RESEARCH ARTICLE OPEN ACCESS

Review and Analysis of Minimum Spanning Tree Using Prim's Algorithm

Jogamohan Medak
Assistant Professor
North Lakhimpur College (Autonomous), North lakhimpur
Assam – India

ABSTRACT
The spanning tree of a graph consist of all vertices some of the edges, so that the graph does not contain a cycle. A Minimum Spanning Tree (MST) is subset of a graph that connects all vertices with minimum possible total edge weights. There are different algorithms that are used to define a Minimum Spanning Tree of a graph i.e. Prim's Algorithm and Kruskal's algorithm. The main objective of this paper is to discuss and analyse the formation of Minimum Spanning Tree using Prim's Algorithm. This paper also includes a graphical representation of the algorithms explaining with the functionalities.
Keywords:- Minimum Spanning Tree (MST), Prim's Algorithm, Kruskal's algorithm,Travelling salesman problem.

I. INTRODUCTION

In graph theory, a Tree is a subset of a directed or undirected connected graph in which vertices are linked with exactly one path without any cycles. A spanning tree of a connected graph can be constructed including all the vertices with minimum possible no of edges. If there are n vertices in the graph, then each spanning tree has n-1 edges. A connected weighted graph where all the vertices are interlinked by some weighted edges can contain multiple numbers of spanning trees. A spanning tree with minimum possible total edge weight is called as a Minimum Spanning Tree (MST). In the minimum spanning tree a set of edges is selected such that there is a path between each node and the sum of the edge weights is minimal.

A minimum spanning tree can be constructed using greedy algorithms i.e. Prim's and Krushkal algorithm. These algorithms are considered as greedy algorithm as they find the best smallest weight edges to build a MST. The kruskal's algorithm firstly processes the edges according to their weights from lowest to largest value. It takes the lowest valued edge and added to the MST. This process continues until all the vertices are visited. The Prim's algorithm is based on graph traversals. It first selects a random node from the graph and in each traversal, examines all edges from the visited node to non-visited nodes to select a minimum edge and then the edge is added to the MST.

II. PRIM'S ALGORITHM

Prim's algorithm is a greedy algorithm that is used to find a minimum spanning tree (MST) of a given connected weighted graph. This algorithm is preferred when the graph is dense. The dense graph is a graph in which there is a large number of edges in the graph. This algorithm can be applied to only undirected connected graph and there should not be any negative weighted edge. In this case, the algorithm is quite efficient. Since there are no non-negative weight cycles, there will be a shortest path whenever there is a path.

The steps to find minimum spanning tree using Prim's algorithm are as follows:

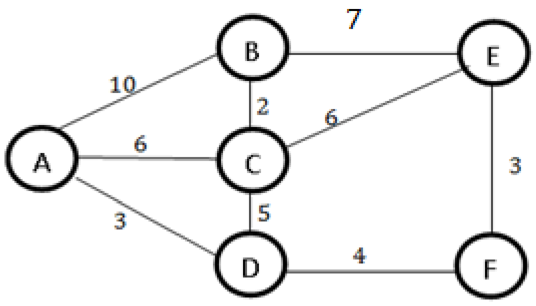
1. If graph has loops and parallel edges than remove loops and parallel edges of that graph.
2. Randomly choose any node, labelling it with a distance of 0 and all other nodes as ∞. The chosen node is treated as current node and considered as visited. All other nodes are considered as unvisited.
3. Identify all the unvisited nodes that are presently connected to the current node. Calculate the distance from the unvisited nodes to the current node.
4. Label each of the vertices with their corresponding weight to the current node, but relabel of a node, if it is less than the previous value of the label. Each time, the nodes are labelled with its weights; keep track of the path with the smallest weight.
5. Mark the current node as visited by colouring over it. Once a vertex is visited, we not need to look at it again.
6. From all the unvisited nodes, find out the node which has minimum weight to the current node, consider this node as visited and treat it as the current working node.
7. Repeat steps 3, 4 and 5 until all nodes are visited.

