Rajdhani
DATE / /

Roll No. 23355

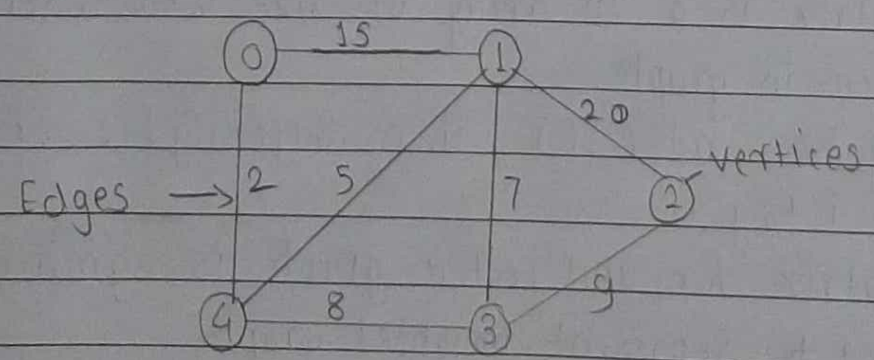
Title : Assignment 7 : Minimum Spanning Tree

Aim : To implement minimum spanning tree using prims and kruskals algorithm.

Problem Statement : Represent a graph of your college campus using adjacency list / adjacency matrix.
Nodes should represent the various departments / institutes and link should represent the distance between them. Find minimum spanning tree using
a) kruskal's algorithm
b) Prim's algorithm
Analyze above two algorithms for space & time complexity

Theory :

Introduction to graph :
- A graph is a non linear data structure consisting of nodes and edges. The nodes are also referred as vertices & the edges are lines or arcs that connect any two nodes in graph.
- Graph consist of finite set of vertices (or nodes) and set of edges which connect a pair of nodes.



- Graph terminologies:

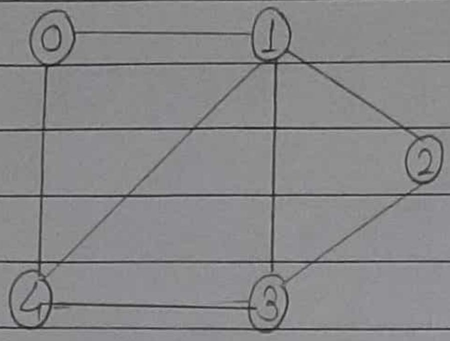
A graph is an ordered pair $G = (V, E)$ comprising a set V of vertices & E edges.
If edges are directed, then it is a directed graph else it is called as undirected graph.
The value assigned to edges is called as weight.

- ADT of Graph

objects: a nonempty set of vertices and set of edges, where each edge is pair of vertices.
functions: for all graph ∈ Graph, v, v_1 and v_2 ∈ vertices

Graph create()
Graph addvertex(graph v)
Graph addEdge(graph, v_1, v_2)
Graph deletevertex(graph, v)
Graph deleteEdge(graph, v_1, v_2)
bool isEmpty(graph)

- Graph representation using ~~or~~ 2D-Array (Matrix)
- Adjacency Matrix is a 2D array of size $V \times V$ 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.
- Adjacency matrix for undirected graph is symmetric.
- It is also used to represent weighted graphs.



Adjacency 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

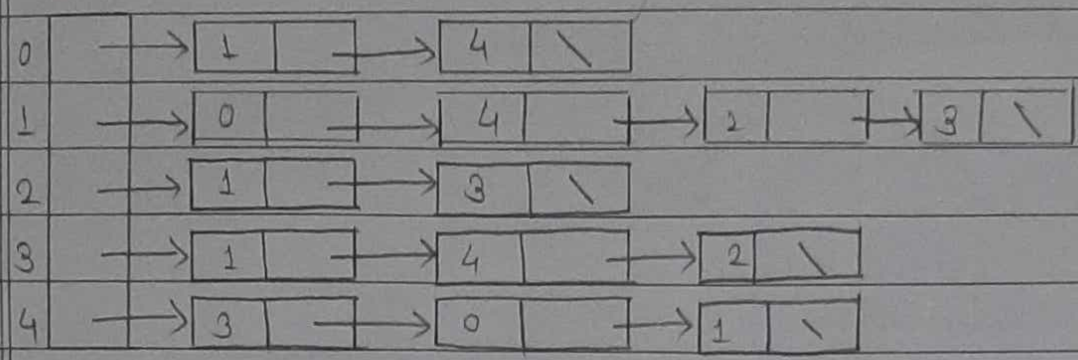
Advantages :

i) Representation is easier to follow
ii) Removing an edge take O(1) time

Disadvantages :

i) Consumes more space.
ii) Takes more time to add an ~~edge~~ vertex

- Graph representation using adjacency list :
- An array of list is used.
- The size of array is equal to number of vertices
- Let arr[] be an array. An entry arr[i] represent list of vertices adjacent to ith vertex.



