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

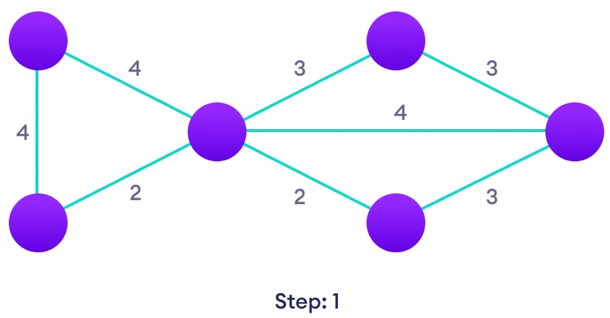
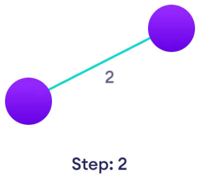
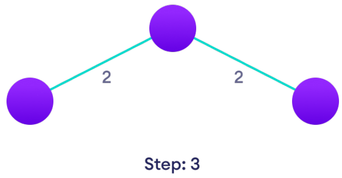


Figure 1: Start with a weighted graph

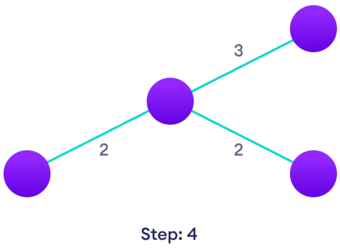


(a) Choose the edge with the least weight, if there are more than 1, choose anyone

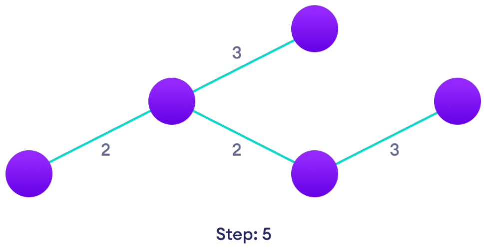


(b) Choose the next shortest edge and add it

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

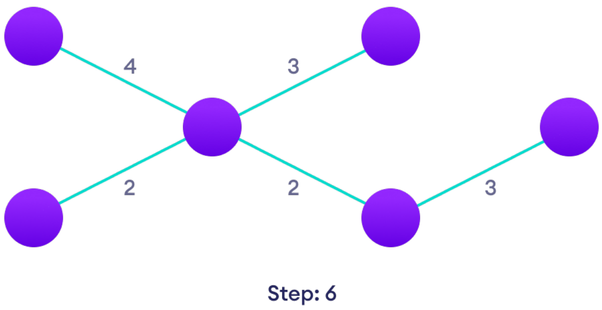


Figure 4: Repeat until you have a spanning tree

3 Algorithm

Algorithm 1: Kruskal Algorithm

KRUSKAL(G):
A = $\emptyset$
for *each vertex* $v \in G.V$: **do**
    MAKE-SET(v)
end
for *each edge* $(u, v) \in G.E$ *ordered by increasing order by weight(u, v):* **do**
    if *FIND-SET(u)* $\neq$ *FIND-SET(v):* **then**
        A = A $\cup$ (u, v)
        UNION(u, v)
    end
end
return A

4 Implementation in C programming language

```c
// Kruskal's algorithm in C

#include <stdio.h>

#define MAX 30

typedef struct edge {
  int u, v, w;
} edge;

typedef struct edge_list {
  edge data[MAX];
  int n;
} edge_list;

edge_list elist;

int Graph[MAX][MAX], n;
edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();

// Applying Krushkal Algo
void kruskalAlgo() {
  int belongs[MAX], i, j, cno1, cno2;
  elist.n = 0;

  for (i = 1; i < n; i++)
    for (j = 0; j < i; j++) {
      if (Graph[i][j] != 0) {
        elist.data[elist.n].u = i;
        elist.data[elist.n].v = j;
        elist.data[elist.n].w = Graph[i][j];
        elist.n++;
```

```c
      }
    }

  sort();

  for (i = 0; i < n; i++)
    belongs[i] = i;

  spanlist.n = 0;

  for (i = 0; i < elist.n; i++) {
    cno1 = find(belongs, elist.data[i].u);
    cno2 = find(belongs, elist.data[i].v);

    if (cno1 != cno2) {
      spanlist.data[spanlist.n] = elist.data[i];
      spanlist.n = spanlist.n + 1;
      applyUnion(belongs, cno1, cno2);
    }
  }
}

int find(int belongs[], int vertexno) {
  return (belongs[vertexno]);
}

void applyUnion(int belongs[], int c1, int c2) {
  int i;

  for (i = 0; i < n; i++)
    if (belongs[i] == c2)
      belongs[i] = c1;
}

// Sorting algo
void sort() {
  int i, j;
  edge temp;

  for (i = 1; i < elist.n; i++)
    for (j = 0; j < elist.n - 1; j++)
      if (elist.data[j].w > elist.data[j + 1].w) {
        temp = elist.data[j];
        elist.data[j] = elist.data[j + 1];
        elist.data[j + 1] = temp;
      }
}
// Printing the result
void print() {
  int i, cost = 0;
  for (i = 0; i < spanlist.n; i++) {
    printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.
        data[i].w);
    cost = cost + spanlist.data[i].w;
  }
  printf("\nSpanning tree cost: %d", cost);
}

```

```
int main() {
  int i, j, total_cost;
  n = 6;

  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

  Graph[1][0] = 4;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 0;
  Graph[1][6] = 0;

  Graph[2][0] = 4;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
  Graph[2][3] = 3;
  Graph[2][4] = 4;
  Graph[2][5] = 0;
  Graph[2][6] = 0;

  Graph[3][0] = 0;
  Graph[3][1] = 0;
  Graph[3][2] = 3;
  Graph[3][3] = 0;
  Graph[3][4] = 3;
  Graph[3][5] = 0;
  Graph[3][6] = 0;

  Graph[4][0] = 0;
  Graph[4][1] = 0;
  Graph[4][2] = 4;
  Graph[4][3] = 3;
  Graph[4][4] = 0;
  Graph[4][5] = 0;
  Graph[4][6] = 0;

  Graph[5][0] = 0;
  Graph[5][1] = 0;
  Graph[5][2] = 2;
  Graph[5][3] = 0;
  Graph[5][4] = 3;
  Graph[5][5] = 0;
  Graph[5][6] = 0;

  kruskalAlgo();
  print();
}
```

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

Following are the edges in the constructed MST

2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4

Minimum Spanning tree cost: 14

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.

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.

(a) MST

(b) Second Best MST

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