Minimum Spanning Tree

Prim's algorithm. Kruska's algorithm.
Application. Implementation.

Team: Popcorn
Team members:
Makhmutova Aruzhan
Jumabekova Balzhan
Kurmanalin Alibek
Kuanov Gizat
Zhunissova Damira

What is a spanning tree?

A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.
How many edges does a minimum spanning tree has?
A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

Application

Minimum Spanning Tree (MST) problem: Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices.
MST is fundamental problem with diverse applications.

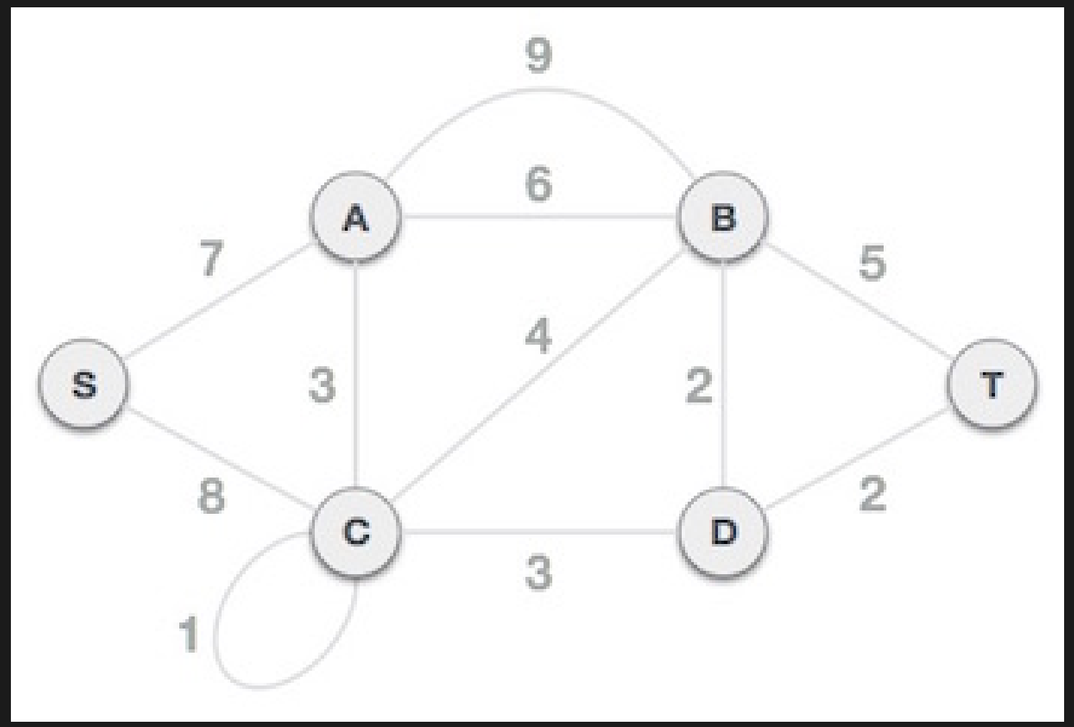
Network design.
– telephone, electrical, hydraulic, TV cable, computer, road

Approximation algorithms for NP-hard problems.
– traveling salesperson problem, Steiner tree
A less obvious application is that the minimum spanning tree can be used to approximately solve the traveling salesman problem. A convenient formal way of defining this problem is to find the shortest path that visits each point at least once.

