**Name : Omkar Nikhal**

**Class : SE-3**

**Roll no : 21342**

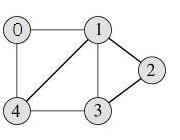
**ASSIGNMENT NO.** 5

|  |  |
| --- | --- |
| Title: | To write a program for Graph creation and find its minimum cost using Prim’s or Kruskal’s algorithm. |
| Problem Statement: | You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures. |
| Objective: | * To understand concept of graph & minimum cost spanning tree. * To understand different minimum cost spanning tree algorithms. * To implement minimum spanning tree algorithms. |
| Outcome: | * Graph implementation using Adjacency matrix or Adjacency list * Find total minimum cost using MST Algorithm |
| S/W Packages and H/W Apparatus used: | 1. 64-bit Fedora 17 or latest 64-Bit update of equivalent open source OS 2. Programming tools (64-Bit) and latest open source update of Eclipse Programming framework, TC++, GTK++ |

Theory

Representation of graph:

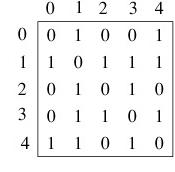
Following is an example undirected graph with 5 vertices.



Using Adjacency Matrix

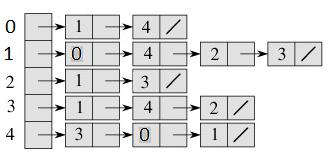
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:



Using Adjacency List

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ith vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists. Following is adjacency list representation of the above graph.



Using Adjacency List

Give a graph G = (V, E), the minimum spanning tree (MST) is a weighted graph G’ = (V, E’) such that:

* E’ Í E
* G’ is connected
* G’ has the minimum cost

A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible. More generally, any undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of the minimum spanning trees for its connected components. There are quite a few use cases for minimum spanning trees. One example would be a telecommunications company which is trying to lay out cables in new neighborhood.

Prim’s Algorithm

1. Select any vertex
2. Select the shortest edge connected to that vertex
3. Select the shortest edge connected to any vertex already connected
4. Repeat Step 3 until all vertices have been connected

Kruskal’s Algorithm

1. Enter number of cities (vertices in graph)
2. Enter the cost of connectivity between each pair of cities (edges in graph)
3. Initialize cost\_of\_connectivity = 0
4. Sort all the edges in non-decreasing order of their cost
5. Pick the smallest cost edge
6. Check if it forms a cycle with the already included edges in the MST
7. If cycle is not formed, include this edge in MST, else discard it
8. Add weight of the selected edge to the cost\_of\_connectivity
9. Repeat step 5, 6, 7 until there are (v-1) edges in the graph
10. cost\_of\_connectivity will have minimum cost in the end

Algorithms

Prim’s Algorithm

// input: a graph G

// output: E: an MST for G

1. Select a starting node, v
2. T ß {v} //the nodes in the MST
3. E ß {} //the edges in the MST
4. While not all nodes in G are in the T do

Choose the edge v’ in G − T such that there is a v in T:

weight (v,v’) is the minimum in

{weight(u,w) : w in G − T and u in T}

T ß T È {v’}

E ß E È {(v,v’)}

1. return E

Kruskal’s Algorithm

// input: a graph G with n nodes and m edges

// output: E: an MST for G

* EG[1…m] // Sort the m edges in G in increasing weights
* E{} // The edges in the MST
* i = 0 // Counter for EG
* While |E| < n-1, do
  + If adding EG[i] to E does not add a cycle, then
    - E.push\_back(EG[i])
    - i++
* Return E

Test-Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Desciption | Input | Output | Result |
| Create Graph | Cities: 5  City 1: Mumbai  City 2: Pune  City 3: Chennai  City 4: Delhi  City 5: Kolkata  Src Dest Dist  1 2 1  1 3 2  1 4 4  1 5 4  2 4 3  2 3 1  3 4 5  4 5 3 | - | Pass |
| Display Graph | - | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | Mumbai | Pune | Chennai | Delhi | Kolkata | | Mumbai | ∞ | 1 | 2 | 4 | 4 | | Pune | 1 | ∞ | 1 | 3 | ∞ | | Chennai | 2 | 1 | ∞ | 5 | ∞ | | Delhi | 4 | 3 | 5 | ∞ | 3 | | Kolkata | 4 | ∞ | ∞ | 3 | ∞ | | Pass |
| Prim’s MST | - | Edge 1:(1, 2) Cost: 1  Edge 2:(2, 3) Cost: 1  Edge 3:(2, 4) Cost: 3  Edge 4:(4, 5) Cost: 3  Minimum cost = 8 | Pass |
| Kruskal’s MST | - | Edge 1:(1, 2) Cost: 1  Edge 2:(2, 3) Cost: 1  Edge 3:(2, 4) Cost: 3  Edge 4:(4, 5) Cost: 3  Minimum cost = 8 | Pass |

Conclusion

We have successfully calculated total minimum cost of graph using minimum spanning tree algorithm.