International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 10, December- 2012

Minimum Spanning Tree Of Undirected Graphs

Aquila Khanam,
PESIT, BSC

Dr. Minita Mathew
Associate Professor, PESIT –BSC

ABSTRACT

This paper presents an approach to finding the minimum spanning tree for simple undirected graphs and undirected multi-graphs. The algorithm involves choosing the minimum edge that connects each disjoint component of the graph, until a single component is formed. This single component is the minimum spanning tree of the given graph. The approach we take is a slight modification to Sollin's algorithm.

1. Introduction

The minimum spanning tree of an undirected graph is an acyclic graph (tree) of minimum weight that connects all the vertices of the graph.

One may use minimum spanning trees or MSTs to set up communication links between cities with minimum cost or minimum length. Similarly, MSTs may be used to set up communication networks, telephone line networks, computer networks or piping in a flow network. Due to the various uses of MSTs in everyday problems, efficient algorithms that can solve graphs with a large number of vertices are required. Traditionally, Prim's and Kruskal's algorithms have been used to create minimum spanning trees, and do so efficiently. Their predecessor is Borůvka's (or Sollin's) algorithm, which is where the MST algorithm in this paper is derived from.

Prim's algorithm involves dividing the vertices of a graph into two sets - visited and unvisited, with the initial visited vertex being arbitrarily chosen as the starting vertex. On each iteration, the minimum edge connecting an unvisited vertex to a visited vertex is added to the tree. The final tree T formed, that spans the vertices of the graph is a minimum spanning tree.

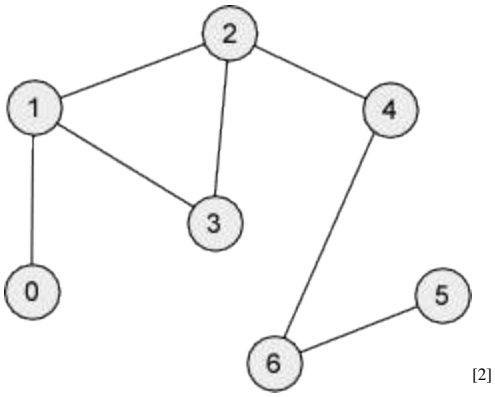
In Kruskal's algorithm, each edge of the graph G is examined in ascending order of weight, and if the chosen edge does not form a cycle in the tree T, it is added to T. The process continues until n-1 edges have been added.

In this MST algorithm, each disjoint component of the spanning tree is connected by adding the minimum edge between any two components to the tree. The algorithm starts with taking all the vertices of the graph as disjoint components and examines each component to determine the minimum edge incident on that component. Effectively, this algorithm continues until only one final component remains, and in the worst case, each iteration halves the number of disjoint components remaining. This selection process also applies to multi-graphs, as only one minimum edge will be chosen between two components that have multiple edges connecting them. The final single component is the required minimum spanning tree.

In this paper, we give a proof as well as a pseudo code for our algorithm and illustrate it with an example.

2. Terminology

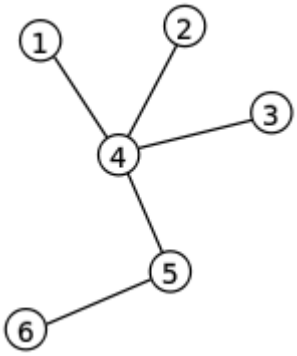
Graph: A graph is an ordered pair G = (V,E), where V is the *vertex set* whose elements are the vertices, or *nodes* of the graph. This set is often denoted V(G) or just V. E is the *edge set* whose elements are the edges, or connections between vertices, of the graph. This set is often denoted as E(G) or just E. If the graph is undirected, individual edges are unordered pairs {u,v}, where u and v are vertices in V. If the graph is directed, edges are ordered pairs (u,v).