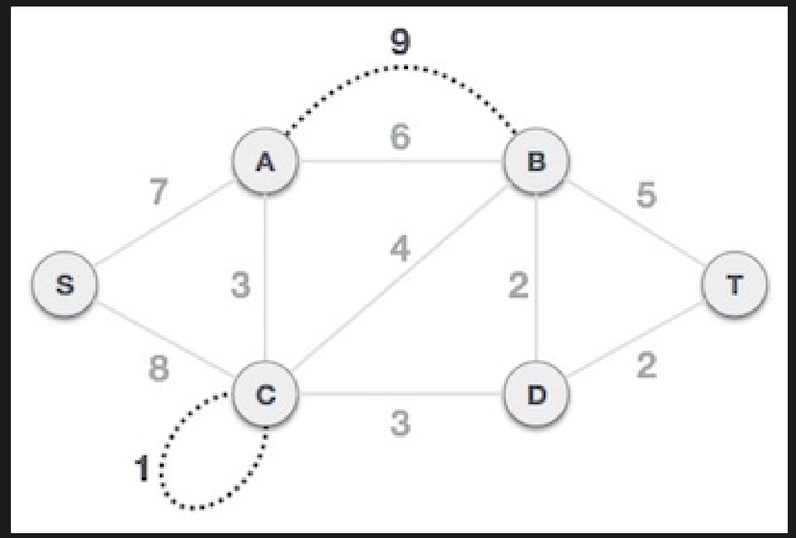
Indirect applications.
– max bottleneck paths
– LDPC codes for error correction
– image registration with Renyi entropy
– learning salient features for real-time face verification
– reducing data storage in sequencing amino acids in a protein
– model locality of particle interactions in turbulent fluid flows
– autoconfig protocol for Ethernet bridging to avoid cycles in a network

Kruskal's algorithm

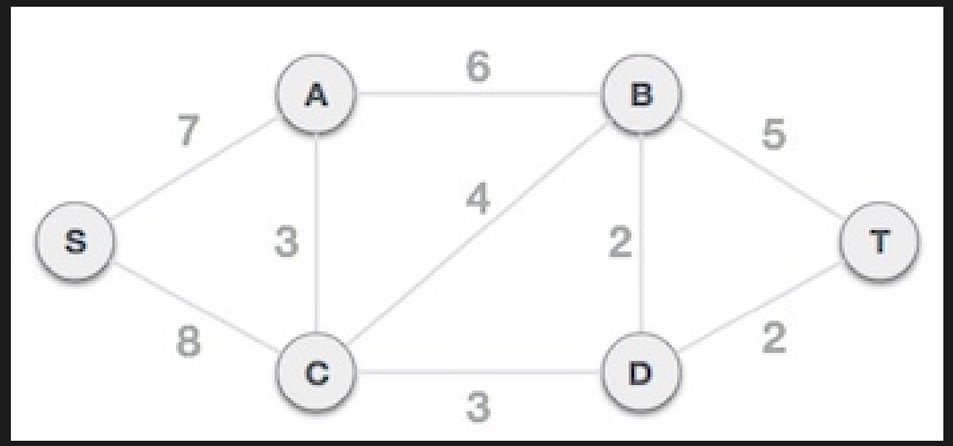
An algorithm to construct a Minimum Spanning Tree for a connected weighted graph. It is a Greedy Algorithm. The Greedy Choice is to put the smallest weight edge that does not because a cycle in the MST constructed so far.
Let's consider this example to understand this algorithm better



Remove all loops and parallel edges from the given graph.



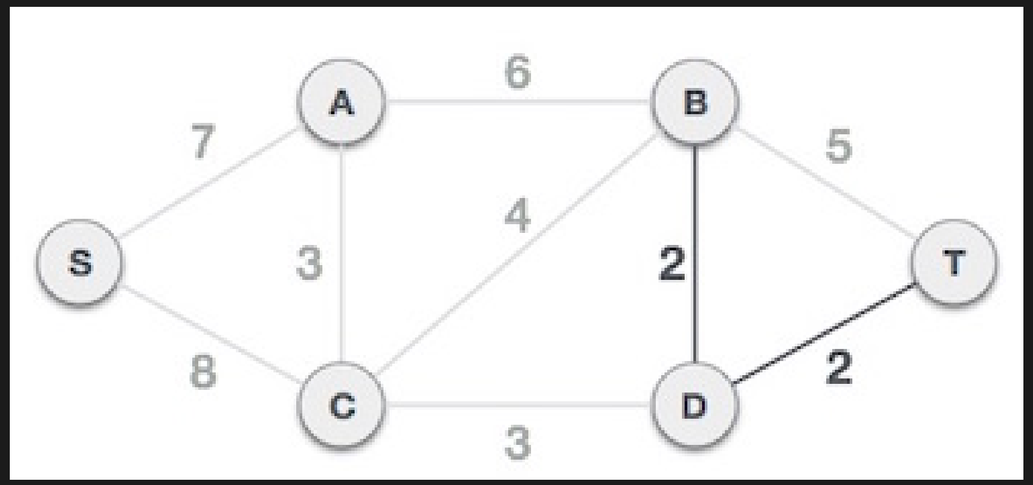
In case of parallel edges, keep the one which has the least cost associated and remove all others.



