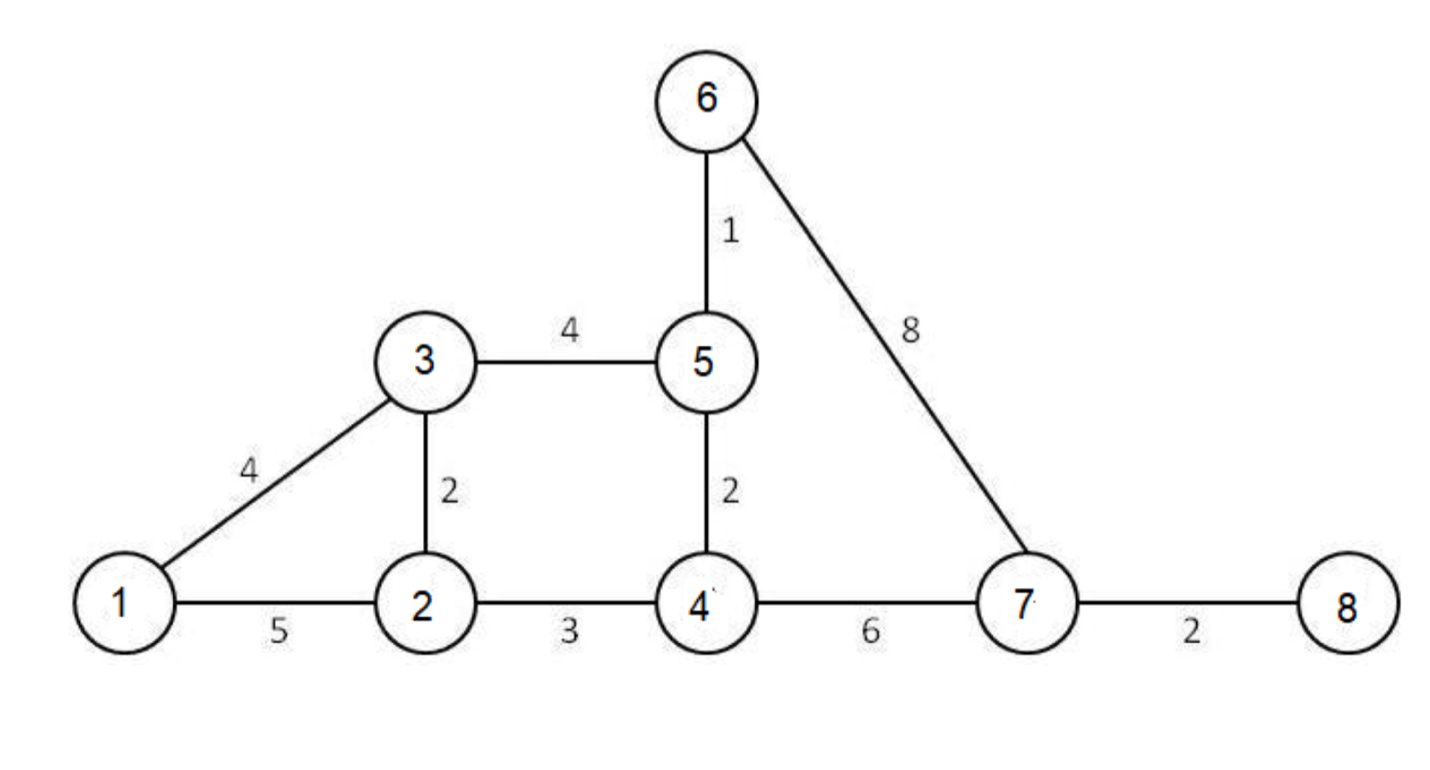
Programming Assignment Unit 4

Develop an implementation of Prim’s algorithms that determines the MST (Minimum Spanning Tree) of the graph from the Unit 2 assignment that we developed the data structure for.



For this assignment, develop an implementation using Java in the Cloud9 environment (or your own Java IDE) that first implements the graph in a data structure and then provides the algorithm that can determine the Minimum spanning tree within this graph in terms of cost. The cost will be the sum of the lengths of the edges that must be traversed. The cost of each edge is represented by the number on the edge. For example, the cost of edge 1,3 is 4 and the cost of edge 6,7 is 8. Your algorithm must output the total cost of spanning the tree as determined by your implementation of Prim’s algorithm.

The algorithm must produce output which is the total cost of the path.

Assessment

You will have ONE WEEK to complete this assignment. It will be due the end of this unit. Your assignment will be assessed (graded) by your peers. You should post this assignment, the results, and other requirements such as the asymptotic analysis in one of the following formats:

Directly cut-and-pasted into the text box for the posting. As a document in either RTF or Word 97/2003 format.

Mark each rubric item using the scale provided. Some scales will be simple yes/no responses. Others will be a scale of 1-4 where 4 indicates that all of the elements of the rubric item are present and 1 indicates that NONE of the elements were present. For example, in the rubric item “Was a java implementation of a minimum spanning tree provided”, then if no code was submitted at all then this item should be 1. If an algorithm was posted (not necessarily working as that would be another rubric item) that attempts to implement a minimum spanning tree traversal, then this item should be 4.

Ans:

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Graph

{

private int V; // No. of vertices

private int graph[][]; // matrix representation of graph

// A utility function to find the vertex with minimum key

// value, from the set of vertices not yet included in MST

int minKey(int key[], Boolean mstSet[])

{

int min = Integer.MAX\_VALUE, min\_index=-1;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min)

{

min = key[v];

min\_index = v;

}

return min\_index;

}

// Function to construct and print MST for a graph represented

// using adjacency matrix representation

void primMST()

{

int parent[] = new int[V];

int key[] = new int [V];

Boolean mstSet[] = new Boolean[V];

int totalCost = 0;

for (int i = 0; i < V; i++)

{

key[i] = Integer.MAX\_VALUE;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1; // First node is always root of MST

for (int count = 0; count < V-1; count++)

{

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

if (graph[u][v]!=0 && mstSet[v] == false &&

graph[u][v] < key[v])

{

parent[v] = u;

key[v] = graph[u][v];

}

}

for (int i = 1; i < V; i++)

totalCost += graph[i][parent[i]];

System.out.println("Total cost of the path: " + totalCost);

}

public Graph(int v)

{

V = v;

graph = new int[V][V];

}

public void addEdge(int src, int dest, int cost)

{

graph[src][dest] = cost;

graph[dest][src] = cost;

}

public static void main (String[] args)

{

Graph g = new Graph(5);

g.addEdge(0, 1, 2);

g.addEdge(0, 3, 6);

g.addEdge(1, 2, 3);

g.addEdge(1, 3, 8);

g.addEdge(1, 4, 5);

g.addEdge(2, 4, 7);

g.addEdge(3, 4, 9);

g.primMST();

}

}