Department of
Computer Science and Engineering

Title: Implement Prim's Algorithm

Algorithms Lab
CSE 206

Green University of Bangladesh

1 Objective(s)

- To learn Prim's algorithm to find MST of a graph.

2 Problem Analysis

2.1 Prim's Algorithm

Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- form a tree that includes every vertex.
- has the minimum sum of weights among all the trees that can be formed from the graph.

2.2 How Prim's algorithm works

It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum. We start from one vertex and keep adding edges with the lowest weight until we reach our goal. The steps for implementing Prim's algorithm are as follows:

- Initialize the minimum spanning tree with a vertex chosen at random.
- Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree.
- Keep repeating step 2 until we get a minimum spanning tree.

2.3 Example of Prim's algorithm

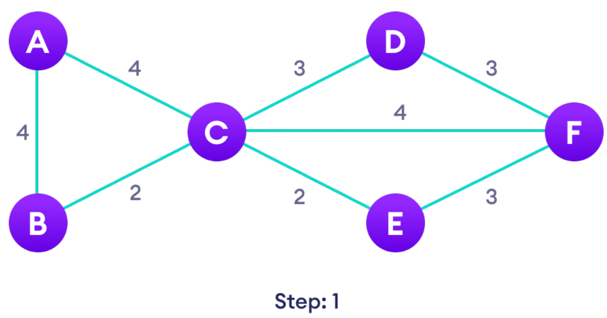
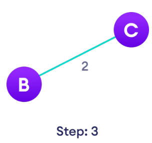


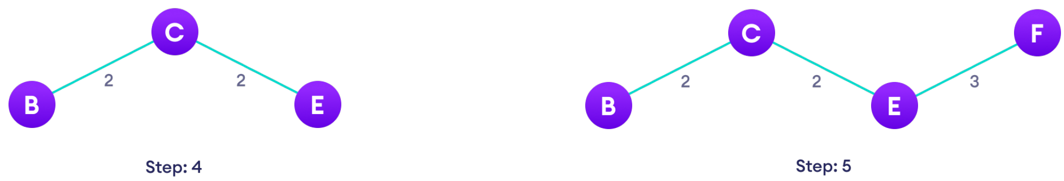
Figure 1: Start with a weighted graph

(a) Choose the edge with the least weight, if there are more than 1, choose anyone



(b) Choose the next shortest edge and add it

Figure 2: Step 2 and 3



(a) Choose the next shortest edge that doesn't create a cycle and add it (b) Choose the next shortest edge that doesn't create a cycle and add it

Figure 3: Step 4 and 5

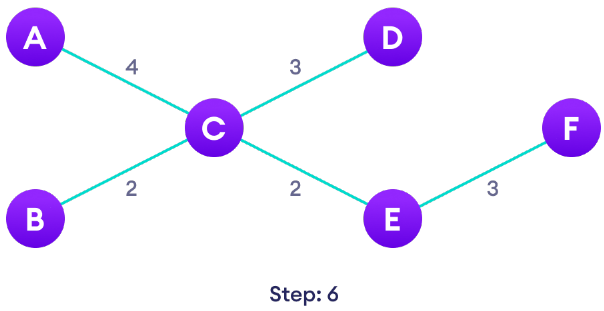


Figure 4: Repeat until you have a spanning tree

3 Algorithm

Algorithm 1: Prim's Algorithm

T = $\emptyset$;
U = 1 ;
while *($U \neq V$)* **do**
    let (u, v) be the lowest cost edge such that u $\in$ U and v $\in$ V - U;
    T = T $\cup$ (u, v)
    U = U $\cup$ v
end

4 Implementation in Java

```java
// A Java program for Prim's Minimum Spanning Tree (MST) algorithm.
// The program is for adjacency matrix representation of the graph

import java.util.*;
import java.lang.*;
import java.io.*;

class MST {
    // Number of vertices in the graph
    private static final int V = 5;

    // 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[])
```

```java
    {
        // Initialize min value
        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;
    }

    // A utility function to print the constructed MST stored in
    // parent[]
    void printMST(int parent[], int graph[][])
    {
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++)
            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
    }

    // Function to construct and print MST for a graph represented
    // using adjacency matrix representation
    void primMST(int graph[][])
    {
        // Array to store constructed MST
        int parent[] = new int[V];

        // Key values used to pick minimum weight edge in cut
        int key[] = new int[V];

        // To represent set of vertices included in MST
        Boolean mstSet[] = new Boolean[V];

        // Initialize all keys as INFINITE
        for (int i = 0; i < V; i++) {
            key[i] = Integer.MAX_VALUE;
            mstSet[i] = false;
        }

        // Always include first 1st vertex in MST.
        key[0] = 0; // Make key 0 so that this vertex is
        // picked as first vertex
        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 thd 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
```

```java
            // 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] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
        }

        // print the constructed MST
        printMST(parent, graph);
    }

    public static void main(String[] args)
    {
        /* Let us create the following graph
        2 3
        (0)--(1)--(2)
        | / \ |
        6| 8/ \5 |7
        | /     \ |
        (3)-------(4)
            9         */
        MST t = new MST();
        int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },
                                      { 2, 0, 3, 8, 5 },
                                      { 0, 3, 0, 0, 7 },
                                      { 6, 8, 0, 0, 9 },
                                      { 0, 5, 7, 9, 0 } };

        // Print the solution
        t.primMST(graph);
    }
}
```

5 Sample Input/Output (Compilation, Debugging & Testing)

Edge Weight

0 - 1 => 2
1 - 2 => 3
0 - 3 => 6
1 - 4 => 5

6 Discussion & Conclusion

Based on the focused objective(s) to understand about the MST algorithms, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

7 Lab Task (Please implement yourself and show the output to the instructor)

1. Write a Program in java to find the Second Best Minimum Spanning Tree using Prim's Algorithm.

7.1 Problem analysis

A Minimum Spanning Tree T is a tree for the given graph G which spans over all vertices of the given graph and has the minimum weight sum of all the edges, from all the possible spanning trees. A second best MST T′ is a spanning tree, that has the second minimum weight sum of all the edges, from all the possible spanning trees of the graph G.

8 Lab Exercise (Submit as a report)

- Find the number of distinct minimum spanning trees for a given weighted graph.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.