Minimum Spanning Tree problem And Single-Source Shortest-Paths problem

ÖMER CENGİZ

University of Beira Interior- Erasmus Student

a47124@ubi.pt

16260056@firat.edu.tr

Portugal, Covilha

Abstract- For any other node v in the G graph, the shortest path between s and v is a path along which the total weight of the edges will be minimized. should find a spanning tree such that it is the shortest path.

If the graph is edge-weighted, we can define the weight of a spanning tree as the sum of the weights of all its edges. A minimum spanning tree is a spanning tree whose weight is the smallest among all possible spanning trees.

Keywords: Minimum Spanning tree, Dijkstra's Algorithm, Prim's Algorithm, Single-Source Shortest-Paths problem

1. Minimum Spanning Tree

A minimum raster tree is a tree produced by using the edges in the graph in such a way that it will generate the lowest total cost and include all the nodes (Vertices/Nodes).

1.1. What is Prim's algorithm?

Prims algorithm is a greedy algorithm that finds the minimum spanning tree in a weighted undirected graph.

Prim's algorithm, which is normally time complexity $O(V^2)$, is optimized as O(ElogV) using minimum binary heap tree or PriorityQueue, O(E+ VlogV) using Fibonacci heap tree. can be done.

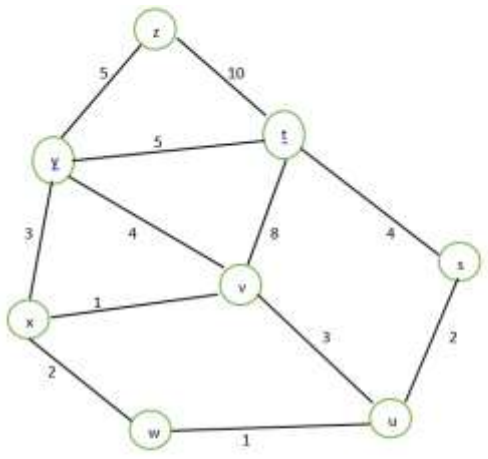
In summary, Prim's algorithm allows us to traverse the nodes in a graph with the least cost.

The most famous algorithms that give the minimum scan tree:

- Kruskal's Algorithm
- Prims Algorithm

1.2. Prim's Algorithm

Primsy, which is a minimal spanning tree (minimum spanning tree), the signs move forward by adding the nearest node to the neighborhoods. Accordingly, let's extract the minimum tree:



In the above graph, a representative letter is assigned for each node and a weight value for each link. Accordingly, the cost of going from each node to the other is determined.

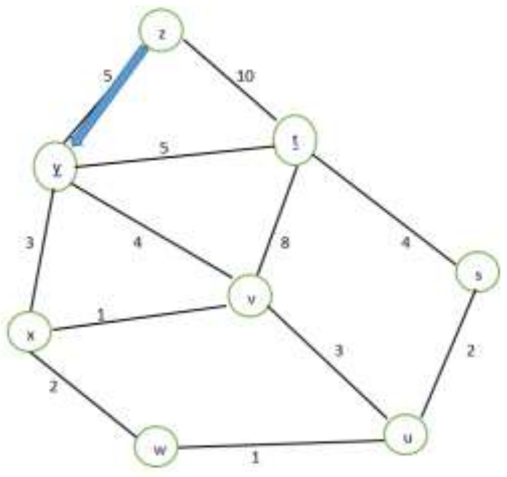
In the premium algorithm, a random starting point is chosen. For example, our starting point is the node z. In this case, the first neighborhood we will examine will be the nodes that can be reached from the "z" node and their costs.

The nodes that can be navigated from the z node and their costs are listed below:

y:5

t:10

The premium algorithm includes the neighbor with the least cost in this list. Accordingly, our new members will be {z,y} and the paths traveled will be {z-y:5}. (The first members list contains nodes that have been visited so far. A node that is already in this list of nodes cannot be added to the list. In the paths list, the cost of going from which node to is kept.) Therefore, the nodes marked by the Prim algorithm in our graph are shown below:



Now let's list the neighbors of all the nodes in the list where our members are standing:

