Sub-Sahara African Academic Research Publications
December, 2020 Editions

JASD
Journal of African Sustainable Development
Vol.17, No.2, ISSN: 2115-4255

EXPLORING TIME BASED COMPARISON OF KRUSKAL AND PRIM ALGORITHMS FOR SINGLE SOURCE SHORTEST PATHS PROBLEM

VICTOR IBOYI
Computer Science Department, Federal Polytechnic, Damaturu.

Abstract
Minimum spanning trees problems (MSTP) of weighted graphs have been instrumental in the design of earliest computer algorithms and provided solutions to weighted graphs with thousands of vertices thus finding applications in communication networks, transportation networks and many more. Two fundamental algorithms namely Kruskal and Prim's Algorithms have been employed for solutions depending on the problems. This mini project compares both algorithms in terms of speed of execution on a given cpu using C++ programming language and recommends a preferable algorithm where speed of execution is premium.

Keywords: *Minimum spanning tree, Kruskal's algorithm, Prim's algorithm, speed of execution, C++ programming.*

Introduction
The minimum spanning tree problems have dominated crucial place in research especially for algorithms and optimization even before the advent of electronic computers in the 1940s. Today, it has found applications in computer networks, communication networks, piping in a flow network, power networks and lots more. The problem is stated as follows:

> *"Given such a weighted graph (a graph G, whose nodes represent cities, whose edges represent possible communication links, and whose edge weight represents the cost of construction, or lengths of the links) one*

> *would then wish to select for construction a set of communication links that would connect all the cities and have a minimum total cost or be of minimum total length. In either case the links contains a circle. Removal of any one of the links on this circle will result in a link selection of lower cost connecting all cities. The cost of a spanning tree would therefore be the costs of the edges in that tree*" (Graham and Hell, 1985).

A Spanning tree summarily is seen as a subset of a graph, which consists of all the vertices covering minimum possible edges and does not have a cycle while a minimum spanning tree is the one that contains the least weight among all the other spanning trees of a connected weighted graph. There can be more than one minimum spanning tree for a graph. Many algorithms over the years have been proposed and being used for minimum spanning tree problems sometimes with graphs up to one hundred and thirty thousand nodes or seventy-five thousand edges. Moret and Shapiro (1991) experimented with algorithms like kruskal, Cheriton Tarjan, Fredman and Tarjan and that of Prim. In this mini project, two of the algorithms Kruskal algorithm and Prim's algorithm are considered under the run of one machine and the efficiency of each of the algorithms is investigated with respect to the time of program execution.

Literature review

According to Barnwal (2020), a spanning tree of a connected and undirected graph is a subgraph tha is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree 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. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree. Given the number of vertices in a given graph V, a minimum spanning tree has V-1 edges. Finding the MST using Kruskal algorithm described in

section three here has been efficient though quite recently Pop, Matei, Sabo and Petrovan (2018) proposed an improvement arriving at what they termed generalized minimum spanning tree problem where the vertices are partitioned into a given number of clusters and then the aim would be to look for a minimum cost tree spanning subset of vertices which will include one vertex from each cluster. For Li, Xia and Wang (2017), two branch Kruskal algorithm (a novel improvement on the Kruskal algorithm) is suggested as it will improve the choosing of a middle value and reduce time complexity. Various researchers have proposed a number of improvement upon this algorithm and researches are still ongoing for improvement as per time, weight and cost domains.

Prim algorithm on the other hand builds a subgraph one edge at a time, a tree one vertex at a time. According to Barnwal (2020), it is represented thus:

```
let T be a single vertex x
while (T has fewer than n vertices)
{
find the smallest edge connecting T to G-T
add it to T
}
```

Kruskal and Prim's algorithms are two very efficient algorithms for solving MSTP. However, depending on the constraints involved, one is better than the other. For example, according to study on the Shengai snd Shenzen stock market by Gao and Wang (2009), Kruskal algorithm is superior to Prim when the number of shares is less than 100, otherwise, Prim's algorithm is superior from the time complexity aspect.

Methodology

In implementation of Kruskal and Prim's Algorithm which are greedy techniques, C++ programming language is used to test for the time efficiency.

Kruskal algorithm

The theory in Kruskal algorithm is further refined by Li, Xia and Wang (2017) who presupposed a connected graph with n vertexes G = {V, E}, non-connected graph without edges T = {V, φ}. Each vertex in graph G is a connected component, increase the cost of all the edges in graph G, and remove the edge with the smallest cost. If the two vertexes of the edge fall on different two connected components, one can select it into T; otherwise, give up it, and choose a minimum cost edge from the remaining edges again. Repeat in this case, the connected components are merged step by step until all vertexes are on the same connected component. M-1 edges were chosen for T, so it is the minimum cost spanning tree with M-1 edges of G

In this project, Union-Find algorithm is implemented to divide the vertices into clusters and allow us to check if two vertices belong to the same cluster or not and hence decide whether adding an edge creates a cycle.

