**SYLLABUS**

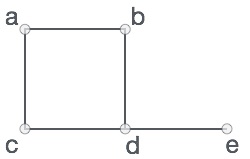
**Module 4**

Advanced Graph Structures : Representation of graphs, Depth First and Breadth First Traversals, Topological Sorting, Strongly connected Components and Biconnected Components Minimum Cost Spanning Tree algorithms- Prim’s Algorithm, Kruskal’ Algorithm,. Shortest Path Finding algorithms – Dijikstra’s single source shortest paths algorithm

**ADVANCED GRAPH STRUCTURES**

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.

Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices. Take a look at the following graph −



In the above graph,

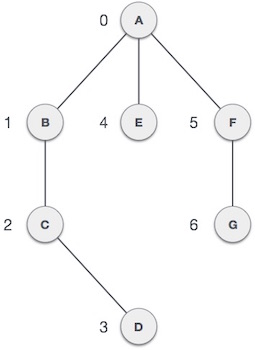
V = {a, b, c, d, e}

E = {ab, ac, bd, cd, de}

**GRAPH DATA STRUCTURE**

Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms −

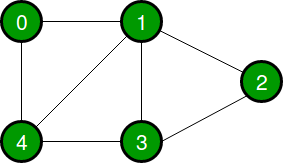
* **Vertex** − Each node of the graph is represented as a vertex. In the following example, the labeled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.
* **Edge** − Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.
* **Adjacency** − Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.
* **Path** − Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.



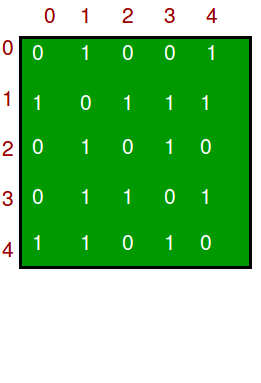
**REPRESENTATION OF GRAPHS**

The following two are the most commonly used representations of a graph.   
**1.** Adjacency Matrix   
**2.** Adjacency List

