To find Minimum Spanning Tree using Prim's Algorithm

Indian Institute of Information Technology, Allahabad

Aditya Aggarwal
iit2019210@iiita.ac.in

Divy Agrawal
iit2019211@iiita.ac.in

Aman Rubey
iit2019212@iiita.ac.in

***Abstract*—In this paper we will use Prim's Algorithm to find the Minimum Spanning Tree. Also in the later part of the paper we have discussed the space and time complexity of the devised algorithm.**

***Index Terms*—Algorithm, Graph's, Minimum Spanning Tree, Prim's Algorithm, Complexity**

I. Introduction

Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

II. Algorithm Design

We have been given a weighted graph by the user whose MST(minimum spanning tree) we have to find. The following steps will help us to find the MST for a given graph.

1) Create a boolean flag array(initialized to false) which has size as number of vertices which will be used to track that which vertex is taken for MST and which not.
2) Create another vector namely, 'min_weight' of same size where we will store the weight of the minimum weighted edge its connected to, such that the other endpoint of the edge is already taken in flag array.
3) Consider any vertex as the starting node and so flag[start_node] will be made true as starting node is included in MST and also min_weight for it will 0.
4) Now, iterate for (vertices-1) times and in every iteration follow next steps:
   a) Get the the minimum weighted index or (u).
   b) A minimum weighted index is that index for which the corresponding vertex is not taken and also it has minimum key in the min_weight vector.
   c) Include u to flag array.
   d) For every v in V update the min_weight , where V is set of all vertices for which graph(u, v)¿0.
   e) To update min_weight for 'v': if graph(u,v) ¡ min_weight[v] then min_weight[v]=graph(u,v)
   f) Here graph is a vector which is used to store the original graph in the form of adjacency matrix.

After performing the above steps the Tree obtained will be a MST.

III. Pseudo Code

MST using Prim's Algorithm

Global Variables:
int vertices;

function main()
Get number of vertices;
Get the edges in form of u v weight(u,v);
store the edges in form adjacency matrix;
graph[u-1][v-1] $\leftarrow$ $weight$;
$graph[v-1][u-1] \leftarrow weight$;
$call\ Prim\_Min\_spanning\_Tree\ function$;
$return\ 0$;

function $Prim\_Min\_spanning\_Tree(adjacency\ matrix\ graph)$
$Define\ vector\ of\ int\ min\_spanning\_tree(vertices)$
$Define\ vector\ of\ int\ min\_weight(vertices)$
$Define\ vector\ of\ bool\ flag(vertices)$
$As\ there\ will\ be\ no\ edges\ in\ MST\ so$
$min\_weight\ will\ be\ initialized\ as\ INT\_MAX$
$flag\ will\ be\ initialized\ as\ false$
$min\_weight[0] \leftarrow 0$
$min\_spanning\_tree[0] \leftarrow -1$
$for(int\ i = 0;\ i < vertices - 1;\ i++)$
$int\ u = getMinWeight(min\_weight,\ flag)$;
$flag[u] = true$;
$for(int\ v = 0;\ v < vertices;\ v++)$
$if\ (graph[u][v]\ and\ flag[v] == false$
$and\ graph[u][v] < min\_weight[v])$
$min\_spanning\_tree[v] = u$;
$min\_weight[v] = graph[u][v]$;
$endif$
$endfor$
$endfor$
$return\ min\_spanning\_tree$

function $getMinWeight(min\_weight,\ flag)$
$Find\ the\ vertex\ not\ visited\ yet$
$and\ having\ minimum\ value\ in\ min\_weight$
$return\ vertex$

IV. Complexity Analysis

In the next two sub-sections we analyse the time and space complexity of algorithm we devised for the given problem.

A. Time-Complexity

Let V denote the number of vertices in the given graph whose MST we have to find.
Taking the reference of the above Pseudo Code, the outer for loop of the Prim_Min_spanning_Tree function is running V times and the getMinWeight function is also running V times. Similarly the Inner for loop is also running V times.
Therefore,

$$T(V) = O(V * (V + V)) + C_1 \tag{1}$$

which is equivalent to,

$$T(V) = O(2V^2) + \Theta(1) \tag{2}$$

which is in turn equivalent to,

$$T(V) = O(V^2) \tag{3}$$

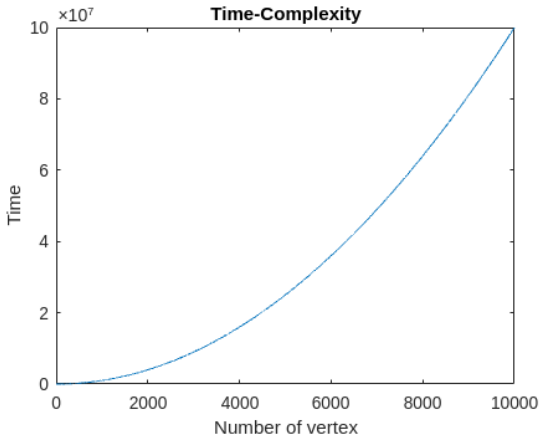

Fig. 1. Time–Complexity Graph ($O(V^2)$)

B. Auxiliary Space-Complexity

Let V denote the number of vertices in the given graph whose MST we have to find.
Taking the reference of the above Pseudo Code, we have created three vectors of size V mainly min_weight, flag, min_spanning_tree.
Therefore,

$$S(V) = O(3 * V) + C_1 \tag{4}$$

which is equivalent to,

$$S(V) = O(V) \tag{5}$$

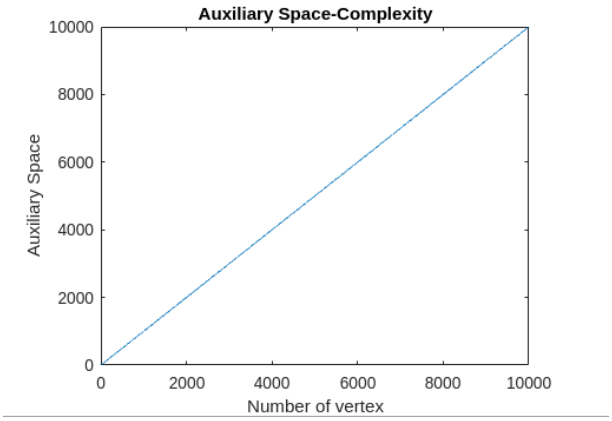

Fig. 2. Auxiliary Space–Complexity Graph (O(V))

V. Conclusion

Therefore, using the concept of Prim's Algorithm we can find the Minimum Spanning Tree of a given graph in $O(V^2)$ time and O(V) auxiliary space.

References

[1] Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
[2] Algorithm Design by J Kleinberg and E Tardos
[3] Prim's Algorithm
[4] Complexity Analysis
[5] IEEE Paper