Kruskal's Algorithm

An Algorithm to Find a Minimum Spanning Tree In an Undirected Graph

Sungjae Cho

Published in Dec. 11, 2016

In this essay, I inspect Kruskal's algorithm, which finds a minimum spanning tree in a connected graph. Before we get into the inspection, we have to know some terms required to start our journey.

Terms

Definition

A ***graph*** is a collection of points and lines connecting some subset of them. In general, a point is called a *vertex* and a line an *edge*.

– Matheworld, http://mathworld.wolfram.com/Graph.html

Definition

A ***tree*** is a simple, undirected, connected, acyclic graph.

Definition

A ***spanning tree*** T of an undirected graph G a subgraph that is a tree which includes all of the vertices of G.

– Wikipedia, https://en.wikipedia.org/wiki/Spanning_tree

A ***spanning tree*** of a graph on n vertices is a subset of $n-1$ edges that form a tree (Skiena 1990, p. 227).

– Mathworld, http://mathworld.wolfram.com/SpanningTree.html

(Intuition: Connecting n vertices with $n-1$ edges means the least number of edges.)

Definition

In a weighted graph, a ***minimum spanning tree*** is a spanning tree that has the minimum weight sum of of its edges.

(Comment: In a connected graph, a minimum spanning can exist only one or more than one.)

Definition

A ***greedy algorithm*** is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

– Wikipedia, https://en.wikipedia.org/wiki/Greedy_algorithm

Minimum Spanning Tree

When We Use Minimum Spanning Trees

Minimum spanning trees are used in civil network planning, computer network routing protocol, and cluster analysis.

Imagine the case that we are requested to build a railroad system in order to connect all the cities in a country. Moreover, our primary goal is to minimize construction expenditure and connect all cities so that citizens can commute between two cities just by using train transportation. Then, first, we investigate the cost to build a railroad between each city. An undirected, connected graph can be made and weighted with respect to the costs. Second, find a spanning tree in the graph that minimizes the weight sum of all connecting edges. The spanning tree is called a minimum spanning tree.

We can see when a minimum spanning tree can be used through the previous case.

How Important to Find a Spanning Tree In a Graph

Finding a spanning tree is a process to transform a graph problem to a tree problem. A tree has recursive structure. That is, a tree can be divided into several subtrees. For the recursive structure, it is easy to design algorithms to solve the problems of a tree.

In the study of algorithms, there is an algorithm design paradigm based on multi-branched recursion, which is called ***divided and conquer***. The existence of the paradigm implies that a great number of algorithms approach problems in a recursive way.

On the other hand, it is much harder to design algorithms for general graphs. Therefore, transforming a graph problem to a tree, namely, finding a spanning tree in a graph is quite important.

Let us see how Kruskal's algorithm really operates.

Kruskal's algorithm

Kruskal’s algorithm operated for an undirected, connected graph. The following procedure is how Kruskal’s algorithm behaves.

1. Arrange all edges in their increasing order of weight.
2. Add the edge which has the least weight if the edge does not make a cycle.
3. Exclude the considered edge from our consideration.
4. If there are remaining edges, execute step 2 again for the remaining edges.
   If there are no remaining edges, then, return the selected edges and end the algorithm.

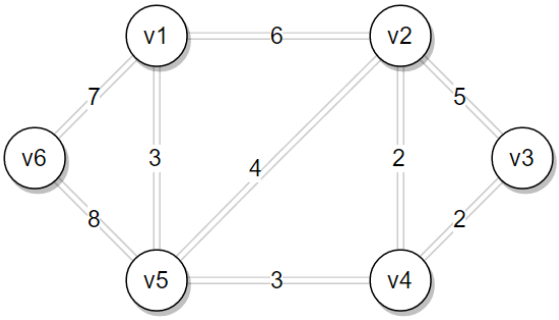
Greedy Algorithms

Since Kruskal’s algorithm is a greedy algorithm, the algorithm is unable to guarantee the optimal time to find a minimum spanning tree. It does not mean the algorithm produces a spanning tree that is not a minimum spanning tree.

