Department of
Computer Science and Engineering

Title: Implement Kruskal's Algorithm

Algorithms Lab
CSE 206

Green University of Bangladesh

1 Objective(s)

- To learn Kruskal's algorithm to find Minimum Spanning Tree (MST) of a graph.

2 Problem Analysis

2.1 Kruskal's Algorithm

Kruskal'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 Kruskal'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 the edges with the lowest weight and keep adding edges until we reach our goal. The steps for implementing Kruskal's algorithm are as follows:

- Sort all the edges from low weight to high.
- Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
- Keep adding edges until we reach all vertices.

2.3 Kruskal's Algorithm Complexity

The time complexity Of Kruskal's Algorithm is: O(E log E).

2.4 Example of Kruskal's algorithm

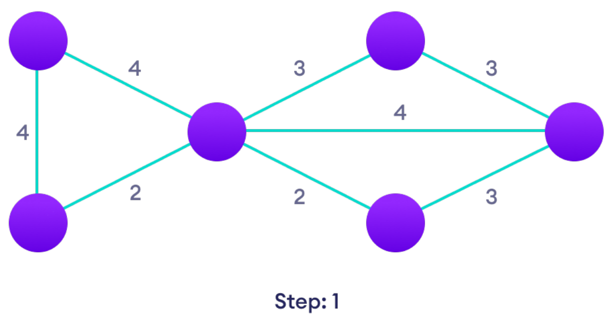
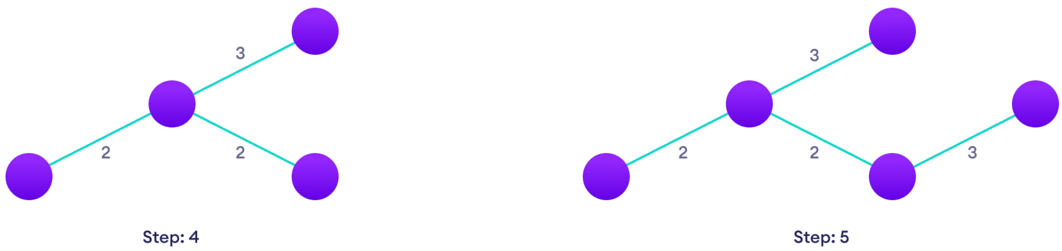


Figure 1: Start with a weighted graph

(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

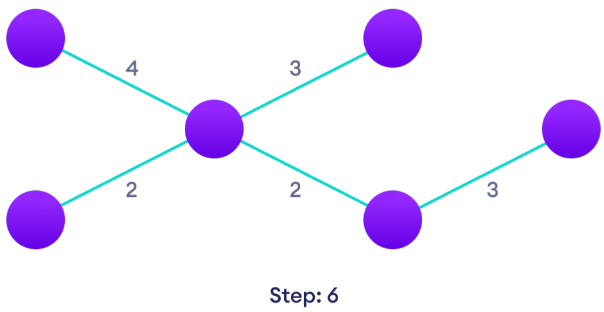


Figure 4: Repeat until you have a spanning tree

3 Algorithm

Algorithm 1: Kruskal Algorithm

```
KRUSKAL(G):
A = ∅
for each vertex v ∈ G.V: do
    MAKE-SET(v)
end
for each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v): do
    if FIND-SET(u) ≠ FIND-SET(v): then
        A = A ∪ (u, v)
        UNION(u, v)
    end
end
return A
```

