

Project Report

On

**Minimum Spanning Tree Using Prims and
Kruskal Algorithm.**

Submitted By
Rohan Chopra

CSE 5311: Design and Analysis of Algorithms
Fall 2019
Instructor(s): Negin Fraidouni

Contents

| | |
|---|-----------|
| Contents..... | 2 |
| 1.Introduction..... | 3 |
| Overview..... | 3 |
| Objective..... | 3 |
| Methodology..... | 3 |
| 2.Working..... | 4 |
| Tools Used..... | 4 |
| Data Structures..... | 4 |
| Algorithms..... | 6 |
| Inputs and Data sets..... | 8 |
| 3.Result..... | 10 |
| Outputs..... | 11 |
| Verdict..... | 11 |
| 4.Analysis and Improvements..... | 12 |
| Prims vs Kruskal..... | 12 |
| Improving Kruskal..... | 14 |
| Improving Prims..... | 14 |
| 5.Conclusion..... | 15 |

1. Introduction

1.1 Overview

This report contains the details about the project created as a Programming project submission for CSE -5311-001 Design and analysis of algorithm. The project choices were posted on the canvas. I have chosen to implement and compare Minimum Spanning Trees Algorithm. It is a GUI python based application that allows a user to perform create a Minimum Spanning Tree using either prims or Kruskal algorithm.

1.2 Objective

The final goal of this project was twofold.

1. Create a minimum spanning tree for any input using prims and Kruskal Algorithm.
2. Compare both the algorithms based on Time – space complexity and find ways to improve the algorithms.

1.3 Methodology

To implement this project, the following methodology needs to be followed :

1. An appropriate data structure is decided.
2. A working, and optimized algorithm needs to be created.
3. The algorithm must be implemented using a programming language and must perform correctly for all inputs.
4. The input file must be proper and structured so that it can be processed.

2. Working

2.1 Tools Used

The following tools, programming languages and frameworks have been used in order to create this project:

1. **Python (3.7)**: This project is written in open source python language. The GUI has also been created using python default library called 'Tkinter'.
2. **PyCharm (Community ver:11)**: PyCharm community is a free python IDE that can be used for python development. I have used the same for this project.

2.2 Data Structures

The implement of an algorithms always requires a data structure. I have used two data structures one for each algorithm.

1. Adjacency Matrix

Prims Algorithm can be implemented using many data structures, but for this project I have used an adjacency Matrix.

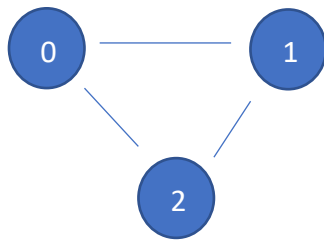
An adjacency matrix is a square matrix. It is commonly used to represent a finite graph. The elements of the matrix are used to deduce that whether the vertices are adjacent or not.

Definition

For an undirected graph with vertex V an adjacency matrix A can be defined as a square matrix of order $|V| \times |V|$ such that A_{ij} is equal to 1 when there's an edge from vertex i vertex j .

For a directed graph the same concept is followed but value of A_{ij} is equal to the weight of that edge.

Example for a graph G:-



Graph G

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Adjacency Matrix

2. Disjoint Sets

Kruskal Algorithm can be implemented using many data structures, but for this project I have used a Disjoint Set.

Disjoint sets in Mathematics are two set of values whose intersection is an empty set.

S1: Set 1

S2: Set 2

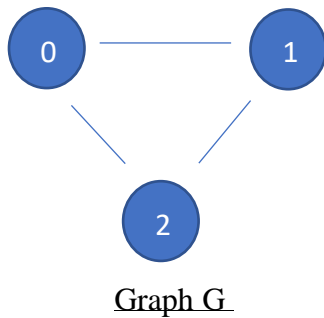
$$S1 \cap S2 = \{\phi\}$$

Definition

For a graph with vertex V a disjoint data structure partitions the vertices into V number of disjoint subsets. There are 2 useful operations that can be performed on this data structure.