Let us see how Kruskal’s algorithm is operated by observing the following example.

Example

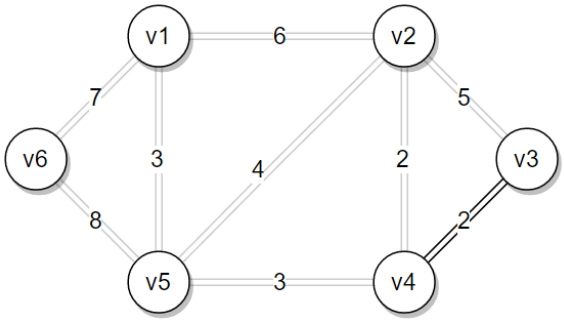
An undirected, connected graph is given.



[Step 1] Arrange all edges in their increasing order of weight.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v3, v4} is an edge that has the minimum weight among all edges. The edge does not make a cycle. Select the edge.

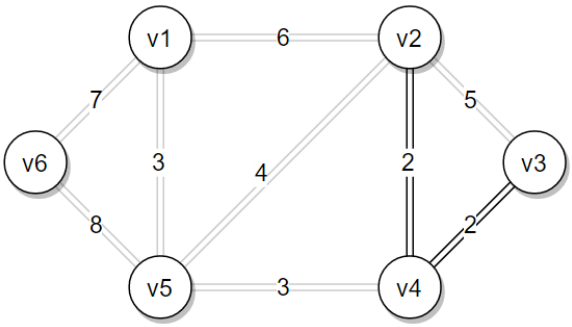


[Step 3] Exclude the considered edge from our consideration.

[Step 4] There are remaining edges. Then, execute step 2 again for the remaining edges.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v2, v4} is an edge that has the minimum weight among all remaining edges. The edge does not make a cycle. Select the edge.

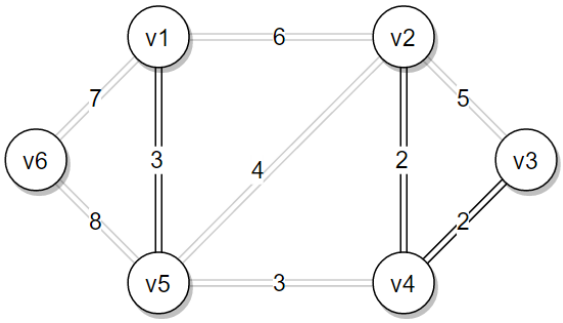


[Step 3] Exclude the considered edge from our consideration.

[Step 4] There are remaining edges. Then, execute step 2 again for the remaining edges.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v1, v5} is an edge that has the minimum weight among all remaining edges. The edge does not make a cycle. Select the edge.

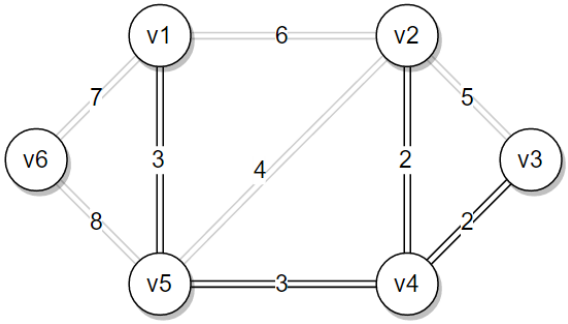


[Step 3] Exclude the considered edge from our consideration.

[Step 4] There are remaining edges. Then, execute step 2 again for the remaining edges.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v4, v5} is an edge that has the minimum weight among all remaining edges. The edge does not make a cycle. Select the edge.

