PICT, PUNE

Assignment 7 DSA

Name : Bhavana Bafna
Class : SE 09
Batch : E 09
Roll no. : 23107

Title : Minimum Spanning Tree

Aim : To implement MST using Prims and Kruskals algorithm

Problem Statement : Represent a graph of your college campus using adjancy list / adjancy matrix. Nodes should represent the various departments / institutes and links should represent the distance between them.

a) Using Krushakal's algorithm.
b) Using Prim's algorithm.

Analyze above two algorithm for space and time complexity.

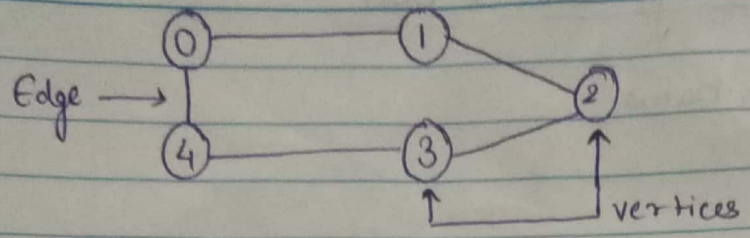
Theory:

1) Introduction to graph -

A graph is a non-linear data structure consisting of nodes and edges.

The nodes are sometimes also refered to as vertices and the edges are lines or arcs that connect any 2 nodes in graph.

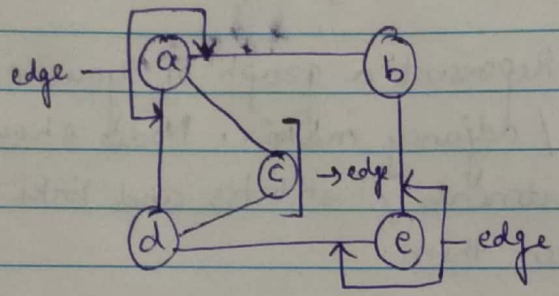
i.e., Graph can be defined as, A graph consists of a finite set of vertices and set of edges which connect a pair of nodes.



2) Terminologies in graph with diagram –

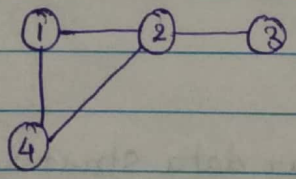
a) Edge:

An edge is one of the two basic units out of which graphs are constructed.



b) Degree:

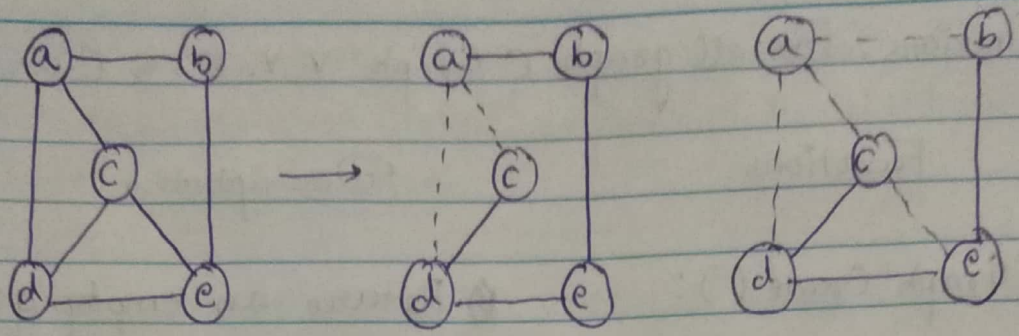
The degree of a vertex in a graph is the number of edges incident on it.



Degree of 1 = 2, degree of 2 = 3
degree of 3 = 1, degree of 4 = 2.

c) Path:

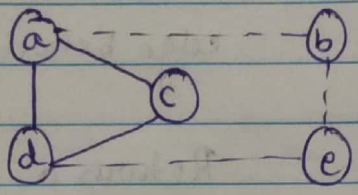
Path is a sequence of alternating vertices and edges such that each successive vertex is connected by edge.



Graph Path

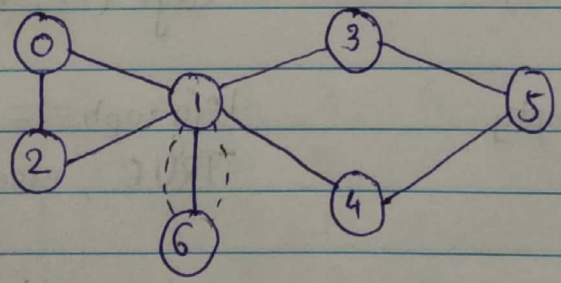
d) Cycle:

Cycle is the path that starts and end at same vertex.



e) Bridge:

A bridge is edge whose removal would disconnect the graph.