1. KRUSKAL(G):
2. A = ∅
3. For each vertex v ∈ G.V:
4. MAKE-SET(v)
5. For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
6. if FIND-SET(u) ≠ FIND-SET(v):
7. A = A ∪ {(u, v)}
8. UNION(u, v)
9. return A

Prim's algorithm

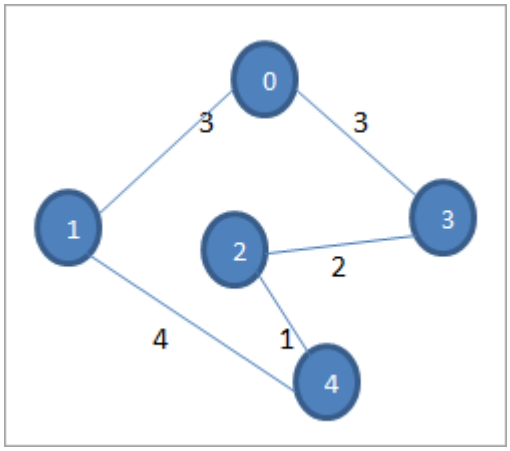
1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
3. Keep repeating step 2 until we get a minimum spanning tree

Here two sets of vertices are created U and V-U. U containing the list of vertices that have been visited and V-U the list of vertices that haven't. One by one, we move vertices from set V-U to set U by connecting the least weight edge.

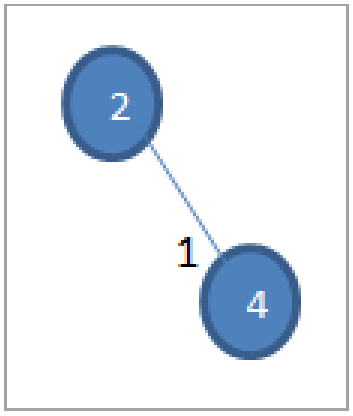
1. T = ∅;
2. U = { 1 };
3. while (U ≠ V)
4. let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
5. T = T ∪ {(u, v)}
6. U = U ∪ {v}

Result and Analysis

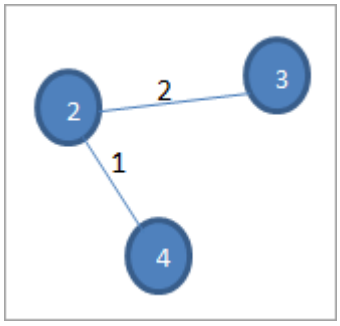
I. Kruskal's algorithm.



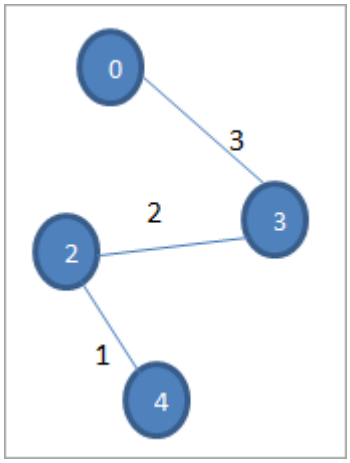
Now we choose the edge with the least weight which is 2-4.



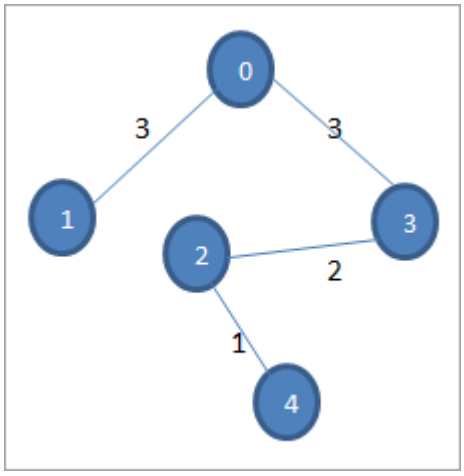
Next, choose the next shortest edge 2-3.



Then we choose next edge with the shortest edge and that does not create a cycle i.e. 0-3



The next step is to choose the shortest edge so that it doesn't form a cycle. This is 0-1.



As we can see, we have covered all the vertices and we have a spanning tree with minimum cost here.

