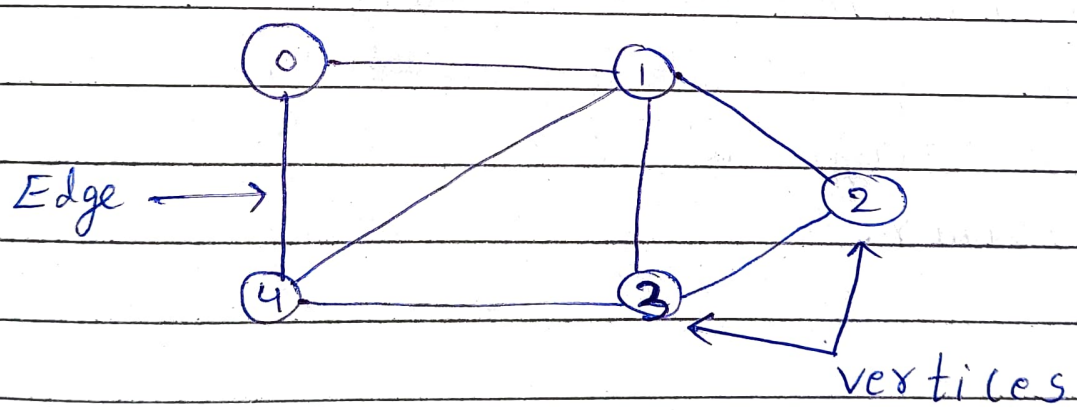
Date.....
Page.....

# DSA LAB - 11

## Theory

1) A. A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and edges are lines or arcs that connect any two nodes in the graph.

So, A Graph consists of a finite set of vertices (or) nodes and set of Edges which connect a pair of nodes.




## Basic Terminologies:

**Vertex:** The elements of graph that are connected through edges are called Vertices. These vertices are also called as nodes. These are labelled with numbers.

Date.....
Page.....

**Edge:** A path or a line between two vertices in a graph. Edge represents a path between two vertices or a line between two vertices. So, each edge can be defined with a pair of vertices. In some cases, the vertices can be same.

**Adjacency:** Two nodes or vertices are adjacent if they are connected to each other through an edge.

**Path:** Path is a sequence of edges between two nodes. It is essentially a traversal starting at one node and ending at another.

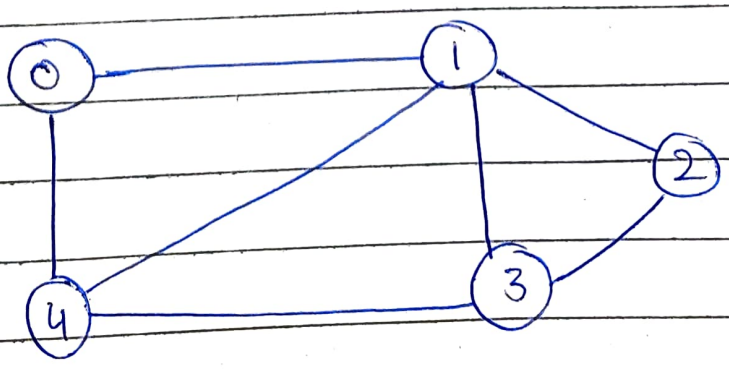
2)A. The most commonly used representations of a graph are:

i) Adjacency Matrix
ii) 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, where w is weight, then there is a edge from vertex i to vertex j with weight w.

The adjacency matrix for below graph is drawn. (For example)



Date.....
Page.....

Adjacency Matrix for above graph.