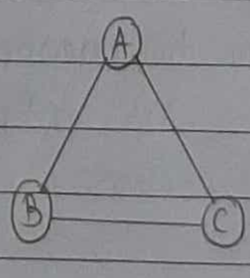
Advantages:
Saves space
Adding vertex is easy

Disadvantage:
Find whether an edge is available or not takes more time

Applications of graph:
1) Maps
2) Computer network
3) Social networking sites
4) To represent flow of computations.

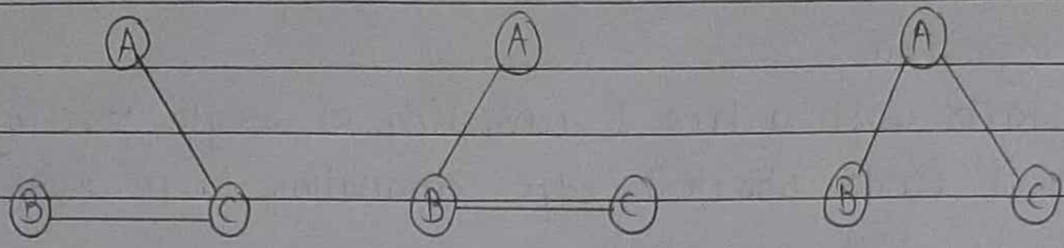
Spanning tree concept:
A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges.
A spanning tree does not have cycles & it cannot be disconnected

Example:



Graph G

Spanning trees :



- Minimum spanning tree :

A minimum spanning tree (MST) for a weighted, connected undirected graph is spanning tree with weight less than or equal to weight of every other spanning tree.
The weight of a spanning tree is the sum of weights given to each edge of spanning tree.

Uses of MST :

1) Telephone wiring
2) Electronic circuits
3) Computer networks.

Algorithm :

1) Prim's algorithm :

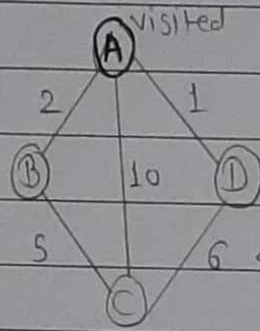
i) It starts with a tree T, consisting of single starting vertex x

ii) Then it finds shortest edge emanating from x that connects T to the rest of graph

iii) It adds this edge & new vertex to tree T.

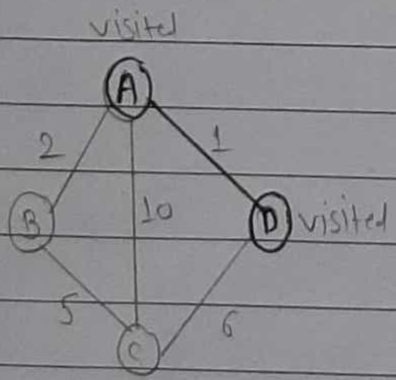
iv) It then picks the shortest edge emanating from the revised tree T that also connects T to the rest of the graph and repeats the process, till all vertices are visited.

Example:

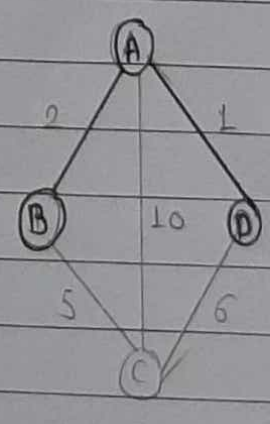


← Step 1

1. Mark A as visited
2. Find the minimum weighted edge connected to vertex A & mark other vertex on this edge as visited.



step 2

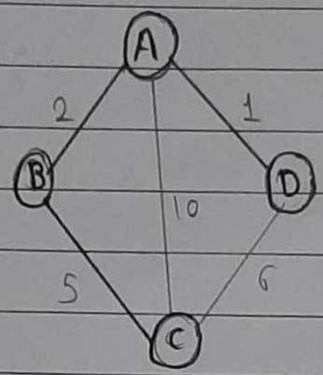


Step 3

Rajdhani
DATE / /

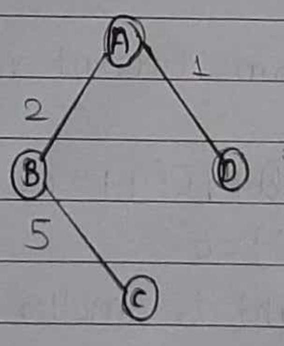
3. Find the minimum weighted edge connected to vertices {A, D} and mark other vertex on this edge as visited