Bridge is (1, 6).

3) ADT of Graph -

Objects: A non-empty set of vertices and a set of undirected edges, where each edge is pair of vertices.

PICT, PUNE

Functions: For all graph ∈ Graph V, v_1 and v_2 ∈ vertices

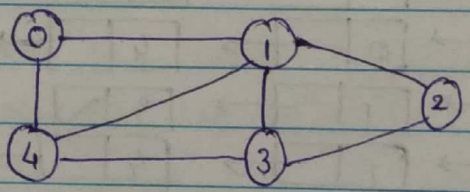
Functions	Description
1) Graph Create ( ):	Return an empty graph
2) Graph Insert vertex (graph, v)	Return a graph with v inserted, v has no incident edge.
3) Graph Insert Edge (graph, v_1, v_2)	Return a graph with new edge between v_1 and v_2
4) Graph Delete Vertex (graph, v)	Return a graph in which v and all edges incident to it removed
5) Graph Delete edge (Graph, v_1, v_2)	Return a graph in which the edge (v_1, v_2) is remove.
6) Boolean IsEmpty (graph)	If(graph == empty graph) return TRUE, else FALSE.
7) List Adjacent (graph, v)	return list of all vertices that are adjacent to V
8) Visit a Graph( ):	if Graph is not Empty, visit all vertices (DFS/BFS)

PICT, PUNE

4) Graph representation using array, diagram, advantages, disadvantages over linked organization, right and left child addresscalculation:

Adjacency matrix is a 2D array of size V×V where V is the number of vertices in graph.
Let, 2D array be adj[][], then adj[i][j]=1 if there is an edge from i to j.
Adjancency matrix for undirected graph is symmetric.
It is ~~the~~ also used to represent weighted graphs.

Diagram:



Adjancy Matrix:

	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

Advantages:

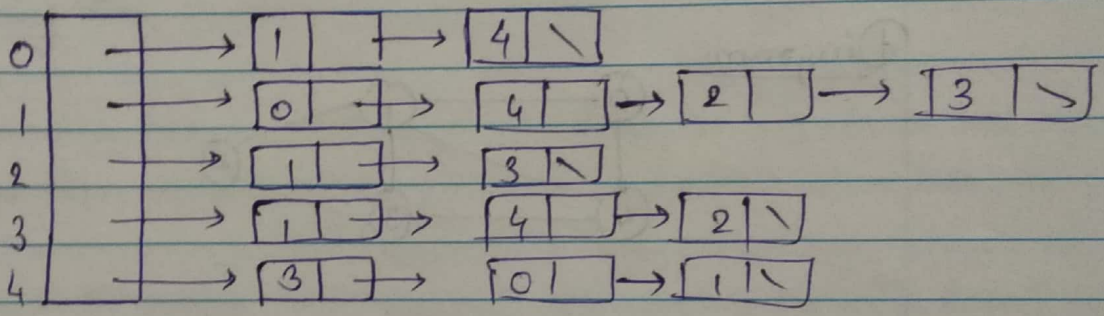
1) Representation is easier to follow.
2) Removing an edge take O(1) time.

PICT, PUNE

2) Disadvantages:
- Consumes more space.
- Takes more time to add on vertex

Graph representation using linked organization:

- An array of list is used.
- The size of array is equal to number of vertex.
- Let, arr[ ] be an array, an entry arr[i] represent list of vertices adjacent to i^{th} vertex.



Advantages:

1. Saves space
2. Adding vertex is easy

Disadvantages:

Find whether an edge is available or not takes more time.