[2]

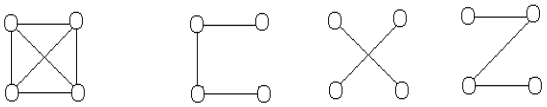
Example of a Graph

Tree**:** A tree is a type of connected graph. A directed graph is a tree if it is connected, has no cycles and all vertices have at most one parent. An undirected graph is considered a tree if it is connected, has vertices one more than the number of edges. Such a graph is acyclic



Example of a tree[3]

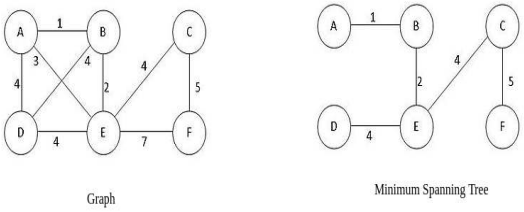
Spanning Tree: For a graph G = (V,E), a spanning tree T = (V,E) is a tree that contains all the vertices of G, and whose set of edges is a subset of the edges of G (E' E).



Graph Spanning Trees

Weighted Trees: A weighted tree associates a label (weight) with every edge in the graph. We denote the weight of an edge "e" as wt(e). The weight of a tree wt(T) is the sum of the weights of the edges.

Minimum Spanning Tree: A spanning tree with the smallest possible weight among all spanning trees for a given graph.[4]



Graph Minimum Spanning Tree

[5]

3. Algorithm

We consider all the vertices of the graph as disjoint components. First to each vertex we identify the edge with minimum weight on it. Ties are broken arbitrarily. These selected edges become the required edges of our minimum spanning tree. In the next stage we are left with several connected components. We then select the edge with the minimum weight between two components. The process is continued till we get a connected component. This selection process also applies to multi-graphs, as only one minimum edge will be chosen between two components. We claim the tree S thus obtained is a minimal spanning tree.

3.1. Proof

S is a spanning graph because the algorithm ensures that all the vertices of G are present. Since the algorithm stops when there is one connected component, the graph G is also connected. Further since we either do not add edges whose vertices are already in S, or only add edges from the vertices of one component to the vertices of the other component, cycles are not formed. Hence S is a spanning tree. We now show that the spanning tree formed by this algorithm and the minimum spanning tree (MST) "T" obtained by any other standard algorithm like Prim's or Kruskal's have the same weight.

Let $e_1, e_2, \ldots e_n$ be the edges of S as added by the algorithm. Suppose the edge $e_k = (u, v)$ differs from that in T. Let P be the path between u and v in T, and let e be the edge by which u is connected to T. Consider $T \cup e_k$. This will create a cycle C in T. Since the algorithm has

picked up e_k, it means that e_k is the edge with the least weight incident on either u or v. Without loss of generality let us assume e_k is the edge of least weight incident on u. Then wt(e) $\geq$ wt(e_k). Therefore, delete e from the tree and replace it by e_k. Continuing this process we can replace every edge that is present in T and not present in S by an edge present in S which is of less than equal to weight. Hence wt(S) $\leq$ wt(T). Since T is a minimum spanning tree. Then wt(S) = wt(T), as T is a minimum spanning tree and the weight of S cannot be less than the weight of T. Therefore, the weight of S must be equal to the weight of T. Hence, S is a minimum spanning tree.

The same proof holds for a multi-graph, as only the least edge for a pair of components is picked up by the algorithm during each iteration. A tie is broken arbitrarily, as it does not affect the weight of the minimum spanning tree.

4. Pseudo Code

The algorithm is as follows:

1 Start
2 For a component u in the graph add the minimum edge e incident on it to it's minimum spanning tree, S.
3 If the number of components in the graph is greater than 1, repeat step 1. Otherwise, if there is only one component, it is required minimum spanning tree, S, of the graph G.
4 End

Pseudo Code

main()