**ADJACENCY MATRIX:**  
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

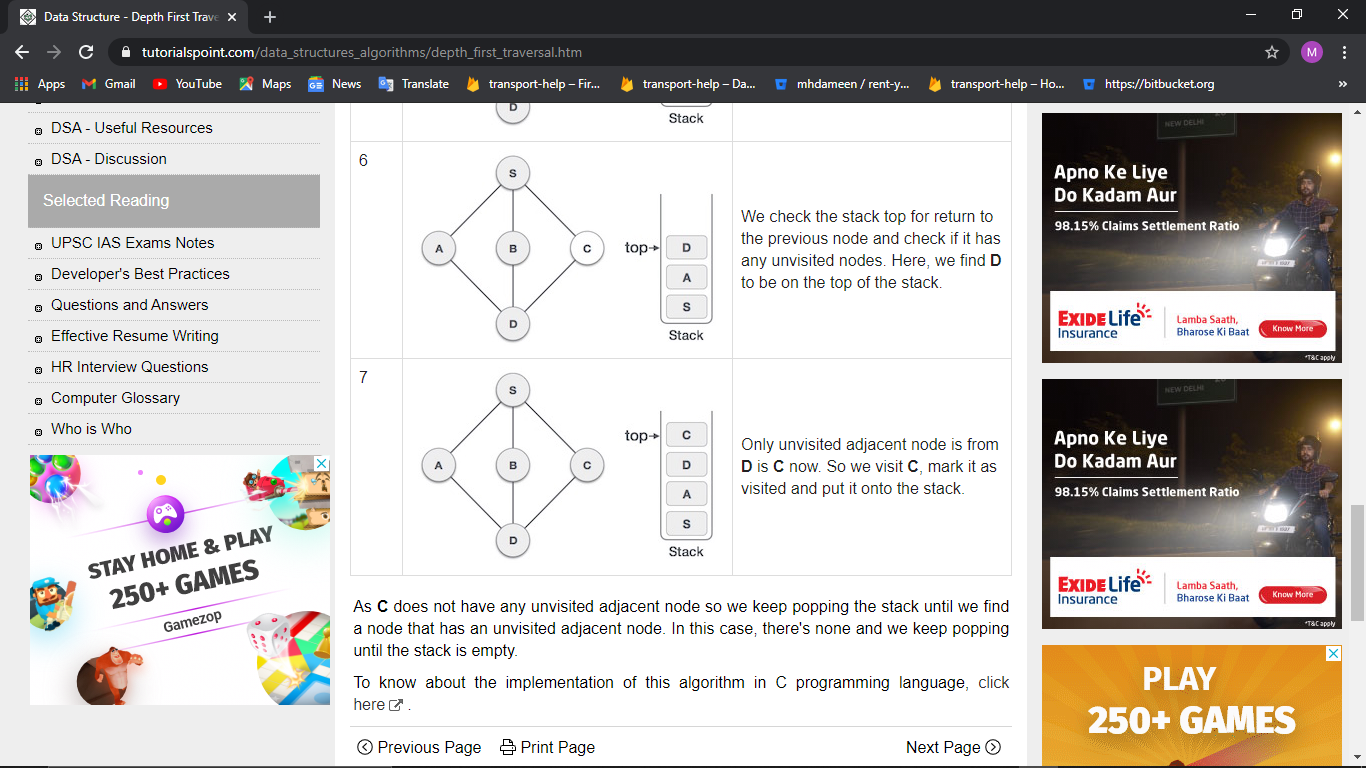
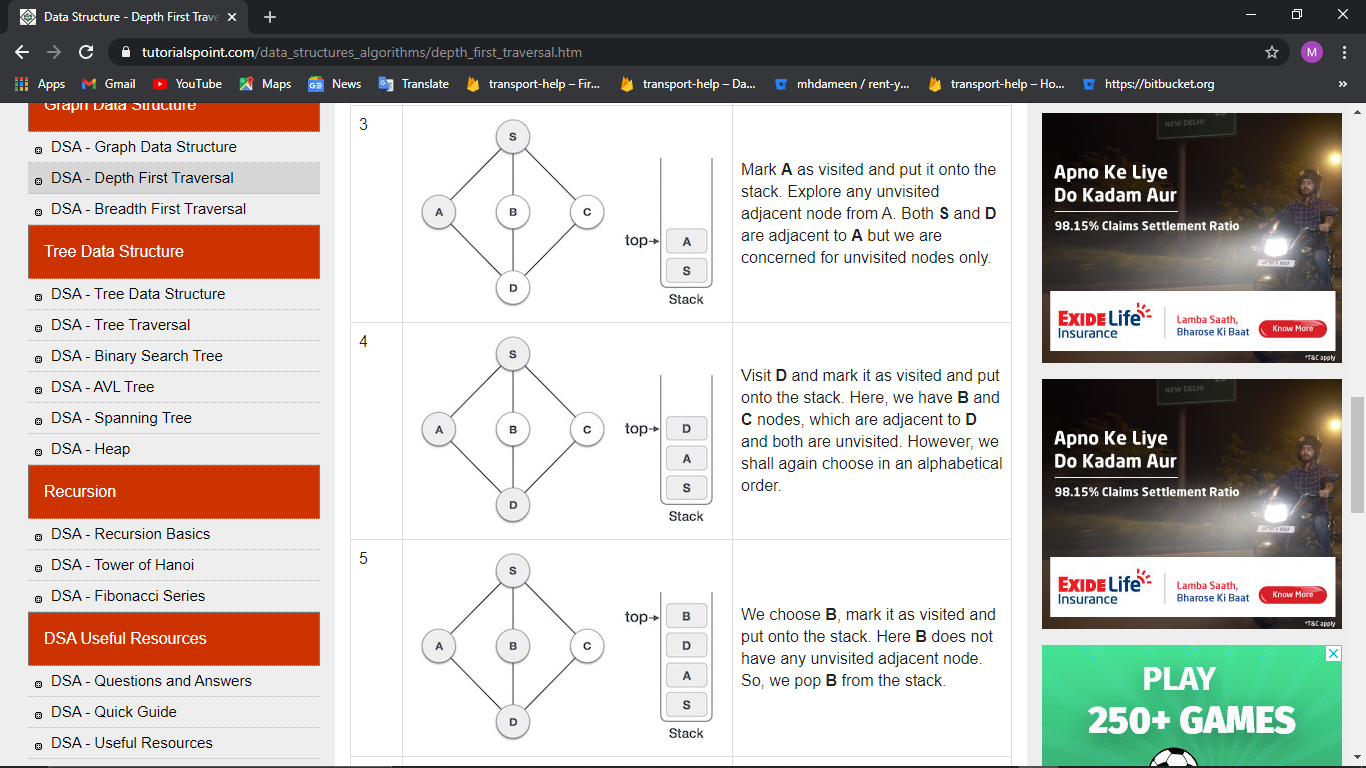
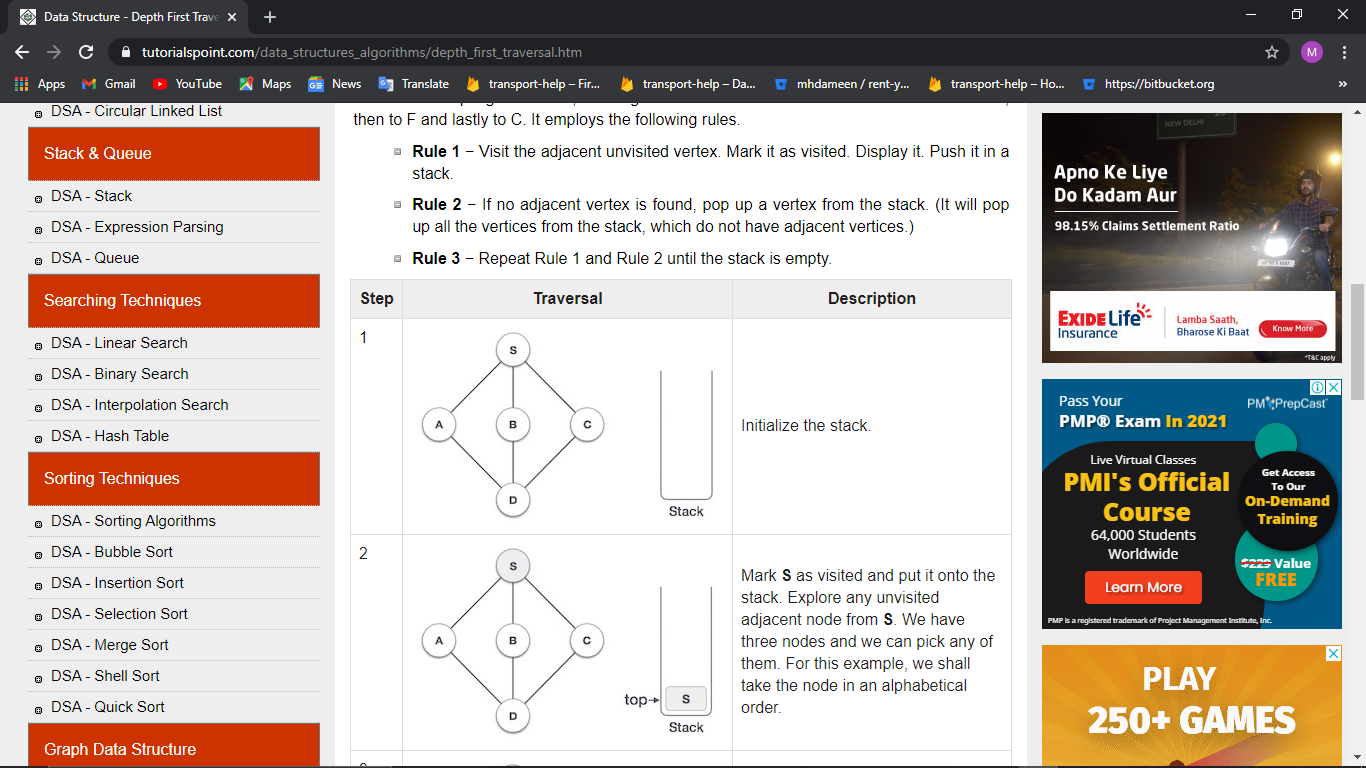


