
# University of Dhaka

## Department of Computer Science and Engineering

**Course:**

Design and Analysis of Algorithms - I Lab

(CSE-2212)

**Assignment Name:**

Prim's and Kruskal's Algorithm for Minimum Spanning Tree Analysis

**Submitted To:**

Dr. Md. Mosaddek Khan

Associate Professor

Department of Computer Science and Engineering, University of Dhaka

**Submitted By:**

Abdullah Ibne Hanif Arean

Roll: FH-12

Registration No: 2019-917-795

**Title:**

Prim’s and Kruskal’s Algorithm for Minimum Spanning Tree Analysis

**Abstract:**

Prim’s Algorithm and Kruskal’s Algorithm are two of the most common minimum spanning tree finder algorithms used in computer science. Both of these algorithms are tested on different implementations and different datasets to determine their effectiveness. Extensive research has been conducted to determine which of the two algorithms is best for a given problem. This has led to the development of other algorithms that are better suited for a specific type of data or a specific type of problem.

**Introduction:**

The prim’s algorithm starts by choosing the root vertex and then moves through nearby vertex pairs. By starting with the shortest weighted edge, Krushal's approach, on the other hand, assists in creating the minimal spanning tree.

Popularly known as a greedy method, Prim's approach aids in locating the lowest spanning tree for a weighted undirected graph. This means that the total weight of all the edges in the tree should be as low as possible because this method likes to explore the subgroup of edges that may create a tree.

In the field of computer science, the shortest spanning tree of a connected graph and the spanning forest of an undirected edge-weighted graph are both found using Kruskal's Algorithm. Essentially, it takes a graph as input and finds the subgroup of its edges.

**Methodology (Materials and Methods)**

Prim’s Algorithm implemented in four different ways, using

1. STL Priority Queue (C++ Standard Template Library)
2. Binary heap (Self Implemented)
3. Linked List (Self Implemented)
4. Array (Self Implemented)

Kruskal’s Algorithm implemented using twelve different ways, using

1. DSU (Disjoint Set Union) with Path Compression, Union by Rank, and STL
2. DSU (Disjoint Set Union) with Path Compression and STL
3. DSU (Disjoint Set Union) with Union by Rank and STL
4. DSU (Disjoint Set Union) and STL
5. DSU (Disjoint Set Union) with Path Compression, Union by Rank, and LinkedList
6. DSU (Disjoint Set Union) with Path Compression and LinkedList
7. DSU (Disjoint Set Union) with Union by Rank and LinkedList
8. DSU (Disjoint Set Union) and LinkedList
9. DSU (Disjoint Set Union) with Path Compression, Union by Rank, and Array
10. DSU (Disjoint Set Union) with Path Compression and Array
11. DSU (Disjoint Set Union) with Union by Rank and Array
12. DSU (Disjoint Set Union) and Array