1. Union – To merge two subsets into a single subset.
2. Find – To find subset in which the element is present.

Example for a graph G:-



Number of subsets - 3

Subsets - $\{0\}, \{1\}, \{2\}$

Find(0) - $\{0\}$

Union(1,0) - $\{0,1\}$

Disjoint Set

2.3 Algorithms

To create this project, I used the following algorithms:

1. Prims Algorithm

```
Prims(adjMatrix)
    MST = []
    VistedNodes = []
    currentVertex := 0
    MinimumWeight := [0,0,infnite]
    EdgesFromCurrentVertex = []

    While( VistedNodes.len != Number of vertices)
        Set current node as visted
        For(i in 0 to number of edges of current vertex)
            If ((weight of edge from EdgesFromCurrentVertex <
                MinimumWeight) and (vertex is not visited))

                MinimumWeight = EdgesFromCurrentVertex

        Set currentVertex whose edge is choosen
        Add this selected edge to MST
        MinimumWeight := [0,0,infnite]

    Return MST
```

In this algorithm, I am passing an adjacency matrix. Then, I run a while loop until I have visited all the nodes.

In this while loop, I set current node as visited. Then I get a list of all edges from the current node and compare with it with minimumWeight(which is set to infinity for every node). This way I get an edge from the current vertex which has the minimum weight. Finally, I set the current vertex as the end point of edge with the minimum weight.

The MST is returned which contains a list of selected edges in format - vertex 1, vertex 2 and weight.

Complexity: - $O(|V|^3)$

V is the number of vertices

2. Kruskal Algorithm

```
Kruskal(G,V)
    MST = []
    MakeSet(V) for all V in Graph

    For all edges(v1 ,V2) ordered by weight(w) in increasing order
        If findSet(v1) != findset(v2)
            MST.append(v1,v2,W)
            Union(findSet(v1),findset(v2))
Return MST
```

MakeSet : It is used to create a disjoint set

findSet(v): It is used to find the set of v.

Union(v1,v2): It is used to combine subset of v1 and v2.

In this algorithm, I first create a disjoint set of all vertices V. Then I sort the edges list in increasing order of their weight W. Then for all values of edges. I compare if the vertices of edges are not in same subset then I add it to the MST and union the subset of the 2 vertices of that edge.

Complexity: - $O(E \cdot \log V)$

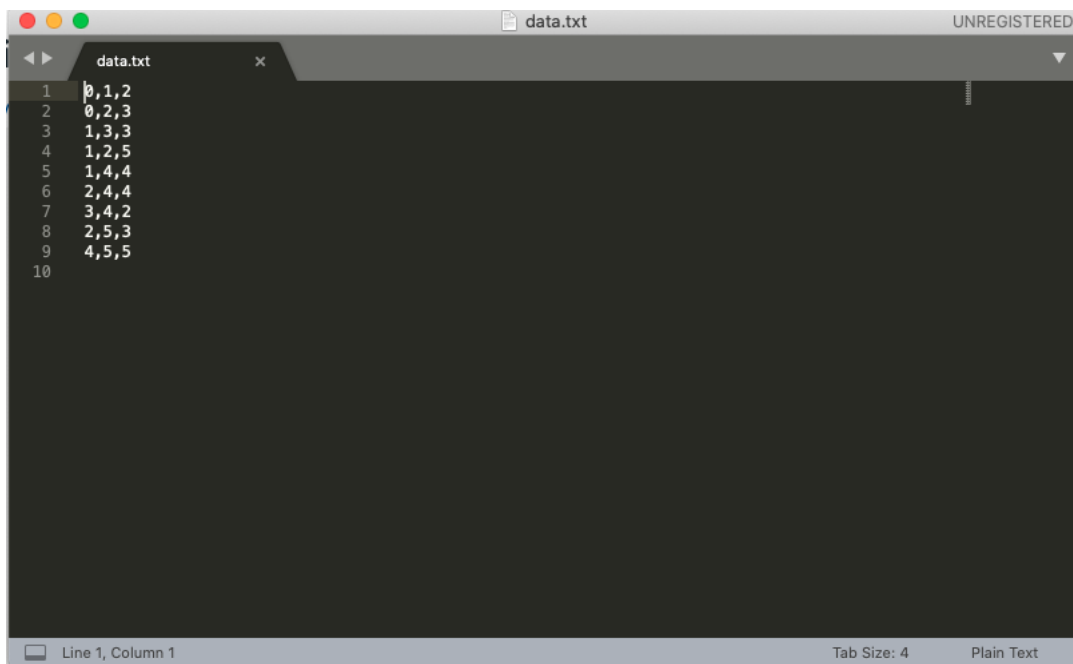
E is the number of Edges and V is the number of vertices.