The adjacency matrix for the above example graph is:

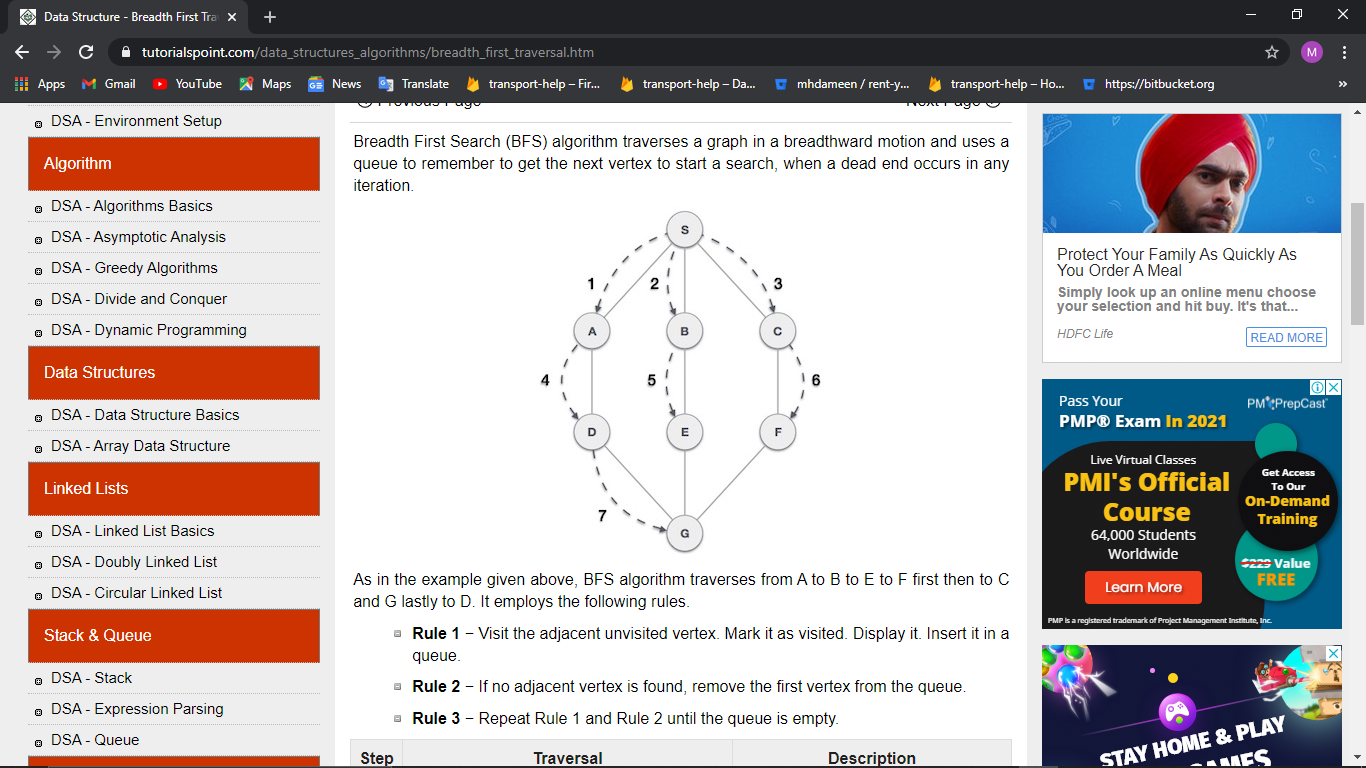


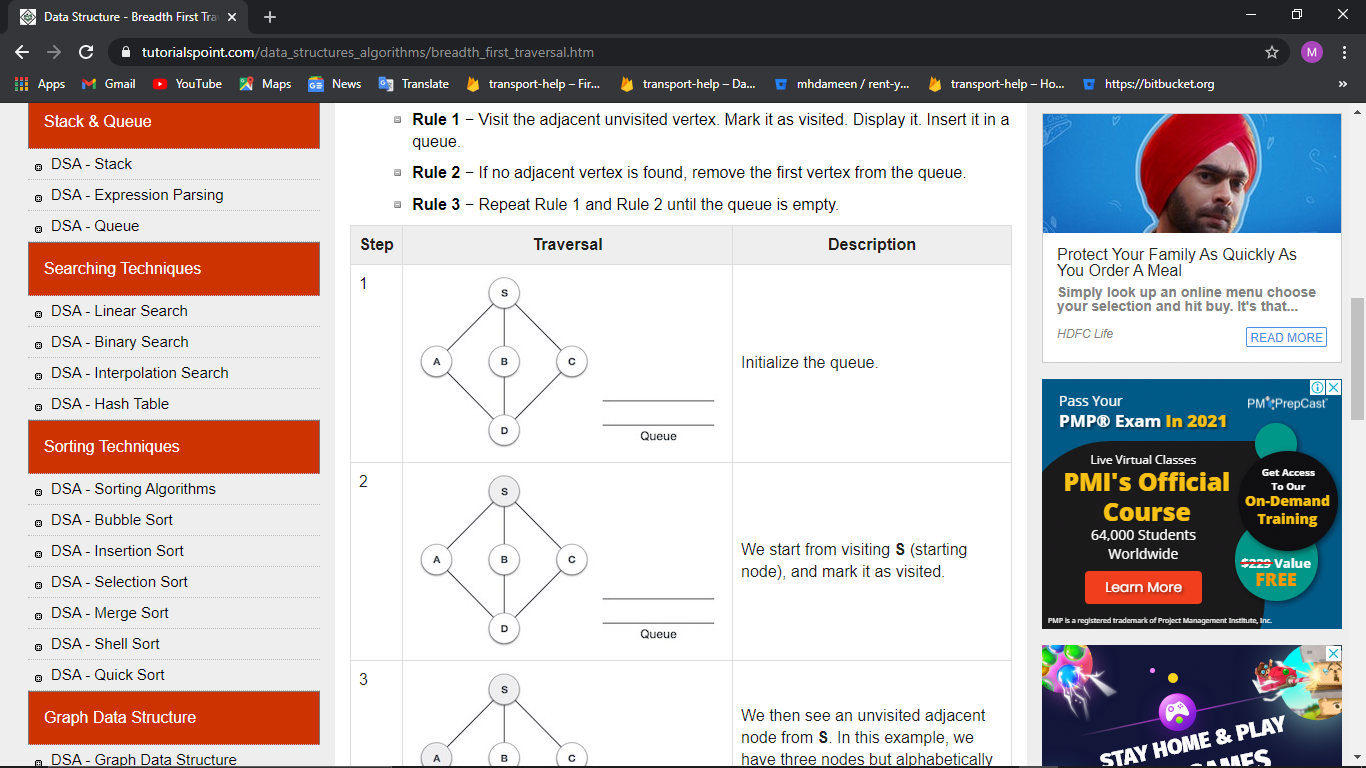
**ADJACENCY LIST:**

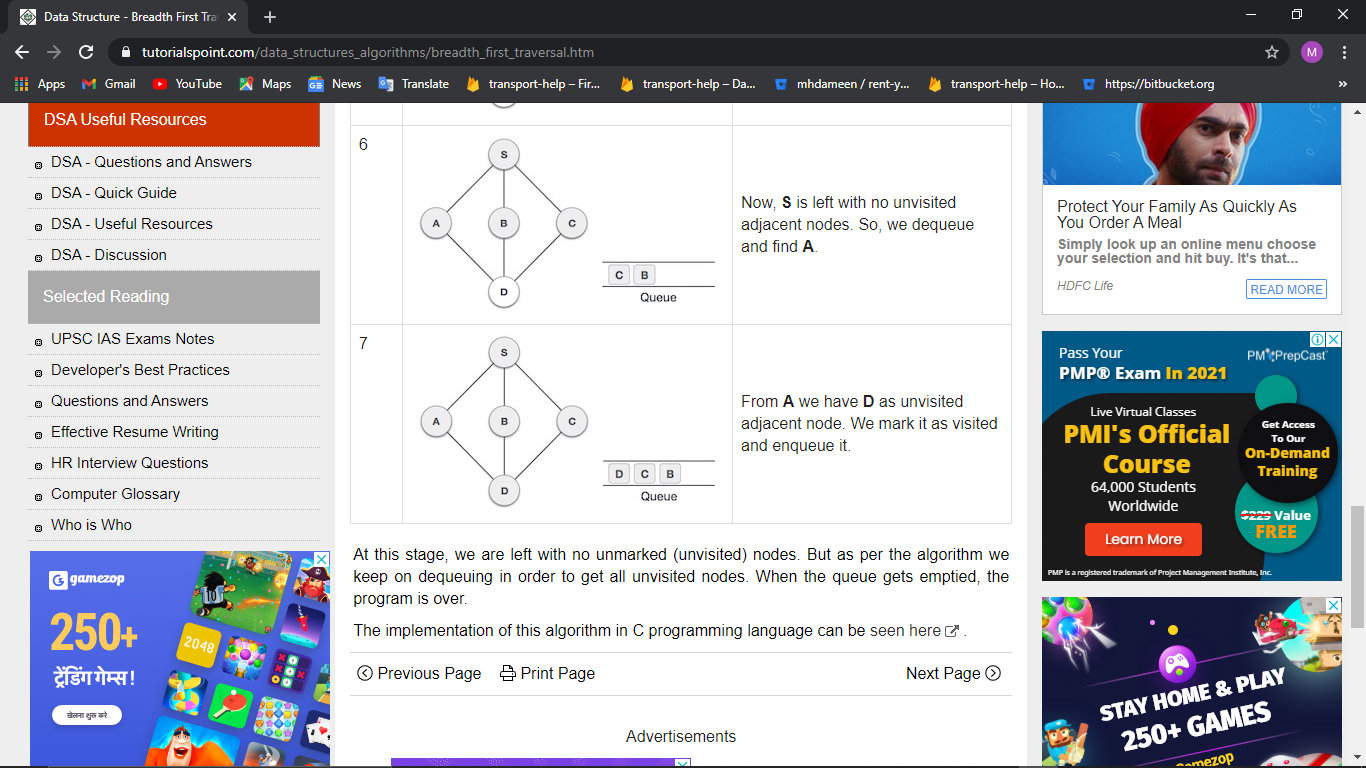
An array of lists is used. The size of the array is equal to the number of vertices. Let the array be an array[]. An entry array[i] represents the list of vertices adjacent to the***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is the adjacency list representation of the above grapAdjacency List Representation of Graph

