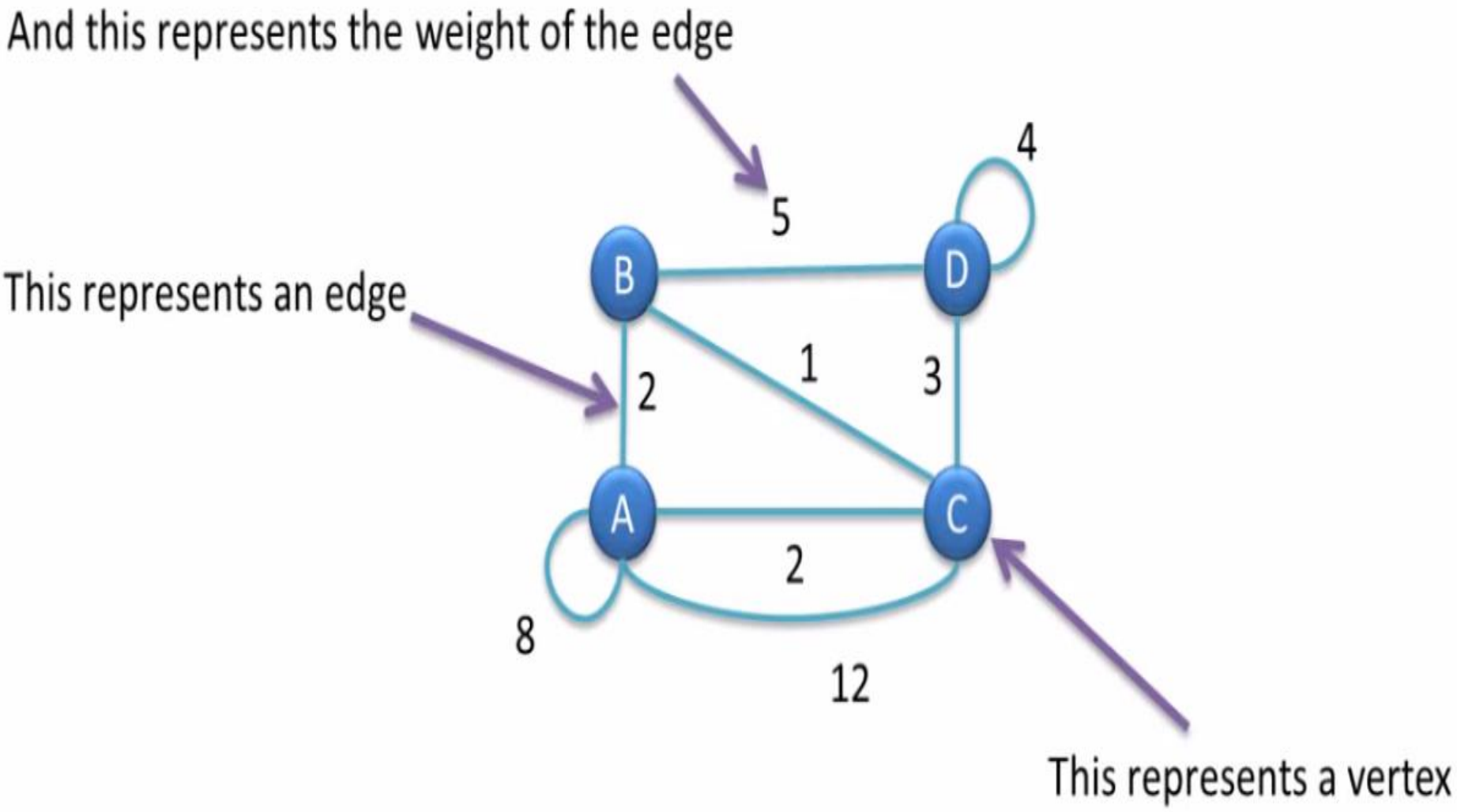
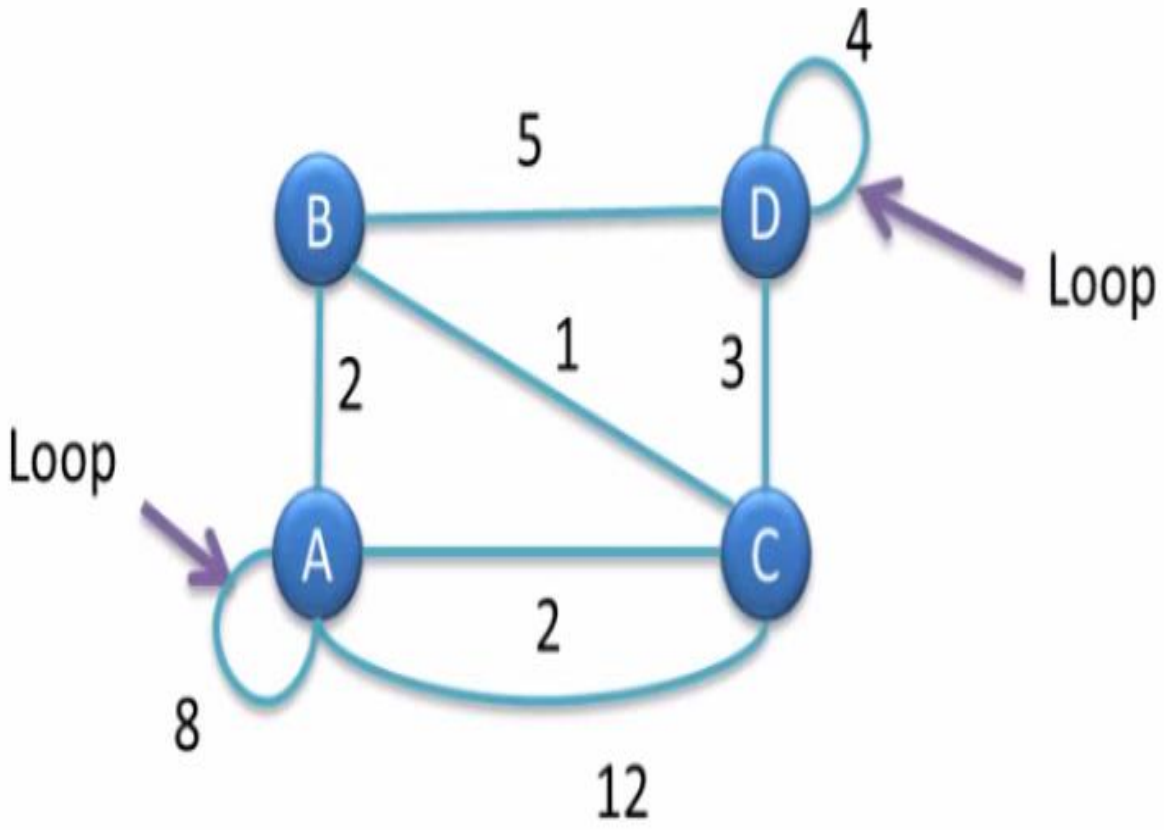
Kruskal's