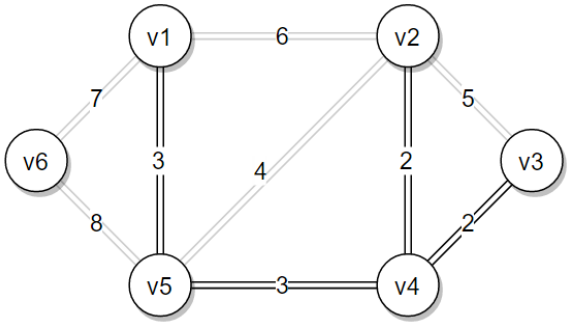


[Step 3] Exclude the considered edge from our consideration.

[Step 4] There are remaining edges. Then, execute step 2 again for the remaining edges.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v2, v5} is an edge that has the minimum weight among all remaining edges. The edge makes a cycle. The edge cannot be an edge in a tree.

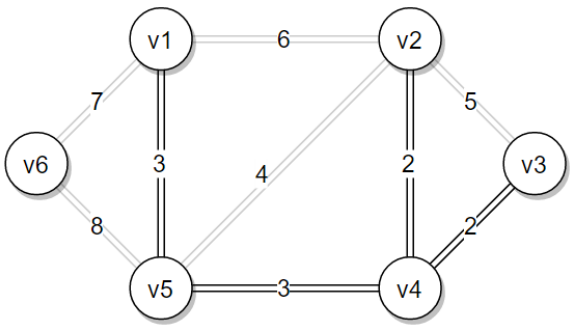


[Step 3] Exclude the considered edge from our consideration.

[Step 4] There are remaining edges. Then, execute step 2 again for the remaining edges.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v2, v3} is an edge that has the minimum weight among all remaining edges. The edge makes a cycle. The edge cannot be an edge in a tree.

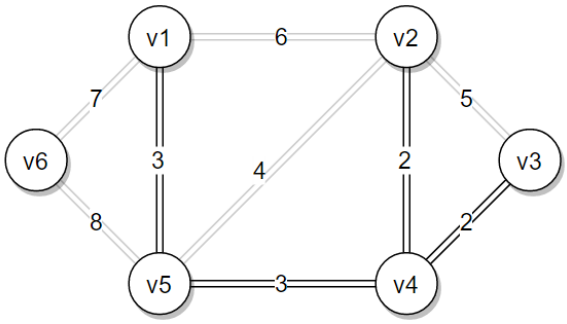


[Step 3] Exclude the considered edge from our consideration.

[Step 4] There are remaining edges. Then, execute step 2 again for the remaining edges.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v1, v2} is an edge that has the minimum weight among all remaining edges. The edge makes a cycle. The edge cannot be an edge in a tree.

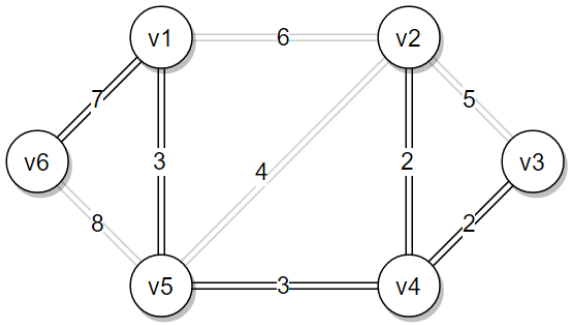


[Step 3] Exclude the considered edge from our consideration.

[Step 4] There are remaining edges. Then, execute step 2 again for the remaining edges.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v1, v6} is an edge that has the minimum weight among all remaining edges. The edge does not make a cycle. Select the edge.

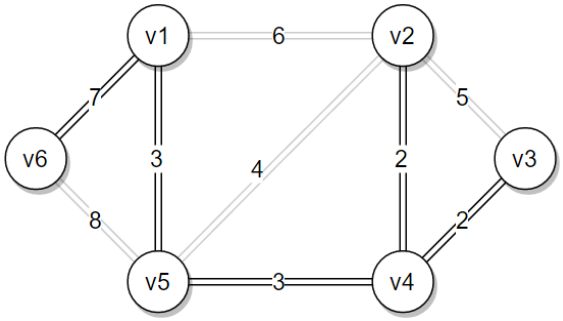


[Step 3] Exclude the considered edge from our consideration.