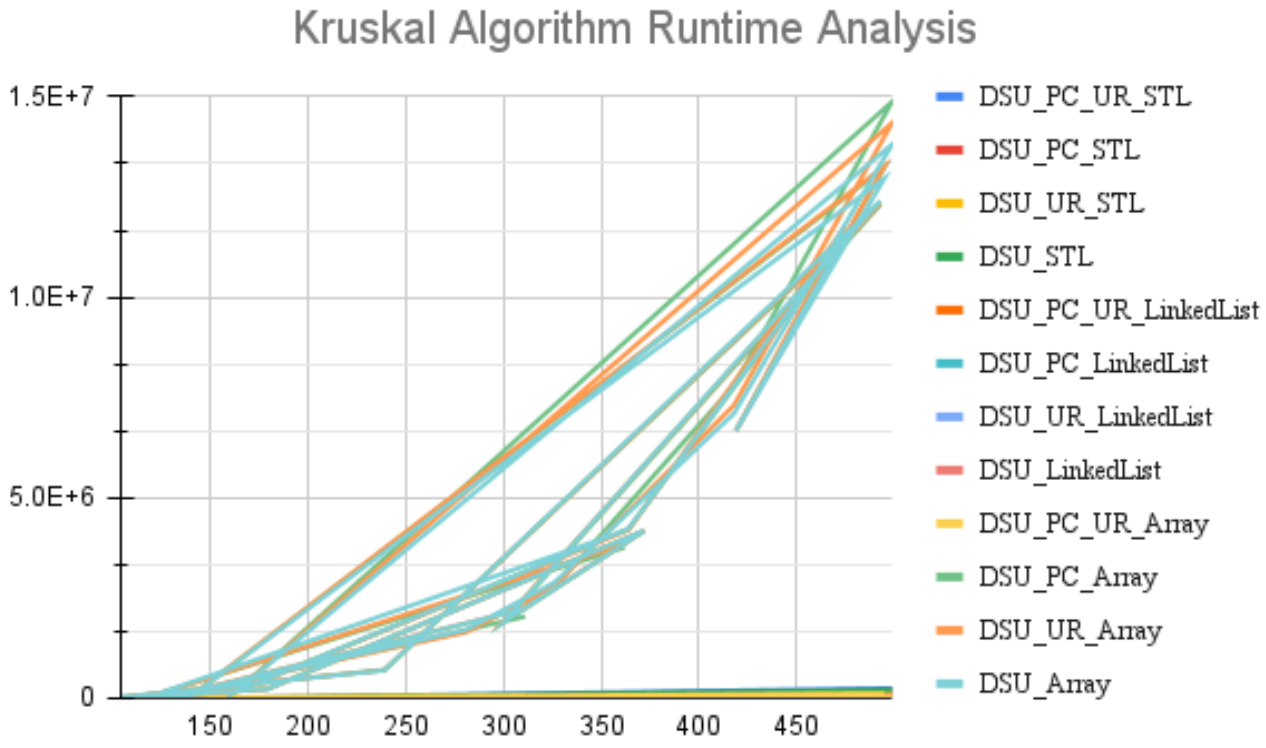
Graph Generated using Networkx Python Library and Algorithms solely coded on C++. To Plot Result We Used

## Results:

Prim Algorithm tested over 12 random graph, the result is as follows: (Time Stamp in Micro Second)

| Vertex | Edge | STL Priority Queue | Min Heap | LinkedList | Array |
|---|---|---|---|---|---|
| 125 | 3830 | 4325 | 2355 | 615112 | 44524514 |
| 147 | 5356 | 6421 | 2475 | 1253267 | 130542611 |
| 139 | 4764 | 6072 | 2544 | 886919 | 88399040 |
| 127 | 3936 | 4702 | 2081 | 595764 | 51879515 |
| 135 | 4484 | 5536 | 2293 | 805164 | 76531923 |
| 111 | 3014 | 3386 | 1501 | 332849 | 23069285 |
| 105 | 2699 | 2684 | 1520 | 264855 | 15996641 |
| 147 | 5356 | 5860 | 2403 | 1187467 | 132391523 |
| 103 | 2601 | 2498 | 1223 | 246962 | 14549981 |
| 139 | 4764 | 5738 | 2782 | 921304 | 92260799 |
| 132 | 4278 | 4851 | 1909 | 704142 | 67298028 |
| 103 | 2601 | 2589 | 1241 | 222180 | 14246695 |
| 150 | 5576 | 6104 | 2454 | 1296131 | 153415316 |

Kruskal's Algorithm Implemented Using 12 Ways and Tested in 20 Random Graphs, result in graph form:



**Discussion:**

Kruskal time complexity worst case is O(E log E),this because we need to sort the edges. Prim time complexity worst case is O(E log V) with priority queue or even better, O(E+V log V) with Heap. We should use Kruskal when the graph is sparse, i.e.small number of edges,like E=O(V),when the edges are already sorted or if we can sort them in linear time. We should use Prim when the graph is dense, i.e number of edges is high , like $E=O(V^2)$.

Analzying the runtime of both the algorithm, we can notice heap implementations are way more faster than LinkedList implementation, array implementation is the slowest. We can also notice that, Standard Template Library implementation is not always best in term of time complexity.

**Links:**

**Github Repository With Codes:**

https://github.com/AbdullahArean/Prim-Kruskal-Graph-Implementation

**Result Data Sheet :**

https://docs.google.com/spreadsheets/d/1o7hJKIJXeZbm71hMhuzeFDwiU1wGHqNCqMy10bQznJA/