The next step is to create a set of edges and weight, and arrange them in an ascending order of weightage (cost).

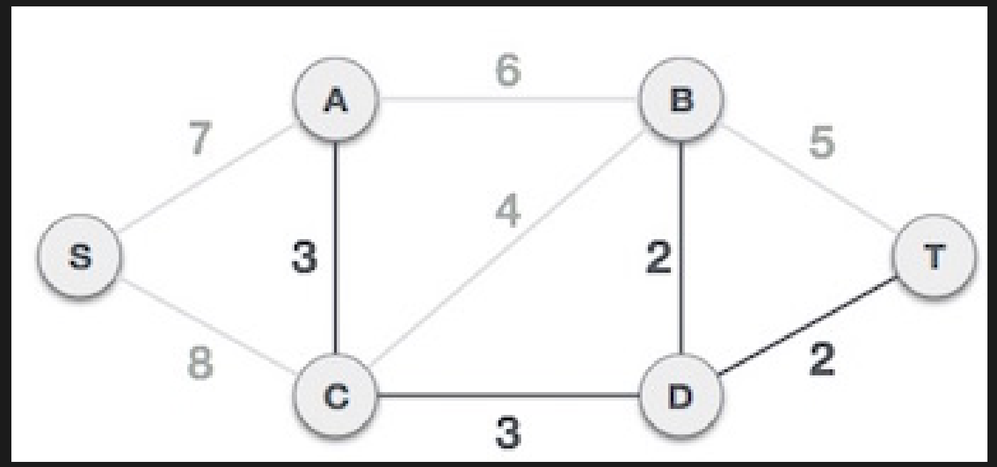
B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

Now we start adding edges to the graph beginning from the one which has the least weight. Throughout, we shall keep checking that the spanning properties remain intact. In case, by adding one edge, the spanning tree property does not hold then we shall consider not to include the edge in the graph.

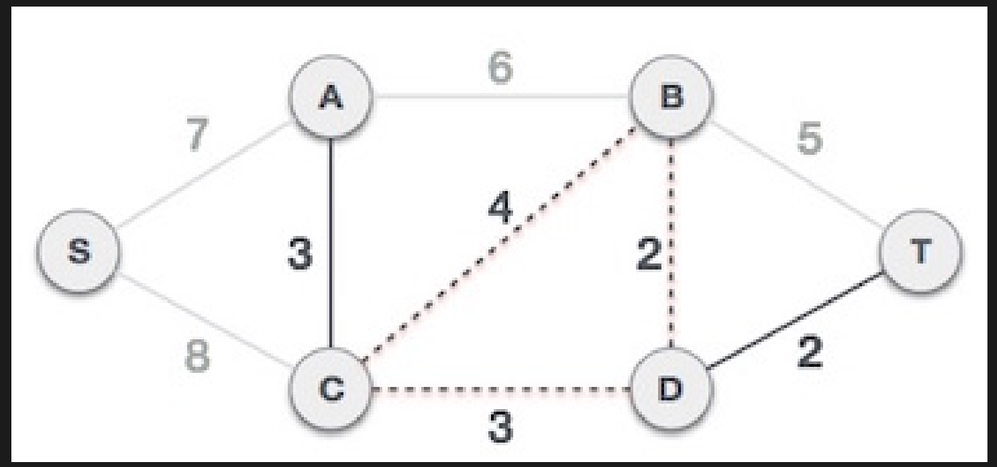


The least cost is 2 and edges involved are B,D and D,T. We add them. Adding them does not violate spanning tree properties, so we continue to our next edge selection.

