IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 11, November 2016

Algorithm to find Minimum Weight Spanning Tree dynamically

Ishwar Baidari[1], S.P. Sajjan[2]

Associate Professor, Department of Computer Science, Karnataka University, Dharwad, India[1]

Research Scholar, Department of Computer Science, Karnataka University, Dharwad, India[2]

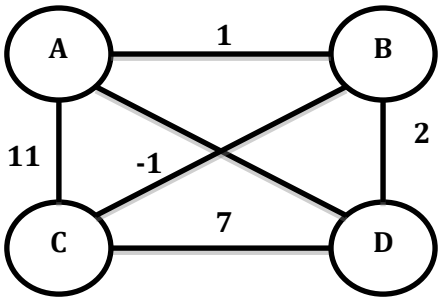
Abstract: A minimum spanning tree is a spanning tree of a connected undirected graph. Each edge is labelled with its weight. It connects all the vertices together with the minimal total weight for its edges. In this paper we designed an algorithm to find minimum weight spanning tree.

Keywords: Spanning tree, minimum spanning tree, minimum weight.

I. INTRODUCTION

Here's a classical task on graphs. We have a group of cities and we must wire them to provide them all with electricity. Out of all possible connections we can make, which one is using minimum amount of wire.
To wire N cities, it's clear that, you need to use at least N-1 wires connecting a pair of cities. The problem is that sometimes you have more than one choice to do it. Even for small number of cities there must be more than one solution as shown on the image bellow.

A. WIRING CITIES



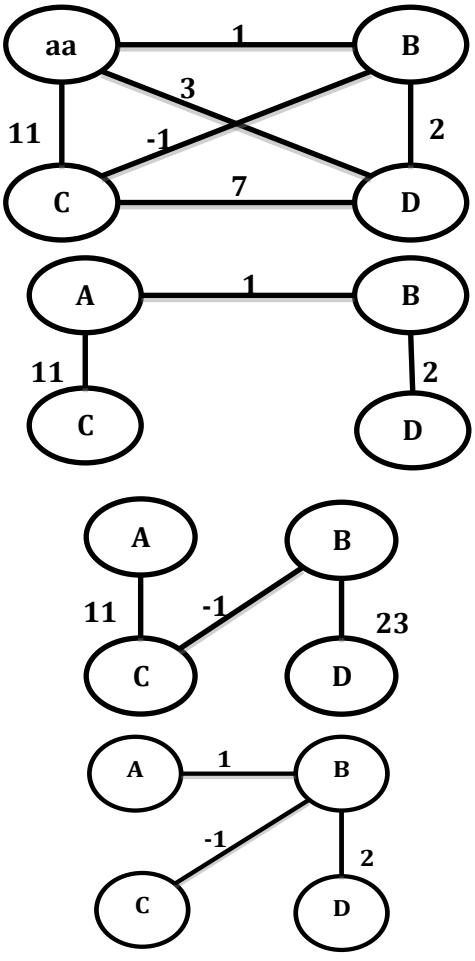
Here we can wire these four nodes in several ways, but the question is, which one is the best one. By the way defining the term "best one" is also tricky. Most often this means which uses least wire, but it can be anything else depending on the circumstances.
As we talk on weighted graphs we can generally speak of a minimum weight solution through all the vertices of the graph. By the way there might be more the one equally optimal (minimal) solutions.

II. OVERVIEW

Obviously we must choose those edges that are enough to connect all the vertices of the graph and whose sum of weights is minimal. Since we can't have cycles in our final solution it must form a tree. Thus we're speaking on a minimum weight spanning tree, as the tree spans over the whole graph.

Does each connected and weighted graph have a minimum spanning tree? The answer is yes! By removing the cycles from the graph G we get a spanning tree, since it's connected. From all possible spanning trees one or more are minimal.