PICT, PUNE

6) Application of graph:

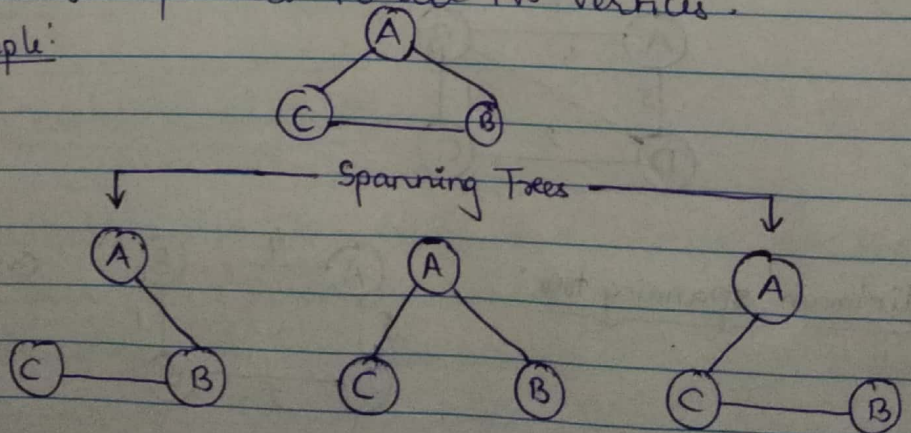
1) Models for communications and electrical networks.
2) Models for computer architectures.
3) Network optimization models for operations analysis, including scheduling and job assignment.
4) Analysis of finite state machines.
5) Parsing and code optimization in compilers.

7) Spanning tree concept and example:

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges.
Hence, a spanning tree does not have cycles and it cannot be disconnected.

By this definition we can draw a conclusion that every connected & undirected Graph G has at least one spanning tree.
A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.

Example:



PICT, PUNE

General properties of spanning tree:

1) A connected graph G can have more than one spanning tree.
2) The spanning tree does not have any cycle.
3) All possible spanning trees of graph G, have the same number of edges & vertices.
4) Adding one edge to the spanning tree will create a circuit or loop.

Application of Spanning tree:

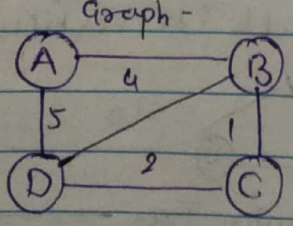
a) Civil Network Planning.
b) Computer Network Routing Protocol.
c) Cluster Analysis.

8) Minimum Spanning Tree and its use cases:

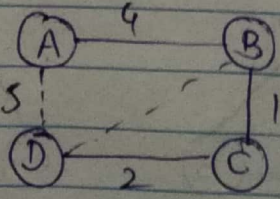
In a weighed graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.

Minimum spanning tree example:

Graph -



Minimum spanning tree:



Cost = 7(4+1+2)

PICT, PUNE

Application of minimum spanning tree:

1. To find paths in the map.
2. To design network like telecommunication networks and electrical grids.

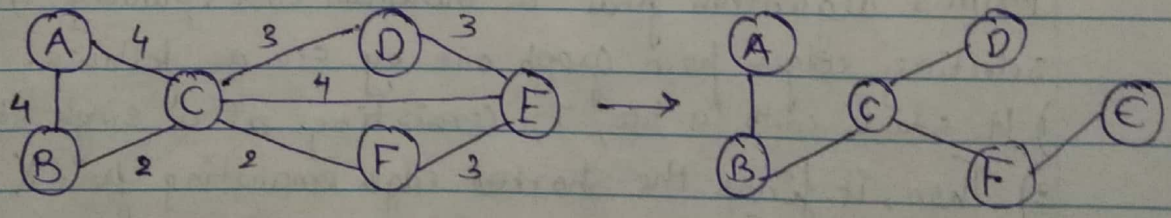
The minimum spanning tree from a graph is found using following algorithms -

1. Prim's algorithms
2. Kruskal's algorithms

1) Prim's Algorithm:

Prim's algorithm to find minimum cost spanning tree uses the greedy approach.
Prim's algorithm treats the nodes as a single tree and keeps on adding new nodes to spanning tree from given graph.

Example:

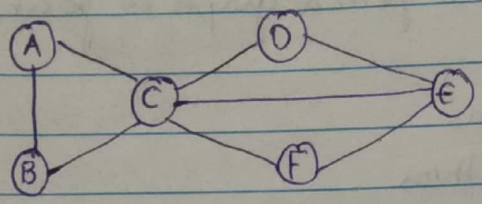


2) Kruskal's algorithm:

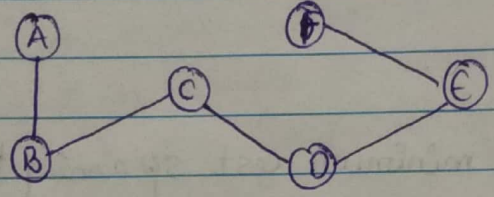
Kruskal's algorithm to find minimum cost spanning tree uses the greedy approach.
This algorithm treats the graph as a forest and every node

it has as individual tree.
A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.