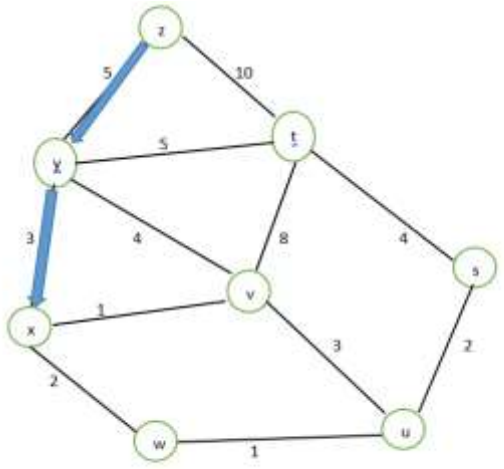
t:5

t:10

v:4

x:3

There are two different routes to node t in the above list (both via z and y). Let's continue with our algorithm and include the smallest path. The nearest neighbor is x:3. In this case our members will be {z,y,x} and our paths will be {z-y:5,y-x:3}. This situation is illustrated in the chart below:



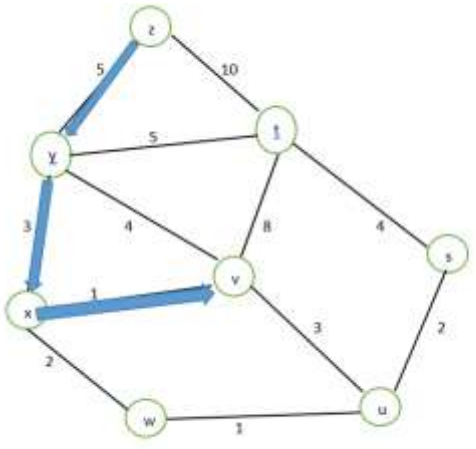
If we list our neighbors again:

t:5

v:1

w:2

If there is more than one route to a node from the island we have included in the list above, the shortest one is taken. In this case, the smallest element of the list, v:1, is preferred, and our members {z,y,x,v} become {z-y:5,y-x:3,x-v:1}. The situation is shown in the chart below:



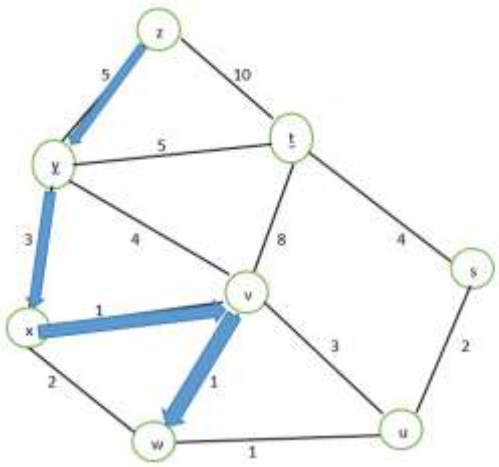
If we list our neighbors again:

t:5

u:3

w:1

If there is more than one route to a node from the island we have included in the list above, the shortest one is taken. In this case, the smallest element of the list, w:1, is preferred, and our members {z,y,x,v,w} become {z-y:5,y-x:3,x-v:1,v-w:1}. The situation is shown in the chart below:

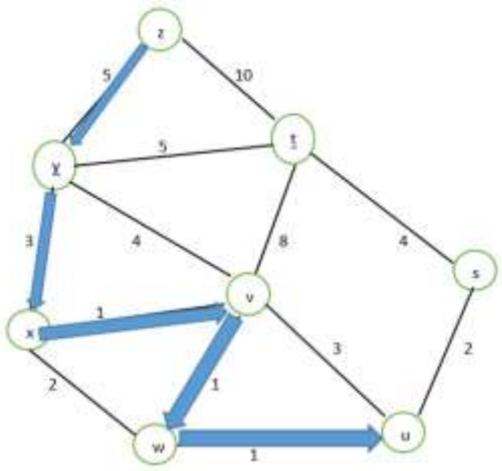


If we list our neighbors again:

t:5

u:1

If there is more than one route to a node from the island we have included in the list above, the shortest one is taken. In this case, the smallest element of the list, u:1, is preferred and our members {z,y,x,v,w,u} are {zy:5,yx:3,xv:1,vw:1,wu:1} happens. The situation is shown in the chart below:

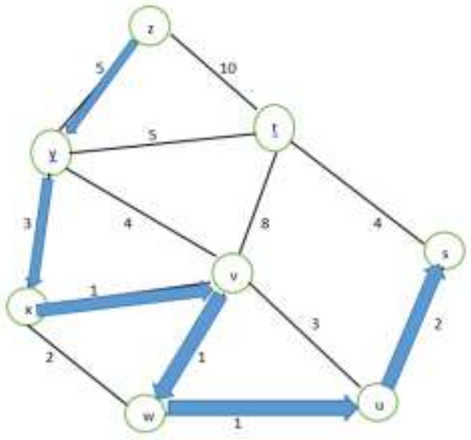


If we list our neighbors again:

t:3

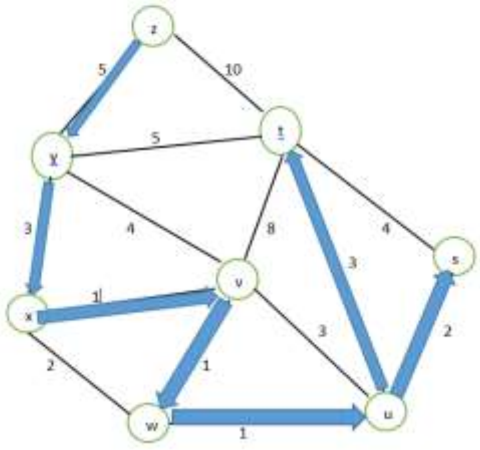
p:2

If there is more than one route to a node from the island we have included in the list above, the shortest one is taken. In this case, the smallest element of the list, s:2, is preferred, and if our members are {z,y,x,v,w,u,s}, {zy:5,yx:3,xv:1,vw:1,wu: 1,us:2}. The situation is shown in the chart below:



The shortest transport for t, our last neighbor

The value t:3 is provided over u. In this case, the smallest element of the list, t:3, is preferred and our members {z,y,x,v,w,u,s,t} are {zy:5,yx:3,xv:1,vw:1, It becomes wu:1,us:2,ut:3}. The resulting minimum hatch tree is given below:

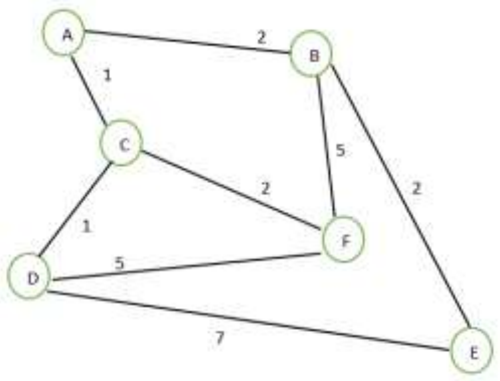


2. Single-Source Shortest-Paths problem

Given a connected, undirected graph G, the shortest path tree rooted at vertex v is a spanning T tree of G such that the path distance from root v to any other vertex u in T is the shortest path from v to u. Is the road distance.

2.1. Dijkstra Algorithm

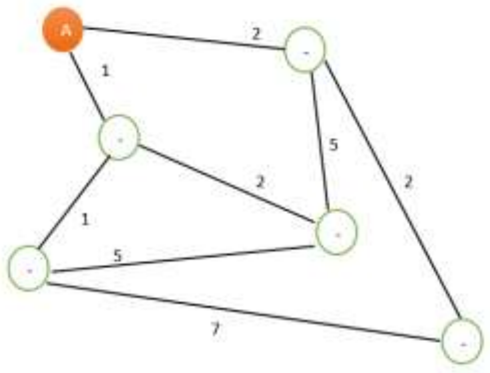
The dijkstra algorithm, which is used in computer science and named after the person who brought the algorithm to the literature, is used to find the shortest path in a given graph.



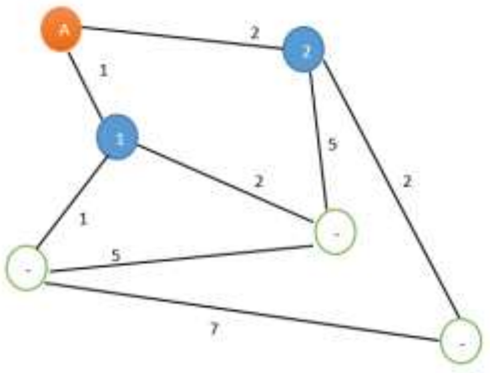
Let's take the figure above as an example. Dijkstra's algorithm calculates the shortest path from a node of any shape to all other nodes.

In our example, let's assume that we have chosen node A as the starting node, and let's show the operation of the algorithm starting from this node.

The algorithm initially assumes that all nodes are not yet accessible and assigns an infinite value. So in the initial state we can't go to any node yet.

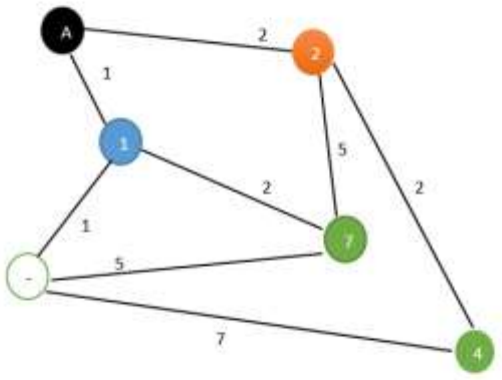


Then it goes around all the neighboring nodes of the starting node and updates the reach distance to these nodes.



