Prims Minimum Spanning Tree Algorithm (Step by Step with examples)

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-92

In the last lecture, we learnt about the cost of a spanning tree, and we discussed what a minimum spanning tree is along with some of its applications. Today, we will see Prim's algorithm which is one of the algorithms to find the minimum spanning tree of a graph.

If you remember, we took the example of *Prem* and solved his dilemma of choosing the best possible route to traverse every city in search of his beloved while spending the least possible money. It is here that his big brother, *Prim*, steps in to help. His brother, *Prim,* has an algorithm named after him, the Prim's algorithm.

Rather than diving into the details of the algorithm right now, let's take a moment to revisit the concepts we've learned so far.

Connected graphs are graphs where every node in the graph has a path to every other node. A **complete graph** is one in which every pair of nodes has a unique edge. You should know those four conditions for a tree to be a **spanning tree** of some other graph. The **Cost of a spanning tree** is the sum of the weights of all the edges in the tree. And **a minimum spanning tree** has the minimum cost over all spanning trees of a graph.

Quick Quiz: Why is a spanning tree called '*spanning*' '*tree*'?

Hint: It spans over the whole graph, and it is a tree.

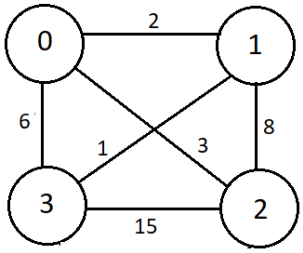
This gives us the green light to move forward with Prim's algorithm.

Prim's Algorithm:

- Prim's algorithm uses the Greedy approach to find the minimum spanning tree.
- There are mainly two steps that this algorithm follows to find the minimum spanning tree of a graph:

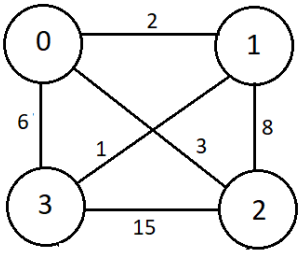
1. We start with any node and start creating the Minimum Spanning Tree.
2. In Prim's Algorithm, we grow the spanning tree from a starting position until n-1 edges are or all nodes are covered.

Let me now take a simple example. Consider the complete graph with 4 vertices, K4, illustrated below:

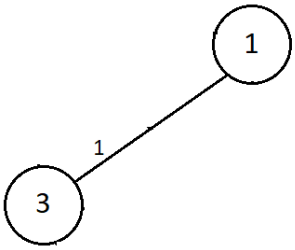


K_4

Now, as the first step says, we can start without any node in the graph. Arbitrarily, we will choose node 1. And since the edge between nodes 1 and 3 is the least weighted, we'll consider this edge in our minimum spanning tree.

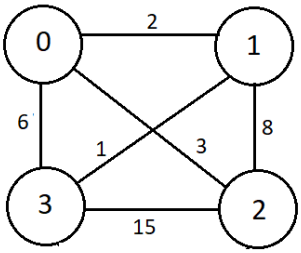


K_4

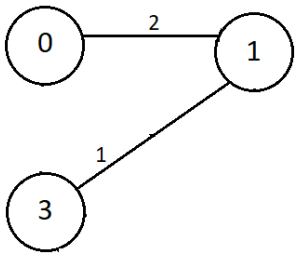


Spanning Tree

Next, we'll try involving node 0 in our spanning tree. Possible candidates for connecting node 0 from either node 3 or node 1 which compose the current spanning tree are edges having weights 6 and 2. Obviously, we'll choose the one with weight 2.

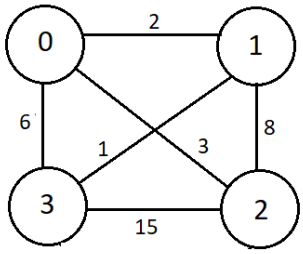


K_4

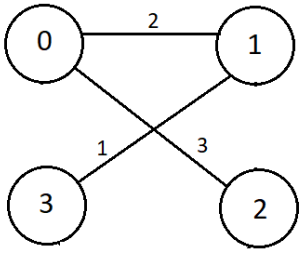


Spanning Tree

Now, the only node left is node 2. Possible candidates for connecting this node to the current spanning tree are edges with weights 15, 3, and 8. And there shouldn't be any doubt while considering the edge with weight 3.