2.4 Inputs and Data Sets

The Program takes a text file from the user which need to be strictly formatted. Each line in the file is an edge with 3 values separated by commas.

Format: -

[startVertex] , [endVertex], [weight]

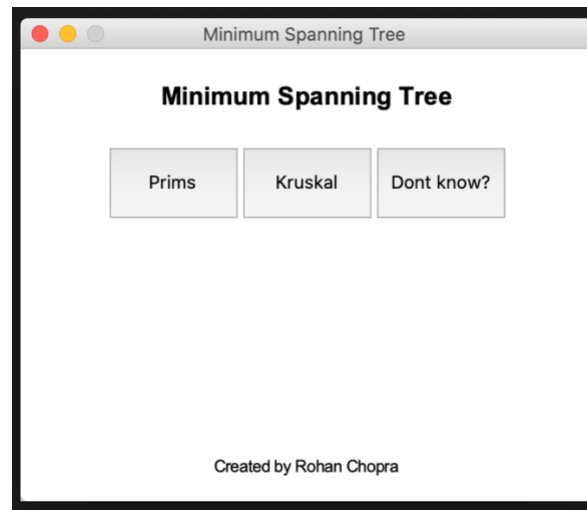


The screenshot shows a text editor window titled 'data.txt' with a tab labeled 'data.txt' and a status bar indicating 'UNREGISTERED'. The editor displays 10 lines of data, each representing an edge with its start vertex, end vertex, and weight separated by commas. The status bar at the bottom shows 'Line 1, Column 1', 'Tab Size: 4', and 'Plain Text'.

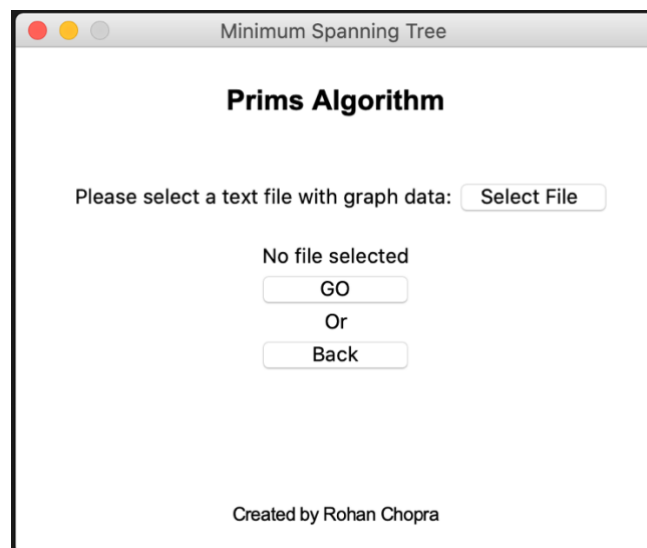
| Line | startVertex | endVertex | weight |
|------|-------------|-----------|--------|
| 1 | 0 | 1 | 2 |
| 2 | 0 | 2 | 3 |
| 3 | 1 | 3 | 3 |
| 4 | 1 | 2 | 5 |
| 5 | 1 | 4 | 4 |
| 6 | 2 | 4 | 4 |
| 7 | 3 | 4 | 2 |
| 8 | 2 | 5 | 3 |
| 9 | 4 | 5 | 5 |
| 10 | | | |

3.Results

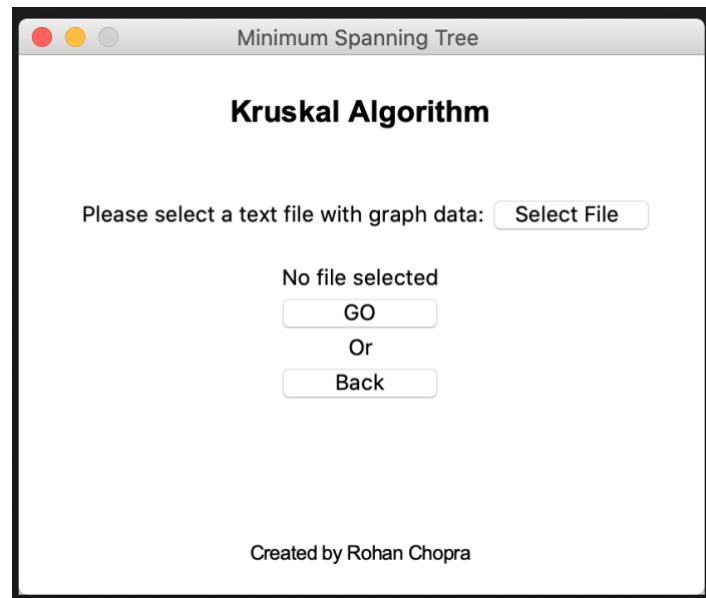
This project has a basic GUI design which has 3 buttons. Prims or Kruskal or Don't Know.



