A Review on Minimum Spanning Tree Algorithms

M. K Salitha
Assistant Professor
Department of Computer Science & Engineering
Musaliar College of Engineering & Technology, Kerala, India

Abstract— The spanning tree is subset of a graph, which consists of all its vertices covered with minimum number of edges. A spanning tree does not contain cycle. A minimum spanning tree of an edge weighted graph is a spanning tree with the smallest possible weight than every other spanning tree for the same graph. Kruskal's algorithm and Prim's algorithm are the two widely used algorithms to obtain the minimum spanning tree of a graph. This paper aims at reviewing the construction of minimum spanning tree using Prim's algorithm and Kruskal's algorithm.

Keywords: Minimum Spanning Tree, Prim's Algorithm, Kruskal's Algorithm

I. INTRODUCTION

A connected and acyclic undirected graph is known as a tree. A spanning tree of a graph G consists of all the vertices connected by minimum number of edges. Thus a spanning tree is still a connected graph with no cycles. A number of spanning trees can be obtained from a connected edge-weighted graph. The weight of a spanning tree is the sum of weight of the edges in the spanning tree. The spanning tree with the minimum weight is called the minimum spanning tree. Minimum spanning trees find application in many areas including pattern recognition [1][4][5], network reliability[1][9], chemical physics[1][3], network designs and cluster analysis.

The two well-known Greedy approaches for constructing a minimum spanning tree are Prim's algorithm and Kruskal's algorithm. The Greedy technique always makes a choice that is best at that instant. Both Prim's and Kruskal's algorithm work on undirected edge-weighted graphs.

II. DESCRIPTION OF PRIM'S ALGORITHM

Prim's algorithm uses greedy technique for finding a minimum spanning tree of an edge-weighted connected graph. The memory access pattern and memory organization has a great impact on the performance of
Prim's algorithm [2][10]
The main steps of Prim's algorithm are as follows:

1) Step 1: Remove all loops and parallel edges if any from the graph. With regard to parallel edges, the one with least cost/weight is retained while all the other nodes are removed.
2) Step 2: A node is chosen randomly as the current node and marked as visited. Its distance is set to zero. The distance of all the other nodes are set to infinity.
3) Step 3: All the unvisited nodes with their edges incident on the current node are considered. The edge with the smallest cost/weight is chosen.
4) Step 4: The other node of the smallest cost/weight edge chosen in step 3 becomes the new current node.
5) Step 5: Steps 3 and 4 are repeated until no more nodes are left unvisited.
6) Step 6: Completion of the above steps results in a minimum spanning tree.

The time complexity of Prim's algorithm is $O(V^2)$ if the graph is represented using an adjacency matrix. Figure1 shows an edge-weighted undirected graph G = (V, E) with six vertices and nine edges labelled with cost/weight.

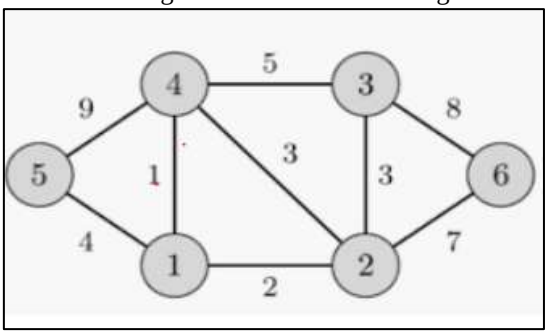

Fig. 1: An edge-weighted undirected graph

The graph can be represented using an adjacency matrix to keep track of the nodes adjacent to a given input node. The adjacency matrix for the graph of figure 1 is shown below (in Table 1).

	1	2	3	4	5	6
1	0	2	0	1	4	0
2	2	0	3	3	0	7
3	0	3	0	5	0	8
4	1	3	5	0	9	0
5	4	0	0	9	0	0
6	0	7	8	0	0	0

Table 1: Adjacency Matrix for Figure 1

The time complexity of Prim's algorithm can be reduced if the graph is represented using an adjacency list.

A. Algorithm 1: Algorithm for implementing Prim's algorithm.

```
/* G=graph, c=cost/weight, r=root node, N=set of nodes
visited, Q=queue */
PRIMS_MST (G, c, r)
{
for each vertex u Є V [G]
//distance of all other nodes set to infinity
distance [u] = ∞
// Predecessor node is initially empty
parent [u] = NIL
// set of nodes visited is initially empty
N = Φ
//distance of root node is set to zero
distance [r] = 0
//All nodes are initially placed in the queue
```



```
Q = v [G]
while (Q ≠ Φ )
u=EXTRACT_MIN (Q)
Q = Q-u
If parent (u) ≠ NIL
/* node and its parent node are included in the visited set N
*/
N = N U (u, parent (u))
/* obtain the nodes adjacent to vertex u from the adjacency
matrix */
for each V Є adj [u]
//compares current distance with existing value
if V Є Q and c (u, v) < distance [v]
parent [v] = u
distance [v] = c (u, v)
}
```

Prim's algorithm is easy to understand. As root node is chosen in the beginning there is clarity about the starting node.

III. Description of Kruskal's Algorithm

Kruskal's algorithm uses greedy approach for finding the minimum spanning tree of an edge-weighted connected graph. This algorithm treats every node in the graph as an individual tree. Thus the graph is treated as a forest.

The main steps of Kruskal's algorithm are as follows:

1) Step 1: Remove all loops and parallel edges if any from the graph. In case of parallel edges, the one with least cost/weight is retained while all others are removed.
2) Step 2: Arrange all the edges in increasing order of their cost/weight.
3) Step 3: Choose an edge with the smallest cost/weight. Check whether including this edge forms a cycle in the spanning tree formed so far. If cycle is formed the edge is discarded. Else it is included.
4) Step 4: Repeat step 3 until all vertices are considered.