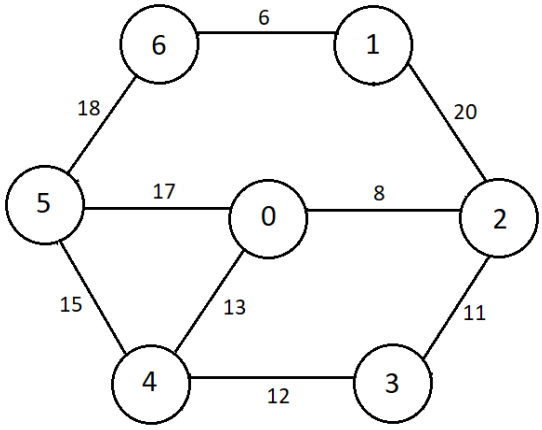
K_4



Spanning Tree

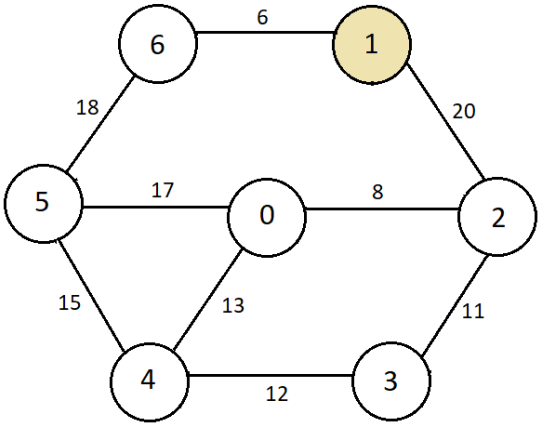
The cost of this spanning tree is (1+2+3), i.e., 6 which is the minimum possible. And that finishes our construction of the spanning tree for graph K4. And this is exactly the way Prim's algorithm works.

Now, I'll give you one relatively tough example to get a better feel of the algorithm. Consider the graph I've illustrated below.



G

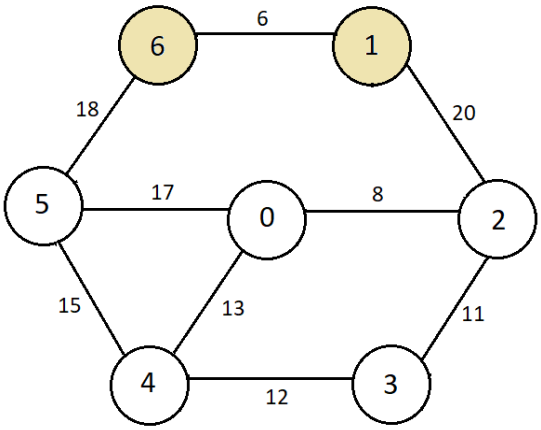
The first step, as the algorithm says, is to choose any arbitrary node and start creating the spanning tree. We have chosen node 1 for the same.

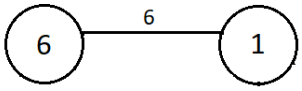


G

Spanning Tree

Colored nodes indicate that they have been considered for the spanning tree while the rest of the nodes are still left to cover. Here, in the Prim's algorithm, we maintain two sets of nodes, one for the nodes counted in the spanning tree, A, and another for the rest of them, V. Nodes connected to node 1 are 6 and 2, connected through edges of weights 6 and 20 respectively. Therefore, we will choose the one with a weight of 6.