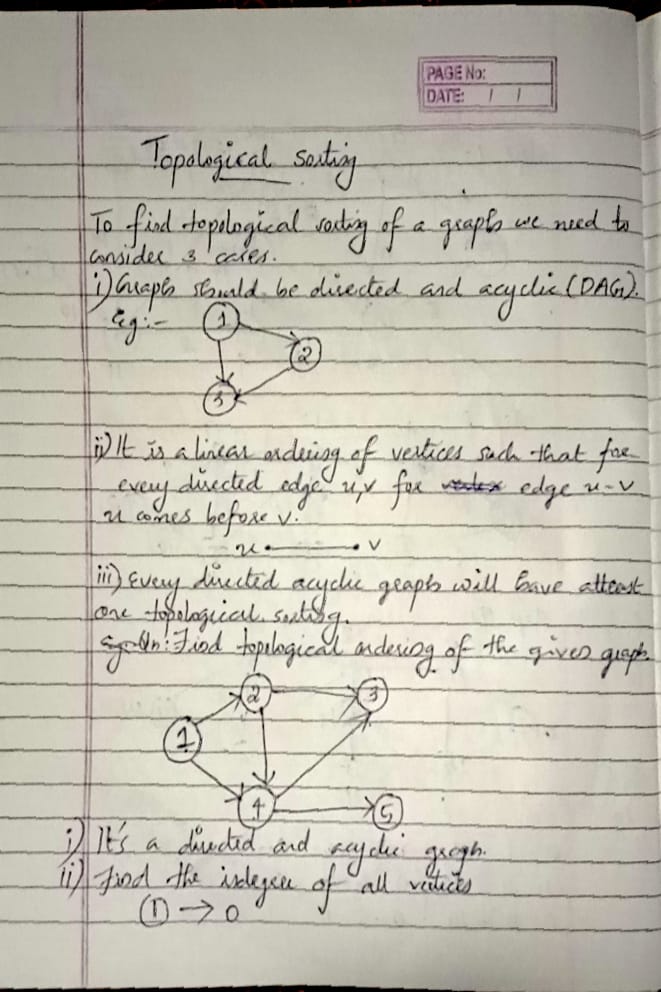
**DEPTH FIRST TRAVERSAL**

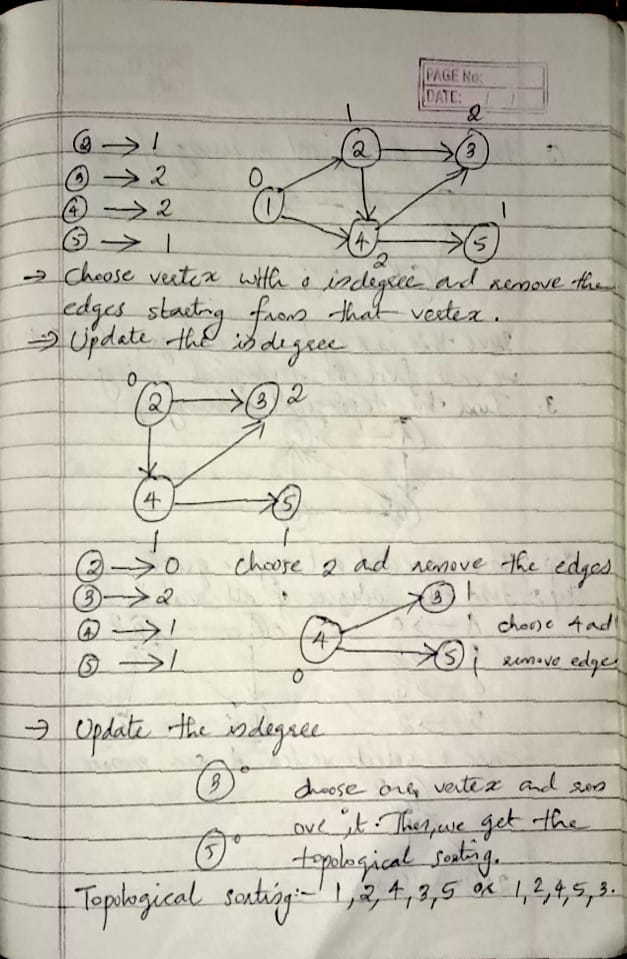
**BREADTH FIRST TRAVERSAL**

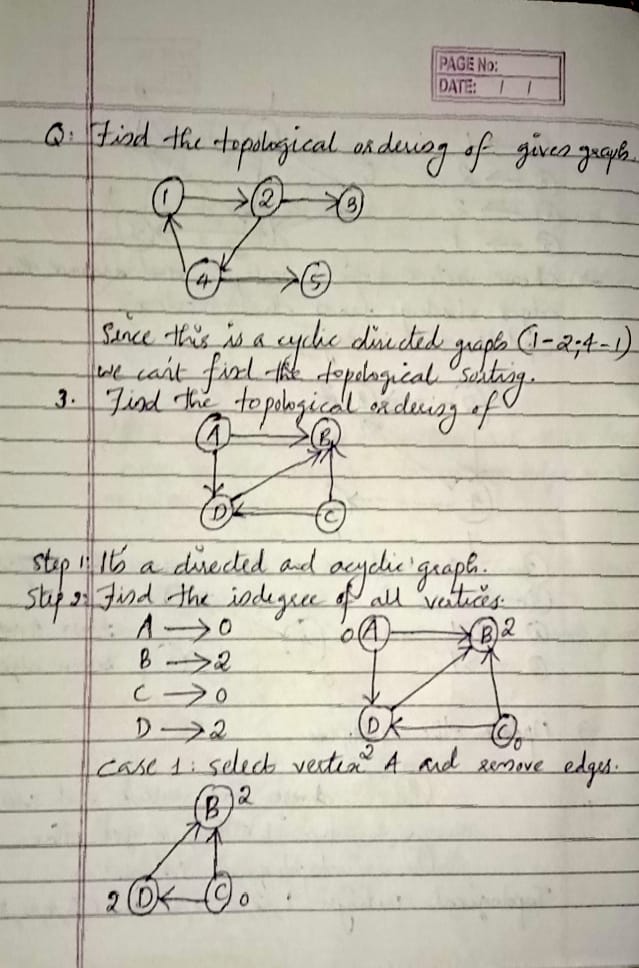
****

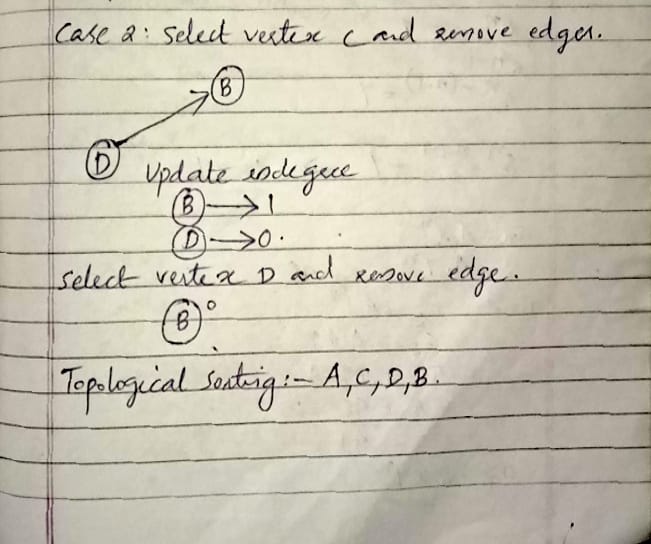
****

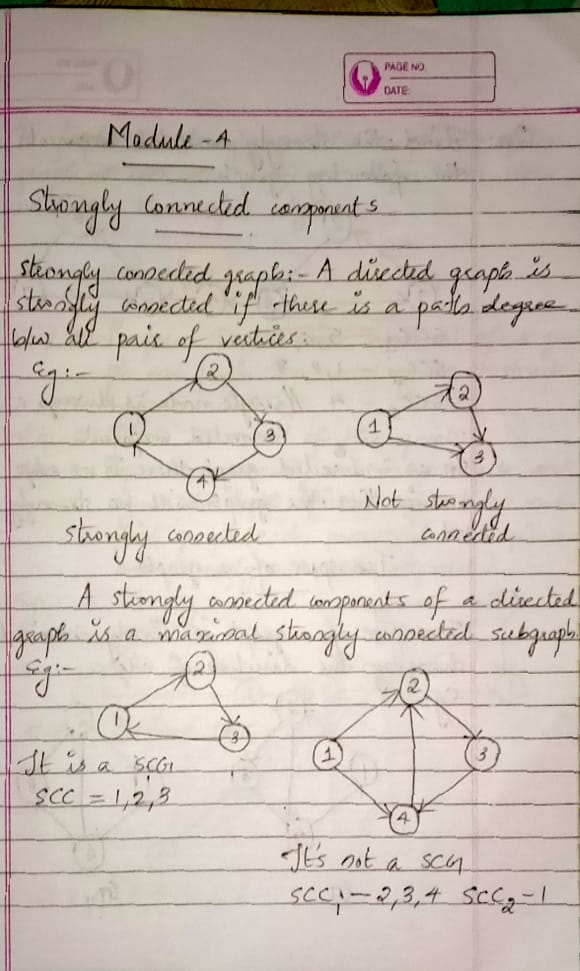
****

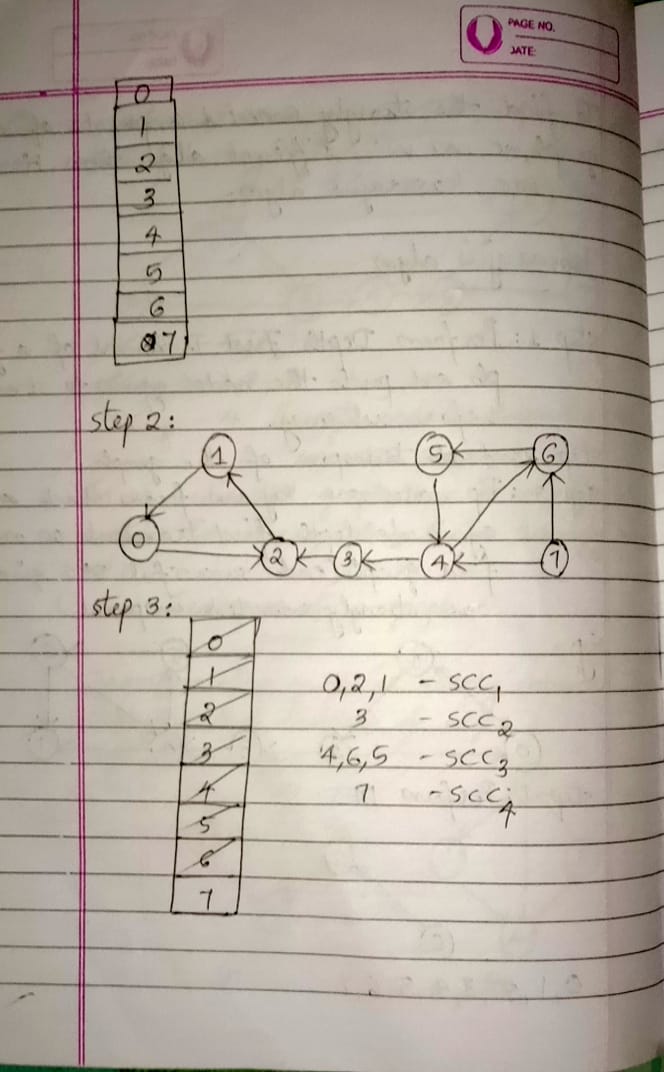


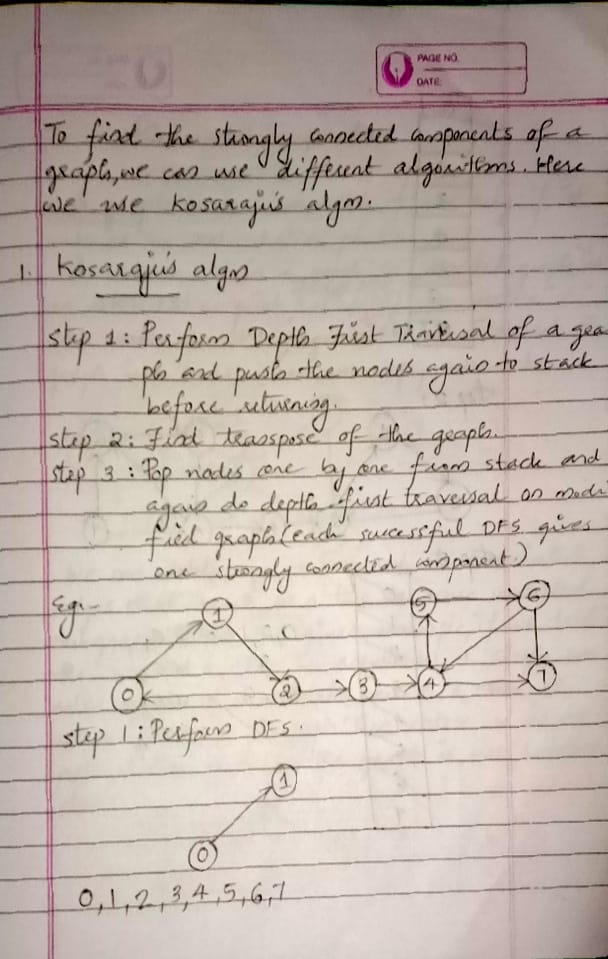


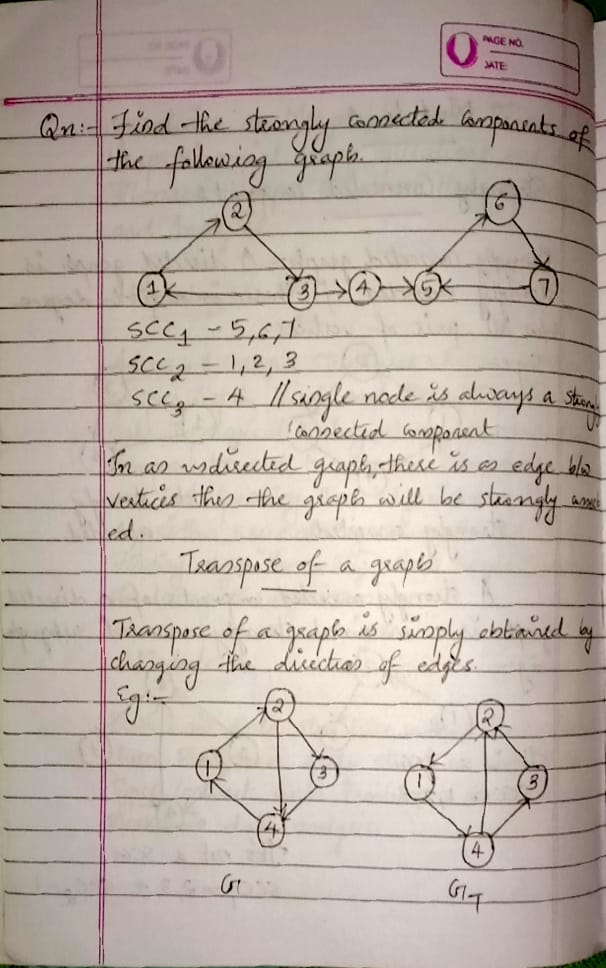












**BICONNECTED COMPONENTS**

A **biconnected component** of a graph is a connected subgraph that cannot be broken into disconnected pieces by deleting any single node . That is , a Graph G is biconnected if and only if it contains no articulation point.

An **articulation point** is a node of a graph whose removal would cause an increase in the number of connected **components**.in other words, A vertex in a graph