The algorithm for implementing Kruskal's algorithm [12] (in Algorithm 1) mainly uses three functions namely,

- MAKE_SET()
- FIND_SET()
- UNION()

MAKE_SET (v) creates a set containing only the element v.
FIND_SET (u) finds the subset to which node u belongs.
UNION (u,v) combines the two subsets.

A. Algorithm 1: Algorithm for implementing Kruskal's algorithm.

```
// G = Graph, c = cost/weight
Kruskals_MST (G, c)
{
//Initialize set of edges N to null
N = Φ
for each vertex v Є V [G]
//creates |v| trees, one containing each vertex
MAKE_SET (v)
Sort the edges of G into increasing order by cost/weight
for each edge (u, v) Є E in increasing order by cost/weight
If FIND_SET (U) ≠ FIND_ SET (V)
// add the edge (u, v) to set N
Then N = N U {(u, v)}
//merges the vertices in the two trees
UNION (u, v)
return N
}
```

Kruskal's algorithm is easy to understand and works well on graphs with large number of edges and vertices.

IV. Comparison between Prim's and Kruskal's Algorithm

The time complexity of Prim's algorithm is O (ElogV) when implemented using a binary heap. However, the time complexity is reduced to O(E+VlogV) if implemented using Fibonacci heap. The space complexity of Prim's algorithm [1] is 4E+5V.

The time complexity of Kruskal's algorithm is O(ElogV) or O(ElogE). The space complexity of Kruskal's algorithm [16] is Θ (E+V).

The time complexity [11] and space complexity of both Prim's algorithm and Kruskal's algorithm is summarized in Table 2.

Algorithm	Time Complexity	Space Complexity
Prim's algorithm	O(ElogV) using binary heap O(E+VlogV) using Fibonacci heap	4E+5V
Kruskal's algorithm	O(ElogV) or O(ElogE)	Θ (E+V)

Table 2: Complexity of Prim's and Kruskal's algorithm

V. Conclusion

This paper presents a review on the two minimum spanning tree construction algorithms namely Prim's algorithm and Kruskal's algorithm. It is observed that Prim's algorithm is faster for dense graphs and Kruskal's algorithm is faster for sparse graphs. The space requirement of Prim's algorithm is less compared to Kruskal's algorithm.

References

[1] R. E. Haymond, J. P. Jarvis and D. R. Shier, Computational Methods for Minimum Spanning Tree Algorithms, SIAM J. SCI. STAT. COMPUT. Vol. 5, No. 1, March 1984
[2] Badri Munier, Muhammad Aleem, Muhammad Arshad Islam, Muhammad Azhar Iqbal, A Fast Implementation of Minimum Spanning Tree Method and Applying it to Kruskal's and Prim's Algorithms, Vol. 1, No. 1 | Jan – June 2017
[3] F. H. Stillinger, JR., Physical clustering, surface tension, and critical phenomena, J. Chem. Phys., 47 (1967), pp. 2513-2533
[4] R. Clark and W. F. Miller, Computer-based data analysis systems, in Methods in Computational Physics, Vol. 5, B. Alder et al., eds, Academic Press, New York, 1966, pp. 47-98

[5] R. E. Osteen and P. P. Lin, Picture'skeletons based on eccentricities ofpoints of minimum spanning trees, SIAM J. Comput., 3 (1974), pp. 23-40
[6] G. J. S. Ross, Algorithm AS13: minimum spanning tree, Appl. Statistics, 18 (1969), pp. 103-104.
[7] E. Horowitz and S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, Potomac, MD, 1978, pp. 174-183
[8] A. V. Aho, J. E. Hopcraft AND J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974
[9] A. Kershenbaum and R. Van Slyke, Computing minimum spanning trees efficiently, Proc. ACM National Conference, 1972, pp. 518-527
[10] J. Adams and E. Shoop, "Teaching shared memory parallel concepts with openmp," Journal of Computing Sciences in Colleges, vol. 30, no. 1, pp. 70–71, 2014
[11] Ilham Mulya Rafid, Performance evaluation for Kruskal's and Prim's Algorithm in Minimum Spanning Tree using Networkx Package and Matplotlib to visualizing the MST Result, May 2019
[12] Thomas H. Cormen, "Introduction to Algorithms", 3rd Edition, MIT Press, 2009
[13] B. Munier, M. Aleem, M. Islam, M. Iqbal and W. Mehmood, "A Fast Implementation of Minimum Spanning Tree Method and Applying it to Kruskal's and Prim's Algorithms", Sukkur IBA Journal of Computing and Mathematical Sciences, vol. 1, no. 1, p. 58, 2017
[14] Haiming Li, Qiyang Xia, Yong Wang, Research and Improvement of Kruskal Algorithm, Journal of Computer and Communications, 2017, 5, 63-69
[15] Jogamohan Medak, Review and Analysis of Minimum Spanning Tree Using Prim's Algorithm, International Journal of Computer Science Trends and Technology (IJCST) – Volume 6 Issue 2, Mar - Apr 2018
[16] Nimesh Patel, Dr. K. M. Patel, A Survey on: Enhancement of Minimum Spanning Tree, Int. Journal of Engineering Research and Applications, Vol. 5, Issue 1(Part 3), January 2015, pp.06-10
[17] GaBow, H. N., Galil, Z., Spencer, T., Tarjan, R. E, "Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graph", Combinatorial - 6(2), 109- 122 (1986)
[18] Sunny Dagar, "Modified Prim's Algorithm", IJCIT, ISSN 2218-5224 (Online), Vol 03, Issue 02, 2012.