A. SPANNING TREES OF G



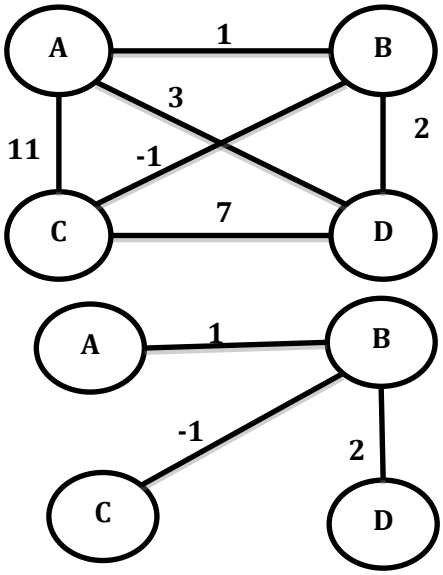
MINIMUM

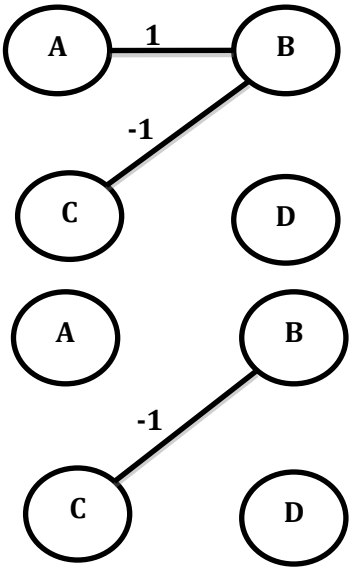
DOI 10.17148/IARJSET.2016.31131

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 11, November 2016

Although there are more than one spanning tree of G, one is the minimum spanning tree (T) If w(u, v) is the weight of the edge (u, v), we can speak of weight of any spanning tree T – w(T) which is the sum of all the edges forming that tree. Thus the weight of the minimum spanning tree is less than the weight of whatever other spanning tree of G.
After we're sure that there is at least one minimum spanning tree for all connected and weighted graphs we only need to find it somehow. We can go with an incremental approach. At the end we'll have the minimum spanning tree (MST), but before that on each step of our algorithm we'll have a sub-set of this final tree, which will grow and grow until it becomes the real MST. This subset of edges we'll keep in one additional set A.

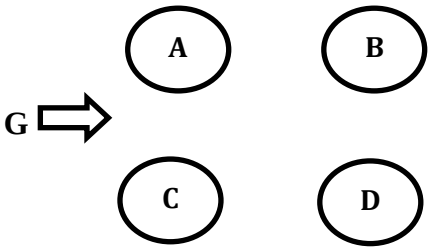
B. GROWING MINIMUM SPANNING TREE





On each step the set of edges forming MST grow with one edge more! So far we know that on each step we have a subset of the final MST, but first we need to answer a couple of questions.

C. How do we start?
Well, we'll start with the empty set of edges. Clearly the empty set is a subset of any other set, thus it will be also a subset of the MST.



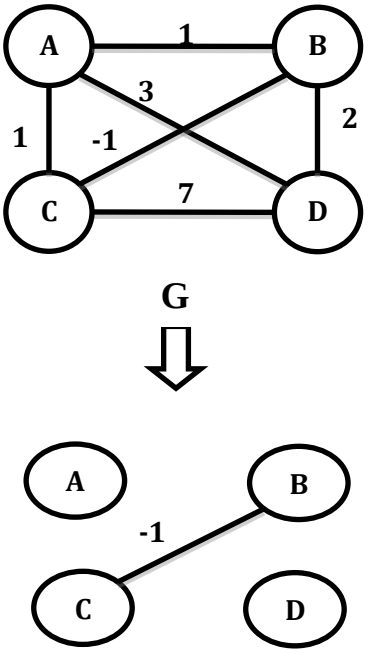
Since the MST is a subset of edges of G we start with the empty set of edges!

D. How do we grow the tree?
Another question we must answer is how to grow the tree. Since we have a MST sub-set (A) on each step how do we add an edge to this set in order to get another (bigger than the previous one) subset of edges, which will be again a subset of the minimum spanning tree?
Clearly we must make a decision which edge to add to the growing subset and this is the tricky part of this algorithm.

III.CHOSE THE LOWEST WEIGHT EDGE!

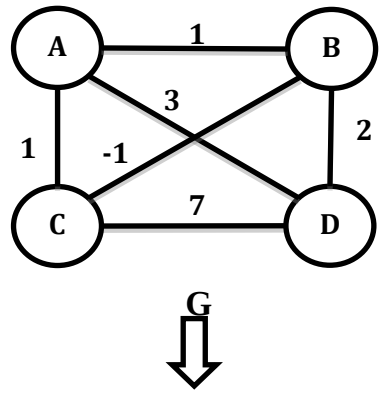
To find the minimum spanning tree on each step we must get the lowest weighted edge that connects our subset (A) with the rest of the vertices.