Example:



- given graph



- Result from Kruskal's algorithm.

7) Algorithm / pseudocodes:

1. Prim's algorithm -

Prim's algorithm finds a minimum cost spanning tree by selecting edges from graph one by one as follows:

1) It starts with a tree, T, consisting of a single starting vertex, a.
2) Then, it finds the shortest edge emanating from X that connects T to the rest of the graph.
3) It adds this edge and the new vertex to tree T.
4) It then picks the shortest edge emanating from the revised tree T that also connects T to the rest of the graph and repeats the process.

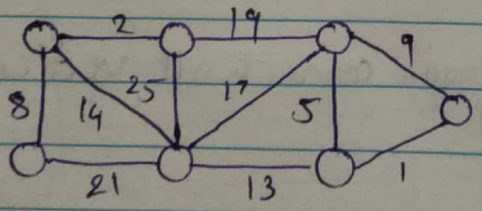
Pseudocode - Prim's Algorithm -

1) For each node v, set v-costs = ∞ and v-known = false.
2) Choose any node v:
   a) Mark v as known.
   b) For each edge (v, u) with weight w; set u.cost = w and u.prev = v.
3) While there are unknown nodes in the graph
   a) Select the unknown node v with lowest cost.
   b) Mark v as known & add (v, v.prev) to output.
   c) for each edge (v, u) with weight w,

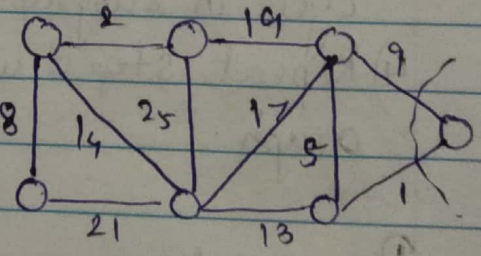
```
if (w < u.cost)
{
  u.cost = w
  u.prev = v
}
```

Example:

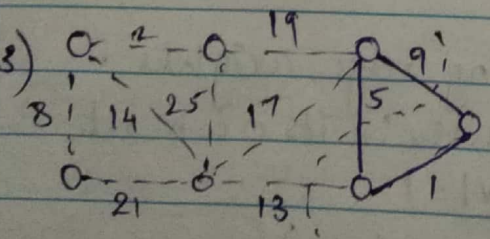
1)



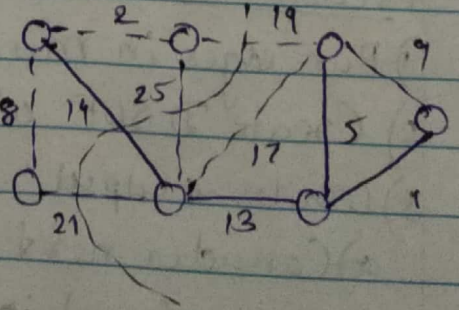
2)



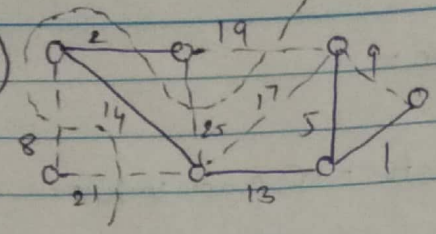
3)



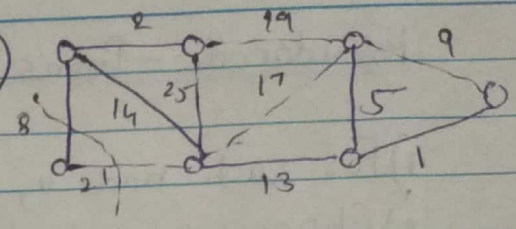
4)



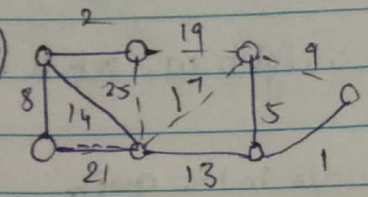
5)



6)



7)



2. Kruskal's algorithm -

Algorithm -

1) First edge choose any edge with minimum weight.
2) Next edge choose any edge with minimum weight from those not yet selected.
3) Continue to choose edges of minimum weight from those not yet selected, except do not select any edge that creates a cycle in subgraph.
4) Repeat step 3 until subgraph connects all vertices of original graph.