8. After completed all steps get desired MST.

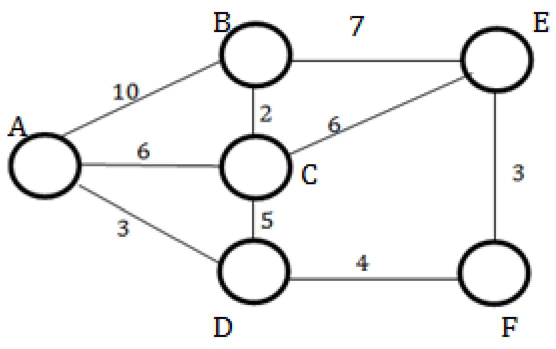
Algorithm:
/* S= set of visited node, Q= Queue, G=Graph, w=Weight */

```
Prim_MST (G, w, S)
{
        Initialization(G,S)
        S ←Ø                                  // the set of visited nodes is initially empty
        Q← v[G]                               // The queue is initially contain all nodes
        while (Q= Ø)                          // Queue not empty
                do u← extract_min(Q)      // select the minimum distance of Q
                        S ←SU{u}              // the u is add in visited set S
                For each vertex v ϵ adj(u)
                        do relax(u, v, w)
}
Initialization (G, S)
{
For each vertex v ϵ adj (u)
 d [v] ←∞                   //Unknown Distance from source node
π[v] ←nil                                 // Predecessor node initially nil.
d[s] ←0                         //Distance of source nodes is zero
}
Relax (u, v,w)
{
 If d[v]>w[u,v]                 // comparing new distance with existing value
        {
                d[v] ← w[u,v]
                π[u] ←u
        }
}
```

<u>Using Prim's algorithm, calculation the minimum spannin tree is given in the following graph .</u>

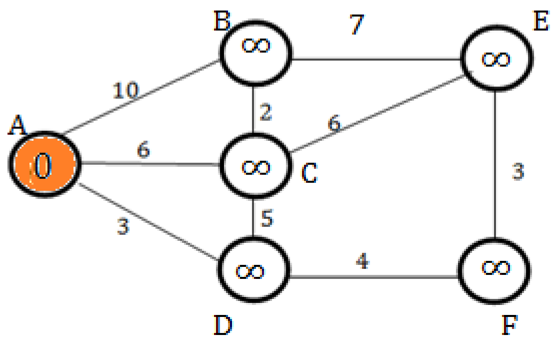


Step1: Redraw the graph as following:



Consider the source node as A. Therefore, A will be the current working node and can be treated as visited.

Nodes	A	B	C	D	E	F
Distance d[v]	0	∞	∞	∞	∞	∞
Distance from π[v]	A					
Status	Visited	Unvisited	Unvisited	Unvisited	Unvisited	Unvisited



Step2:

We can calculate the weights of unvisited adjacent nodes of current node A. The unvisited nodes of A are B, C, and D.

Now, Distance [B]> distance (A, B) ⇨ ∞ > 10

So, relabel is required in node B.

Distance [C]> distance (A, C) ⇨ ∞ > 6

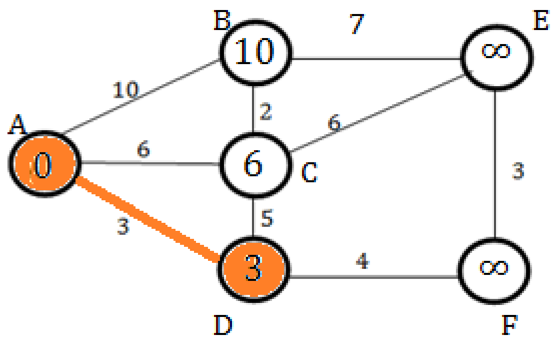
So, relabel is required in node C.

Distance [D]> distance (A, D) ⇨∞ > 3

So, relabel is required in node D.

Nodes	A	B	C	D	E	F
Distance d[v]	0	10	6	3	∞	∞
Distance from π[v]	A	A	A	A		
Status	Visited	Unvisited	Unvisited	Unvisited	Unvisited	Unvisited

We can discover the smallest distance from A to the unvisited node which is D, traversing through A to D and now D can be considered as the current node.