Output:

The Minimum Spanning Tree (MST) according to Kruskal's Algorithm:

Edge : Weight

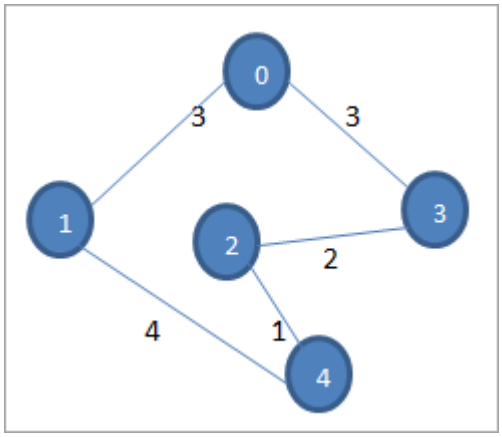
2 – 4 : 1

2 – 3 : 2

0 – 1 : 3

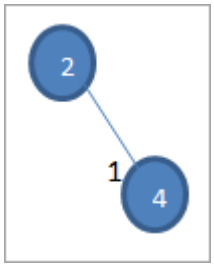
0 – 3 : 3

II. Prim's algorithm

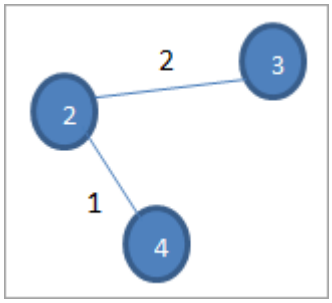


Let us select node 2 as the random vertex.

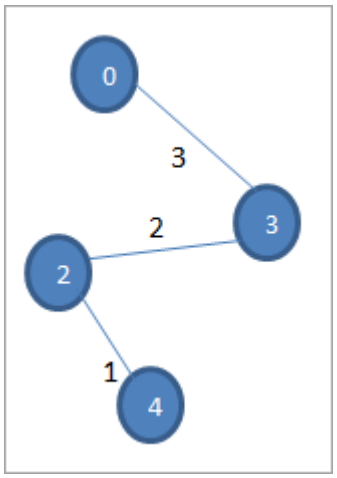
Next, we select the edge with the least weight from 2. We choose edge 2-4.



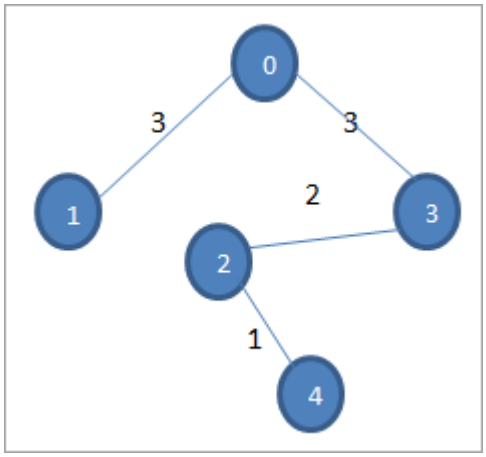
Next, we choose another vertex that is not in the spanning tree yet. We choose the edge 2-3.



Now let us select an edge with least weight from the above vertices. We have edge 3-0 which has the least weight.



Next, we choose an edge with the least weight from vertex 0. This is the edge 0-1.



From the above figure, we see that we have now covered all the vertices in the graph and obtained a complete spanning tree with minimum cost.

Output:
The Minimum Spanning Tree as per Prim's Algorithm:
Edge : Weight
0 – 1 : 3
0 – 3 : 3
3 – 2 : 2
2 – 4 : 1

RESULT

	Kruskal's Algorithm	Prim's Algorithm
Starting Time(sec)	0.00sec	0.00sec
Completion Time(sec)	0.06101sec	0.06769sec
Compilation Time(sec)	8.39sec	6.77sec

Comparison
In terms of time efficiency, Kruskal's Algorithm which has a lower completion time of 0.06101 seconds is more efficient than Prim's Algorithm which has a completion time of 0.06769seconds.

Conclusion
The results of the investigation from the executed program show that in terms of time efficiency, Kruskal's Algorithm which has a lower completion time of 0.06101 seconds is more efficient than Prim's algorithm which has a completion time of 0.06769 seconds for the construction of a minimum spanning tree for a weighted graph. Therefore, where speed is the premium, Kruskal algorithm is the recommended over Prim's.

Appreciation:
I thank Haruna Chiroma (PhD) for his support and corrections in this mini project.

Reference
Barnwal, A. (2020). Kruskal's Minimum Spanning Tree Algorithm - Greedy Algo-2. Available online at https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Graham, R.L. and Hell, P. (1985). On the history of the minimum spanning tree problem. Annals of the history of computing. 7, 43-57. doi: 10.1109/MAHC.1985.10011.

Huang, F., Gao, P. and Wang, H. (2009). "Comparison of Prim and Kruskal on Shanghai and Shenzhen 300 Index Hierarchical Structure Tree," International Conference on Web Information Systems and Mining, Shanghai,, pp. 237-241. doi: 10.1109/WISM.2009.56.