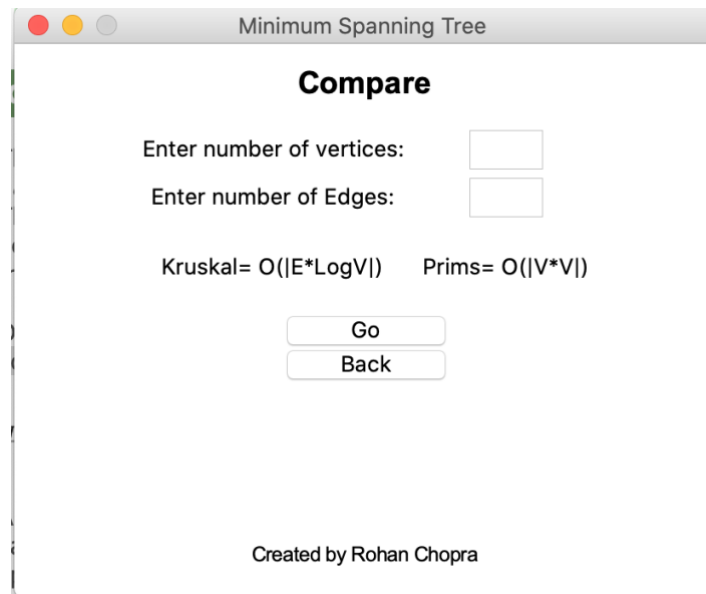
Prims Button is used to perform prims algorithm on a dataset selected by the user. The file must be a valid file and should be extracted



Similarly, Kruskal Button is used to perform Kruskal algorithm on a dataset selected by the user. The file must be a valid file and should be extracted