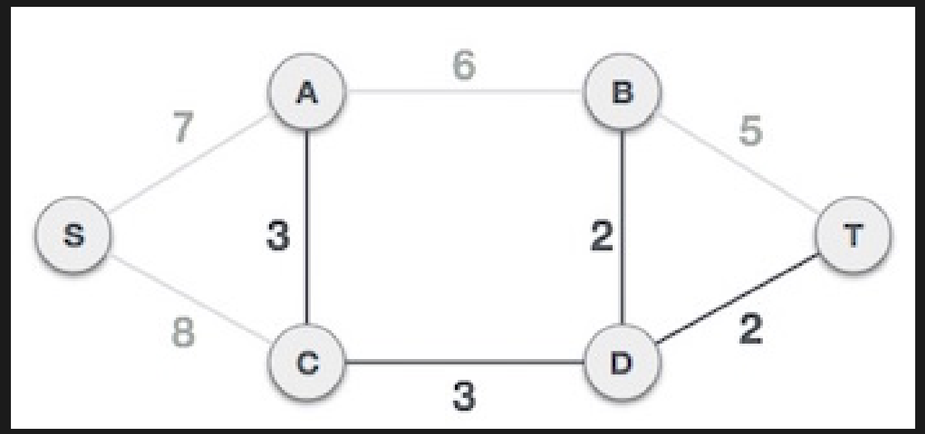
Next cost is 3, and associated edges are A,C and C,D. We add them again –



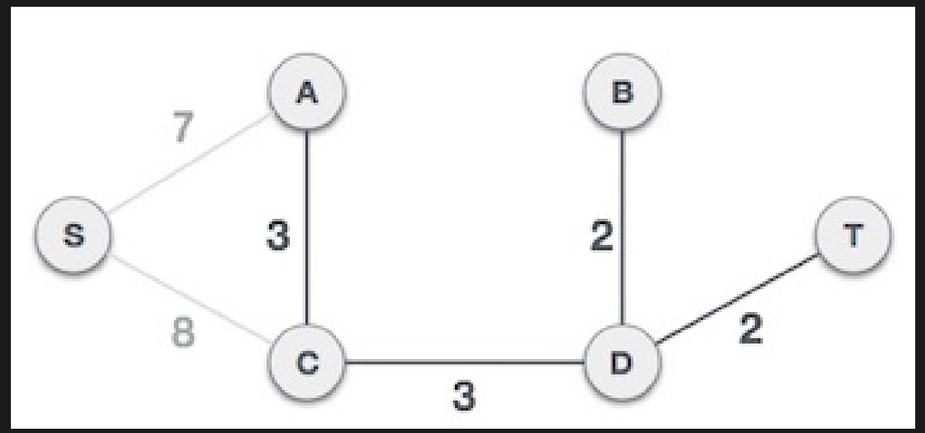
Next cost in the table is 4, and we observe that adding it will create a circuit in the graph. –



