UNIT IV NON LINEAR DATA STRUCTURES - GRAPHS

Definition – Representation of Graph – Types of graph - Breadth-first traversal - Depth-first traversal – Topological Sort – Bi-connectivity – Cut vertex – Euler circuits – Applications of graphs.

4.1 Definition - RESENTATION OF GRAPHS

- A graph G = {V, E} consists of a set of vertices V and set of edges E.
- Vertices are referred to as nodes in graph and the line joining the two vertices are referred to as Edges.

Types of Graphs

Graphs are of two types

- Directed graphs.
- Undirected Graphs.

(i) Directed Graphs

Directed graph is a graph which consists of *directed edges.* It is also referred as *Digraph.*

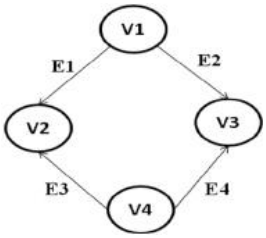


Fig: 4.1 Directed graphs.

In Directed graph, *the edges between the vertices are ordered*. *E1 is the edge between the vertices V1 and V2*.

- *V1* is called the *Head* and *V2* is called the *Tail*.
- So, *E1 is a set of (V1, V2)* and *not of (V2, V1).*

(ii) Undirected Graphs:

- Undirected graph is a graph, which consists of undirected edges.

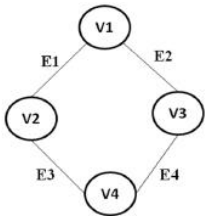


Fig: 4.2 undirected graphs.

- In Undirected graph, the edges between the vertices are not ordered.
- So, E1 is a set of (V1, V2) or (V2, V1).

Terms Related To Trees

- **Adjacent nodes :** two nodes are adjacent if they are connected by an edge
- **Path:** a sequence of vertices that connect two nodes in a graph
- **Length of path of graph:** is the number of edges in the path
- ***In-degree*** of a node x in G is the number of edges coming to x.
- ***Out-degree*** of x is the number of edges leaving x.

Degree and Neighbor:

Let G be an undirected graph

- The ***degree*** of a node x is the number of edges that have x as one of their end nodes
- The neighbors of x are the nodes adjacent to x

Weighted Graphs:

A graph is said to be weighted graph if every edge in the graph is assigned a weight or value. It can be either a directed or an undirected graph.

Fig: 4.3 weighted graphs.

Complete Graph:

- A complete graph is a graph in which there is an edge between every pair of vertices.
- A complete graph with ***n vertices*** will have ***n(n - 1) /2 edges***.

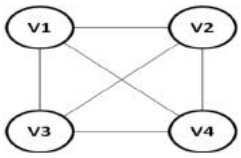


Fig: 4.4 complete graphs.

Sub Graph

A sub graph G' of G is a graph G such that the set of vertices and set of edges of G' are proper subset of the set of edges of G.

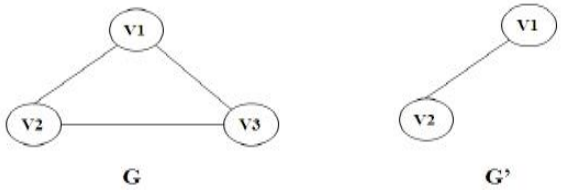


Fig: 4.5 Sub graphs.

Connected Graphs

An undirected graph is said to be connected if for every pair of distinct vertices Vi and Vj, there is a path from Vi to Vj in G.

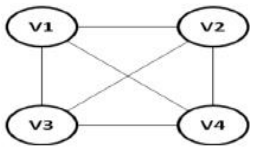


Fig: 4.6 Connected graphs.

Strongly Connected Graphs:

A directed graph is said to be strongly connected if and only if, for each pair of distinct vertices Vi and Vj, there is a path from Vi to Vj in G.

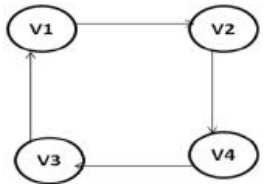
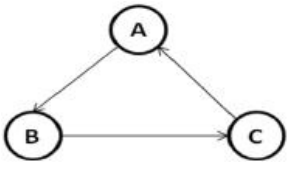


Fig: 4.7 strongly connected graphs.

Cyclic Graphs

A directed graph is said to be a cyclic graph in which no vertex is repeated except the first and last vertex are the same.



Cycle = A -> B-> C ->A

Fig: 4.8 Directed Cyclic graphs.

An undirected graph is said to be a cyclic graph in which if any edge appears more than once it appears with the same orientation.

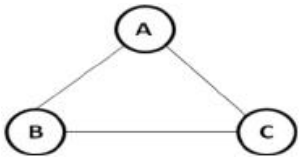
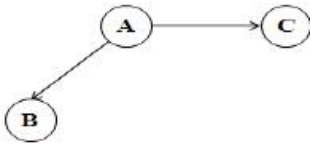


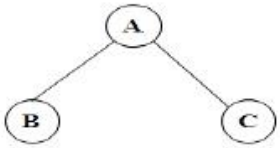
Fig: 4.9 Undirected Cyclic graphs.

Acyclic Graphs

A graph is said to be a acyclic graph ***if it has no cycles.***



DAG – Directed acyclic graph (no specific cycle)



Undirected acyclic graph

Fig: 4.10 Acyclic graphs.

4.2 TYPES OF GRAPHS -Representation

There are two representations of graphs:

- Adjacency matrix representation
- Adjacency lists representation

(i) Representing the graph as adjacency matrix

- In this representation, each graph of n nodes is represented by an n x n matrix A, that is, a two-dimensional array A.
- The nodes are labeled 1,2,…,n.
  A[i][j] = 1 if (i,j) is an edge
  A[i][j] = 0 if (i,j) is not an edge

Data Structures for Graphs as Adjacency Matrix

- A two-dimensional matrix or array that has one row and one column for each node in the graph

- For each edge of the graph (Vi, Vj), the location of the matrix at row i and column j is 1
- All other locations are 0
- For an undirected graph, the matrix will be symmetric along the diagonal
- For a weighted graph, the adjacency matrix would have the weight for edges in the graph, zeros along the diagonal, and infinity (∞) every place

Advantage

- Simple to implement
- Easy and fast to tell if a pair (i,j) is an edge: simply check if A[i][j] is 1 or 0

Disadvantage

- Even if there are few edges, the matrix takes O(n2) in memory

(ii) Representing the graph as adjacency list

A graph of n nodes is represented by a one-dimensional array L of linked lists, where

- A list of pointers, one for each node of the graph
- L[i] is the linked list containing all the nodes adjacent from node i.
- The nodes in the list L[i] are in no particular order

Data Structures for Graphs as Adjacency List

- A list of pointers, one for each node of the graph.
- These pointers are the start of a linked list of nodes that can be reached by one edge of the graph.
- For a weighted graph, this list would also include the weight for each edge.