Here is our graph

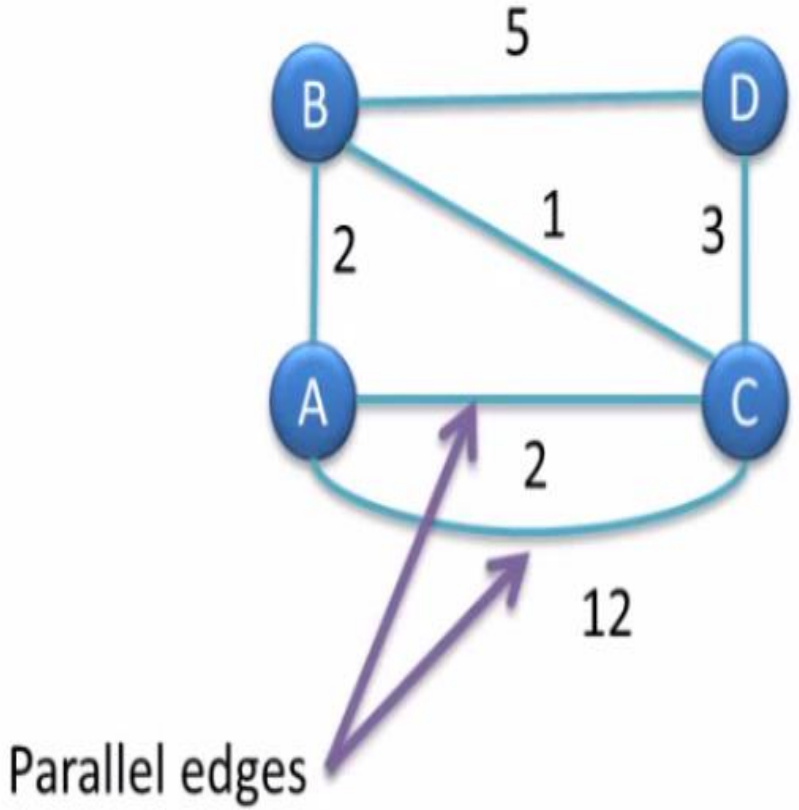


Step 1: Remove all the loops

Note!
Any edge that starts and ends at the same vertex is a loop.

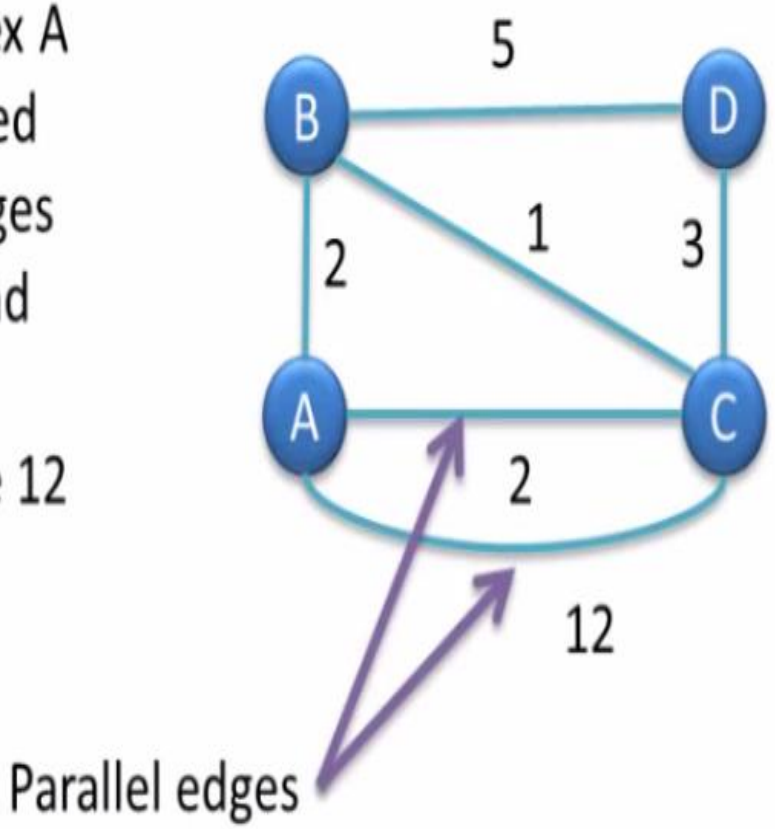


Step 2: Remove all parallel edges between two vertex except the one with least weight