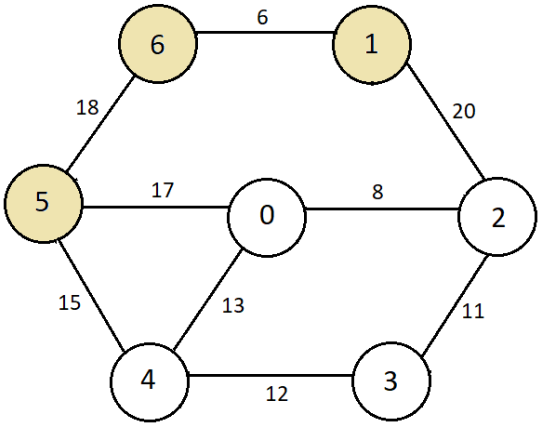


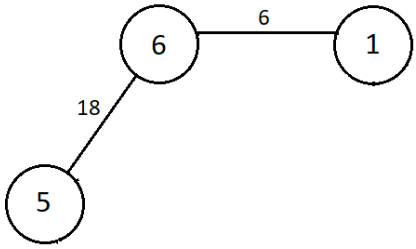


G

Spanning Tree

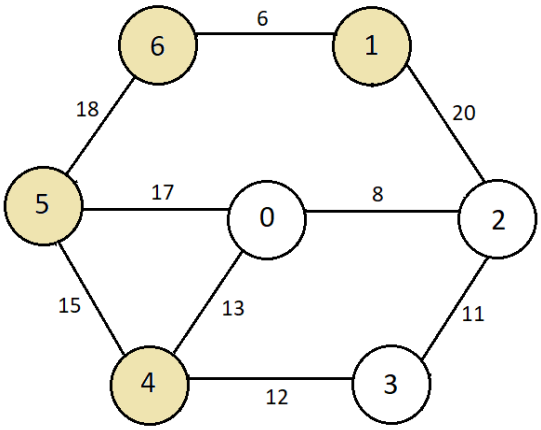
Now, 6 gets added to the set containing the covered elements. Nodes possible to get connected to the current spanning tree are 5 and 2. Node 5 and node 2 are connected to node 6 and node 1 respectively through edges of weights 18 and 20. We will obviously choose the one with less weight, i.e., weighted edge 18.

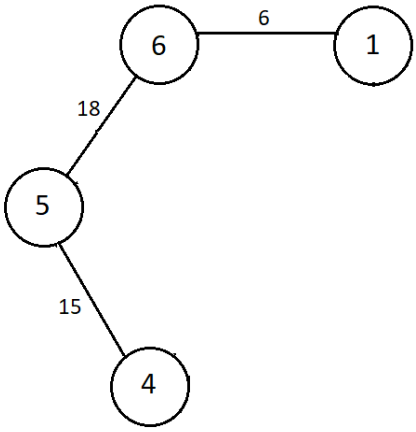




G Spanning Tree

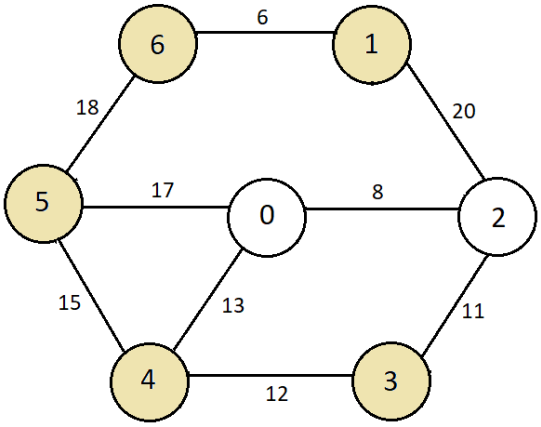
Now, 5 gets added to the set containing the covered elements. Possible nodes to get connected to the current spanning tree are 4, 0, and 2. Node 4 and node 0 are connected to node 5 through edges of weights 15 and 17 and node 2 is connected to node 1 through an edge of weight 20. We will, therefore, choose the one with weight min (15,17,20) i.e., 15. Hence, node 4 gets added to the current spanning tree.



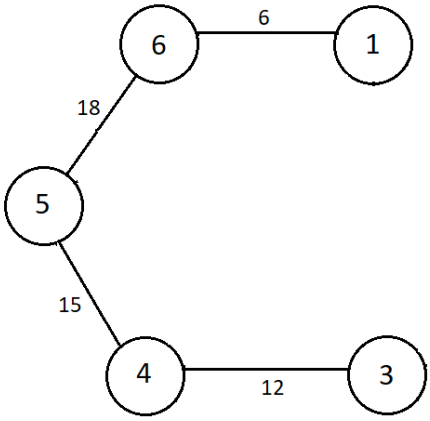


G Spanning Tree

Now, we have nodes 0, 2, and 3 as possible candidates to get connected to the current spanning tree. Node 0 is connected to node 5 and node 4 through edges of weights 17 and 13 respectively. Node 3 is connected to node 4 through an edge of weight 12, and node 2 is connected to node 1 through an edge of weight 20. Therefore, we'll choose the one with weight min (17,13, 12, 20) i.e., 12. Hence, node 3 gets added to the current spanning tree.

