Kruskal's Minimum Spanning Tree

CSE 4614 Technical Report Writing

Shakleen Ishfar
ID: 160041029

Department of Computer Science and Engineering
Islamic University of Technology
Bangladesh

04 August, 2019

Contents

1 Introduction **1**

2 Problem Formulation **1**

3 General Solution **2**

3.1 Union-Find by Rank 2

3.2 Union by Rank in Cycle Detection 2

3.3 Procedure 3

3.4 Kruskal's Algorithm Pseudocode 3

4 Example **4**

5 Complexity Analysis **7**

List of Figures

1 Example graph 4

List of Tables

1 Result of sorting edges of example graph 4

2 Going through the graph one edge at a time 5

1 Introduction

Kruskal's algorithm is a *minimum-spanning-tree (MST)* algorithm which finds an edge of the least possible weight that connects any two trees in the forest. The algorithm was written by *Joseph Kruskal* and published in 1956 at the *Proceedings of American Mathematical Society.*

2 Problem Formulation

Before getting to the problem statement there are some terminologies which need to be addressed.
Spanning tree: A *spanning tree* of a connected and undirected graph is a sub-graph which is a tree connecting all the vertices together. A single graph can have many different spanning trees. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.
Minimum Spanning Tree: A *minimum spanning tree (MST)*, for a weighted, connected and undirected graph, is a spanning tree with weight less than or equal to the weight of every other spanning tree.
Problem Statement: Given, a weidghted, connected and undirected graph

having V vertices. Kruskal's algorithm will return a graph which will be tree having $V-1$ edges. The total weight of the MST, W is supposed to be equal or less than all other possible spanning trees. Here W is the following:

$$W = \sum_{i=1}^{V-1} W_i \tag{1}$$

where W_i is the weight of the i-th edge of the MST.

3 General Solution

Before jumping into the steps of Kruskal's Algorithm some basic knowledge is required to understand the procedure of the algorithm. These are discussed here.

3.1 Union-Find by Rank

The union by rank algorithm is used in Kruskal's MST algorithm for cycle detection. A short explanation of Union by Rank is given here for convenience.
Rank: A rank is basically the maximum depth value of a tree.
Union-Find Algorithm: In Union-Find algorithm, a tree having a lower rank is attached to a tree having a higher rank. In other words, the shorter tree is attached to the larger tree. What this algorithm does is that all the vertices in a tree is thought of as being under the same set. Let's say we have a tree with vertices 1, 2, 3 and 4. Then all the vertices of this tree will be seen as being under the same set. Again, let there be two trees respectively having vertices 1, 2, 3 and 4, 5. Then the vertices of each tree will belong to a separate set. Each set is given a value, typically one of the values of the vertices. So a set having 1, 2, 3 might be assigned the value of either 1 or 2 or 3 and the set with members 4, 5 might be given the value of 4 or 5. Now there are essentially 3 operations in this algorithm.

1. **Union(u, v):** Unites the sets u and v. So merging sets 1, 2, 3 and 4, 5 will result in 1, 2, 3, 4, 5.
2. **FindParent(u):** FInds the parent node of the set u. Meaning, the assigned value for that set.
3. **MakeSet(u):** Creates a set with one member u and assigns the set to have u as it's assigned value.

3.2 Union by Rank in Cycle Detection

Let's say that initially a vertice will belong to a set that is assigned a value of the vertice value. A group of connected vertices will be under the same set, represented by a value of one of the member vertices (Following the rule of

Union-Find algorithm). We shall call this specific vertice as the parent vertice of this group of vertices. Now, when we add a vertice to this graph of connected vertices through an edge we are essentially adding the new vertice set to the group vertice set. Simply, we are merging their set. This is done using Union(u, v) operation. Now, a cycle will form if, we try to add a vertice through an edge that is already in the set of the group of vertice set. So, checking for the set assigned value using FindParent(u) is enough to detect a cycle in the graph. This is how we use the Union-Find by Rank algorithm in Kruskal to detect cycles. There is a much more efficient implementation of Union-Find algorithm that implements path compression to reduce time complexity even more. However, discussion of this implementation is outside the scope of this tutorial.

3.3 Procedure

This is a Greedy Algorithm. Here the Greedy Choice is to pick the smallest weight bearing edge that does not form a cycle in the MST constructed so far. The steps to finding the MST of a graph is written below:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it. Cycle detection is done using *Union by Rank and Path Compression* algorithm.
3. Repeat step 2 until there are $V - 1$ edges in the spanning tree.

3.4 Kruskal's Algorithm Pseudocode

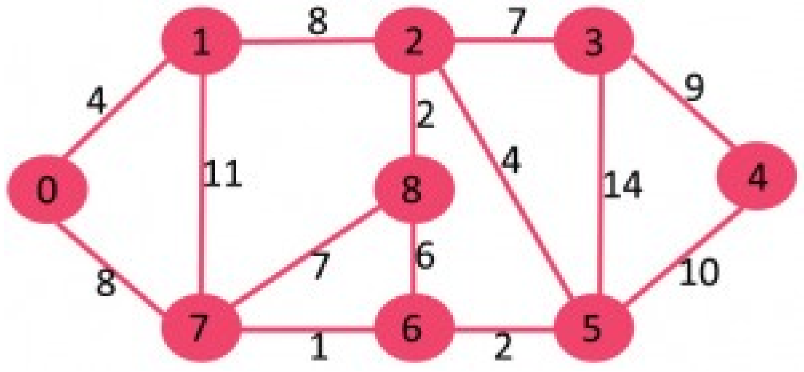
Algorithm 1 Kruskal's algorithm

```
procedure KRUSKAL($G$)
    $A = \emptyset$                                  ▷ Will store MST to be produced
    for $v \in G.V$ do                          ▷ Each vertice originally in it's own set
        MakeSet(v)                                  ▷ Makes vertice v a set of v
    end for
    Sort G.E by non-decreasing weight               ▷ Sorts edges of graph G
    for $(u, v) \in G.E$ do                     ▷ For each edge in graph G
        if FindSet(u) $\neq$ FindSet(v) then        ▷ Check cycle formulation
            $A = A \cup (u, v)$                     ▷ Add to MST graph
            Union(FindSet(u), FindSet(v))           ▷ Unite u and v parent sets
        end if
    end for
    return A
end procedure
```