G(connected graph) is an articulation point if and only if we delete the vertex v and all its edges then it disconnect the graph into 2 or more non empty components

We can delete any vertex and its associates edges that result in two or more connected subgraph , the given example :

-If we delete the vertex 1 then it result in a single connected graph

.-If we delete vertex 3 then it result in 2 subgraph then,vertex 3 is an Articulation point,So this graph is not biconnected.

Eg: for biconnected graph

Q;Find the articulation point in a graph

Step 1: Construct depth first traversal and provide number for each node according to the

Traversal.Find the lowercase number of parent for each node.

Step 2:If a root node has atleast 2 children then it will be articulation point.Also,leafnode has no Articulation point.

Eg:

Dfs for the above graph

d=1

d=2

d=3

d=4

d=5 d=6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| vertices | 1 | 2 | 3 | 4 | 5 | 6 |
| Discovery time(d) | 1 | 6 | 3 | 2 | 4 | 5 |
| Lowest discovery number(L) | 1 | 1 | 1 | 1 | 3 | 3 |

Here leaf node has no articulation point so , leaf node is biconnected.

To find the articulation point consider 2 edges u,v.here,u is the parent and v is the child

then u,v become an articulation point if and only if

L[v]>=d[u]

If this satisfies then u is an articulation point.

Consider the next vertex v=4 and u=3

* L [4]>=d[3]

1>=3

Here the condition is note satisfied.

Consider the next 2 vertex v=5 and u=3

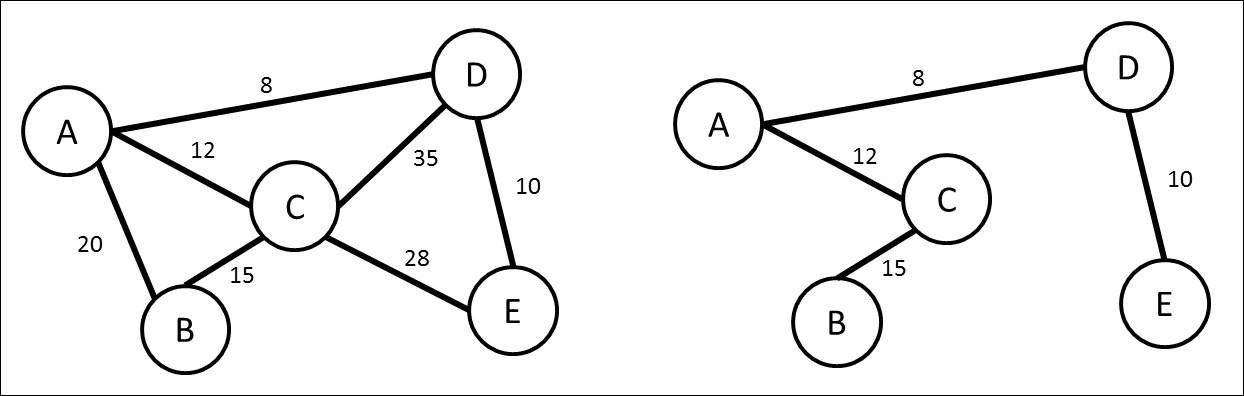
* L[5]>=d[3]

3>=3

Here the condition is satisfied .therefore, u is an articulation point.so,3 is an articulation point.