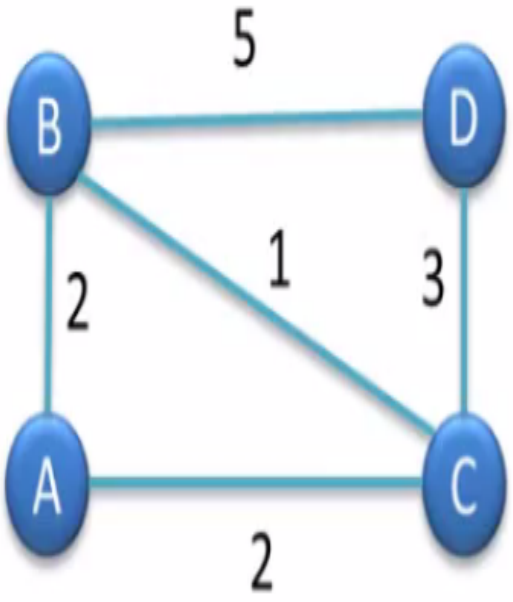
Step 2: Remove all parallel edges between two vertex except the one with least weight

Note!
In this graph, vertex A and C are connected by two parallel edges having weight 2 and 12 respectively.
So, we will remove 12 and keep 2.



Step 3: Create the edge table

An edge table will have name of all the edges along with their weight is ascending order.



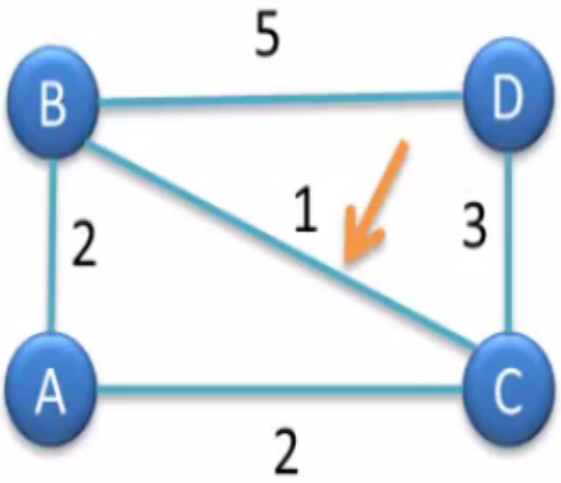
Now if you look at the graph, then you will notice that there are total 5 edges.

So our edge table will have 5 columns.

Edge					
Weight					

Now look at the graph and fill the 1st column with the edge of minimum weight.

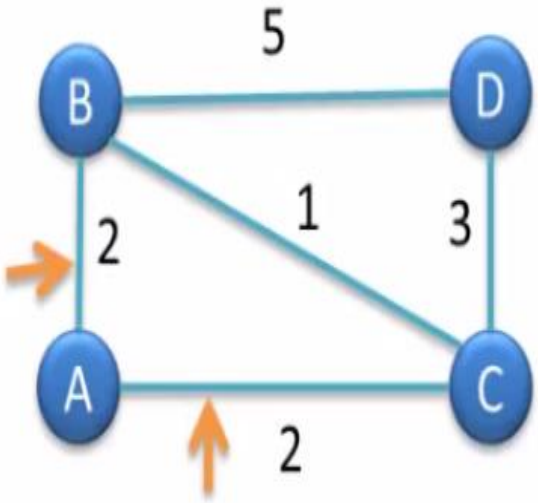
In this case edge BC is of least weight so we will select it.



Edge	BC				
Weight	1				

Now look at the graph again and find the edge of next minimum weight.

In this case edge AB and edge AC are having the minimum weight so we will select them both.

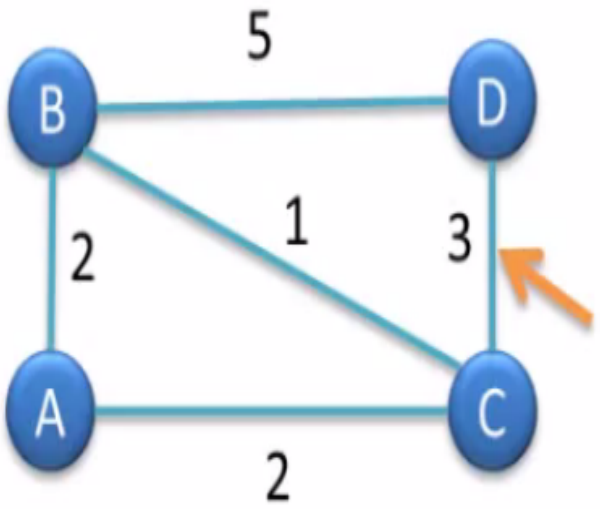


Note!
In case we have two or more edges with same weight then we can write them in any order in the edge table.

Edge	BC	AB	AC		
Weight	1	2	2		

Now look at the graph again and find the edge of next minimum weight.

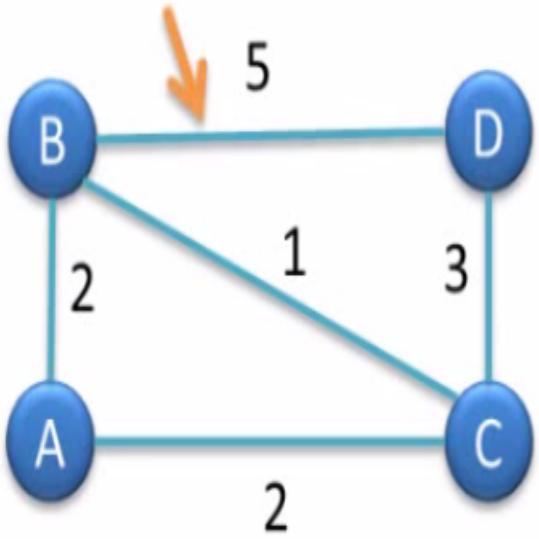
In this case edge DC has the minimum weight so we will select it.