4. Find the minimum weighted edge connected to vertices {A, D, B} & mark other vertex on this edge as visited.



Step 4

5. All vertices are visited. MST is:



Total cost = 8

Algorithms with adjacency matrix:

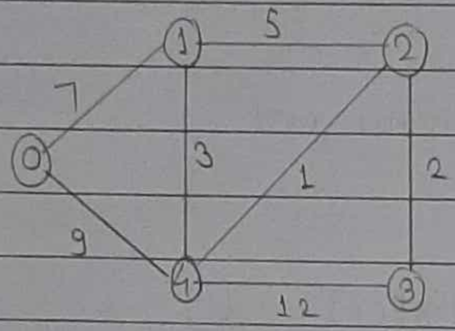
```
totalvisited = 0
~~visi~~ for i <- 0 to V
        dist [i] = 5000
        visited [i] = 0
        path [i] = 0

// Start from vertex 0
current = 0
visited [current] = 1
totalvisited = 1
distance [current] = 0

while totalvisited ≠ V
    // find distance from current vertex to all other
    for i <- 0 to V
        If weight [current] [i] != 0
            If visited [i] = 0
                // If weight is smaller than previous distance,
                // replace it
                If weight [current] [i] < dist [i]
                    dist [i] = weight [current] [i]
                    path [i] = current
    End for

    // find edge with minimum distance
    mincost = 32767
    ~~for i <- 0 to V~~
```

```
for i <- 0 to V
    If visited[i] = 0
        If dist[i] < mincost
            mincost = dist[i]
            current = i
End for
// mark current vertex as visited
visited[current] = 1
totalvisited = totalvisited + 1
End while

//Display MST & calculate Total cost
cost = 0
for i <- 0 to V
    print i, path[i], dist[i]
    cost = cost = dist[i]
End for
print cost

End
```

- kruskal's algorithm:

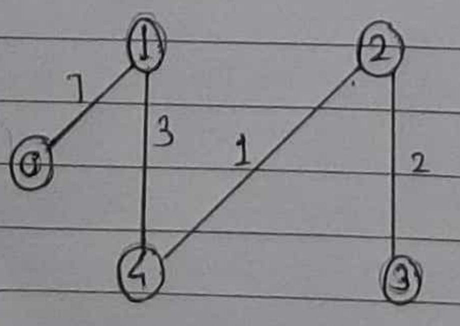
1. Sort all the edges in ascending order of weight
2. Pick the smallest edge. check if it forms cycle with spanni tree formed. If cycle is not formed, include this edge Else discard it
3. Repeate step 2 till there are (V-1) edges.

Example:



1. smallest edge is from (2, 4) select it
2. Select (2, 3)
3. select (1, 4)
4. (1, 2) will create cycle, discard it
5. select (0, 1)
6. There are v-1 i.e. 4 edges, so stop.

Spanning Tree:



Total cost = 13

- Algorithm:

```
// edge represents u ~~as~~ & v as vertices & w as weight
// Sort the edges
int connt [E] // E is total no. of edges
val = 1 , cnt = 0 , j = 0

while cnt < E-1 and j < E
    // If both vertices are not visited
    If connt [edge [j]. u] = connt [edge [j]. v] = 0
        // Select edge
        temp [cnt] = edge [j]
        connt [edge [j]. u] = connt [edge [j]. v]] = val
        val = val + 1
         cnt = cnt + 1
    // If both vertices have different connection values
    ELSE If connt [edge [j]. u] ≠ connt [edge [j]. v]
        // Select edge  temp [cnt] = edge [j]
        // if both vertices are visited, make same connection
        // values
        for i = 0 to E
            connt [i] = connt [edge [j]. u]

      // if first vertex is visted only
    Else If connt [edge [j]. u] != 0 and connt [edge [j]. v] = 0
            ~~connt [v]~~ connt [edge [j]. v] = connt [edge [j]. u]]

    Else
        connt [edge [j]. u] = connt [edge [j]. v]
    cnt = cnt + 1
```

Rajdhani
DATE / /

// If both vertices have same connect value, reject it
ELSE
    // reject

j = j + 1

End while

// display MST
for i = 0 to cnt
    print temp[i].u, temp[i].v, temp[i].w
    cost = cost + temp[i].w
End for
print cost

End

- Test cases :
1) complete undirected graph with no loops, parallel edges
2) connected undirected graph with no loops, parallel edges

- validations
1) No. of vertex & No. of edges are positive integers
2) Start & end vertices are within the number of vertices provided by user

Rajdhani
DATE / /

Conclusion:
Time's complexity of prim's algorithm is $O(V^2)$ for adjacency matrix. It can be reduced to $O(E \log V)$ with adjacency list. It works faster with dense graphs.
Time complexity of kruskal's algorithm is $O(E \log E)$ or $O(E \log V)$. It runs faster in sparse graphs.