4.3 BREADTH-FIRST SEARCH

Graph traversal is technique used for searching a vertex in a graph. The graph traversal is also used to decide the order of vertices to be visit in the search process. A graph traversal finds the egdes to be used in the search process without creating loops that means using graph traversal we visit all verticces of graph without getting into looping path.

There are two graph traversal techniques and they are as follows...

1. **BFS (Breadth First Search)**
2. **DFS (Depth First Search)**

BFS (BREADTH FIRST SEARCH)

BFS traversal of a graph, produces a **spanning tree** as final result. **Spanning Tree** is a graph without any loops. We use **Queue data structure** with maximum size of total number of vertices in the graph to implement BFS traversal of a graph.

We use the following steps to implement BFS traversal...

Step 1: Define a Queue of size total number of vertices in the graph.

Step 2: Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.

Step 3: Visit all the **adjacent** vertices of the verex which is at front of the Queue which is not visited and insert them into the Queue.

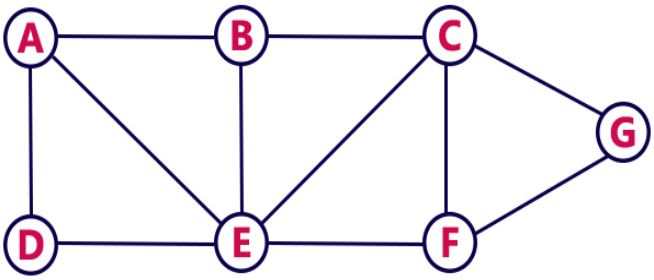
Step 4: When there is no new vertex to be visit from the vertex at front of the Queue then delete that vertex from the Queue.

Step 5: Repeat step 3 and 4 until queue becomes empty.

Step 6: When queue becomes Empty, then produce final spanning tree by removing unused edges from the graph

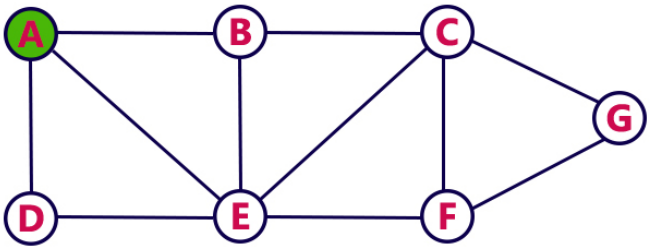
Example:

Consider the following example graph to perform BFS traversal



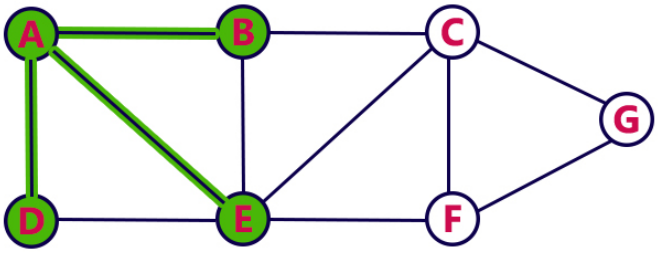
Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.



Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..

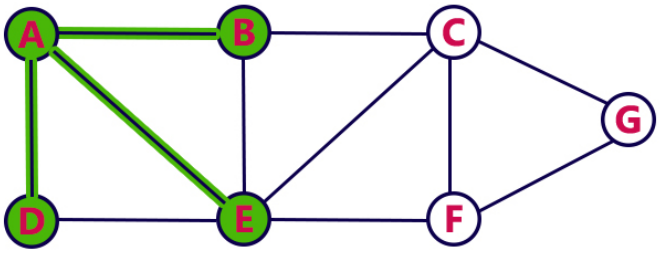


Queue

	D	E	B			

Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.

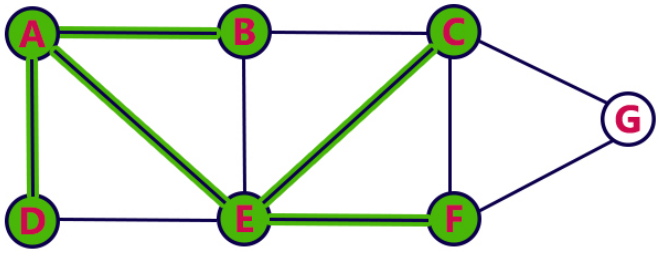


Queue

		E	B			

Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.

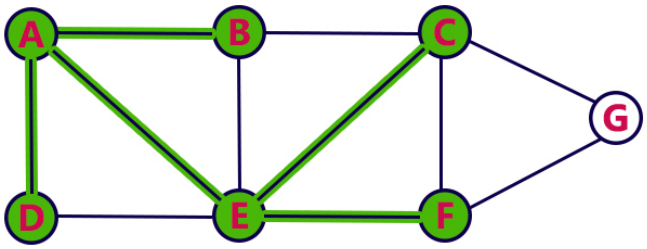


Queue

			B	C	F	

Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.

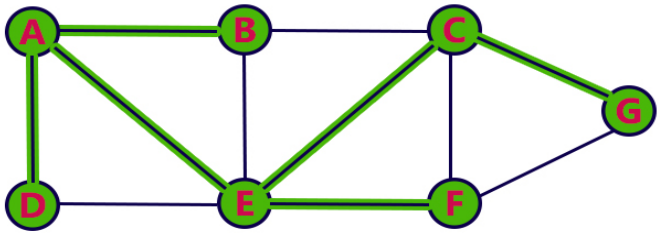


Queue

				C	F	

Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.