Step3:

We can calculate the weights of unvisited adjacent nodes of current node D. The unvisited nodes of D are C and F. Now,

Distance[C] > Distance (D, C) ⇨ 6 > 5

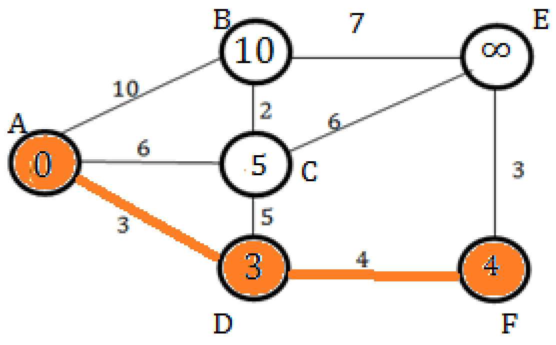
So, relabel is required in node C.

Distance [F] > Distance (D, F) ⇨ ∞ > 4

So, relabel is required in node F.

Nodes	A	B	C	D	E	F
Distanced[v]	0	10	5	3	∞	4
Distance from π[v]	A	A	D	A		D
Status	Visited	Unvisited	Unvisited	Visited	Unvisited	Unvisited

We can discover the smallest distance from D to the unvisited node which is F, traversing through D to F and now F can be considered as the current node.



Step 5:

Unvisited adjacent of current working node of F is E

Now,

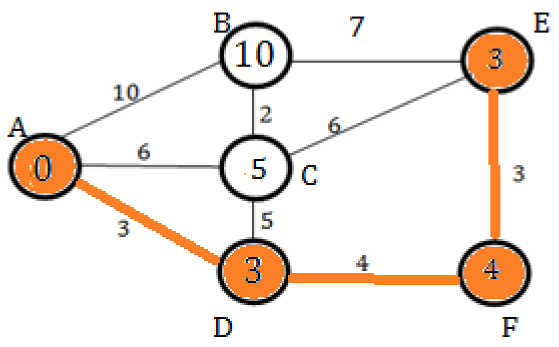
Distance [E]> Distance (F, E) ⇨ ∞ > 3

So, relabel is required in node E.

Nodes	A	B	C	D	E	F
Distance d[v]	0	10	5	3	3	4
Distance from π[v]	A	A	D	A	F	D

Status	Visited	Unvisited	Unvisited	Visited	Unvisited	Visited

We can discover the smallest distance from F to the unvisited node which is E, traversing from F to E and now E can be considered as the current node.



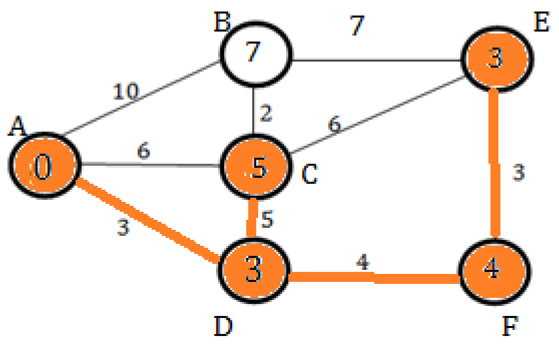
Step 5:

Unvisited current node of E is C and B.
Now,

Distance [B]> Distance (E, B) ⇨ 10 > 7
So, relabel is required in node.
Distance [C]> Distance (E, C) ⇨ 5< 6
So, relabel is not required.

Nodes	A	B	C	D	E	F
Distance d[v]	0	7	5	3	3	4
Distance from π[u]	A	E	D	A	F	D
Status	Visited	Unvisited	Unvisited	Visited	Visited	Visited

We can discover the smallest distance from D to the unvisited node which is C, traversing from D to C and now C can be considered as the current node.



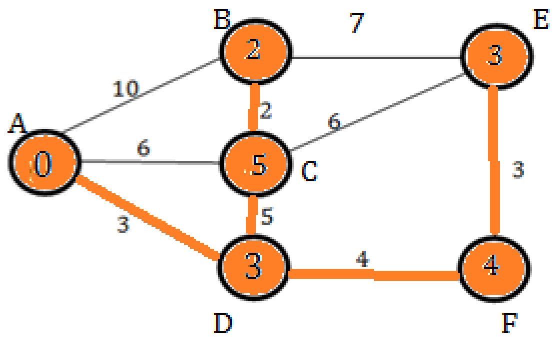
Step6:

From current node C the unvisited node is B.
Distance [B] > Distance(C, B).
So, relabel is required.