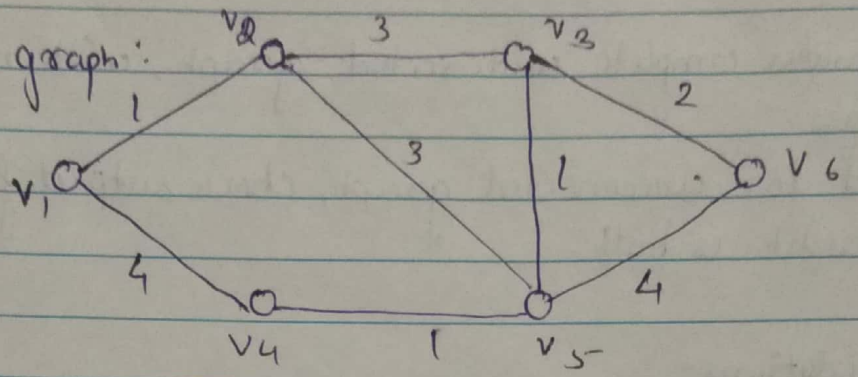
Pseudocode -

1) Put edges in min-heap using edge weights.
2) Create DSUF with each vertex in its own set
3) While output size $< |V| - 1$
   a) Consider next smallest edge (u, v).
   b) If find (u, v) indicates u and v are in different sets.
      - output (u, v)
      - union (u, v)

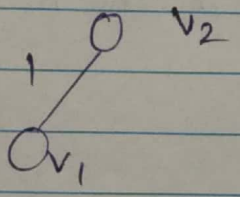
PICT, PUNE

Example-

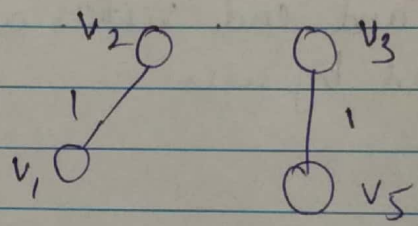
Given graph:



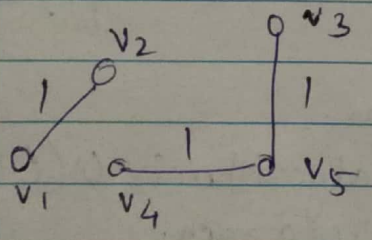
After 1st iteration -



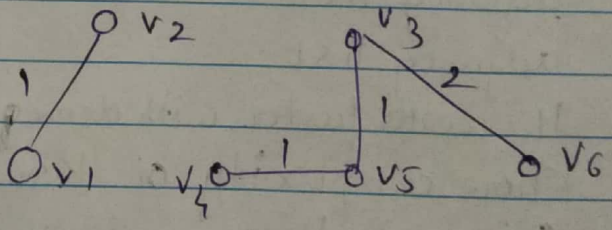
After second iteration -



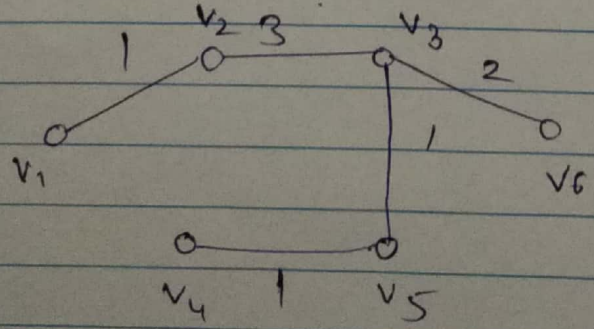
After 3rd iteration -



After 4th iteration -



After 5th iteration -



PICT, PUNE

Test cases-

1) Consider complete undirected graph, with no loop parallel edges.
2) Test for unconnected graph, check output for prims and Krushkals both.

Validations-

1) No. of veiter & no. of edges are positive integer no.
2) Test for -ve weight.
3) Start and end vertices are within the no. of users provided by user.

Conclusion-

Time complexity of Prim's algorithm is $O(V^2)$ for adjancy matrix. It can be reduced to $O(E \log V)$ with adjancy list.
It works faster with dense graphs.
Time complexity of Krushkal's algorithm is $O(E \log E)$ or $O(E \log V)$. It runs faster in sparse graph.