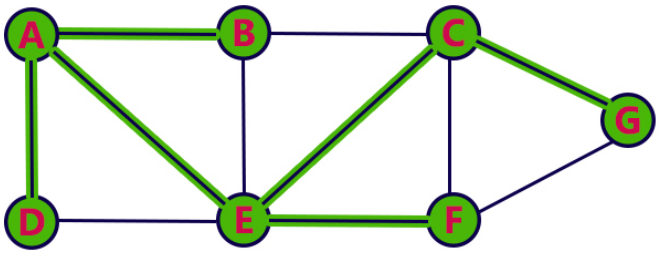


Queue

					F	G

Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.

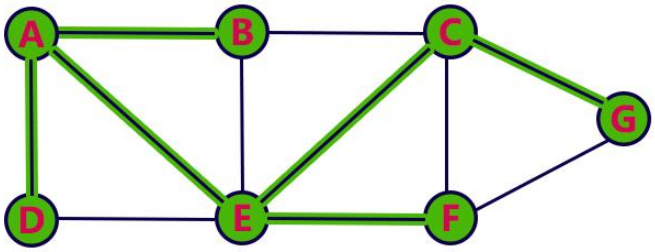


Queue

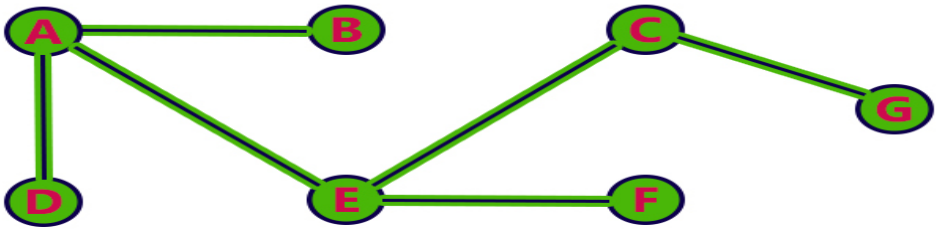
						G

Step 8:

- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.



- Queue became Empty. So, stop the BFS process.
- Final result of BFS is a Spanning Tree as shown below...



Application of breadth first search

- To check whether the graph is connected or not

Program for breadth first search

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v) {
    for (i=1;i<=n;i++)
      if(a[v][i] && !visited[i])
       q[++r]=i;
    if(f<=r) {
            visited[q[f]]=1;
            bfs(q[f++]);
    }
}
void main() {
    int v;
    clrscr();
    printf("\n Enter the number of vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++) {
```

```
            q[i]=0;
            visited[i]=0;
    }
    printf("\n Enter graph data in matrix form:\n");
    for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
       scanf("%d",&a[i][j]);
    printf("\n Enter the starting vertex:");
    scanf("%d",&v);
    bfs(v);
    printf("\n The node which are reachable are:\n");
    for (i=1;i<=n;i++)
      if(visited[i])
       printf("%d\t",i); else
       printf("\n Bfs is not possible");
    getch();
}
```

Output:

```
Enter Number of Vertices:4
Enter graph data in Matrix form:
1   0   1   0
0   0   1   1
0   1   0   1
1   1   1   1