Minimum cost spanning tree

A **Minimum Spanning Tree** (**MST**) works on graphs with directed and weighted (non-negative costs) edges. Consider a graph G with n vertices. The spanning tree is a subgraph of graph G with all its n vertices connected to each other using n-1 edges. Thus, there is no possibility of a cycle with the subgraph. If the spanning tree does have a cycle, then it is advisable to remove any one edge, probably the one with the highest cost. The spanning tree with the least sum of edge weights is termed as a MST. It is widely used in applications such as laying of power cables across the city, connecting all houses using the least length of power cables. Here, the weight of each edge is the length of the cable, and the vertices are houses in the city. The most common algorithms to find the minimum cost spanning tree are Prim's algorithm and Kruskal's algorithm. Figure 8.11 shows the minimum cost spanning tree for an undirected-weighted graph.



1 .prim’s Algorithum

Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

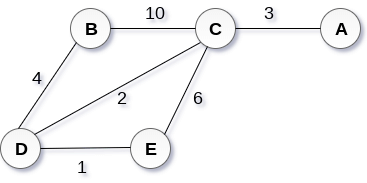
The algorithm is given as follows.

Algorithm

* **Step 1:** Select a starting vertex
* **Step 2:** Repeat Steps 3 and 4 until there are fringe vertices
* **Step 3:** Select an edge e connecting the tree vertex and fringe vertex that has minimum weight
* **Step 4:** Add the selected edge and the vertex to the minimum spanning tree T  
  [END OF LOOP]
* **Step 5:** EXIT

Example :

Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



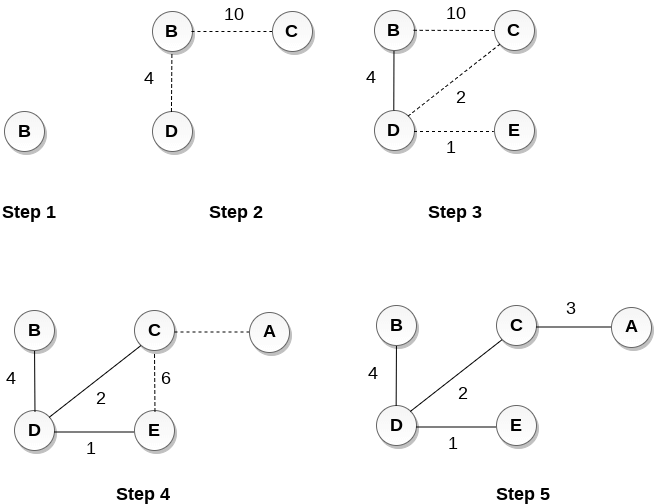
Solution

* **Step 1 :** Choose a starting vertex B.
* **Step 2:** Add the vertices that are adjacent to A. the edges that connecting the vertices are shown by dotted lines.
* **Step 3:** Choose the edge with the minimum weight among all. i.e. BD and add it to MST. Add the adjacent vertices of D i.e. C and E.
* **Step 3:** Choose the edge with the minimum weight among all. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C i.e. E and A.
* **Step 4:** Choose the edge with the minimum weight i.e. CA. We can't choose CE as it would cause cycle in the graph.

The graph produces in the step 4 is the minimum spanning tree of the graph shown in the above figure.

The cost of MST will be calculated as;

cost(MST) = 4 + 2 + 1 + 3 = 10 units.



[**Next →**](https://www.javatpoint.com/linear-search)**← Prev**

**2. Kruskal's Algorithm**

Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which, we can traverse every vertex of the graph. Kruskal's algorithm follows greedy approach which finds an optimum solution at every stage instead of focusing on a global optimum.

The Kruskal's algorithm is given as follows.

* Step 1: Remove all loops and parallel edges.
* Step 2: Arrange all edges in the increasing order of their weight.
* Step 3: Add the edges which have least weight.

### Example :

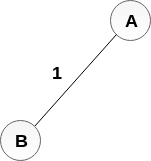
**Apply the Kruskal's algorithm on the graph given as follows.**



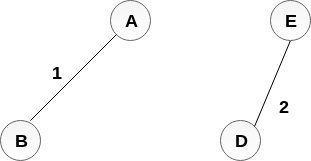
## Solution:

Start constructing the tree;

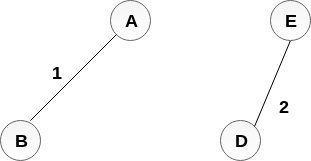
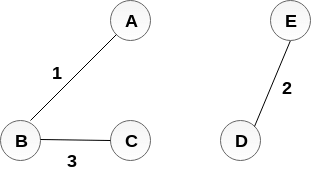
Add AB to the MST;



Add DE to the MST;



Add BC to the MST;

The next step is to add AE, but we can't add that as it will cause a cycle.

The next edge to be added is AC, but it can't be added as it will cause a cycle.

The next edge to be added is AD, but it can't be added as it will contain a cycle.

Hence, the final MST is the one which is shown in the step 4.

the cost of MST = 1 + 2 + 3 + 4 = 10.