4 Example

To better understand Kruskal's algorithm let us take a look at an example. The example graph is shown in figure 1 We can see that it is a bidirectional weighted graph. So Kruskal's algorithm can be applied to it to find it's MST. Let us simulate the running of the algorithm and see what happens in each step to get a good grasp of how it is working.

Figure 1: Example graph



The first step is to sort all the edges in non-decreasing order of weight. Any sorting algorithm can be used for this purpose. Sorting the edges yields the following ordering of edges shown in table 1.

Table 1: Result of sorting edges of example graph

Weight	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now that our edges have been sorted in non-decreasing order we can begin to include the edges to form our MST. The rule to remember is that we include an edge to our MST only if it doesn't create a cycle. If the edge causes a cycle we will not include that edge. We shall do this as long as we don't have $V-1$ number of edges. Recall V is the number of vertices present in a graph. For our example graph of figure 1 $V=9$. So we will keep including edges until there are exactly 8 edges in our MST. So let's go through the sorted list and try to formulate our graph one edge at a time, as shown in table 2.

Table 2: Going through the graph one edge at a time

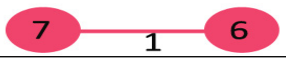
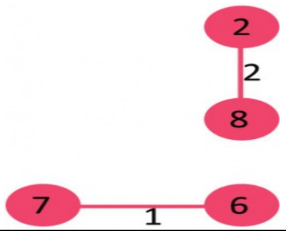
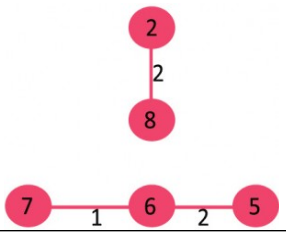
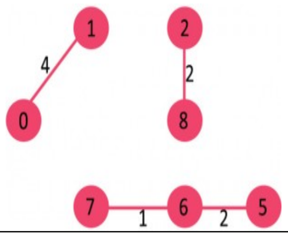
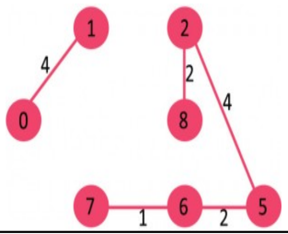
Begin of Table			
Edge	Cycle?	Verdict	Figure
7-6	No	Include	
8-2	No	Include	
6-5	No	Include	
0-1	No	Include	
2-5	No	Include	

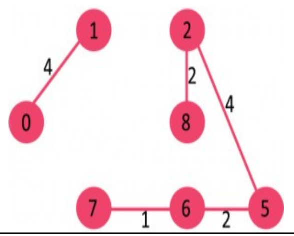
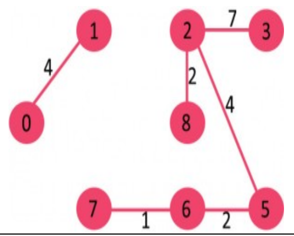
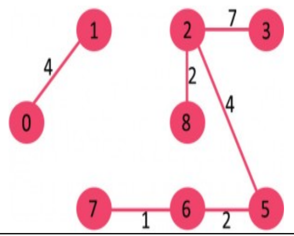
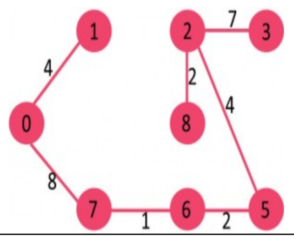
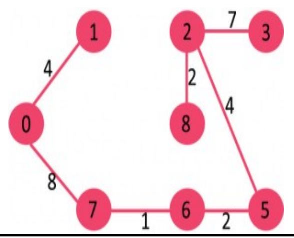
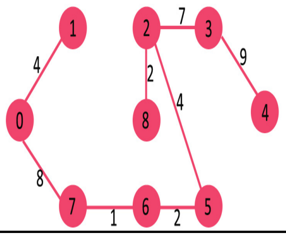
Table Continued			
Edge	Cycle?	Verdict	Figure
8-6	Yes	Discard	
2-3	No	Include	
7-8	Yes	Discard	
0-7	No	Include	
1-2	Yes	Discard	

Table Continued			
Edge	Cycle?	Verdict	Figure
3-4	No	Include	 1 2 3 0 8 4 7 6 5 4 7 9 2 4 8 1 2
End of Table			

The final figure in table 2 is our MST. As we'd already added $V - 1 = 8$ edges, we did not need to go through all the edges in table 1. This is how Kruskal's algorithm can be used to process a graph and build an MST.

5 Complexity Analysis

Kruskal's algorithm has a complexity of O(VlogV). We get this complexity in the following way:

- To sort the edges in non-decreasing order of weight, we can use merge or quick sort. This takes O(ElogE) time.
- We iterate through all edges E and apply find-union algorithm. The find and union operations can take at most O(logV) time.
- To perform both these operations complexity becomes O(ElogE + ElogV).
- The value of E can be at most O(V^2), so O(logV) are O(logE) same. Thus, ultimately time complexity becomes O(VlogV).