Enter the Starting Vertex: 1

The nodes which are reachable are

1   2     3    4
```

4.4 DEPTH FIRST SEARCH

DFS (Depth First Search)

DFS traversal of a graph, produces a **spanning tree** as final result. **Spanning Tree** is a graph without any loops. We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal of a graph.

We use the following steps to implement DFS traversal...

Step 1: Define a Stack of size total number of vertices in the graph.

Step 2: Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.

Step 3: Visit any one of the **adjacent** vertex of the verex which is at top of the stack which is not visited and push it on to the stack.

Step 4: Repeat step 3 until there are no new vertex to be visit from the vertex on top of the stack.

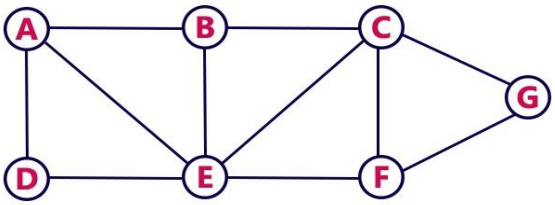
Step 5: When there is no new vertex to be visit then use **back tracking** and pop one vertex from the stack.

Step 6: Repeat steps 3, 4 and 5 until stack becomes Empty.

Step 7: When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

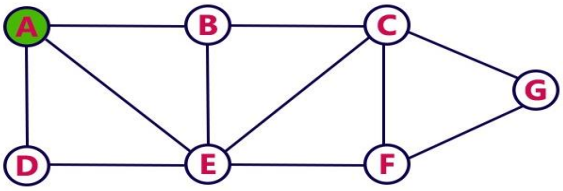
Example:

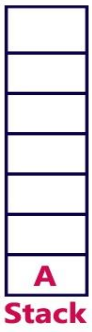
Consider the following example graph to perform DFS traversal



Step 1:

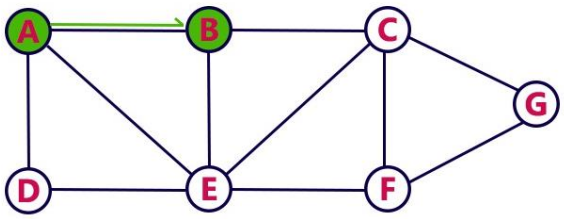
- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.

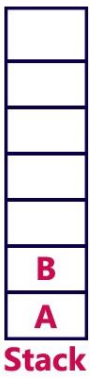




Step 2:

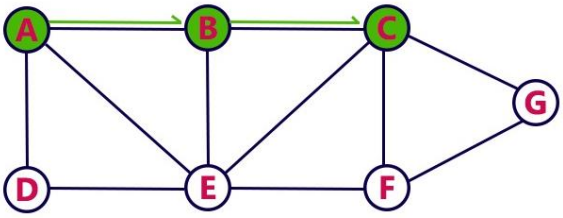
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.

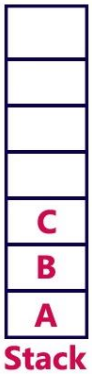




Step 3:

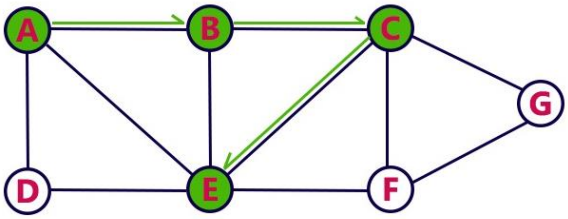
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.





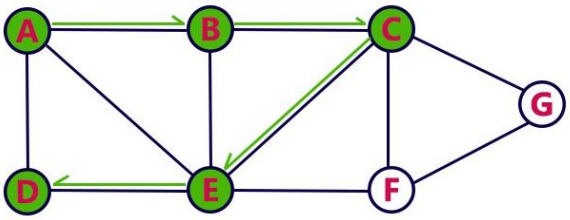
Step 4:

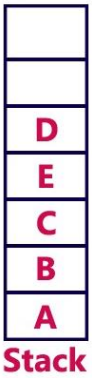
- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack



Step 5:

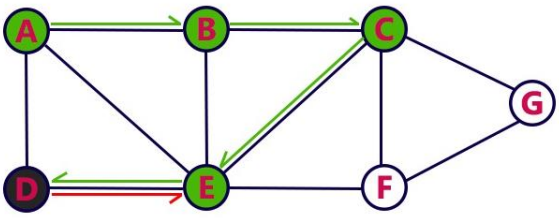
- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack

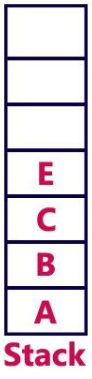




Step 6:

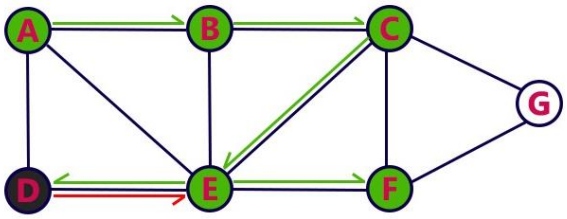
- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.





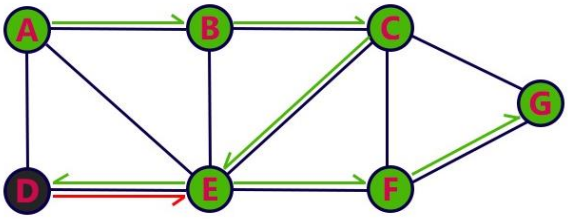
Step 7:

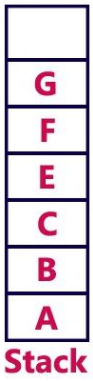
- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.



Step 8:

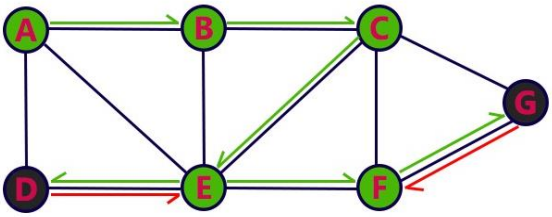
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.





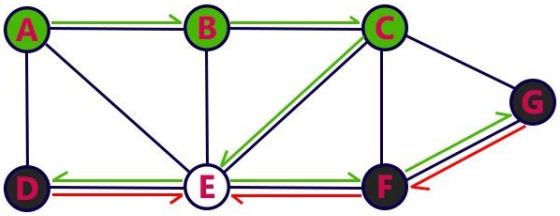
Step 9:

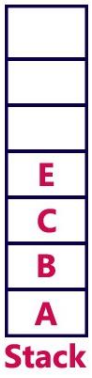
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.



Step 10:

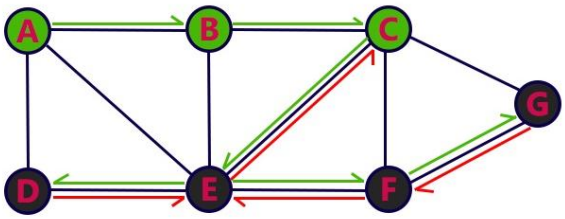
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.

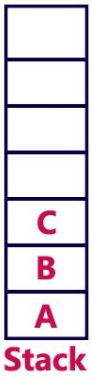




Step 11:

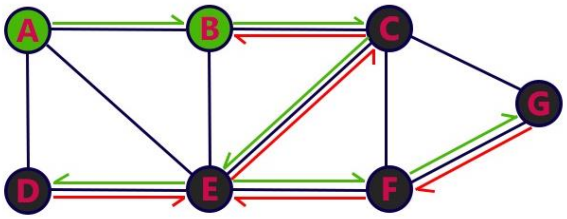
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.

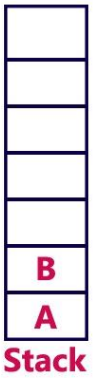




Step 12:

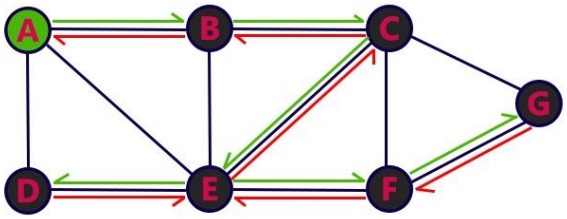
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.





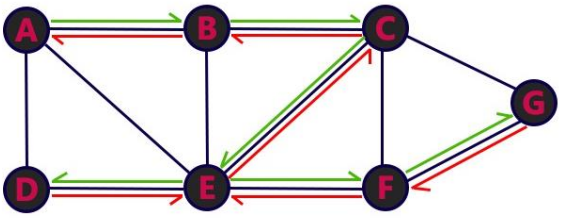
Step 13:

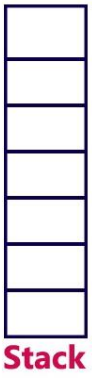
- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.



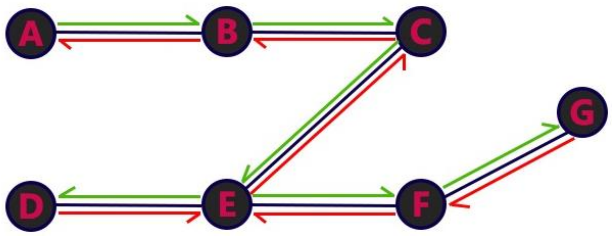
Step 14:

- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.





- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.



Application of Depth First Search

- To check whether the undirected graph is connected or not.
- To check whether the connected undirected graph is biconnected or not.
- To check the Acyclicity of the directed graph.

Program for Depth First Search