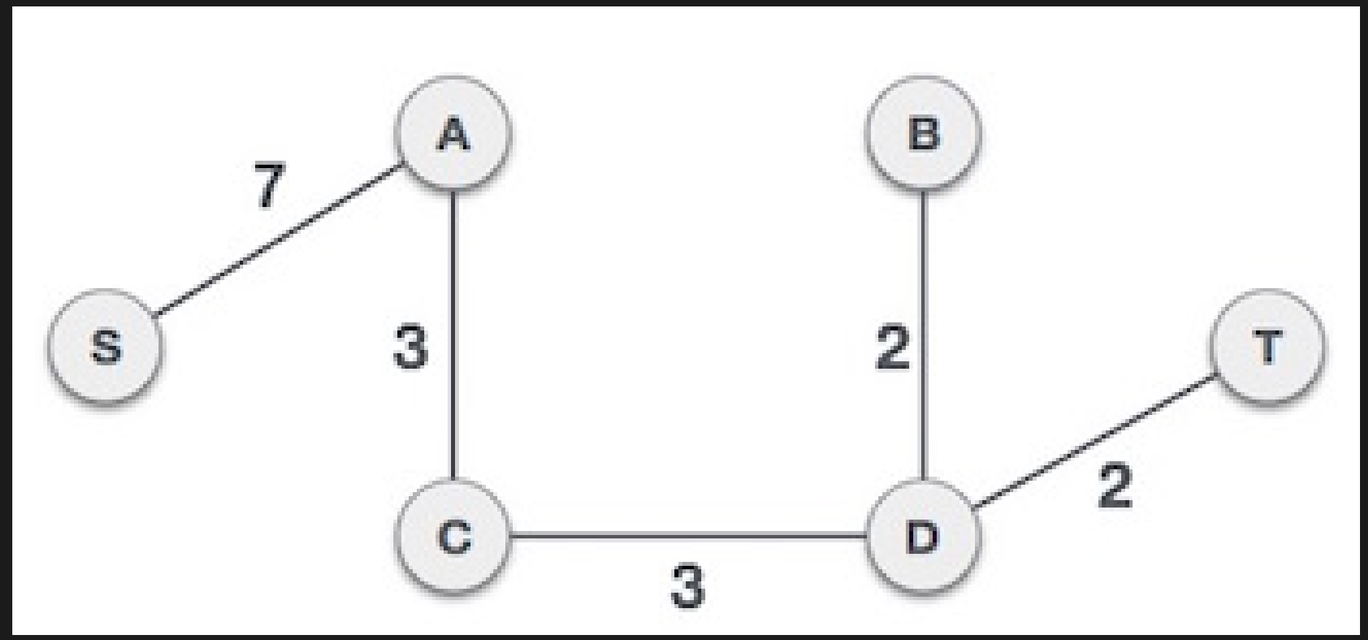
We ignore it. In the process we shall ignore/avoid all edges that create a circuit.



We observe that edges with cost 5 and 6 also create circuits. We ignore them and move on.



Now we are left with only one node to be added. Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.

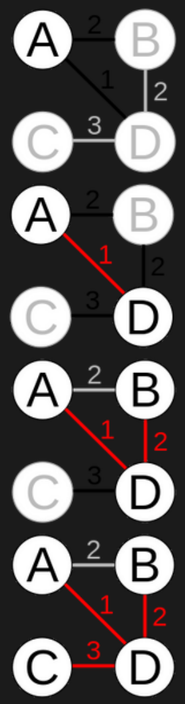


By adding edge S,A we have included all the nodes of the graph and we now have minimum cost spanning tree.

Prim's algorithm

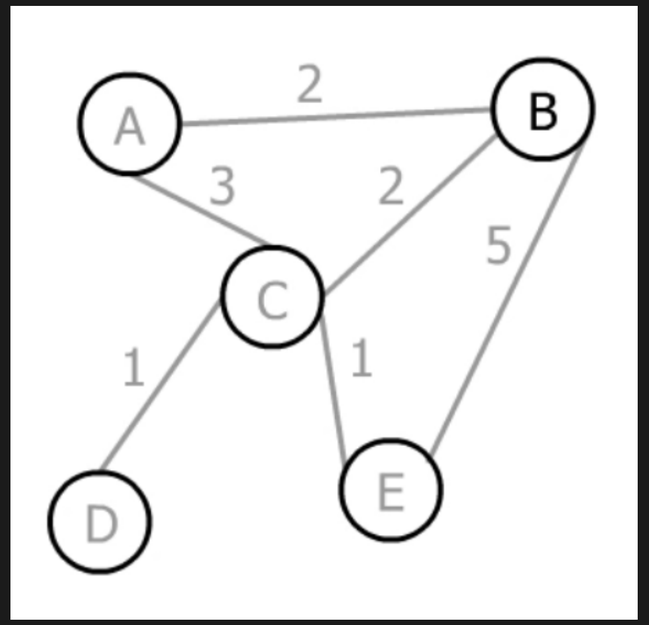
Prim's algorithm takes a weighted, undirected, connected graph as input and returns an MST of that graph as output. It works in a greedy manner.

In the first step, it selects an arbitrary vertex. Thereafter, **each new step adds the nearest vertex to the tree constructed so far** until there is no disconnected vertex left.