A. CHOSING AN EDGE



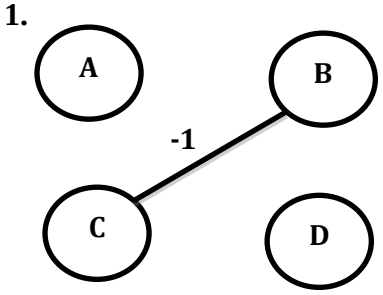
IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 11, November 2016

Each step choosing lowest weight edge
However can we be sure that by choosing the less weighted edge we'll get the MST? Well, let's assume that isn't right in order to prove that wrong!
OK so on some step of our growing sub-tree we don't get the lightest edge (u, v), because we somehow doubt this rule, and we get another edge – let's say (x, y). Mind that w(x, y) >= w (u, v).

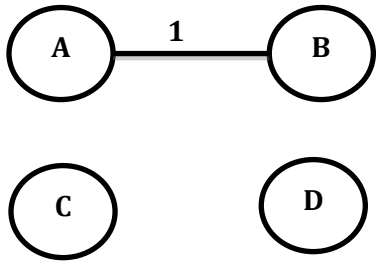


G

1.



2.



W(B, C) < w(A, C)

Thus our final MST will contain somewhere in its set of edges the edge (x, y), but the weight of MST w(T) is minimal, and if we get another spanning tree that contains the exact same edges as T but instead of (x, y) contains (u, v) we'll get a smaller weight!
That isn't possible! Thus we proved that on each step we must get the less weighted edge.
This particular approach is called "greedy", because on each step we get the best possible choice. However greedy algorithms don't always get the right or optimal solution. Fortunately for MST this isn't true so we can be greedy as much as we can!

OK let's make a summary of our algorithm in the following pseudo code.

B. Pseudo Code

1. We start with an empty set (A) subset of the final MST;
2. Until A does not form T:
   a. Get the less weighted edge u from G;
   b. Add u to A;
3. Return A

C. Feature work

In a graph G and Minimum Spanning Tree(MST) T, suppose that we have decrease the weight of one the edge not in T(Increase the edge, which is in T). Written an algorithm for finding the MST in the modified graph.

D. Weight Decreases

Whenever the weight of a non-tree edge (i, j) is decreased, one has to find the maximum weight edge (x, y) along the path from i to j in T and to remove it if w(x, y) > w(i, j). This can be accomplished by using any dynamic tree data structure to store the MST.
The procedure to update the minimum spanning tree is described in Algorithm 2. Vertex i is made the new root of the tree. The maximum weight edge (x, y) in the path from j to i is computed in line. If the weight of edge (x, y) is larger than that of edge (i, j) (comparison in line 5), then the former is removed from the tree by the Cut(x, y) operation in line 6 and the new edge (i, j) is inserted by the Link (i, j, w(i, j)) operation in line 7. The updated MST is returned in line 9. The efficiency of the above computations depends on the underlying structure used to maintain the MST and to implement the path operations. If RD-trees or DRD-trees are used, then Algorithm 2 runs in time O(|V|). It runs in time O(log |V|) if a more complex implementation (such as ST-trees) is used.

E. Pseudo Code (Decrease weight of non-tree edge)

1. We start with an existing MST;
2. Decrease the weight non tree edge
I. If Decrease non tree edge weight
II. Repeat until A does not form T:
a. Get the less weighted edge u from G;
b. Add u to A;
3. Return A

F. Weight Increasing

Whenever the weight of a graph edge (i, j) is increased, one has to find the minimum weight edge (x, y) along the path from i to j in G and to remove it if w(x, y) < w(i, j). This can be accomplished by using any dynamic tree data structure to store the MST.