```
#include<stdio.h>
 #include<conio.h>
 int a[20][20],reach[20],n;
 void dfs(int v)
 {
      int i;
      reach[v]=1;
      for (i=1;i<=n;i++)
        if(a[v][i] && !reach[i])
      {
                printf("\n %d->%d",v,i);
                dfs(i);
      }
 }
 void main()
 {
      int i,j,count=0;
      clrscr();
      printf("\n Enter number of vertices:");
      scanf("%d",&n);
      for (i=1;i<=n;i++)
      {
                reach[i]=0;
                for (j=1;j<=n;j++)
                  a[i][j]=0;
```

```
        }
        printf("\n Enter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++)
           scanf("%d",&a[i][j]);
        dfs(1);
        printf("\n");
        for (i=1;i<=n;i++) {
                if(reach[i])
                  count++;
        }
        if(count==n)
          printf("\n Graph is connected"); else
          printf("\n Graph is not connected");
        getch();
    }
```

OUTPUT:

Enter Number of Vertices:4
Enter the Adjacency Matrix:
0 1 0 1
1 1 1 1
0 1 1 0
0 1 1 1

1→2
2→3
2→4

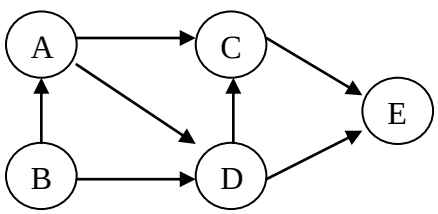
Graph is connected

4.5 TOPOLOGICAL SORTING

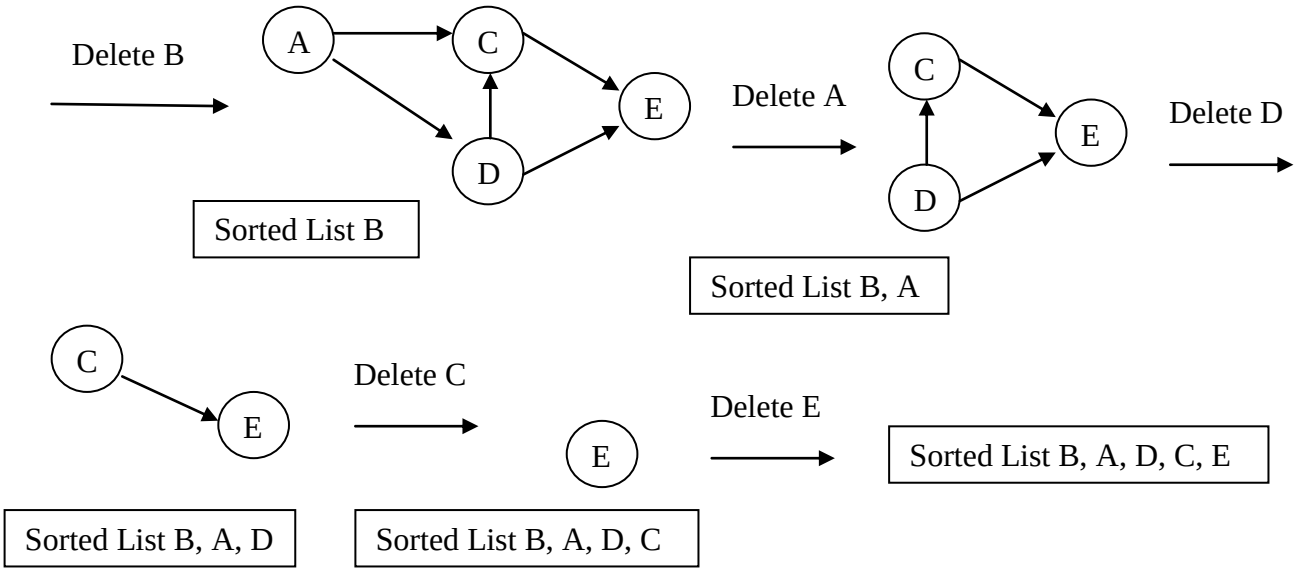
Algorithm:

1. From a given graph find a vertex with no incoming edges. Delete it along with an the edges outgoing from it. If there are more than one such vertices then break the tie randomly.
2. Note the vertices that are deleted.
3. All these recorded vertices give topologically sorted list.

Example 1:



Choose vertex B, because it has no incoming edge, delete it along with its adjacent edges.

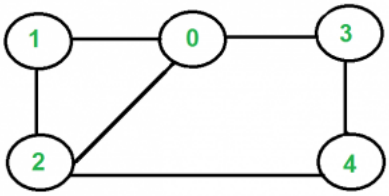


4.6 BI CONNECTED GRAPH

- An undirected graph is called Biconnected if there are two vertex-disjoint paths between any two vertices.
- In a Biconnected Graph, there is a simple cycle through any two vertices. By convention, two nodes connected by an edge form a biconnected graph, but this does not verify the above properties. For a graph with more than two vertices, the above properties must be there for it to be Biconnected.

Following are some examples.

Biconnected



Biconnected

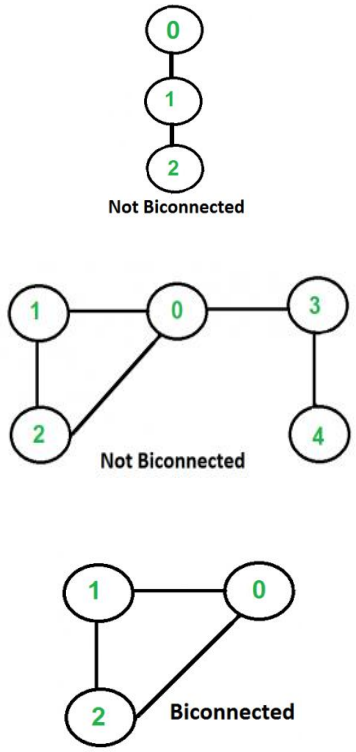


Fig: 4.11 Bi-connected graphs.

- A connected graph is Bi connected if it is connected and doesn't have any Articulation Point.
- We mainly need to check two things in a graph.
  1) The graph is connected.
  2) There is not articulation point in graph.
- We start from any vertex and do DFS traversal. In DFS traversal, we check if there is any articulation point.
- If we don't find any articulation point, then the graph is Biconnected.
- Finally, we need to check whether all vertices were reachable in DFS or not. If all vertices were not reachable, then the graph is not even connected.

4.7 CUT VERTEX - Articulation Points (or Cut Vertices) in a Graph

Definition:

A vertex V is an Undirected graph G is called a cut vertex if removing it disconnects the graph.

- The cut vertex is also called as articulation point.
- The following example represents the concept of cut vertex.

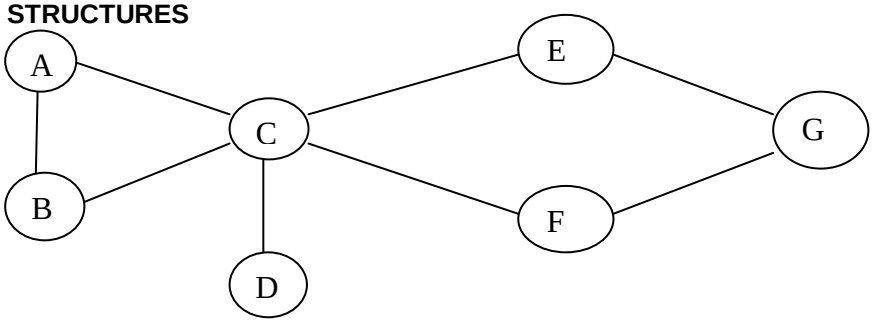


Fig: 4.12 Cut Vertex.

Here Vertex C is a CUT VERTEX

- On removing C we get the disconnected Component as.

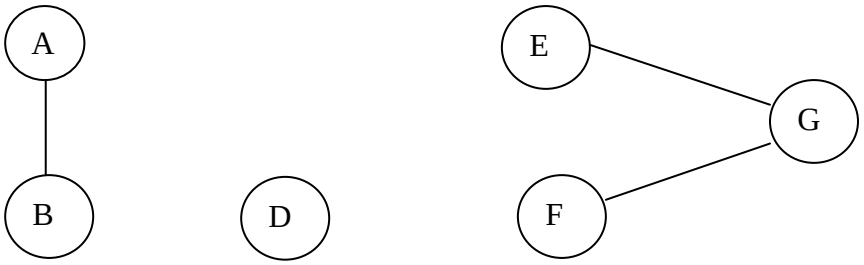


Fig: 4.13 Cut Vertex (Disconnected Component)

- The concept of vertex are useful for designing reliable networks

4.8 EULER CIRCUITS

- In Graph Theory there is a famous problem known **Konigsberg Bridge Problem**.
- In this problem the main theme was cross the seven bridge exactly once to visit various cities.
- From this problem, the concept of Euler Circuit is developed. Let us define the terminologies Euler path and Euler Circuit.

Euler Path:

- A Path in a graph G is Called Euler Path if it includes every edge exactly once and every vertex gets visited.
- Euler Circuit on graph G is an Euler path that visits each vertex of graph G and uses every edge of G.

For Example

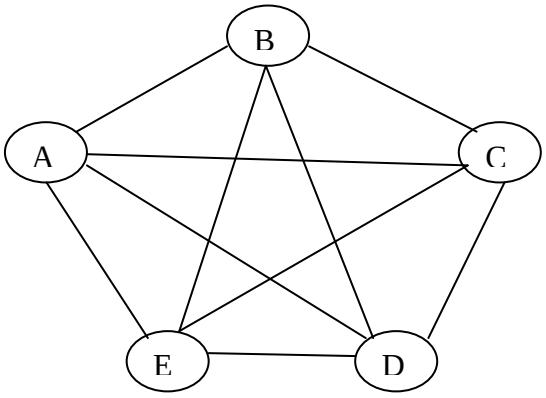


Fig: 4.13 Euler Circuit

The Euler Circuit is A-B-E-A-D-B-C-E-D-C-A

4.9 APPLICATION OF GRAPHS

- In Computer networking such as Local Area Network (LAN), Wide Area Networking (WAN), internetworking.