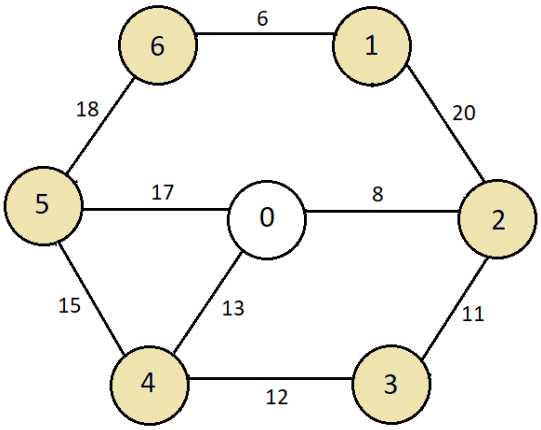


G

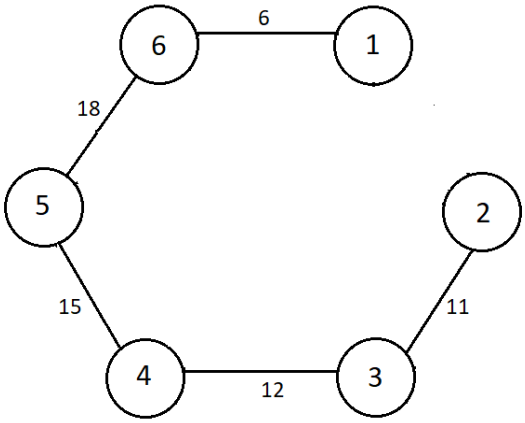


Spanning Tree

Next, we are left with only two nodes to cover. Node 0 is connected to node 5 and node 4 through edges of weights 17 and 13 respectively. Similarly, Node 2 is connected to node 1 and node 3 through edges of weights 20 and 11 respectively. Therefore, we'll choose the one with weight min (17,13, 20, 11) i.e., 11. Hence, node 2 gets added to the current spanning tree through node 3.

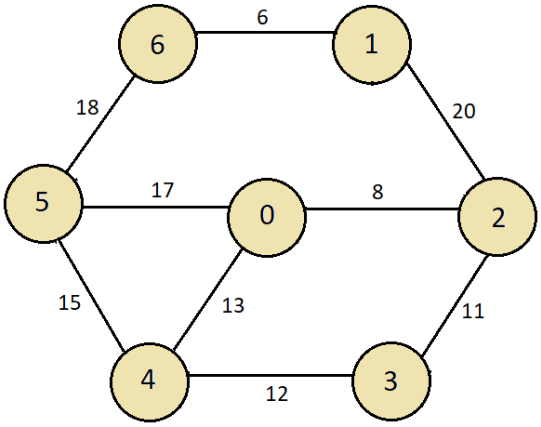


G

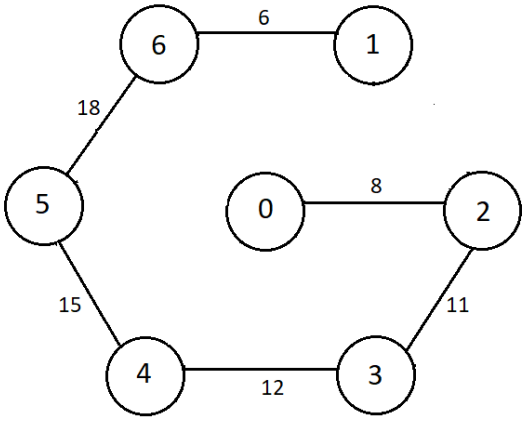


Spanning Tree

The only node left now is node 0. We have 3 possible ways to connect this node to the rest of the spanning tree. There are edges of weights 17, 13, and 8 though nodes 5, 4, and 2 respectively. Therefore, we'll choose the edge with a weight of 8.



G



Spanning Tree

This completes our spanning tree with 7 nodes and 6 edges. The cost of this spanning tree sums to (6+18+15+12+11+8), which is equal to 70. And you could verify if this is actually the minimum of all possible spanning trees. This is how Prim's algorithm works. It maintains the sets of elements covered and uncovered and sees the least possible edge to connect any one of the covered nodes to the uncovered ones. And the algorithm stops functioning once the set with uncovered nodes gets emptied.

And this is where we wrap up for today. I hope you could understand Prim's algorithm, and ultimately, we could solve Prem's dilemma as well. Another algorithm that is often used in finding the spanning tree of a graph is the Kruskal's. We'll see it too in the coming lectures. Stay tuned.

Thank you for being with me throughout the session. If you appreciate my work, please let your friends know about this channel too. You haven't saved this graph playlist already, you just have to move on to codewithharry.com or my YouTube channel to access them. See you all there in the next lecture. Till then keep learning.