4 Implementation in Java

```java
// Java program for Kruskal's algorithm to
// find Minimum Spanning Tree of a given
//connected, undirected and  weighted graph
import java.util.*;
import java.lang.*;
import java.io.*;

class Graph {
    // A class to represent a graph edge
    class Edge implements Comparable<Edge>
    {
        int src, dest, weight;

        // Comparator function used for
        // sorting edgesbased on their weight
        public int compareTo(Edge compareEdge)
        {
            return this.weight - compareEdge.weight;
        }
    };

    // A class to represent a subset for
    // union-find
    class subset
    {
        int parent, rank;
    };

    int V, E; // V-> no. of vertices & E->no.of edges
    Edge edge[]; // collection of all edges

    // Creates a graph with V vertices and E edges
    Graph(int v, int e)
    {
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i = 0; i < e; ++i)
```

```
            edge[i] = new Edge();
    }

    // A utility function to find set of an
    // element i (uses path compression technique)
    int find(subset subsets[], int i)
    {
        // find root and make root as parent of i
        // (path compression)
        if (subsets[i].parent != i)
            subsets[i].parent
                = find(subsets, subsets[i].parent);

        return subsets[i].parent;
    }

    // A function that does union of two sets
    // of x and y (uses union by rank)
    void Union(subset subsets[], int x, int y)
    {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        // Attach smaller rank tree under root
        // of high rank tree (Union by Rank)
        if (subsets[xroot].rank
            < subsets[yroot].rank)
            subsets[xroot].parent = yroot;
        else if (subsets[xroot].rank
                 > subsets[yroot].rank)
            subsets[yroot].parent = xroot;

        // If ranks are same, then make one as
        // root and increment its rank by one
        else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }

    // The main function to construct MST using Kruskal's
    // algorithm
    void KruskalMST()
    {
        // Tnis will store the resultant MST
        Edge result[] = new Edge[V];

        // An index variable, used for result[]
        int e = 0;

        // An index variable, used for sorted edges
        int i = 0;
        for (i = 0; i < V; ++i)
            result[i] = new Edge();

        // Step 1:  Sort all the edges in non-decreasing
        // order of their weight.  If we are not allowed to
        // change the given graph, we can create a copy of
```

```
        // array of edges
        Arrays.sort(edge);

        // Allocate memory for creating V ssubsets
        subset subsets[] = new subset[V];
        for (i = 0; i < V; ++i)
            subsets[i] = new subset();

        // Create V subsets with single elements
        for (int v = 0; v < V; ++v)
        {
            subsets[v].parent = v;
            subsets[v].rank = 0;
        }

        i = 0; // Index used to pick next edge

        // Number of edges to be taken is equal to V-1
        while (e < V - 1)
        {
            // Step 2: Pick the smallest edge. And increment
            // the index for next iteration
            Edge next_edge = edge[i++];

            int x = find(subsets, next_edge.src);
            int y = find(subsets, next_edge.dest);

            // If including this edge does't cause cycle,
            // include it in result and increment the index
            // of result for next edge
            if (x != y) {
                result[e++] = next_edge;
                Union(subsets, x, y);
            }
            // Else discard the next_edge
        }

        // print the contents of result[] to display
        // the built MST
        System.out.println("Following are the edges in "
                           + "the constructed MST");
        int minimumCost = 0;
        for (i = 0; i < e; ++i)
        {
            System.out.println(result[i].src + " -- "
                               + result[i].dest
                               + " == " + result[i].weight);
            minimumCost += result[i].weight;
        }
        System.out.println("Minimum Cost Spanning Tree "
                           + minimumCost);
    }

    // Driver Code
    public static void main(String[] args)
    {

        /* Let us create following weighted graph
```

```
                  10
             0--------1
             |   \      |
            6|    5\    |15
             |       \ |
             2--------3
                 4        */
        int V = 4; // Number of vertices in graph
        int E = 5; // Number of edges in graph
        Graph graph = new Graph(V, E);

        // add edge 0-1
        graph.edge[0].src = 0;
        graph.edge[0].dest = 1;
        graph.edge[0].weight = 10;

        // add edge 0-2
        graph.edge[1].src = 0;
        graph.edge[1].dest = 2;
        graph.edge[1].weight = 6;

        // add edge 0-3
        graph.edge[2].src = 0;
        graph.edge[2].dest = 3;
        graph.edge[2].weight = 5;

        // add edge 1-3
        graph.edge[3].src = 1;
        graph.edge[3].dest = 3;
        graph.edge[3].weight = 15;

        // add edge 2-3
        graph.edge[4].src = 2;
        graph.edge[4].dest = 3;
        graph.edge[4].weight = 4;

        // Function call
        graph.KruskalMST();
    }
}
```

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

Following are the edges in the constructed MST

2 – 3 == 4
0 – 3 == 5
0 – 1 == 10

Minimum Cost Spanning Tree: 19

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 Kruskal 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.

7.2 Using Kruskal's Algorithm

We can use Kruskal's algorithm to find the MST first, and then just try to remove a single edge from it and replace it with another.

1. Sort the edges in O(ElogE), then find a MST using Kruskal in O(E).
2. For each edge in the MST (we will have V−1 edges in it) temporarily exclude it from the edge list so that it cannot be chosen.
3. Then, again try to find a MST in O(E) using the remaining edges.
4. Do this for all the edges in MST, and take the best of all.
   Note: we don't need to sort the edges again in for Step 3.

(a) MST (b) Second Best MST

Figure 5: In this figure left is the MST and right is the second best MST

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.