The procedure to update the minimum spanning tree is described in Algorithm 2. Vertex i is made the new root of the tree. The minimum weight edge (x, y) in the path from j to i is computed in line. If the weight of edge (x, y) is larger than that of edge (i, j) (comparison in line 5), then the former is removed from the tree by the Cut(x, y) operation in line 6 and the new edge (i, j) is inserted by the Link (i, j, w(i, j)) operation in line 7. The updated MST is

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 11, November 2016

returned in line 9. The efficiency of the above computations depends on the underlying structure used to maintain the MST and to implement the path operations. If RD-trees or DRD-trees are used, then Algorithm 2 runs in time O(|V|). It runs in time O(log |V|) if a more complex implementation (such as ST-trees) is used.

G. Pseudo Code (Increasing weight of tree edge)
1. We start with an existing MST;
2. Decrease the weight non tree edge
I. If Decrease non tree edge weight
II. Repeat until A does not form T:
a. Get the less weighted edge u from G;
b. Add u to A;
3. Return A

IV. COMPLETE ALGORITHM: MINIMUM WEIGHT SPANNING TREE DYNAMICALLY

INPUT: GRAPH G = (V, E), WEIGHTS W.
1: BUILD A LIST A WITH ALL (I, J) ∈ E;
2: SORT LIST A BY NON-DECREASING ORDER OF WEIGHTS;
3: USE LIST A TO COMPUTE THE MST(BY USING KRUSKAL OR PRIMS) T = (V, E′);
4: ENTER NEW EDGE VALUE(EITHER INCREASE/DECREASE)
5: INPUT NEW WEIGHT VALUE M AND N
6: FOR I=0 TO N DO //UPDATE MATRIX
7: FOR J=0 TO M DO
8: IF(I==P-1 && J==Q-1)
9: INPUT M VALUE
10: COST[I][J]=M;
11: COST[J][I]=M;
12: BREAK;
13: END IF
14: END FOR
15: END FOR
16: LET (I, J) ∈ E BE THE EDGE WHOSE WEIGHT WILL CHANGE TO W_{NEW} ;
17: S ← A(I, J);
18: SET THE NEW EDGE WEIGHT: W(I, J) ← w_{new} ;
19: RECOMPUTED THE MST(BY USING KRUSKAL OR PRIMS) T = (V, E′);
20: END

V. COMPLEXITY

A graph (positive weight edges) with a MST If some edge, e is modified to a new value, what is the best way to update the MST without completely remaking it. I think this can be done in linear time. Also, it seems that I would need a different algorithm based on whether 1) e is already a part of the MST and 2) whether the new edge, e is larger or smaller than the original.

A. There are 4 cases:
Edge is in MST and you decreasing value of edge: Current MST is still MST

Edge is not in MST and you decreasing value of edge: Add this edge to the MST. Now you've got exactly 1 cycle. Based on cycle property in MST you need to find and remove edge with highest value that is on that cycle. You can do it using BFS or DFS. Complexity O(n).

Edge is in MST and you increasing its value of edge: Remove this edge from MST. Now you have 2 connected components that should be connected. You can calculate both components in O(n) (BFS or DFS). You need to find edge with smallest value that connects these components. Iterate over edges in ascending order by their value. Complexity O(n).

Edge is not in MST and you increasing its value of edge: Current MST is still MST

B. Application
Actually this algorithm is used firstly by Borůvka which started to wire Moravia in 1926. Even without knowing that the "greedy" approach will lead him to the right solution he optimally covered Moravia with electricity.
However this algorithm is too general and there are two main algorithms – the Prim's algorithm and the Kruskal's algorithm that we shall see in future posts.
The thing is that on each step we must get the less weighted edge and both algorithms use different approaches to do that.

REFERENCES

[1] Ellis Horowitz and Sartaj Sahni: Fundamental of Computer Algorithms (1993) Galgotia Publications.
[2] Narsingh Deo, "Graph Theory with Applications to Engineering and Computer Science", PHI Learning, 204
[3] Mr. Hassan, "An efficient method to solve least cost minimum spanning tree", Computer and Information Sciences (2012) 24. 101-105.
[4] Jothi, Raja, Raghavachari, Balaji. Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design. ACM Transaction on Algorithms (2) 265-282, 2005.
[5] Zhou, Gengai. Cao, Zhenya. Cao, Jian, Meng, Zhiqing, A genetic algorithm approach on capacitated minimum spanning tree problem". International conference on computational intelligence and security, 215-218, 2006.