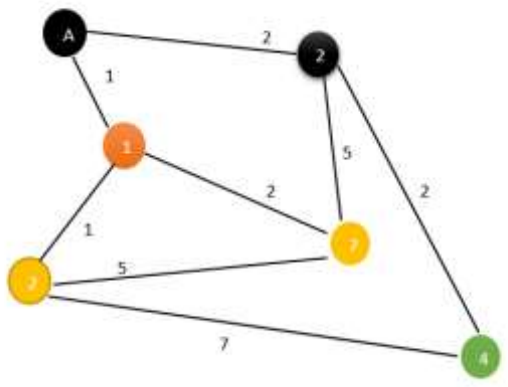
After this update, it updates the neighbors of the updated nodes and this process continues until all the nodes are updated and there is no new update on the figure.

For example, nodes (B and C) which are neighbors of node A above have been updated. In the next step, the neighbors of these nodes will be updated. Let's start with a node we want. For example, let's first update the neighbors of node B and then C:



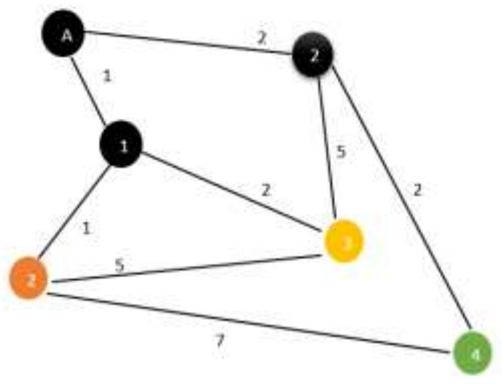
In the figure above, nodes E and F, which are neighbors of node B, have been updated. During this update process, the current cost on node B and the cost of going to nodes E and F are summed.

For example, 2 + 5 = 7 for E node and 2 + 2 = 4 for F node. Now let's update from the next node, node C:



The nodes updated at this stage are nodes D and F, which are neighbors of node C. Of these nodes, the node F was previously updated and had a value of 7, but the new value was calculated as 1 + 2 = 3 over C for F, and 7 was overwritten as 3 since this value was lower than the previously calculated value of 7.

Let's continue the work of the algorithm by updating the next nodes. This time, for example, let's update the neighbors of node D (Currently, nodes D, F and E are equally newly updated, starting with either of them is suitable for the Dijkstra algorithm. However, if a recursive coding is done, of course, the nearest neighbors will be updated according to the code)

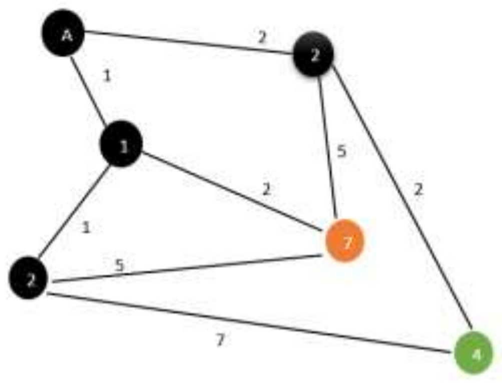


No neighbors of node D have been updated because the calculated values are higher than the current values and therefore no better result can be found.

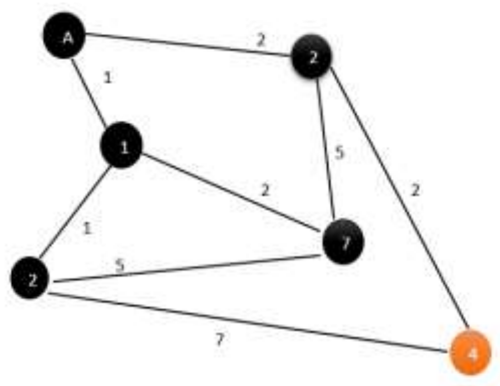
for F 2 + 5 = 7 > 3

For E 2 + 7 = 9 > 4

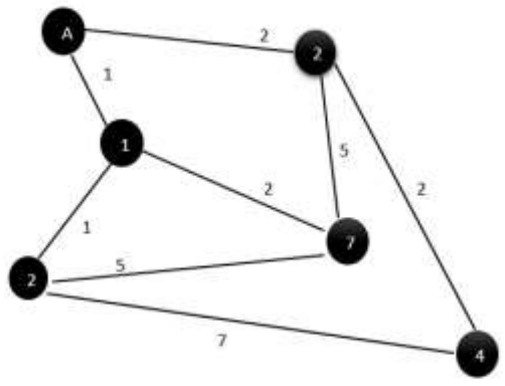
So we continue by choosing the next node



There is no change in the neighbors of node F, and finally node E is tried:



There is no change in our final state, and our shape (graph) is circulated and finished in a stable state (since there is no further change).