Edge	BC	AB	AC	DC	
Weight	1	2	2	3	

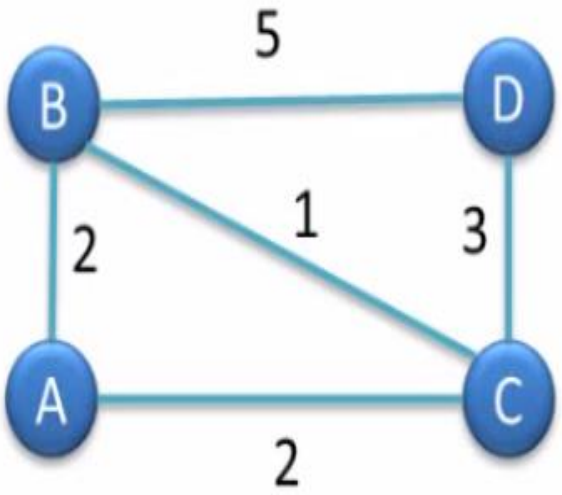
Now look at the graph again and find the edge of next minimum weight.

In this case edge BD has the minimum weight so we will select it.



Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

Time to find the minimum spanning tree.

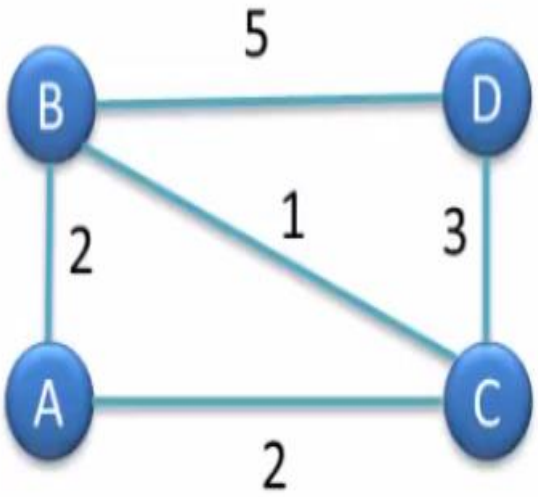


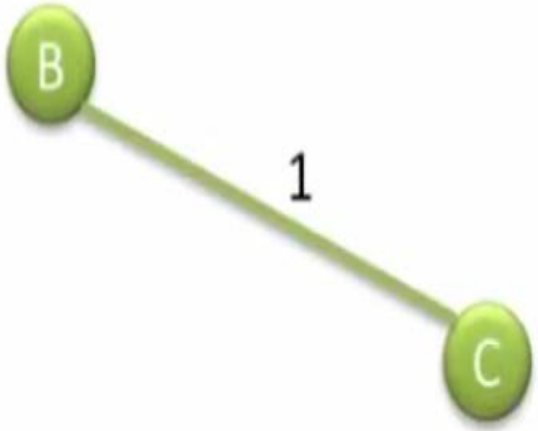
How we will proceed

To find the MST (Minimum Spanning Tree) we will start from the smallest weight edge and keep selecting edges that does not form any circuit with the previously selected edges.

Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

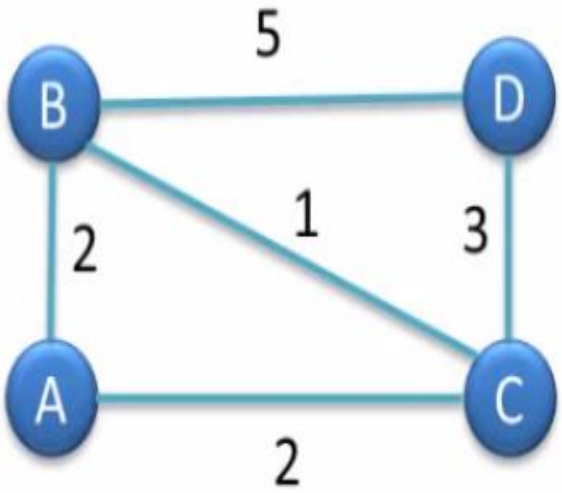
1 is the smallest weight so we will select edge BC

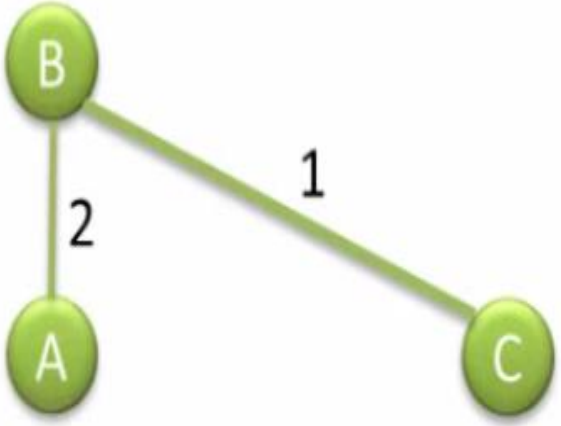




Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

2 is the next smallest weight and edge AB does not form a circuit with the previously selected edges so we will select it.