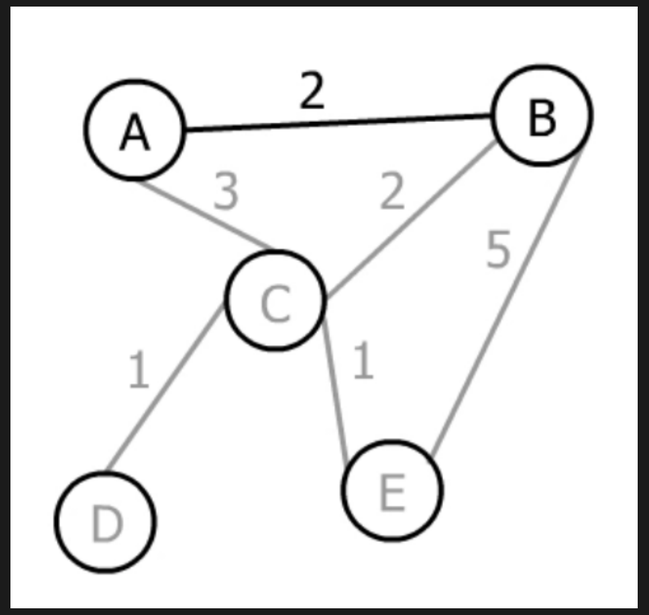


In the illustration, the edges that are already included in the MST are highlighted in red, and the current candidates are highlighted in black, from which the edge with the minimum weight is selected.

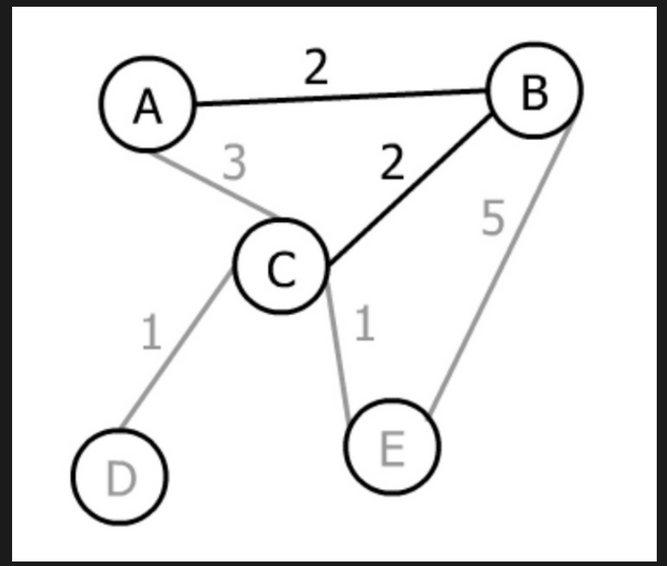
Let's run Prim's algorithm on this graph step-by-step:



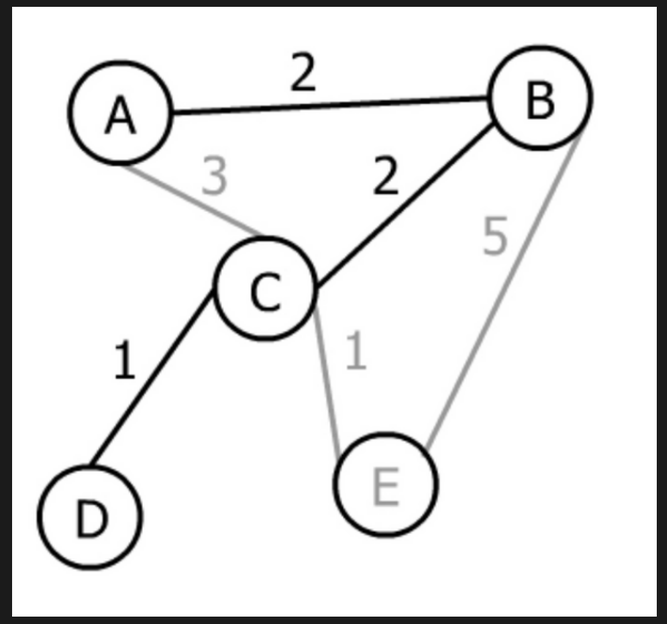
Assuming the arbitrary vertex to start the algorithm is B, we have three choices A, C, and E to go. The corresponding weights of the edges are 2, 2, and 5, therefore the minimum is 2. In this case, we have two edges weighing 2, so we can choose either of them (it doesn't matter which one). Let's choose A:



