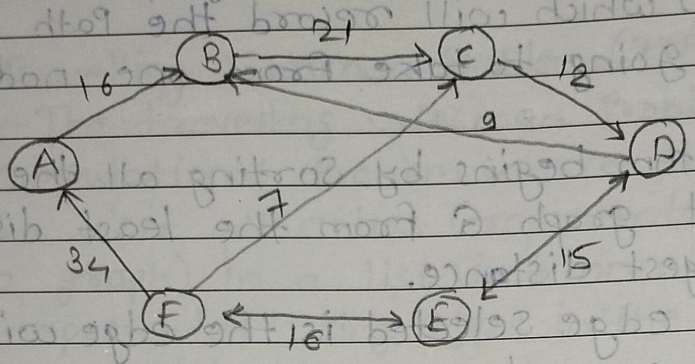
3] The first edge selected is the edge with least distance, and one of the two vertices (say A and B) being the origin node (say A)

4] Then among the adjacent edges of the node other than the origin node (B), find the least cost edge and add it onto the output graph.

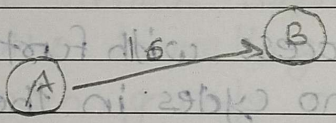
5] Continue the process with further nodes making sure there are no cycles in the output graph and the path reaches back to the origin node A.

6] However, if the origin is mentioned in the given problem, then the solution must always start from that node only.

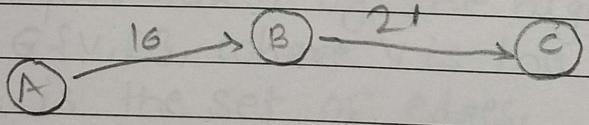
Example:
Consider the following graph with six cities and the distances between them -



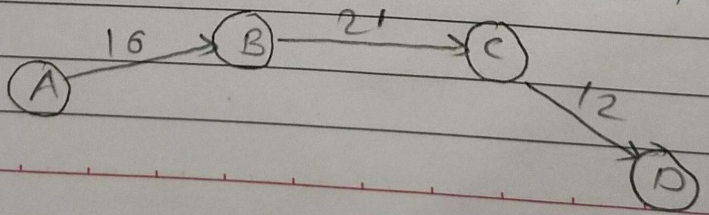
From the given graph, since the origin is already mentioned, the solution must always start from that node. Among the edges leading from A, A → B has the shortest distance.



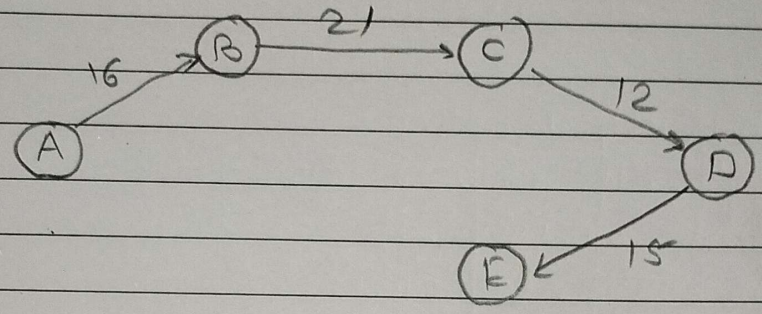
Then, B → C has the shortest and only edge between, therefore it is included in the output graph.



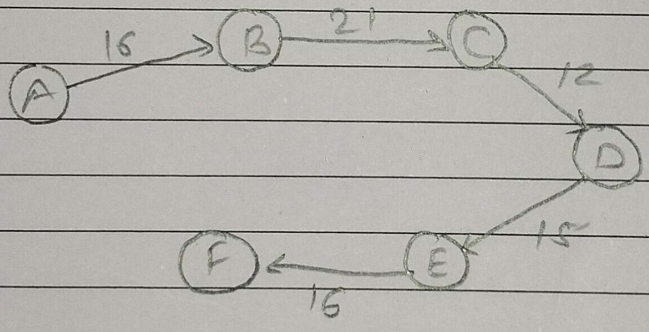
There's only one edge between C → D, therefore it is added to the output graph.



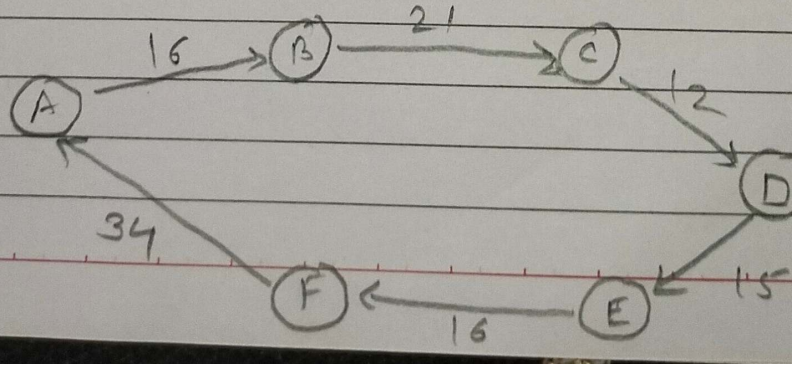
There's two outward edges from D. Even though D → B has lower distance than D → E, B is already visited once and it would from a cycle if added to the output graph. Therefore, D → E is added into the output graph.



There's only one edge from e, that is E → f. Therefore it is added into the output graph.



Again even though F → c has lower distance than F → A, F → A is added into the output graph in order to avoid the cycle that would from and c is already visited once.



The shortest path that originates and ends at A is $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow A$

The cost of the path is:

$16 + 21 + 12 + 15 + 16 + 34 = 114$