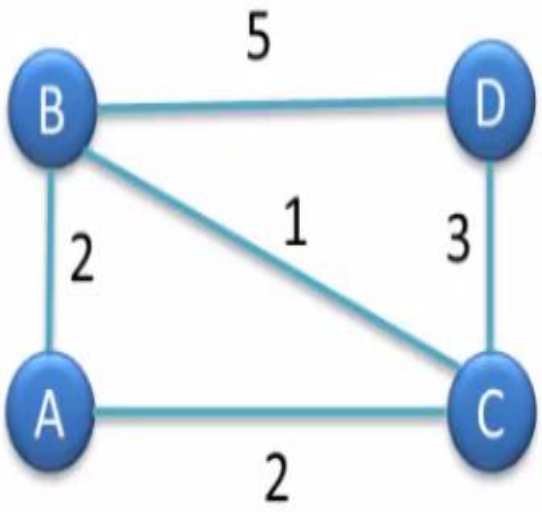


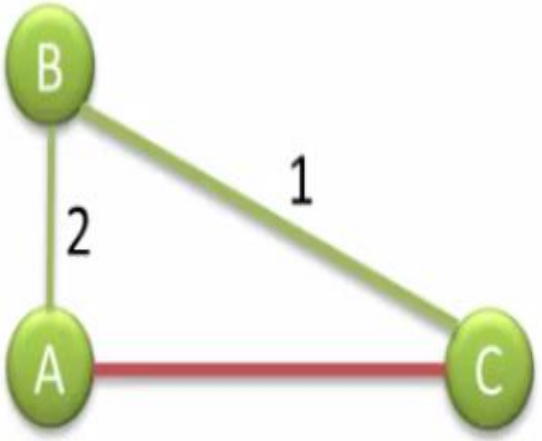
Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

2 is the next smallest weight.

But if we select edge AC then it will form a circuit.

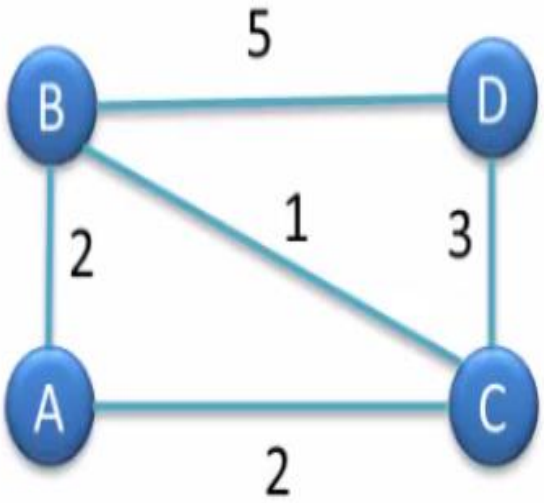
So we are going to reject edge AC.

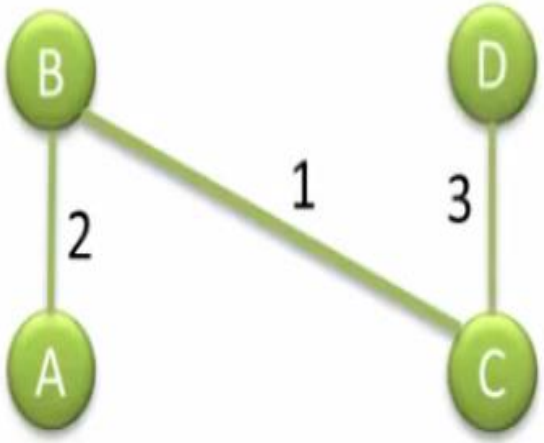




Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

3 is the next smallest weight and edge DC does not form a circuit with the previously selected edges. So we will select it.

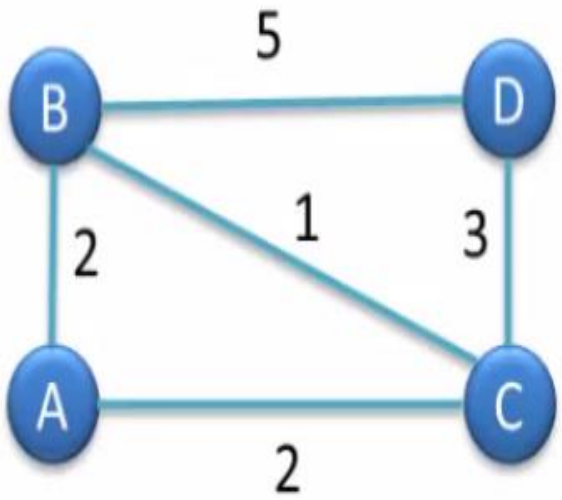


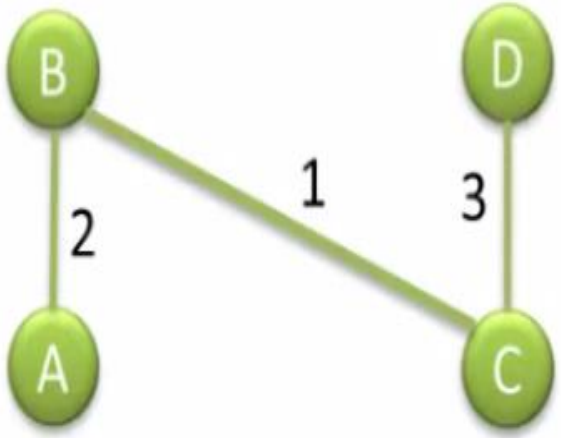


Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

Since we have got the 3 edges so we will stop here.

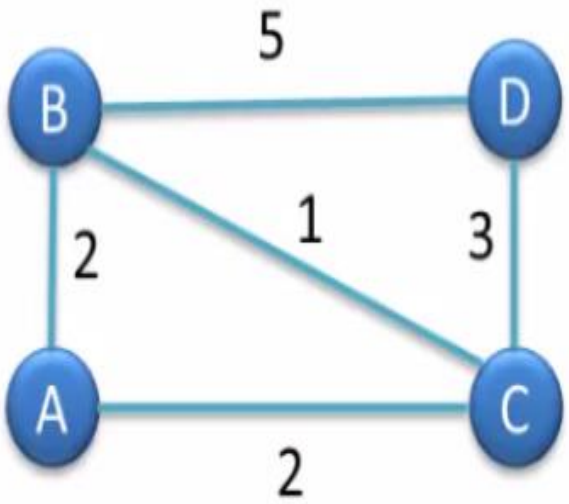
Note!
Our graph has 4 vertices so, our MST will have 3 edges.