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 1 | 0 | 1 | 0 |

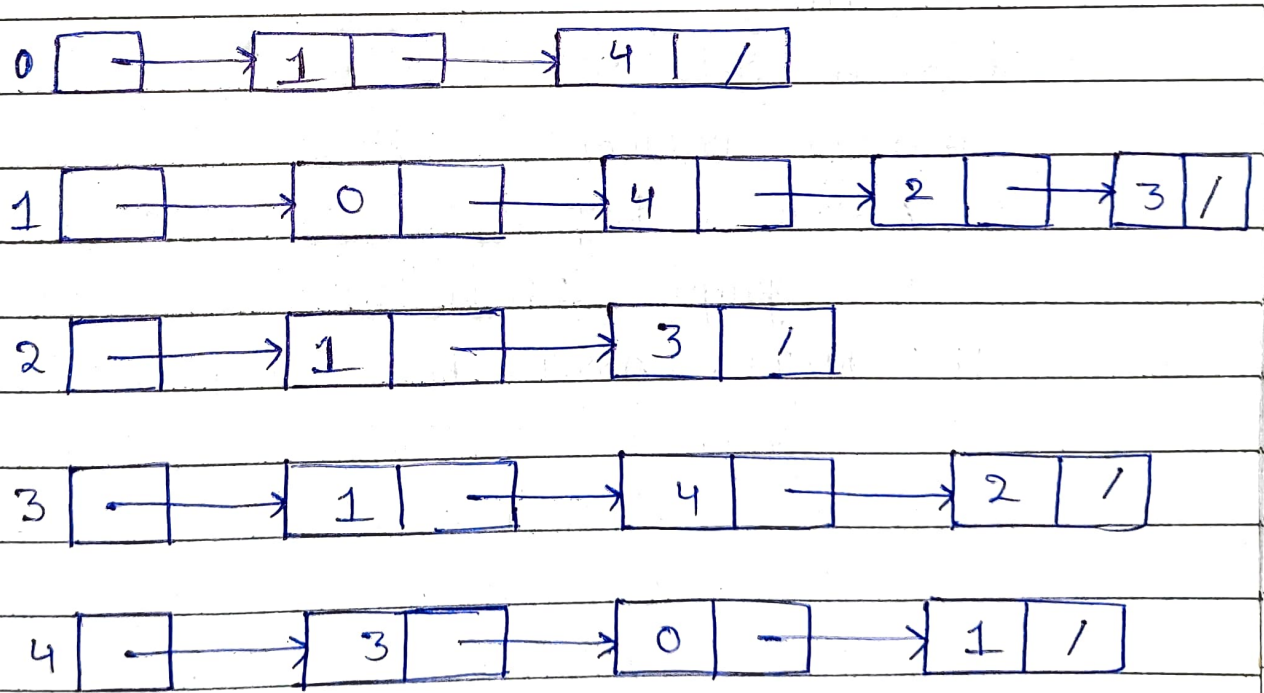
Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex 'u' to vertex 'v' are efficient and can be done O(1).

Cons: Consumes more space O(V^2). Even if the graph is sparse (contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.

Date.....
Page.....

## Adjacent 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 entire array [i] represents the list of vertices adjacent to the $i^{th}$ vertex. This representation can also be used to represent a weighed graph. The weights of edges can be represented as lists of pairs. Following is the adjacency list representation before mentioned graph.



Date.....
Page.....

3) A. Prim's Algorithm:

The idea behind Prim's algorithm is simple, a spanning time means all vertices must be connected. So the two disjoint subsets of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

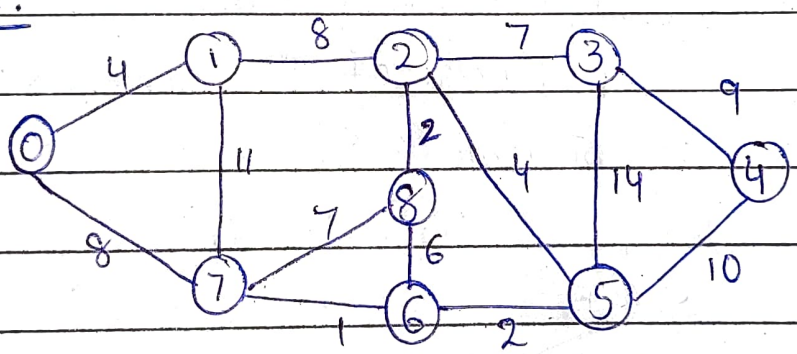
Algorithm:

1) Create Set mstSet that keeps track of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3) while mstSet doesn't include all vertices
   a) Pick a vertex u which is not there in mstSet and has minimum key value.
   b) Include u to mstSet
   c) Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every

adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v.