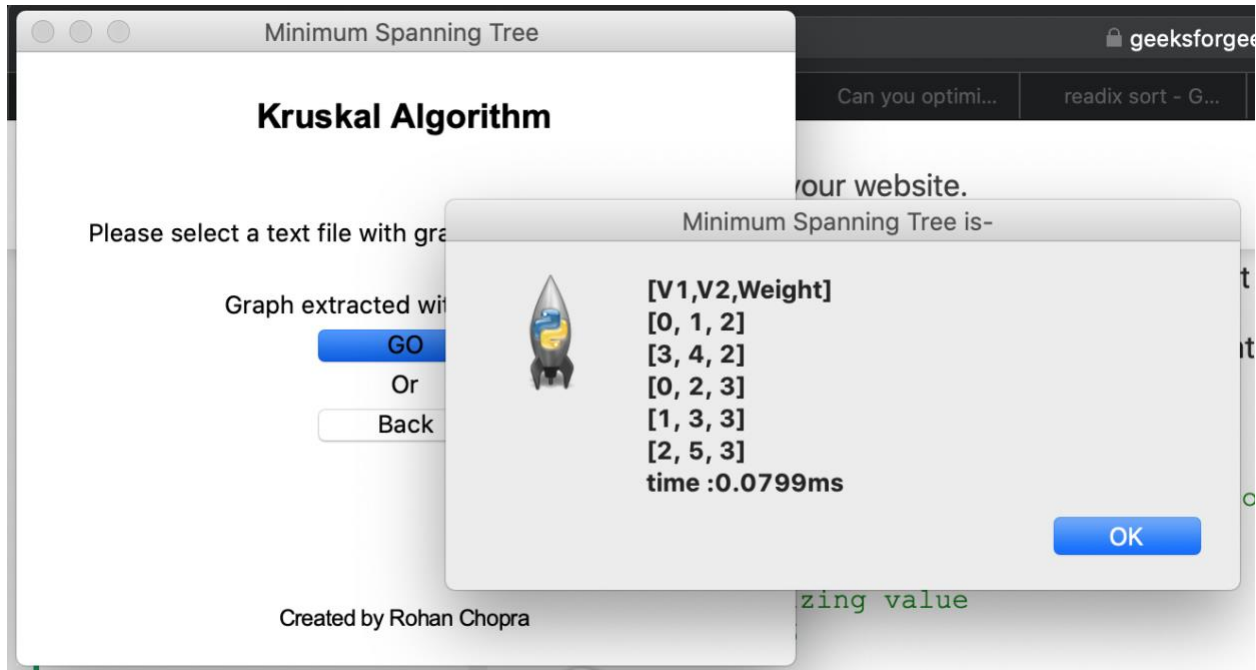


Lastly, Don't Know Button is used compare the two algorithms on a very basic level. It uses the number of vertices and number of edges to tell which algorithm should be used. Based on the inputs it tells if the graph is sparse or dense and which algorithm will be more suitable. I used density function check to predict this.



3.1 Outputs

The final output, Minimum spanning tree is given by showing the edges with their format in a dialog box.



3.2 Verdict

For any size of the data the complexity for prim's algorithm is always $O(V^2)$ where V is the number of vertices.

For any size of the data the complexity for Kruskal algorithm is always $O(E \log V)$ where V is the number of vertices and E is number of edges.

The Selection of algorithm depends upon density function:

$$D = \frac{2 * |E|}{(|V| * (|V| - 1))}$$

If D is close to 0 then graph is sparse else if it is close to 1 its dense.

4. Analysis and Improvements

4.1 Prims vs Kruskal

Theoretical Results: -

After comparing the complexity for both of the algorithms. We can conclude that prims algorithm is suitable when number of Vertices is significantly smaller than the number of edges. Hence the graph is dense.

For a sparse graph we are better off using Kruskal's algorithm.

Experimental Results: -

These results are purely experimental and are performed in the same environment.

Assumption:

For a graph where number of vertices are 49 and number of edges are 100.

According to our theoretical knowledge we would choose Kruskal.

Since density $D = (2 * |E|) / (|V| * (|V| - 1))$

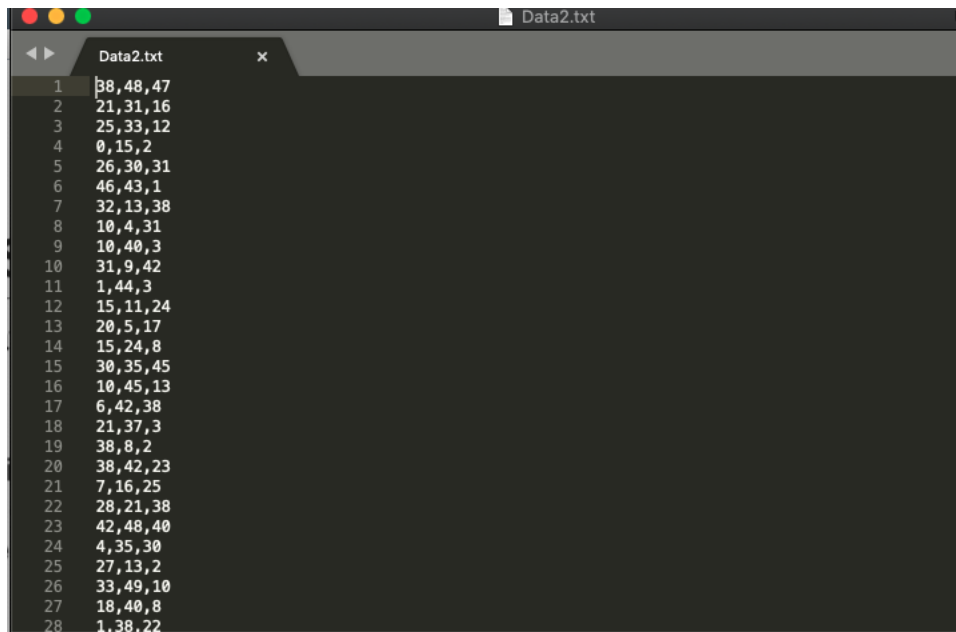
$$D = (2 * 100) / (49 * (48)) = 0.085$$

So, we can say Graph is sparse. Since D is close to 0 ($\text{floor}(D) < 0.5$).

