UNIT – V

Graphs

Fundamentals, Representation of graphs,**Graph Traversals:** BFS, DFS, **Minimum cost spanning tree:** Definition, Prim's Algorithm, Kruskal's algorithm.
Hashing: Hash Table, Hash Function, Collison resolution Techniques- separate Chaining, open addressing.

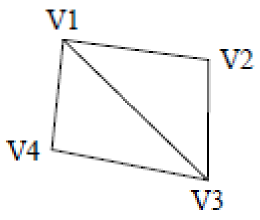
Definition:- Graph is a non linear data structure. A graph consists of 2 sets V, E.

(i) V is a finite set of vertices
(ii) E is a finite set of edges

Vertices are nothing but nodes of the graph where as Edges are nothing but connections between the nodes.

So, a graph can be represented as G= (V, E).

Eg:-



For the above graph, V= {V1, V2, V3, V4}

E={ (V1, V2), (V1, V3), (V1, V4), (V2, V3), (V3, V4)}

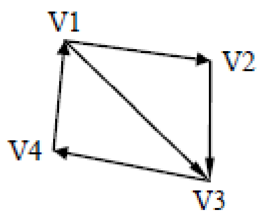
Applications of Graph:-

1. Graphs are used in the design of computer networks.
2. To implement routing system in airlines.
3. To find the shortest path in computer networks.
4. The web pages are connected like a graph in websites.

Graph Terminology:-

1.Digraph (or) Directed graph:- It is a graph such that G=(V, E) where V is a set of vertices and E is a set of edges with direction.

Eg:-

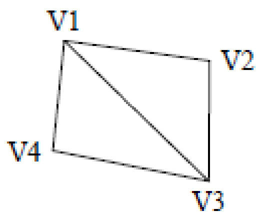


For the above graph, V= {V1, V2, V3, V4}

E={ (V1, V2), (V1, V3), (V2, V3), (V3, V4), (V4, V1)}

2.Undirected graph:- It is a graph such that G=(V, E) where V is a set of vertices and E is set of edges with no direction.

Eg:-

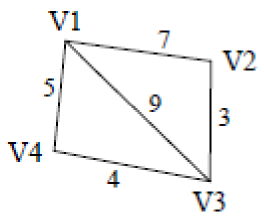


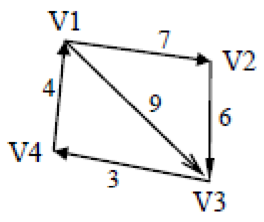
For the above graph, V= {V1, V2, V3, V4}

E={ (V1, V2), (V1, V3), (V1, V4), (V2, V3), (V3, V4)}

3.Weighted graph:- It is a graph in which all the edges are labelled with some weights.

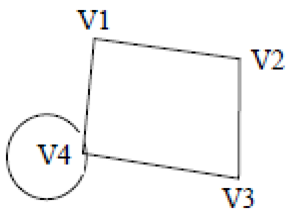
Eg:-





4.loop (or) self loop:- If there is an edge whose starting and ending vertices are same i.e. (Vi, Vi) then that edge is called as Cycle.

Eg:-

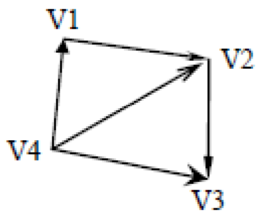


In the above graph vertex V4 and V2 has self loop.

5.Cyclic Graph:- A graph that have cycle is called Cyclic graph.

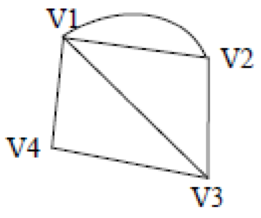
Eg:-

6.Acyclic Graph:-A graph that does not have cycle is called Acyclic graph.



7.Parallel edges:- If there is more than one edge between the same pair of vertices, then they are known as parallel edges.

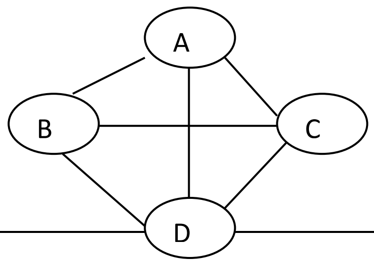
Eg:-



In the above graph 2 parallel edges between vertices V1 and V2.

8. Connected Graph:-If there is an edge between every pair of distinct vertices then the graph is called as Connected Graph.

