IT300 Design and Analysis of Algorithms

Report on Mini-Project

MINIMUM SPANNING TREE

Group No: 16

Shaulendra Kumar – 201IT159

Mayur Jinde –201IT135

Sanket Hanagandi –201IT154

Details of the research paper

Click here to see the research paper

Authors:

Paryatil

UPN "Veteran" Yogyakarta, Country Indonesia

Krit Salahddine

Ibn Zohr University, Agadir, Country Marocco

Abstract:

The minimum spanning tree, as its name implies, is the tree with the least possible length among all spanning trees. The spanning tree of a graph is generated when each and every vertex of a graph is connected having no cycles in them. A recurrent issue throughout the years has always been determining the minimal spanning tree of a graph. For this problem, a lot of effective algorithms have already been created. In order to reduce the number of edges taken into account while computing the minimal spanning tree of a graph, an alternative approach has been provided in this study.

I. Introduction:

MSTP (minimum spanning tree problem) is a well-known combinatorial optimization problem. It deals with the challenge of constructing a minimumweight tree that spans all of the vertices of a weighted, undirected, and linked graph, with the tree's weight equal to the sum of its edges' weights. It is widely used in a variety of scientific and technological domains, including computer and communication networks, knowledge engineering, wire connections, VLSI circuit design, and a wide range of optimization challenges. The concepts of minimal spanning tree have been extensively used in recent approaches to assessing various biomedical problems such as medical imaging, bio-terrorism, and so on (MST). In reality, MST ideas have been used in recent developments in clustering algorithms.

The two most significant spanning tree algorithms are known as greedy algorithms and are Kruskal's Algorithm and Prim's Algorithm, respectively. For a linked weighted graph, these greedy techniques find the lowest spanning tree. They locate a tree of that graph that has all of the vertex information and has an edge weight total that is either less than or equal to that of every spanning tree that can be found.

II. Existing Methodology:

Base paper Summary:

Prim’s Algorithm of MST:

The basic premise of Prim's algorithm is that all vertices of a spanning tree must be connected. Therefore, in order to create a Spanning Tree, the two distinct subsets of vertices (explained above) need to be joined. And for it to be a Minimum Spanning Tree, they must be connected to the minimum weight edge.

Steps:

1. Initialise the tree with a single vertex from a given graph.

2. Choose the shortest outgoing edge from the node and add this to the tree.

3. The edge formed by the two vertices is treated as a single node, we check for the shortest adjacent edge to the node and add this to the tree.

4. 4. Repeat the steps until all the vertices are added to the tree.

Kruskal's Algorithm of MST:

The greedy method is used by Kruskal's algorithm to determine the minimal cost spanning tree. Picking the minimum weight edge that prevents a cycle in the MST as it has been built thus far is the greedy option.

Steps:

1. Sort all the edges in the increasing order of their cost.

2. Choose the edge which has the smallest cost and check for the cycle. If the edge forms the cycle with the spanning tree discard that edge else include the edge in the spanning tree.

3. Repeat step 2 until all the vertices are included in the spanning tree.

Limitations:

By reading this paper we came to know that Prim's algorithm works more efficiently than Kruskal's algorithm in dense connected graph. But Prim's algorithm requires the need of sorting the edges, so in the proposed algorithm, there is no need of sorting the edges as this method is based on loop detection through node.

III. Proposed Methodology:

Our proposed algorithm is based on detecting the cycle associated with edges/vertices. There are many simple, and almost linear cycle detection techniques available. We select the vertex u with the highest degree from the set of unmarked vertices (V-S), where S is the set of marked vertices. Next, it is determined whether the selected vertex is a part of a cycle or knot. If so, we locate the edge with the highest edge weight (i.e., the most expensive edge) and eliminate it from the cycle as well as the list of edges E. In the event of a tie, the edge with the greatest number of terminal vertices is chosen.

The suggested method chooses the vertex with the highest degree at each stage. This idea is based on the observation that as a cycle's degree rises, so does the likelihood that a vertex will join it. A vertex is more likely to be connected to a cycle the higher its degree. Additionally, when deleting an edge, the tie is determined by the degree of the edge's terminal

vertices. The method's greedy nature is demonstrated by the fact that each stage selects the vertex with the highest degree in order to check cycle associativity.

Snapshot of implemented algorithm:

```
void MST_Algo(int e, int v){
    // set<int>s;
    //cout<<"check2\n";
    int t = 25; //e>v-1
    while (e > (v - 1))
    {
        int node = max_degree.top().first;
        int deg = max_degree.top().second;
        //cout<<"check3\n";
        //cout<<node<<" "<<deg<<" "<<e<<endl;
        // e--;
        // max_degree.pop();
        int k = 1;
        while (1)
        {
            //cout<<"check4\n";
            int color[1000] = { 0 };
            int par[1000] = { 0 };
            int mark[1000] = { 0 };
            int cno = 0;
            vector<int>cycles[1000];
            dfs_cycle(1, 0, color, mark, par, cno);
            //cout<<"check7\n";
            vector<int>temp;
            // cout<<node_member_cycle(e,mark,cno,node,cycles)<<"\n";
            // int m_val=INT_MIN;
            // for(auto x:adj[node])if(x.second>m_val)m_val=x.second;
            // cout<<"\nmval="<<node<<" "<<m_val;
```

IV. Results:

This section depicts the algorithm's application to the input graph illustrated in Figure 1. Because node B has the highest degree, the algorithm chooses it first. It then recognises that node B is linked to the B-A-E-B cycle. Then it chooses the costliest edge and discovers that AE and BE are tied. The terminal vertices of BE, on the other hand, have the highest degree, hence BE is deleted, as illustrated in figure 2 by the dotted red lines.