- In Telephone cabling graph theory is effectively used
- In Job Scheduling algorithm

4.9.1 FINDING SHORTEST PATH

Shortest Path Algorithm

The shortest path algorithm determines the minimum cost of the path from source to every other vertex. There are two types of shortest path problem such as

1. The single source shortest path problem.
2. The all pair shortest path problem.

The single source shortest path algorithm finds the minimum cost from single source vertex to all other vertices. Dijkstra's algorithm is used to solve this problem which follows the greedy technique.

All pairs shortest path problem finds the shortest distance from each vertex to all other vertices. To solve this problem dynamic programming technique known as floyd's algorithm is used.

These algorithms are applicable to both directed and undirected weighted graphs provided that they do not contain a cycle of negative length.

DIJKSTRA'S SHORTEST PATH

The **Dijkstra's shortest path** algorithm suggests the shortest path from some source node to the some other destination node. The source node or the node from we start measuring the distance is called the start node and the destination node is called the end node. In this algorithm we start finding the distance from the start node and find all the paths from it to neighbouring nodes. Among those which ever is the nearest node that path is selected. This process of finding the nearest node is repeated till the end node. Then whichever is the path that path is called the shortest path.

Since in this algorithm all the paths are tried and then we are choosing the shortest path among them, this algorithm is solved by a greedy algorithm. One more thing is that we are having all the vertices in the shortest path and therefore the graph doesn't give the spanning tree.

Example:

P = Set which is for nodes which have already selected.
T = Remaining node

Step 1:

v = a

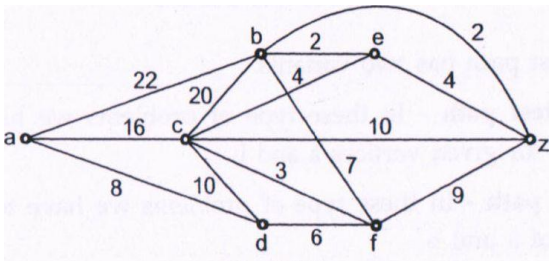
P = {a}, T = {b,c,d,e,f,z}
dist(b) = min[old dist(b),dist(a)+w(a,b)]
dist(b) = min[∞,0+22|
dist(b) = 22
dist(c) = 16
dist(d)= 8 ← minimum node
dist(e) = ∞
dist(f) = ∞
dist(z) = ∞

So minimum node is selected in P i.e. node d.

Step 2 :

v = d

P = {a,d} T = {b,c,e,f,z}
dist(b) = min{old dist(b),dist(d)+w(b,d)}
dist(b) = min{22,8+∞}
dist(b) = 22
dist(c) = min{16,8+10} = 16
dist(e) = min{∞,8+∞} = 8
dist(f) = min{∞,8+6} = 14
dist(z) = min{∞,8+∞} = ∞



Step 3 :

v = f

P = {a,d,f} T = {b,c,e,z}

dist(b) = min{22,14+7} = 21

dist(c) = min{16,14+3} = 16

dist(e) = min{ ∞,14 +∞ } = ∞

dist(z) = min{∞, 14+9) = 23

Step 4 :

v = c

P = {a,d,f,c} T = {b,e,z}

dist(b)= min{21,16+20} = 21

dist(e) = min{∞ ,16+4}= 20

dist(z) = min{23,16+10} = 23

Step 5:

v = e

P = (a,d,f,c,e(T = |b,z|

dist(b) =min|21,20+2| = 21

dist(z) =min|23,20+4)= 23

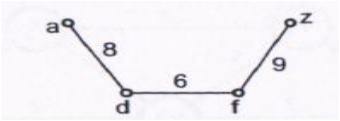
Step 6:

v = b

P = (a,d,f,c,e,b) T = (z)

dist(z) = min(23,21+2) = 23.

Now the target vertex for finding the shortest path is z. Hence the length of the shortest path from the vertex a to z is 23. The shortest path shown below



PART A

GRAPHS - REPRESENTATION OF GRAPHS

1. Define graph. [L1]
2. Define directed graph or digraph. [L1]
3. Define undirected graph. [L1]
4. Define path in a graph. [L1]
5. Define a cycle in a graph. [L1]
6. Define a strongly connected graph. [L1]

7. Define a weakly connected graph. [L1]
8. Define a weighted graph. [L1]
9. Define adjacency matrix. [L1]
10. What does traversing a graph mean? State the different ways of traversing a graph.[L1]
11. What is a simple graph? [L1]
12. When a graph is said to be bi-connected? (APR/MAY 2010) [L2]
13. What are the applications of graph? [L1]
14. How a graph is represented? [L1]
15. Define complete graph. [L1]
16. Define acyclic graph. [L1]
17. What is activity node graph? [L1]
18. Define indegree and outdegree of a graph. [L1]
19. When does a graph become a tree? [L1]
20. Define connected components. [L1]

BREADTH FIRST SEARCH & DEPTH FIRST SEARCH

21. What is breadth-first search? [L1]
22. Give the applications of DFS. [L1]
23. Differentiate BFS and DFS. [L1]