Eg:-



<u>Representation of a Graph:-</u> A graph can be represented in 3 ways.

1. Set representation

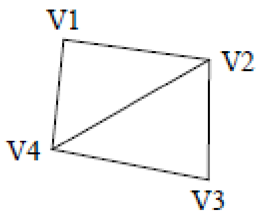
2. Adjacency matrix representation (**or**) Array representation

3. Adjacency list representation (**or**) Linked list representation

<u>1. Set representation:-</u> This is the straight forward method used for representing a graph.

In this method 2 sets are maintained V (Set of Vertices) and E (Set of Edges).

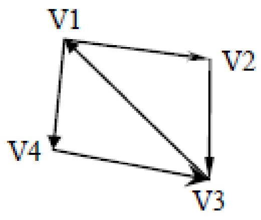
<u>Eg1:-</u>



V= {V1, V2, V3, V4}

E={ (V1, V2), (V1, V4), (V2, V3), (V2, V4), (V3, V4)}

<u>Eg2:-</u>



V= {V1, V2, V3, V4}

E={ (V1, V2), (V1, V4), (V2, V3), (V3, V1), (V4, V3)}

<u>2. Adjacency matrix representation (or) Array representation:-</u>

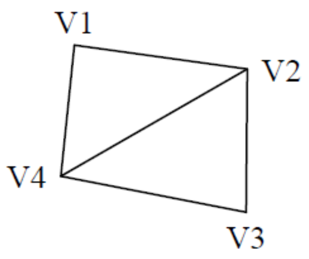
This representation uses a square matrix of order n*n, where n is number of vertices in graph.

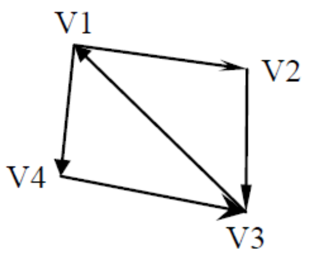
Consider a graph 'G' with 'n' number of vertices and adjacency matrix "adj".

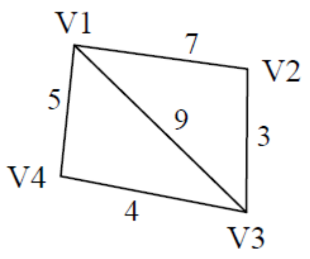
If there is an edge present between vertices Vi and Vj then adj[i][j]=1 otherwise adj[i][j]=0.

In an Undirected graph if adj[i][j]=1 then adj[j][i]=1

Eg: G1 G2 G3



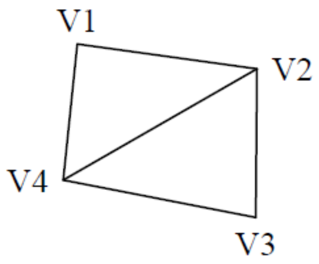


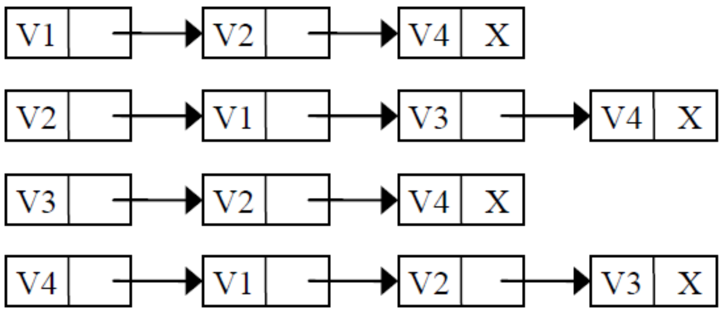


$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 \\ 4 & 1 & 1 & 1 & 0 \end{array}
\qquad
\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 1 \\ 2 & 0 & 0 & 1 & 0 \\ 3 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 \end{array}
\qquad
\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 7 & 9 & 5 \\ 2 & 7 & 0 & 3 & 0 \\ 3 & 9 & 3 & 0 & 4 \\ 4 & 5 & 0 & 4 & 0 \end{array}$$

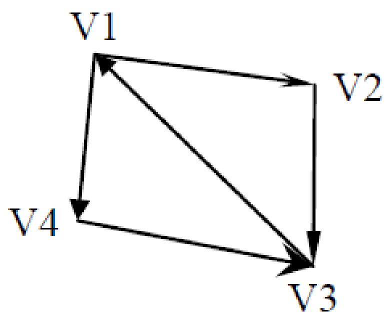
3. Adjacency list representation (or) Linked list representation:- We uses linked list representation of graphs when number of vertices of the graph is not known in advance.