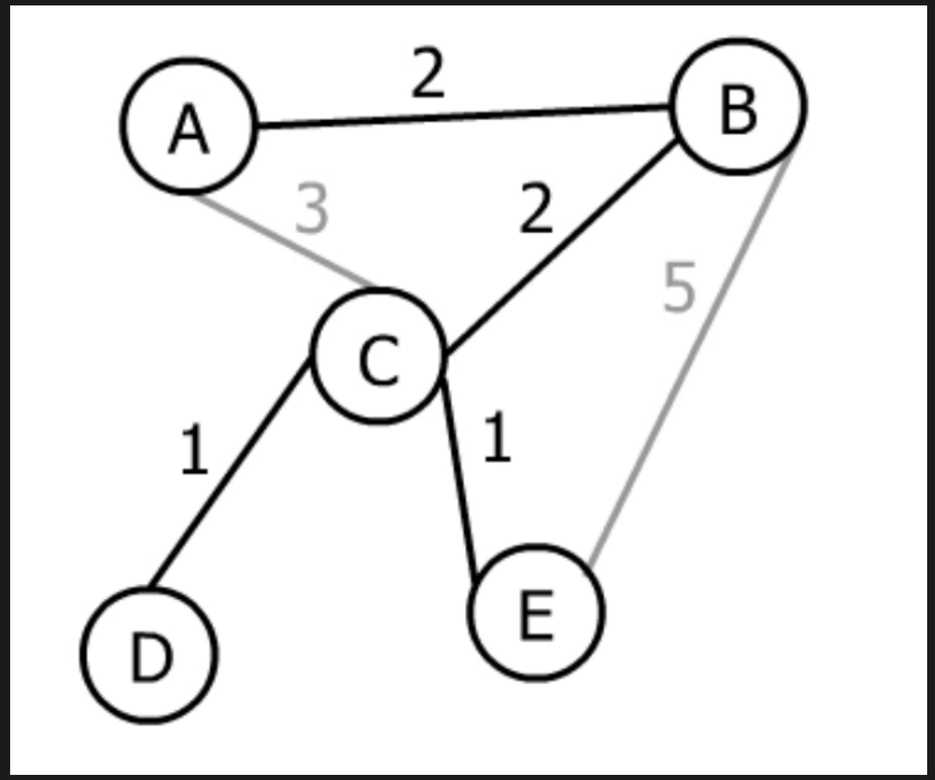
Now we have a tree with two vertices A and B. We can select any of A or B's edges not yet added that lead to an unadded vertex. So, we can pick AC, BC, or BE.
Prim's algorithm chooses the minimum, which is 2, or BC:



Now we have a tree with three vertices and three possible edges to move forward: CD, CE, or BE. AC isn't included since it wouldn't add a new vertex to the tree. The minimum weight amongst these three is 1.
However, there are two edges both weighing 1. Consequently, Prim's algorithm chooses one of them (again doesn't matter which one) in this step:



There is only one vertex left to join, so we can pick from CE and BE. The minimum weight that can connect our tree to it is 1, and Prim's algorithm will choose it:



As all vertices of the input graph are now present in the output tree, Prim's algorithm ends. Therefore, this tree is an MST of the input graph.

And the differences between these two algorithms are

Prim's Algorithm	Kruskal's Algorithm
The tree that we are making or growing always remains connected.	The tree that we are making or growing usually remains disconnected.
Prim's Algorithm grows a solution from a random vertex by adding the next cheapest vertex to the existing tree.	Kruskal's Algorithm grows a solution from the cheapest edge by adding the next cheapest edge to the existing tree / forest.
Prim's Algorithm is faster for dense graphs.	Kruskal's Algorithm is faster for sparse graphs.