[Step 4] There are remaining edges. Then, execute step 2 again for the remaining edges.

v3, v4	v2, v4	v1, v5	v4, v5	v2, v5	v2, v3	v1, v2	v1, v6	v5, v6
2	2	3	3	4	5	6	7	8

[Step 2] The edge {v5, v6} is an edge that has the minimum weight among all remaining edges. The edge makes a cycle. The edge cannot be an edge in a tree.



[Step 3] Exclude the considered edge from our consideration.

[Step 4] There is no remaining edge. Then, execute step 2 again for the remaining edges. Then, return the selected edges and end the algorithm.

Finally, we made a minmum spanning tree. Let us check whether the result is a minimum spanning tree.

First, because the result is connected and has no cycle, it is a tree. Also, there are 6 vertices and 5 edges, which is the mimimum number of edges we can choose in a graph of 6 vertices. Thus, the tree is a spanning tree of the graph.

Second, we have to prove the result tree has really the minimal number of edges. We need a mathematical proof.

Proof of Correctedness

The proof that Kruskal's algorithm generates a minimum spanning tree requires two proofs.

1. Prove that the algorithm produces a spanning tree.
2. Prove the constructed spanning tree is of minimal weight.

Spanning Tree

- Let G be an undirected, connected graph.
- Following the algorithm, every edge is checked whether the edge is a cycle.
- If the edge makes a cycle, the two vertices of the edge are already connected.
- If the edge does not make a cycle, the vertices of the edge are not connected yet.
- After getting through all process of the algorithm, all vertices become connected. Besides, any cycle cannot be added in a new graph produced by the algorithm. Hence, the new graph is a tree. Consequently, since the new graph is a connected graph in G, the tree is a minimum spanning tree in G.

Minimality

We show that the following proposition P is true by induction. If F is the set of edges chosen at any stage of the algorithm, then there is some minimum spanning tree that contains F.

- Clearly P is true at the beginning, when F is empty: any minimum spanning tree will do, and there exists one because a weighted connected graph always has a minimum spanning tree.
- Now assume P is true for some non-final edge set F and let T be a minimum spanning tree that contains F. If the next chosen edge e is also in T, then P is true for $F + e$. Otherwise, $T + e$ has a cycle C and there is another edge f that is in C but not F. (If there were no such edge f, then e could not have been added to F, since doing so would have created the cycle C.) Then $T - f + e$ is a tree, and it has the same weight as T, since T has minimum weight and the weight of f cannot be less than the weight of e, otherwise the algorithm would have chosen f instead of e. So $T - f + e$ is a minimum spanning tree containing $F + e$ and again P holds.
- Therefore, by the principle of induction, *P* holds when F has become a spanning tree, which is only possible if F is a minimum spanning tree itself.

The proof of minimality is quoted from the article of Kruscal's algorithm in English Wikipedia, https://en.wikipedia.org/wiki/Kruskal's_algorithm#Proof_of_correctness.

References

1. https://en.wikipedia.org/wiki/Kruskal's_algorithm
2. https://www.tutorialspoint.com/data_structures_algorithms/kruskals_spanning_tree_algorithm.htm
3. http://mathworld.wolfram.com/SpanningTree.html
4. https://www.tutorialspoint.com/data_structures_algorithms/spanning_tree.htm
5. https://en.wikipedia.org/wiki/Minimum_spanning_tree
6. https://en.wikipedia.org/wiki/Divide_and_conquer_algorithms
7. https://en.wikipedia.org/wiki/Greedy_algorithm

Further Reading

1. Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". *Proceedings of the American Mathematical Society*. **7**: 48–50
    - This is the first writing Kruskal's algorithm first appeared.