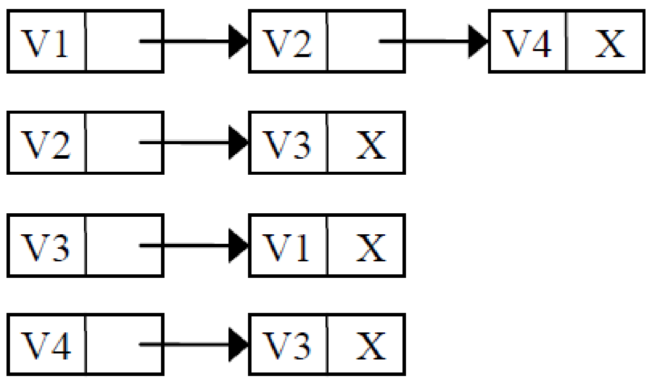
Eg1:-





Eg2:-





Graph searching techniques:- There are 2 searching techniques.

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

1.Breadth First Search (BFS):- BFS can be called as Level-by-Level traversal.

Here any vertex at level ‘i’ will be visited only after visiting all the vertices present at the preceding level i-1.

BFS uses queue data structure for its implementation.

Initially each vertex of the graph is in Unvisited State.

Algorithm:-

Input:- Adjacent matrix of a graph

Output:- BFS traversal of a graph

```
1. Take any vertex as starting vertex, enqueue that vertex and set its status as visited.
2.  While queue is not empty
        a) DEQUEUE a vertex V from queue and print it.
        b) ENQUEUE all the unvisited adjacent vertices of V into queue and set their status as visited.
```

C Program to implement BFS:-

```
#include<stdio.h>

#include<conio.h>

#define MAX 6

main()

{
```

```
            int visited[MAX]={0};

            int adj[MAX][MAX],i,j;

            printf("\nenter adjacency matrix\n");

             for(i=0;i<MAX;i++)

                        for(j=0;j<MAX;j++)

                                    scanf("%d",&adj[i][j]);

            bfs(adj,visited,0);

           getch();

}

void bfs(int adj[][MAX],int visited[], int start)

{

            int queue[MAX],rear=-1,front=-1,i;

           queue[++rear]=start;

           visited[start]=1;

            while(rear!=front)

           {

                      start=queue[++front];

                      printf("%c\t",start+65);

                      for(i=0;i<MAX;i++)

                      {

                                 if(adj[start][i]==1 && visited[i]==0)

                                 {

                                            queue[++rear]=i;

                                            visited[i]=1;

                                 }

                      }

           }
```

```
}
```

2.Depth First Search (DFS):-

The basic principle of DFS is quite sample, visit all the vertices up to the deepest level and so on.

DFS uses stack for its implementation.

Initially all the vertices of the graph are in unvisited state.

Algorithm:-

Input:-Adjacency matrix representation of a graph

Output:- DFS traversal of graph

```
1. Take any vertex as starting vertex, push that vertex onto a stack and set its status as visited.
2. While stack is not empty
   a) pop a vertex V from stack and print it.
   b) push all the unvisited neighbours of vertex onto a stack and set their status as visited.
```

C Program to implement DFS:-

```
#include<stdio.h>

#include<conio.h>

#define MAX 4

void dfs(int [][],int [],int);

main()

{

        int visited[MAX]={0};

        int adj[MAX][MAX],i,j;

        printf("\nenter adjacency matrix\n");

        for(i=0;i<MAX;i++)

            for(j=0;j<MAX;j++)

        scanf("%d",&adj[i][j]);

        dfs(adj,visited,0);

        getch();

}
```

```
void dfs(int adj[MAX][MAX],int visited[], int start)
{
        int stack[MAX],top=-1,i;
        stack[++top]=start;
        visited[start]=1;
        while(top!=-1)
       {
            start=stack[top--];
            printf("%c\t",start+65);
            for(i=0;i<MAX;i++)
            {
                 if(adj[start][i]==1 && visited[i]==0)
                 {
                      stack[++top]=i;
                      visited[i]=1;
                 }
            }
      }
 }
```