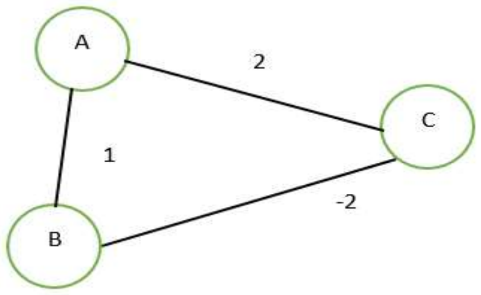


In this graph above, the values written on the nodes give the shortest path distance to each node, starting from node A. For example, the cost of reaching node F in the figure is calculated as 3, and Dijkstra's algorithm claims that there is no shorter path from node A to node F (equally different paths can be found, but the shortest path is still 3)

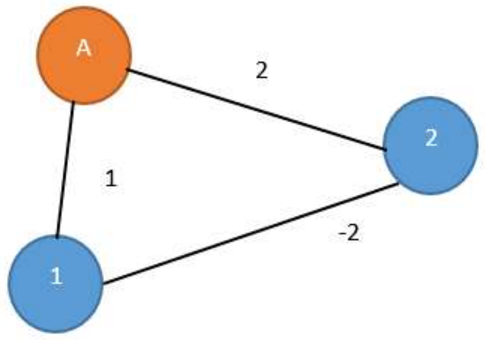
2.2. Weakness of Dijkstra's Algorithm

Unfortunately, the algorithm does not work successfully if there is a negative (-) edge. The reason for this is that the negative (-) value edge constantly produces a better result than the current situation and the algorithm never becomes stable.

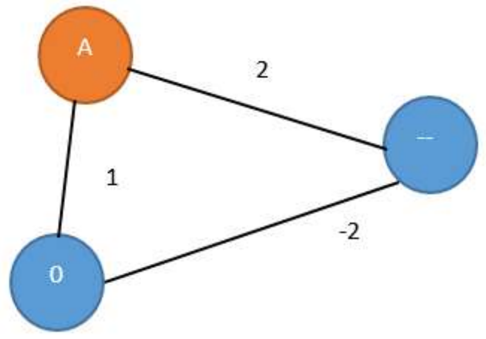
For example, let's take the following simple form:



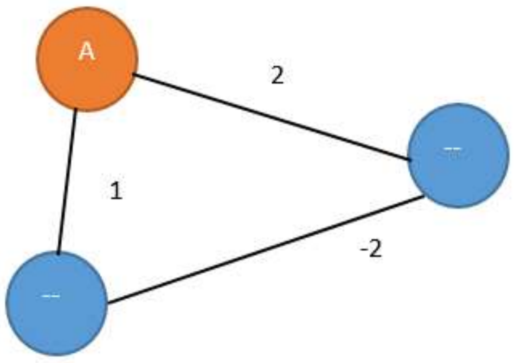
In this figure above, the B-C edge is given as -2. In this case, if we want to calculate the shortest paths starting from node A, we will first update from node A to its neighbor nodes:



Then the neighbors of these nodes will be updated. It will find a shorter path by adding the value -2 to each other at the two nodes, and will get shorter results by updating each other.



This process is the beginning of an endless process, and the process of finding a better result never ends.



A better state will be found continuously than the current state and smaller values will be updated. Therefore, Dijkstra's algorithm does not work well for graphs with negative (-) values.

REFERENCES

[1]. https://www.baeldung.com/cs/minimum-spanning-vs-shortest-path-trees***, linked list |set 1,author: ashwani khemani/geekforgeeks, last updated : 16 Sep, 2020, Accessed :3 June 2021***

[2]. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/ ***, C language, Programmer Sought, last updated : 2018-2021,Accessed: 3 June 2021***

[3]. https://medium.com/algoritma-ve-veri-yap%C4%B1lar%C4%B1/prims-algoritmas%C4%B1-nedir-aa3d05e0ec2e***, About C programming syntax, Last updated : 20 Jul 2021, Accessed : 2 June 2021***