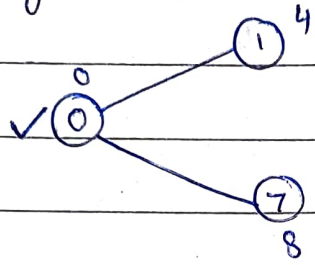
The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

Example:

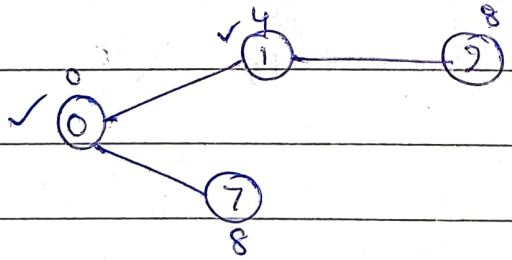



The set mstSet is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with the minimum key value. The vertex 0 is picked, include it in mstSet. So mstSet becomes {0}. After including to mstSet, update key values of adjacent vertices. Adjacent vertices 0 are 1 and 7. The key values of 1 and 7 are updated 4 and 8. Following

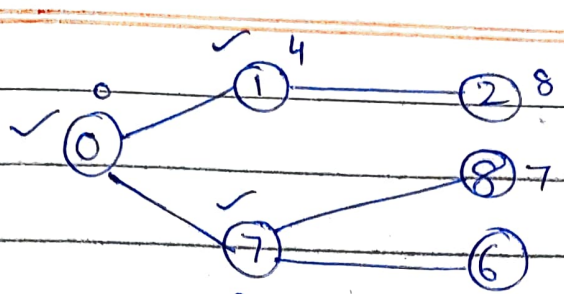
subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



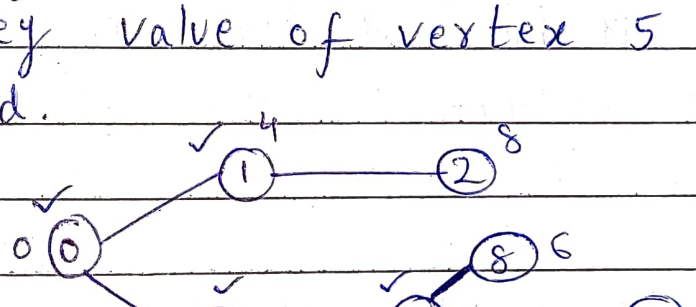

Pick the vertex with minimum key value and not already included in MST. The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



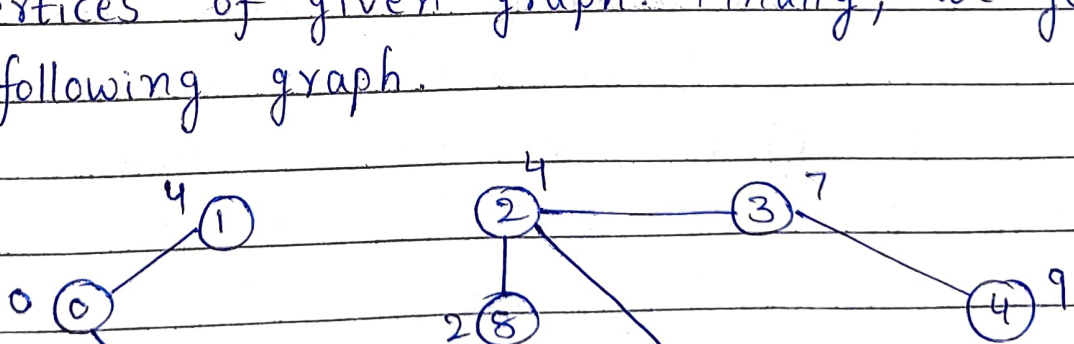

Pick the vertex with minimum key value and not already included in MST. we can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key values of vertex 6 and 8 becomes finite (1 and 7 respectively).




Pick the vertex with minimum key value and not already included in MST. vertex 6 is picked. So mstSet now becomes {0,1,7,6}. Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.




we repeat above steps until mstSet includes all vertices of given graph. Finally, we get the following graph.



Date.....
Page.....

## Krushkal's algorithm:

Krushal's algorithm finds a minimum spanning forest of an undirected edge-weighted graph. If graph is connected, it finds a minimum spanning tree. This algorithm treats the graph as a forest and every node it has an individual tree. A tree connects to another only and only if, it has at least cost among all available options and does not violate MST properties.