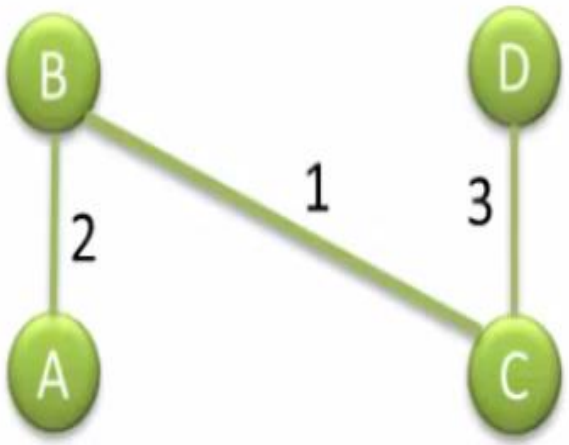


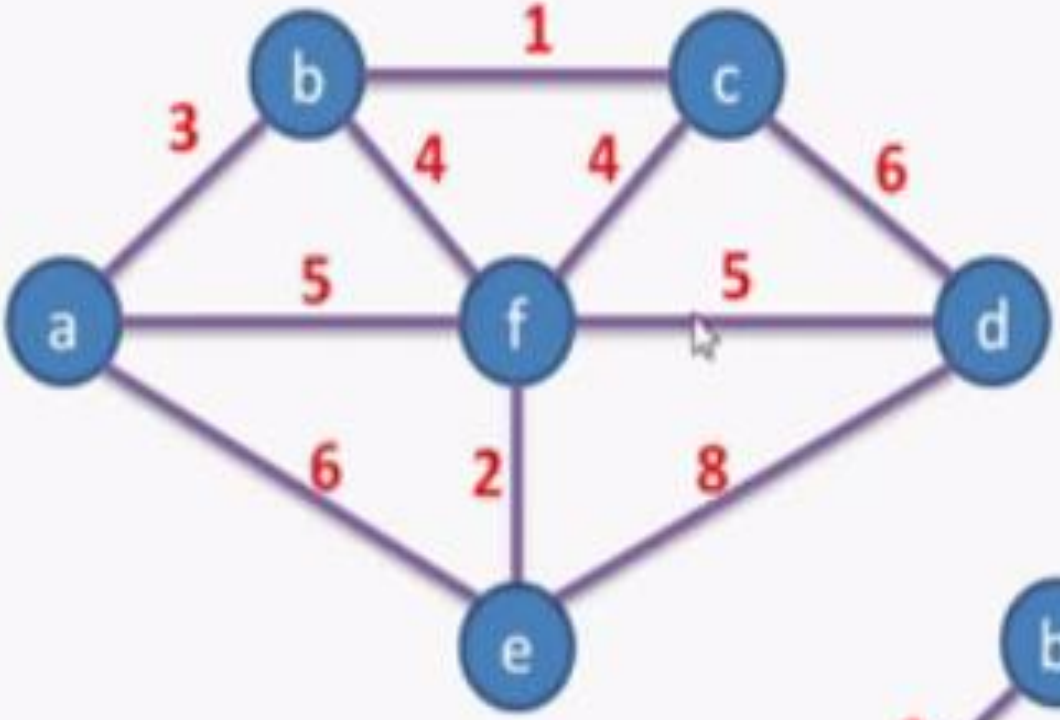


Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

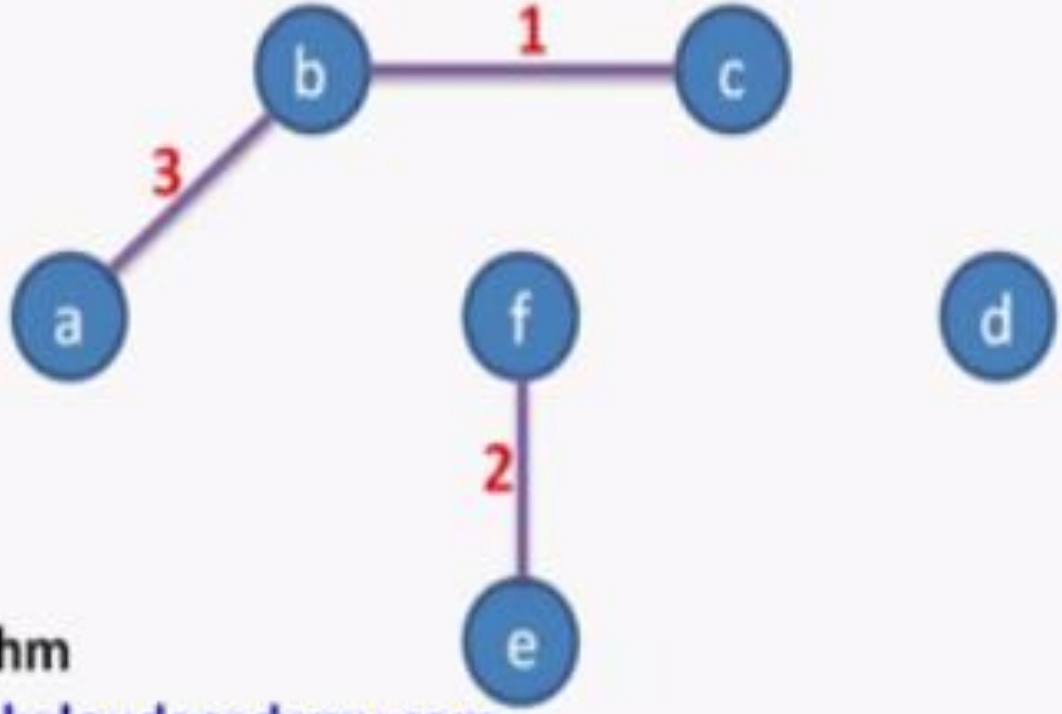
Therefore, our required Minimum Spanning Tree is







