**Graph algorithms - practical work no. 4**

Andreas R. Toadere **C++** | Group 917

Problem 6: Write a program that, given an undirected connected graph, constructs a minimal spanning tree using the Prim's algorithm.

**Prim’s algorithm and its complexities:**

Like Kruskal’s algorithm, Prim’s algorithm is also a [Greedy algorithm](https://www.geeksforgeeks.org/archives/18528). It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.  
A group of edges that connects two set of vertices in a graph is called [cut in graph theory](http://en.wikipedia.org/wiki/Cut_%28graph_theory%29). So, at every step of Prim’s algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).

**How does Prim’s Algorithm Work?**

 The idea behind Prim’s algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) 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 a 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](http://en.wikipedia.org/wiki/Cut_(graph_theory)). 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.

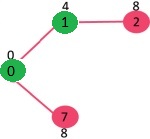
Let us understand with the following example:



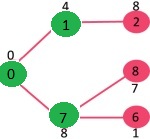
The set mstSet is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum key value. The vertex 0 is picked, include it in mstSet. So mstSet becomes {0}. After including to mstSet, update key values of adjacent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



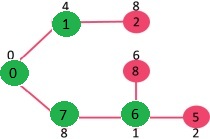
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



Pick the vertex with minimum key value and not already included in MST (not in mstSET). 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 value of vertex 6 and 8 becomes finite (1 and 7 respectively).



Pick the vertex with minimum key value and not already included in MST (not in mstSET). 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 the above steps until mstSet includes all vertices of given graph. Finally, we get the following graph.



**Time Complexity** of the algorithm is O(V^2). If the input [graph is represented using adjacency list](https://www.geeksforgeeks.org/archives/27134), then the time complexity of Prim’s algorithm can be reduced to O(E log V) with the help of binary heap.

**The code of the program**

// Function to construct and print MST for   
// a graph represented using adjacency   
// matrix representation   
void primMST(int graph[V][V])  
{  
 // Array to store constructed MST  
 int parent[V];  
 // Key values used to pick minimum weight edge in cut  
 int key[V];  
 // To represent set of vertices not yet included in MST  
 bool mstSet[V];  
  
 // Initialize all keys as INFINITE  
 for (int i = 0; i < V; i++)  
 key[i] = INT\_MAX, mstSet[i] = false;  
  
 // Always include first 1st vertex in MST.  
 // Make key 0 so that this vertex is picked as first vertex.  
 key[0] = 0;  
 parent[0] = -1; // First node is always root of MST  
  
 // The MST will have V vertices  
 for (int count = 0; count < V-1; count++)  
 {  
 // Pick the minimum key vertex from the  
 // set of vertices not yet included in MST  
 int u = minKey(key, mstSet);  
  
 // Add the picked vertex to the MST Set  
 mstSet[u] = true;  
  
 // Update key value and parent index of  
 // the adjacent vertices of the picked vertex.  
 // Consider only those vertices which are not  
 // yet included in MST  
 for (int v = 0; v < V; v++)  
  
 // graph[u][v] is non zero only for adjacent vertices of m  
 // mstSet[v] is false for vertices not yet included in MST  
 // Update the key only if graph[u][v] is smaller than key[v]  
 if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])  
 parent[v] = u, key[v] = graph[u][v];  
 }  
}