Data Set: -

Number of vertices $V = 49$

Number of edges $E = 100$

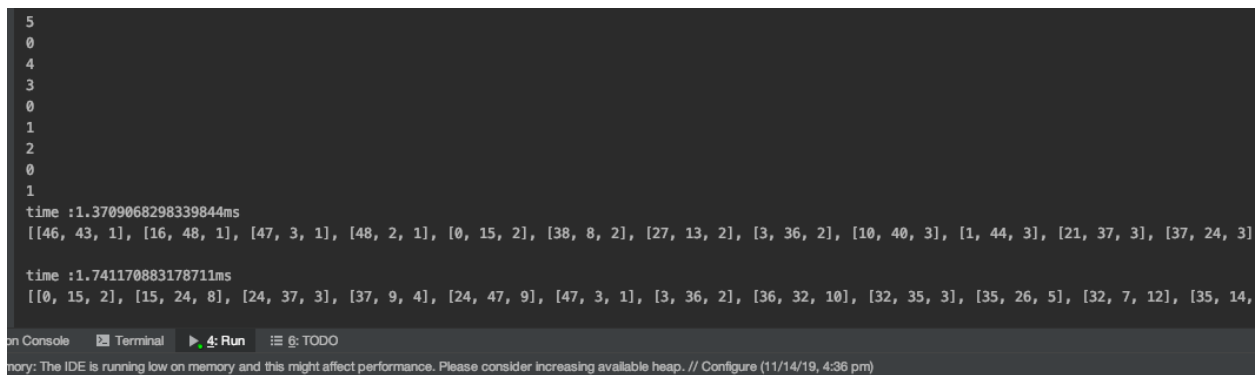


```
1 38,48,47
2 21,31,16
3 25,33,12
4 0,15,2
5 26,30,31
6 46,43,1
7 32,13,38
8 10,4,31
9 10,40,3
10 31,9,42
11 1,44,3
12 15,11,24
13 20,5,17
14 15,24,8
15 30,35,45
16 10,45,13
17 6,42,38
18 21,37,3
19 38,8,2
20 38,42,23
21 7,16,25
22 28,21,38
23 42,48,40
24 4,35,30
25 27,13,2
26 33,49,10
27 18,40,8
28 1,38,22
```

Experimental Results :

Using Kruskal: We get a time of 1.37 ms

Using Prims: We get a time of 1.74 ms



```
5
0
4
3
0
1
2
0
1
time :1.3709068298339844ms
[[46, 43, 1], [16, 48, 1], [47, 3, 1], [48, 2, 1], [0, 15, 2], [38, 8, 2], [27, 13, 2], [3, 36, 2], [10, 40, 3], [1, 44, 3], [21, 37, 3], [37, 24, 3]]

time :1.741170883178711ms
[[0, 15, 2], [15, 24, 8], [24, 37, 3], [37, 9, 4], [24, 47, 9], [47, 3, 1], [3, 36, 2], [36, 32, 10], [32, 35, 3], [35, 26, 5], [32, 7, 12], [35, 14,
```

Conclusion:

We can clearly see that our assumption was correct. For a dense matrix prims algorithm works better and for a sparse matrix Kruskal is better.

4.2 Improving Kruskal Algorithm

In order to improve Kruskal's algorithm's Time complexity I can use a better sorting technique since sorting is dominant in Kruskal's Algorithm. I can use a radix sort or make sure that graph is already sorted. So., if I am able to get linear time complexity for sorting then it would might improve the time complexity.

4.3 Improving Prims Algorithm

In order to improve Prims algorithm, I can use a different data structure. If I use a Binary heap instead of Kruskal algorithm it would decrease the time. This would make the time complexity logarithmic.

Using binary heap I would get $O(\log V)$ for the implementing the minimum priority queue and it would run $2E$ times .

Therefore total complexity = $O(E * \log V)$

5. Conclusion

We can conclude that Prim's algorithm works better when the graph is dense while Kruskal's algorithm works better when the graph is sparser.

Prim's has a time complexity of $O(V^2)$ while Kruskal has a time complexity of $O(E \cdot \log V)$