1 Start
2 Input the number of vertices
3 Initialize variables
```
int i,j=0,min,u=0,v=0, components=n;
```
4 Find the edges incident on each vertex that are of minimum weight
```
for(i=0;i<n;i++)
    findMin(G,i);
```
5 Build an array of all the vertices connected to vertex 1.
```
buildVisited(0);
```
6 If the number of components is greater than 1, it means that the minimum spanning tree has not been formed, and the least edge between each component must be added to the tree. If the number of components is 1, go to step
```
while (components>1)
        {
            min = 99;
            for(i=0;i<n;i++)
            {
                if (visited[i] == 1)
                 {

    for(j=0;j<n;j++)
                        {
                            if (visited[j]==0 && min>G[i][j])
                             {

    min = G[i][j];

    u = i;

    v = j;

                            }
                        }
                    }
                    connect[v][u] = 1;
                    buildVisited(u);
            }
            System.out.println(weight);
    }
```
7 End

findMin()

1 Start
2 Initialize the variables
```
int j,min=99,pos=0;
```
3 Find the minimum edge incident on each vertex and add those edges to the tree
```
for(j=0;j<n;j++)
    {
        if(min>X[i][j])
        {
            min = X[i][j];
            pos = j;
        }
```

```
        }
        connect[i][pos] = 1;
        }
```

4 End

buildVisited(u)

1 Start

2 Declare local variables
int i;

3 Set the source vertex u as visited
visited[u] = 1

4 For each unvisited vertex connected to the source vertex, connect the least edge, and recursively find the least edge connected to this vertex as the source

```
        for(i=0;i<n;i++)
        {
                if (connect[i][u] == 1 && visited[i] == 0)
                {
                        components--;
                        System.out.println(i+" - "+u);
                        weight += G[i][u];
                        buildVisited(i);
                }
        }
```

5 End

5. Illustration

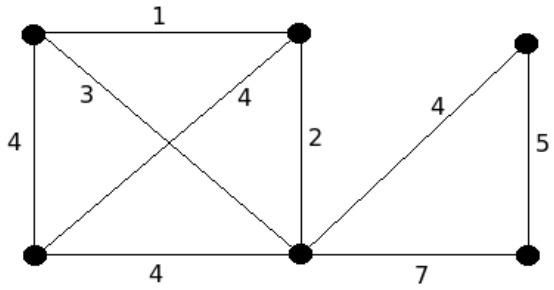


Fig. 1

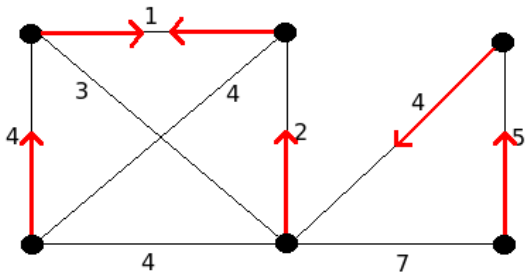


Fig. 2

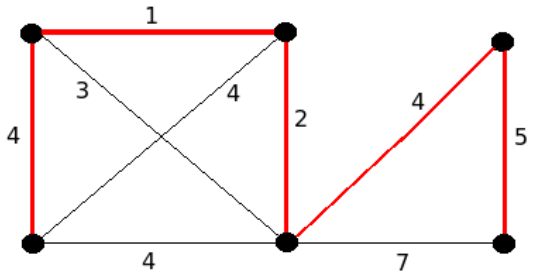


Fig. 3

6. Conclusion

The Minimum spanning tree algorithm has been designed and tested to have a complexity of $O(n^2)$, which is comparable to Prim's algorithm in its adjacency matrix implementation. The applications of MSTs vary over numerous fields; from circuit design-to minimize the number of wires used to connect pins; to biotech-reducing data storage in sequencing amino acids in a protein; to image registration with Renyi entropy or to connect islands with a minimum number of bridges or group of locations with a minimum number of roads.

7. References

[1] http://en.wikibooks.org/wiki/Graph_Theory/Definitions
[2] (http://mjwilcox.net/index/wp-content/uploads/2011/04/undirected_graph_example1.gif)
[3] (http://en.wikipedia.org/wiki/Tree_(graph_theory))
[4] (parallel.ru/info/reference/hpccgloss.html)
[5](http://en.wikipedia.org/wiki/File:Msp1.jpg)