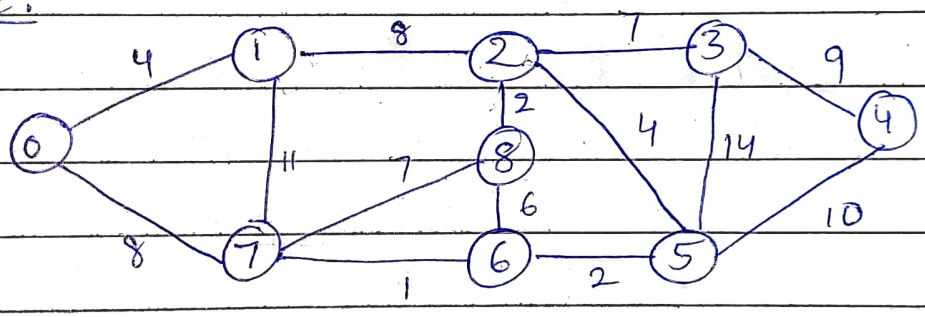
### Algorithm:

1) Sort all the edges in non-decreasing order of their weight.
2) Pick the smallest edge. check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge, Else discard it.
3) Repeat step 2 until there are (V-1) edges in spanning tree. where V is number of vertices in given graph.

Step 2 uses the Union - Find algorithm to detect cycles.

This algorithm detects cycle in graph. This algorithm is a Greedy algorithm. The Greedy choice is to pick the smallest weight edge that does not cause a cycle in MST constructed so far.

Example:




The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9-1) = 8 edges.

Date.....
Page.....

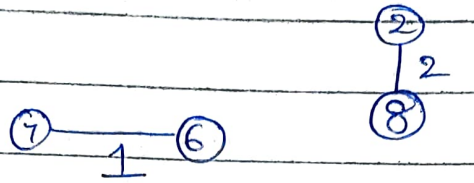
After sorting:

| Weight | Src | Dest |
|---|---|---|
| 1 | 7 | 6 |
| 2 | 8 | 2 |
| 2 | 6 | 5 |
| 4 | 0 | 1 |
| 4 | 2 | 5 |
| 6 | 8 | 6 |
| 7 | 2 | 3 |
| 7 | 7 | 8 |
| 8 | 0 | 7 |
| 8 | 1 | 2 |
| 9 | 3 | 4 |
| 10 | 5 | 4 |
| 11 | 1 | 7 |
| 14 | 3 | 5 |

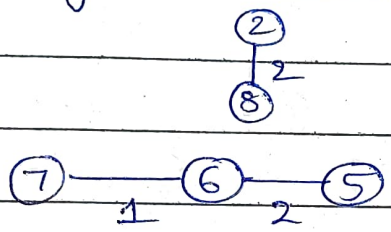
Now pick all edges one by one from the sorted list of edges.

1) Pick edge 7-6 ; No cycle is formed, include it.

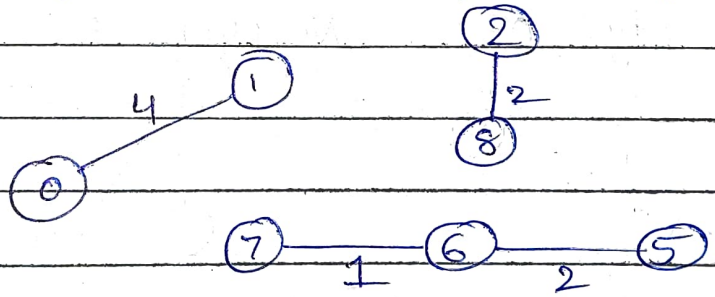
2) Pick edge 8-2 : No cycle is formed, include it.



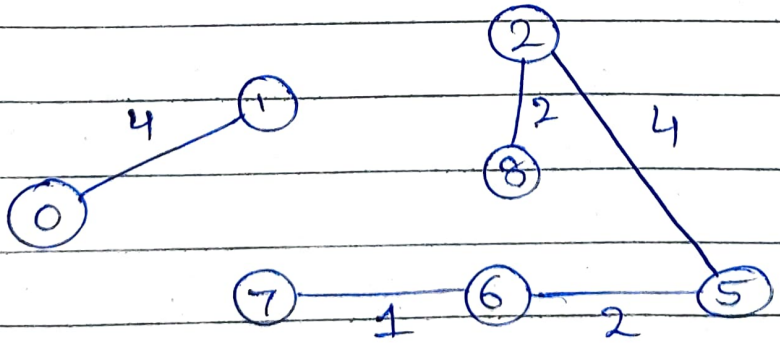

3) Pick edge 6-5 : No cycle is formed, include it.