Li, H., Xia, Q., Wang, Y. (2017). Research and Improvement of Kruskal Algorithm. *Journal of Computer and Communications*, 5(12). doi: 10.4236/jcc.2017.512007

Moret, B.M.E. and Shapiro, H.D. (1991). An empirical analysis of algorithms for constructing a minimum spanning tree. In: Dehne F., Sack J. R., Santoro N. (eds) Algorithms and Data Structures. Lecture Notes in Computer Science, 519. Springer, Berlin, Heidelberg. doi:10.1007/BFb0028279

Pop, C. P., Matei, O., Sabo, C. and Petrovan, A. (2018). A two-level solution approach for solving the generalized minimum spanning tree problem. *European Journal of Operational Research,* 265 (2), 478-487. doi: 10.1016/j.ejor.2017.08.015.

APPENDIX I

C++ CODE FOR KRUSKAL ALGORITHM

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
        #define graph_edge pair<int,int>
        class Graph {
        private:
        intV;     // number of nodes in graph
        vector<pair<int, graph_edge>> G; // vector for graph
        vector<pair<int, graph_edge>> T; // vector for mst
        int *parent;
        public:
        Graph(int V);
        void AddEdge(int u, int v, intwt);
        intfind_set(inti);
        void union_set(int u, int v);
        void kruskal_algorithm();
        void display_mst();
        };
        Graph::Graph(int V) {
        parent = new int[V];
        for (inti = 0; i < V; i++)
        parent[i] = i;
G.clear();
        T.clear();
        }
        void Graph::AddEdge(int u, int v, intwt) {
        G.push_back(make_pair(wt, graph_edge(u, v)));
        }
        intGraph::find_set(inti) {
          // If i is the parent of itself
          if (i == parent[i])
            return i;
```

```
    else
      //else recursively find the parent of i
      return find_set(parent[i]);
}
void Graph::union_set(int u, int v) {
  parent[u] = parent[v];
}
void Graph::kruskal_algorithm() {
inti, uSt, vEd;
  sort(G.begin(), G.end()); // sort the edges ordered on increasing weight
  for (i = 0; i<G.size(); i++) {
uSt = find_set(G[i].second.first);
vEd = find_set(G[i].second.second);
    if (uSt != vEd) {
T.push_back(G[i]); // add to mst vector
union_set(uSt, vEd);
    }
  }
}
void Graph::display_mst() {
cout<< "Edge :" << " Weight" <<endl;
  for (inti = 0; i<T.size(); i++) {
cout<< T[i].second.first<< " - " << T[i].second.second<< " : "
<< T[i].first;
cout<<endl;
  }
}
intmain() {
  Graph gmst(5);
gmst.AddEdge(0,1,3);
gmst.AddEdge(0,3,3);
gmst.AddEdge(2,3,2);
gmst.AddEdge(2,4,1);
gmst.AddEdge(1,4,4);
gmst.kruskal_algorithm();
cout<<"The Minimum Spanning Tree according to Kruskal's Algorithm:"<<endl;
gmst.display_mst();
  return 0;
}
```

APPENDIX II

```
#include <iostream>
#include <cstring>
using namespace std;
    #define INF 9999
    // graph contains 5 vertices
    #define V 5
    // an array G that stores adjacency matrix for input graph
    int G[V][V] = {
```

```
    {0, 3, 0, 3, 0},
    {3, 0, 0, 0, 4},
    {0, 0, 0, 2, 1},
    {3, 3, 2, 0, 0},
    {0, 4, 1, 0, 0}};
int main () {
intnum_edge;          // number of edge
 // mst_vertex - array to track vertices selected for spanning tree
intmst_vertex[V];
// set selected false initially
memset (mst_vertex, false, sizeof (mst_vertex));
 // set number of edge to 0
num_edge = 0;
  //let 0th vertex be the first to be selected
mst_vertex[0] = true;
intx;          //  row
inty;          //  col
  // print details of MST
cout<<"The Minimum Spanning Tree as per Prim's Algorithm:"<<endl;
cout<< "Edge" <<" : " << "Weight";
cout<<endl;
 while (num_edge< V - 1) {
  //Prim's algorithm code
int min = INF;
   x = 0;
   y = 0;
for (inti = 0; i < V; i++) {
if (mst_vertex[i]) {
for (int j = 0; j < V; j++) {
if (!mst_vertex[j] && G[i][j]) { // not in mst_vertex and there is an edge
if (min > G[i][j]) {
min = G[i][j];
                    x = i;
                    y = j;
                }
            }
          }
        }
      }
}
cout<< x <<  " - " << y << " :  " << G[x][y];
cout<<endl;
mst_vertex[y] = true;num_edge++;
  }
return 0;
```