Nodes	A	B	C	D	E	F
Distance d[v]	0	2	5	3	3	4

Distance from π[v]	A	C	A	A	F	D
Status	Visited	Unvisited	Visited	Visited	Visited	Visited

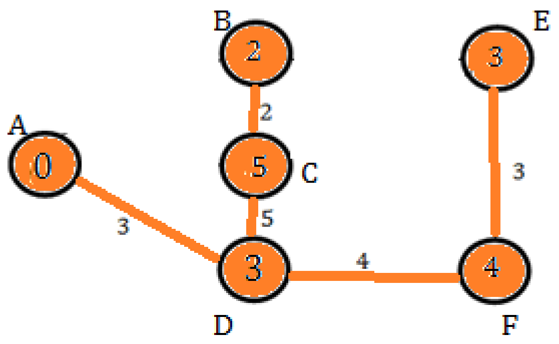
Now, only unvisited node is B. So, Traverse from C to B and mark B as visited.



The final table is:

Nodes	A	B	C	D	E	F
Distance d[v]	0	2	5	3	3	4
Distance from π[v]	A	C	A	A	F	D
Status	Visited	Visited	Visited	Visited	Visited	Visited

Therefore, the minimum spanning tree of the given graph is given below:



The weight of minimum spanning tree will be:

= (A, D) + (D, F) + (F, E) + (D, C) + (C, B)

= 3+4+3+5+2

= 17

III. COMPLEXITY

Finding the minimum distance is O (V) and overall complexity with adjacency list representation is O (V^2). If queue is kept as a binary heap, relax will need a decrease-key operation which is O (logV) and the overall complexity is O (V log V + E log V) i.e. O (E log V). If queue is kept as a Fibonacci heap, decrease-key has an amortized complexity O (1) and the overall complexity is O (E+V log V).

IV. CONCLUSION

The time complexity for Prim's algorithm is acceptable in terms of its overall performance to finding minimum spanning tree. Minimum Spanning Tree concept is mostly used in network analysis, water supply network, transportation network, cluster analysis, circuit design and to solve the travelling salesman problem of the graph theory. The Prim's algorithm is preferred to dense graphs but for sparse graph the Kruskal's algorithm is used for better results.

REFERENCE

[1].Srivastava, S.K, Srivastava, Deepali. Data Structures Through C in Depth. BPB publications.2011.

[2].Baluja, G.S. Data Structures Through C. Dhanpat Rai & Co.(Pvt) Ltd.2008

[3]. Kairanbay, M., Jani, H. M.(2013). A Review And Evaluations Of Shortest Path Algorithms. international journal of scientific & technology research volume 2, issue 6.

[4]. Ojekudo, N.A., Akpan, N. P.(2017). Anapplication of Dijkstra's Algorithm to shortest route problem. Volume 13, Issue 3, P 20-32.

[5]. Jean-Paul Tremblay & Paul G .An Introduction to Data Structures with Applications. Sorenson Publisher - Tata McGraw Hill.

[6]. Patel, R.B..Expert in Data Structure with C (3 rd Edition). New Delhi: Khanna Book Publishing.

[7]. Ellis Horowitz & Sartaj Sahni: Fundamentals of Computer Algorithms (1993), Galgotia Publications.

[8]. Minimum spanning tree. Available at https://en.wikipedia.org/wiki/Minimum_spanning_tree.

[9]. Greedy Algorithms. Available at https://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-minimum-spanning-tree-mst-2/.

[10]. Prim's Algorithm. Available at https://en.wikipedia.org/wiki/Prim%27s_algorithm.

[11]. Prim's Algorithm for Minimum Spanning Trees Explained Step by Step. Available at https://www.youtube.com/watch?v=X-QdRmgekpE&t=1s.

[12]. Prims Algorithm : Greedy Technique. Available at https://www.youtube.com/watch?v=kuNfKbyOnDs&t=529s.

[13]. Difference between Prim's and Kruskal's Algorithm. Available at https://www.youtube.com/watch?v=Ryo0Mu5SeqU.

[14]. Prim's algorithm for Minimum Spanning Tree. Available at https://www.youtube.com/watch?v=xENj6Ns3CfU&t=18s.