TOPOLOGICAL SORT

24. What is topological sort? Give algorithm. [L1]

MINIMUM SPANNING TREE

25. What is a minimum spanning tree? [L1]
26. When a graph is said to be bipartite ? [L2]
27. Define a depth first spanning tree. [L1]
28. What are the methods to solve minimum spanning tree? [L1]
29. What is the minimum number of spanning trees possible for a complete graph with n vertices? [L1]

PART-B

1. Explain in detail about various types of Graphs. **[Pg.No:111]** [L1]
2. Explain in detail different ways of representation of Graphs. **[Pg.No:114]** [L1]
3. Explain Graph traversal with suitable algorithm and example. (or)
   Explain breadth first search algorithm for traversal of any graph with suitable examples. [Nov-Dec 2014] **[Pg.No:115]** [L2]
4. Explain depth first search algorithm for traversal of any graph with suitable

examples.[Nov-Dec 2015] **[Pg.No:120]** [L2]

5. Using Dijiktra's algorithm, find the shortest path from the source to all other Nodes [Nov-Dec 2014, Nov-Dec 2015] **[Pg.No:130]** [L4]

PART A

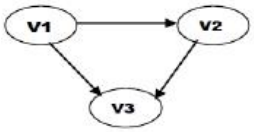
GRAPHS - REPRESENTATION OF GRAPHS

1. Define graph.

A graph is a collection of two sets of V and E where V is finite non empty set of vertices and E is a finite non empty set of edges.

Vertices are nothing but the nodes in the graph and two adjacent vertices are joined by edges. The graph is denoted by G = {V, E}.

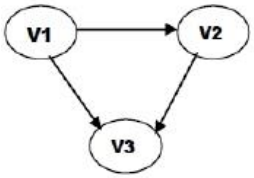
Example:



2. Define directed graph or digraph.

If an edge between any two nodes in a graph is directionally oriented, a graph is called as directed .it is also referred as digraph.

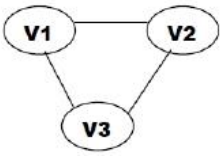
Example:



3. Define undirected graph.

If an edge between any two nodes in a graph is not directionally oriented, a graph is called as undirected .it is also referred as unqualified graph

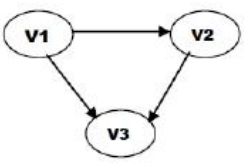
Example:



4. Define path in a graph.

A path in a graph is defined as a sequence of distinct vertices each adjacent to the next except possibly the first vertex and last vertex is different.

Example:

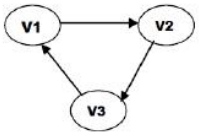


From the diagram, the path from V1 to V2 is V1,V2,V3.

5. Define a cycle in a graph.

A cycle is a path containing at least three vertices such that the starting and the ending vertices are the same

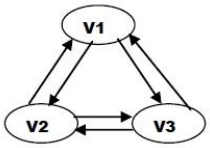
Example:



6. Define a strongly connected graph

A graph is said to be a strongly connected graph, if for every pair of distinct vertices there is a directed path from every vertex to every other vertex. It is also referred as a complete graph.

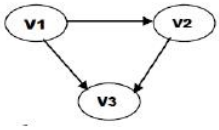
Example:



7. Define a weakly connected graph.

A directed graph is said to be a weakly connected graph if any vertex doesn't have a directed path to any other vertices.

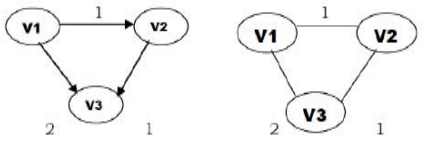
Example:



8. Define a weighted graph.

A graph is said to be a weighted graph if every edge in the graph is assigned some weight or value. The weight of an edge is a positive value that may be representing the distance between the vertices or the weights of the edges along the path.

Example:



9. Define adjacency matrix.

The adjacency matrix A, for a graph G = (V, E) with n vertices is an n*n matrix, such that
Aij=1,if there is an edge Vi to Vj
Aij=0,if there is no edge.

10. What does traversing a graph mean? State the different ways of traversing a graph.

Traversing a graph means visiting all the nodes in the graph. The two important graph traversal methods are

- Depth first traversal or depth first search (DFS)
- Breadth first traversal or breadth first search (BFS)

11. What is a simple graph?

A simple graph is a graph, which has not more than one edge between a pair of nodes than such a graph is called a simple graph.

12. When a graph is said to be bi-connected? (APR/MAY 2010)

A connected undirected graph is bi-connected if there is no vertices whose removal disconnects the rest of the graph.

13. What are the applications of graph?

- In computer networking such as Local Area Network(LAN), wide Area Networking (WAN) internetworking the graph is used.