4) Pick edge 0-1 : No cycle is formed, include it.

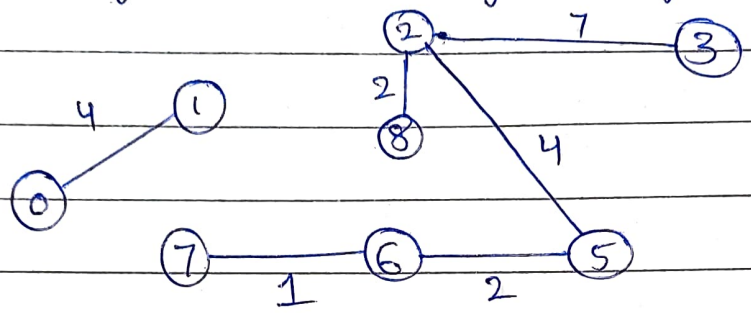



5) Pick edge 2-5 : No cycle is formed, include it.



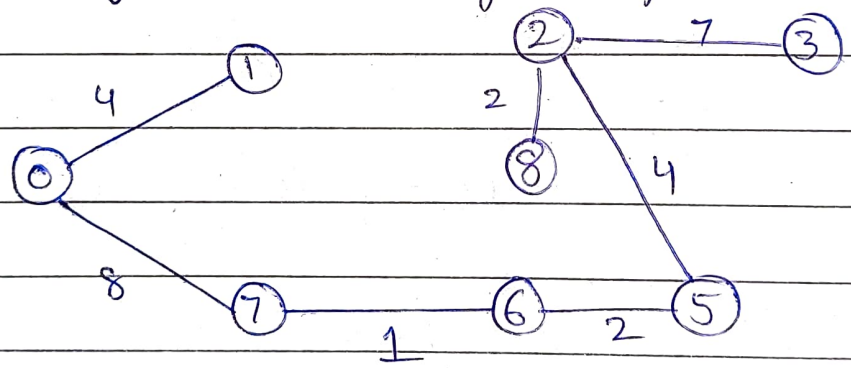
6) Pick edge 8-6 : Since including this edge results in the cycle, discard it.

7) Pick edge 2-3 : No cycle is formed, include it.



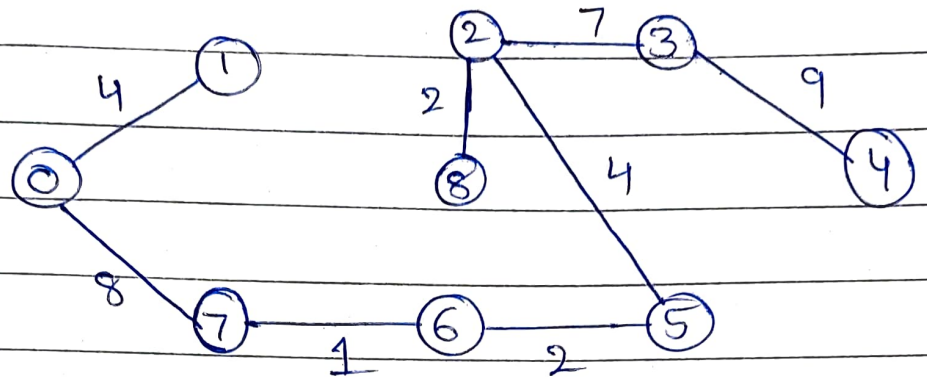

8) Pick edge 7-8 : Since including this edge results in cycle, discard it.

9) Pick edge 0-7 : No cycle is formed, include it




10) Pick edge 1-2 : Since including this edge results in cycle, discard it.

11) Pick edge 3-4 : No cycle formed, include it.




Since the number of edges equals, $(V-1 = 9-1 = 8)$, the algorithm stops here.