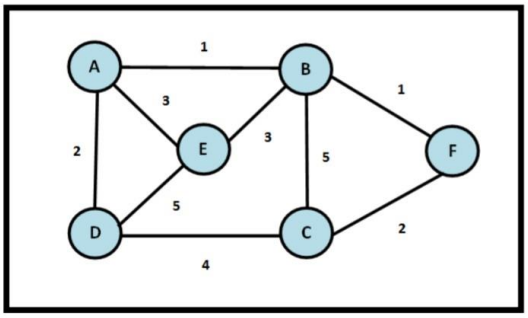

Figure 1: Input Graph

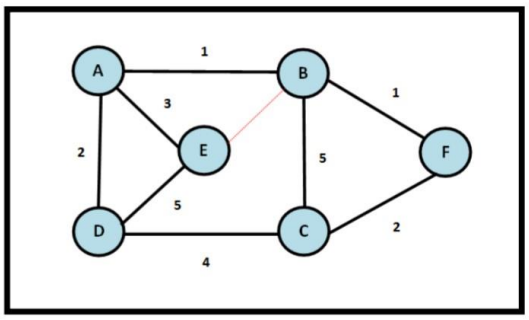

Figure 2: BE deleted

Now the algorithm detects that B is also a member of the cycle B-C-F-B and removes the most expensive edge BC from it (see Fig. 3). Next the cycle BA-D-C-F-B is detected and DC is deleted (see Fig. 4). At this point the node B becomes free of all cycles and hence vertex B is added to the list of marked vertices S.

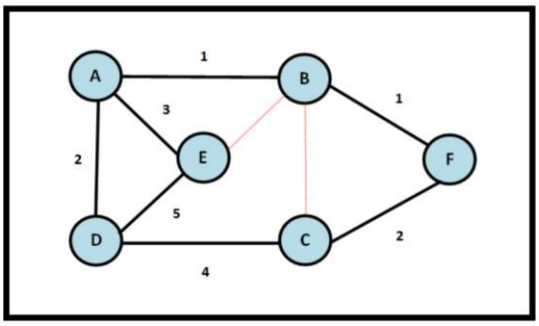

Figure 3: BC deleted

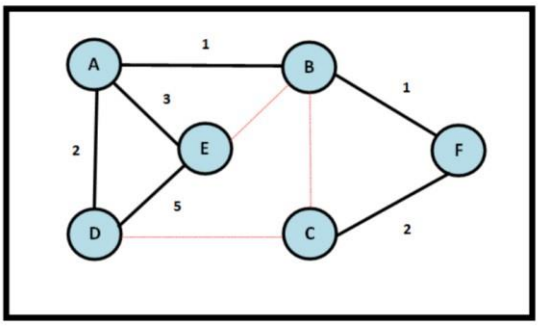

Figure 4: DC deleted

A is the next vertex to be tested. Because node A is linked to the cycle A-D-E-A, the most expensive edge, DE, is deleted, as seen in figure 5. Then A is added to the list of vertices that have been marked.

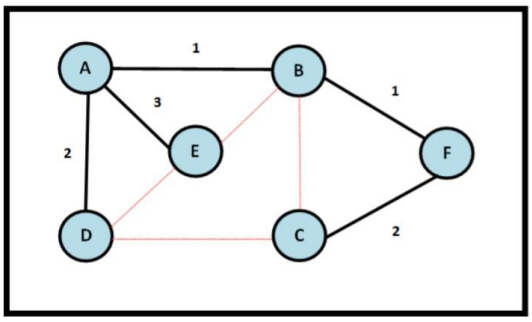

Figure 5: DE deleted

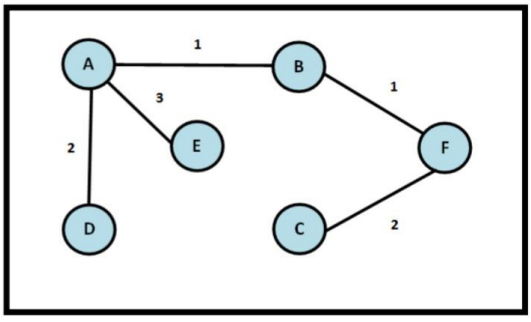

Figure 6: Minimal Spanning Tree

The method finishes at this point because the number of edges is one less than the number of vertices, resulting in the minimal spanning tree illustrated in figure 6. This approach has the advantage of eliminating the requirement to sort the edges. In most circumstances, removing one edge also results in the removal of many cycles. Figure 3 shows

how removing BC from the graph removes four cycles. The collection of marked vertices S assures that the vertices are not chosen twice.

V. Conclusion:

In our study, we have demonstrated a technique that is easily implementable on a single system. It can be improved for distributed systems using a variety of parallel computing approaches (distributed graphs). The benefit of this method is that sorting the weighted graph's edges is not necessary. Furthermore, it becomes considerably faster when numerous cycles are broken by removing one of the edges from a chosen cycle. We may therefore conclude that it is an effective technique for figuring out the smallest spanning tree of a weighted undirected graph.

VI. Reference:

1) AKL, Selim G, 2019, The Desien and analysis of krus kall Algorithms, New Jersey Prentis Hall lnc
2) Budd Timothy, 2019, Object Oriented Programming Reading, Addison Wiley Publisher co lnc
3) Finkel,R.A and j.L.Bentley, 2019, Quadtrees A Data Structures For Retrieval on Composite Key, A cta Informatika Vol 4. Rlo.1