- In telephone cabling graph theory is effectively used.
- In job scheduling algorithm the graph is used.

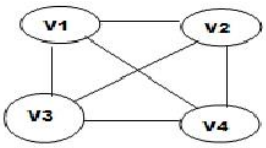
14. How a graph is represented?

There are two way of representing the graph are

- Adjacency matrix representation
- Adjacency list representation

15. Define complete graph.

A complete graph is a graph in which there is an edge between every pair of vertices. A complete graph n vertices will have n(n-1)/2 edges.



Number of vertices is 4
Number of edges is 6

16. Define acyclic graph.

A directed graph which has no cycles is referred to acyclic graph. It is abbreviated as DAG → Directed Acyclic Graph.

17. What is activity node graph?

Activity node graph represents a set of activity's and scheduling constraints. Each node represent activity (task) and an edge represent the next activity**.**

18. Define indegree and outdegree of a graph.

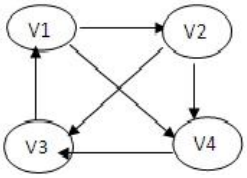
Indegree

Indegree of a vertex in a digraph is the number of edges that are incident on it.

Outdegree

Outdegree of vertex is the number of edges that leave the vertex.

Eg:



Indegree(V1) = 1
Indegree(V2) = 1
Outdegree(V3) = 1
Outdegree(V4) = 1

19. When does a graph become a tree?

A graph can be a tree it is connected.

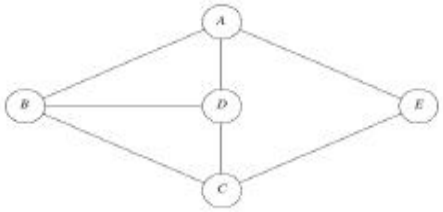
20. Define connected components.

Undirected Graphs

A undirected graph is 'connected ' if and only if a depth first search starting from any node visits every node.

Biconnectivity

A connected undirected graph is biconnected if there are no vertices if there are no vertices whose removal disconnects the rest of the graph.



BREADTH FIRST SEARCH & DEPTH FIRST SEARCH

21. What is breadth-first search?

Breath First Search (BFS) of a graph, G starts from an unvisited vertex u. Then all unvisited vertices vi adjacent to u are visited and then all unvisited vertices wj adjacent to vi are visited and so on. The traversal terminates when there are no more nodes to visit.

22. Give the applications of DFS.

- Finding connected component.
- Topological sorting.
- Finding 2-(edge or vertex) connected components.

- Finding 3-(edges or vertex)-connected components.
- Finding the bridge of a graph.
- Checking the acyclicity of the directed graph.

23. Differentiate BFS and DFS

Sl.NO	DFS	BFS
1	Backtracking is possible from a dead end	Backtracking is not possible.
2	Vertices from which exploration is incomplete are processed in a LIFO order.	The vertices to be explored are organized as a FIFO queue.
3	Search is done in one particular direction.	The vertices in the same level are maintained parallel.

TOPOLOGICAL SORT

24. What is topological sort? Give algorithm.

Topological sort is defined as an ordering of vertices in a directed acyclic graph. Such that if there is a path from Vi to Vj, then appears after Vi in the ordering.

- Find the vertex with no incoming edges.
- Print the vertex and remove it along with its edges from the graph.
- Apply the sample strategy to the rest of the graph.
- Finally all recorded vertices give topological sorted list.

MINIMUM SPANNING TREE

25. What is a minimum spanning tree?

A minimum spanning tree of an undirected graph G is a tree formed from graph edges that connects all the vertices of G at the lowest total cost.

26. When a graph is said to be bipartite ?

A bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V. A bipartite graph is a graph that does not contain any odd-length cycles.

27. Define a depth first spanning tree.

The tree that is formulated by depth first search on a graph is called as depth first spanning tree. The depth first spanning tree consists of tree edges and back edges.

28. What are the methods to solve minimum spanning tree?

- Prims algorithm.
- Kruskal's algorithm.

29. What is the minimum number of spanning trees possible for a complete graph with n vertices?

There are n^{n-2} number of spanning trees for a complete graph with n vertices. For example if there are 3 vertices in a complete graph i.e. K_3 then there are $3^{3-2}=3$ spanning trees possible.