Kruskal's Algorithm
Example



Kruskal's Algorithm

Assume that all the edges are Sorted in ascending order of weights

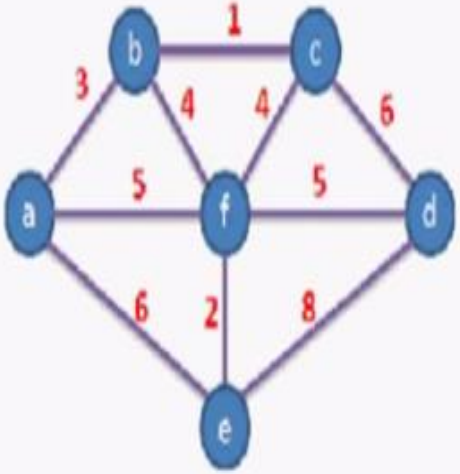
Et → Array of edges which is our solution

To keep track of how many edges are there in our edge set.

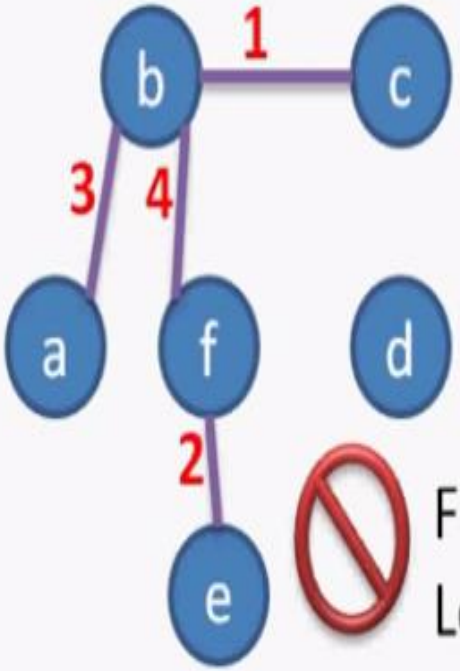
Till we connect all the vertices with edges go ahead

```
Algorithm Kruskal (G)
    Sort E in ascending order of weights
    Et ← 0 //no edges selected
    encounter ← 0 //no of edges selected
    k ← 0
    while encounter < |V| - 1
        k ← k + 1
        if Et U {eik} is acyclic
            Et ← Et U {eik}
            encounter += 1
    return Et
```

Kruskal's Algorithm



Edge	Lengths
b→c	1
f→e	2
a→b	3
b→f	4
c→f	4
a→f	5
d→f	5
a→e	6
c→d	6
e→d	8



Forms Loop !!!

Algorithm Kruskal (G)

Sort E in ascending order of weights

Et ← 0 //no edges selected

encounter ← 0 //no of edges selected

k ← 0

while encounter < |V| - 1

    k ← k + 1

    if Et U {eik} is acyclic

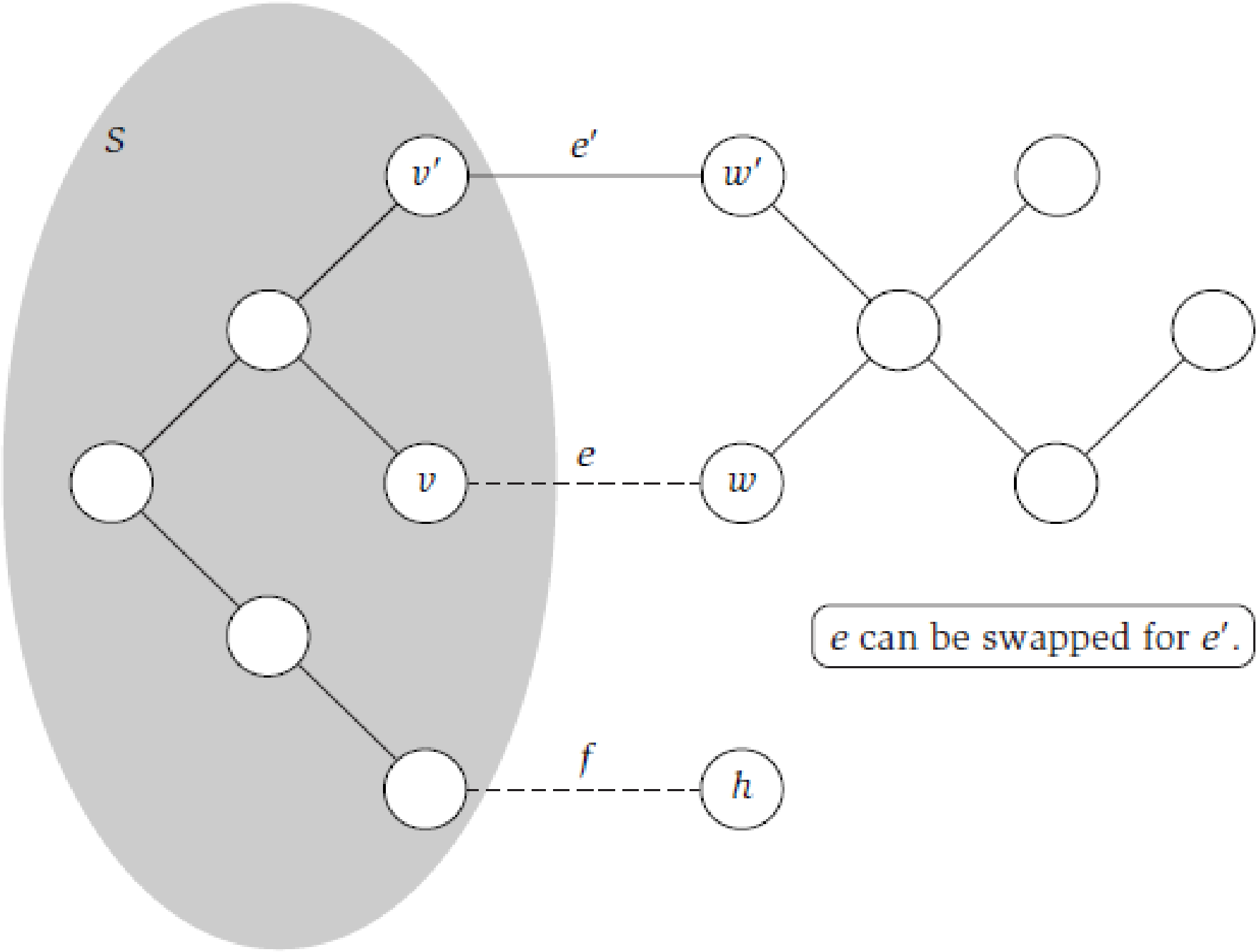
        Et ← Et U {eik}

        encounter += 1

return Et

Kruskal's Algorithm

ANALYSIS


S
v′
e′
w′
v
e
w
f
h
e can be swapped for e′.

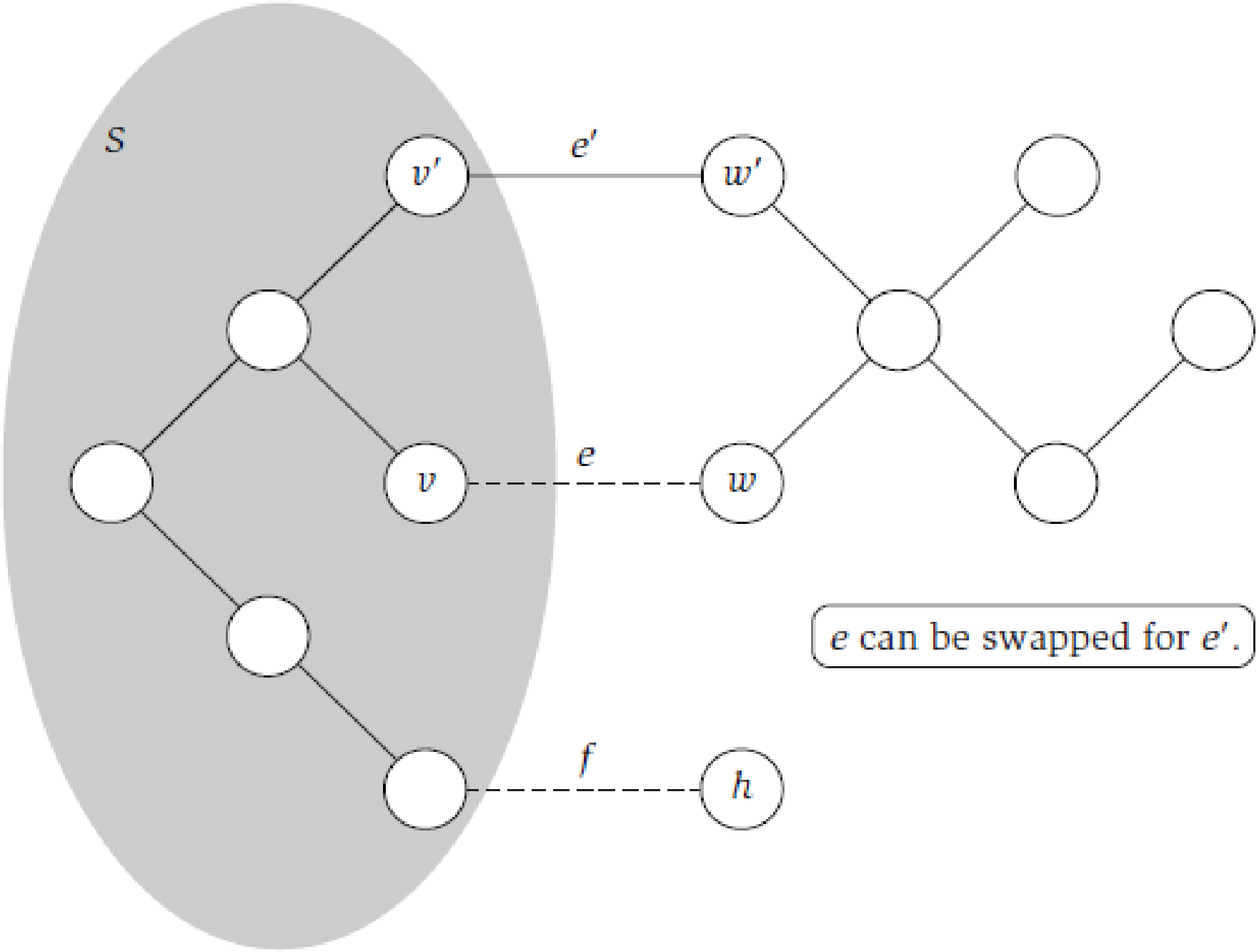
- ***Let T be a minimum-cost solution to the network design problem defined above. Then (V, T) is a tree.***

- **Proof.** By definition, (V, T) must be connected.
- We show that it also will contain no cycles. Indeed, suppose it contained a cycle C.

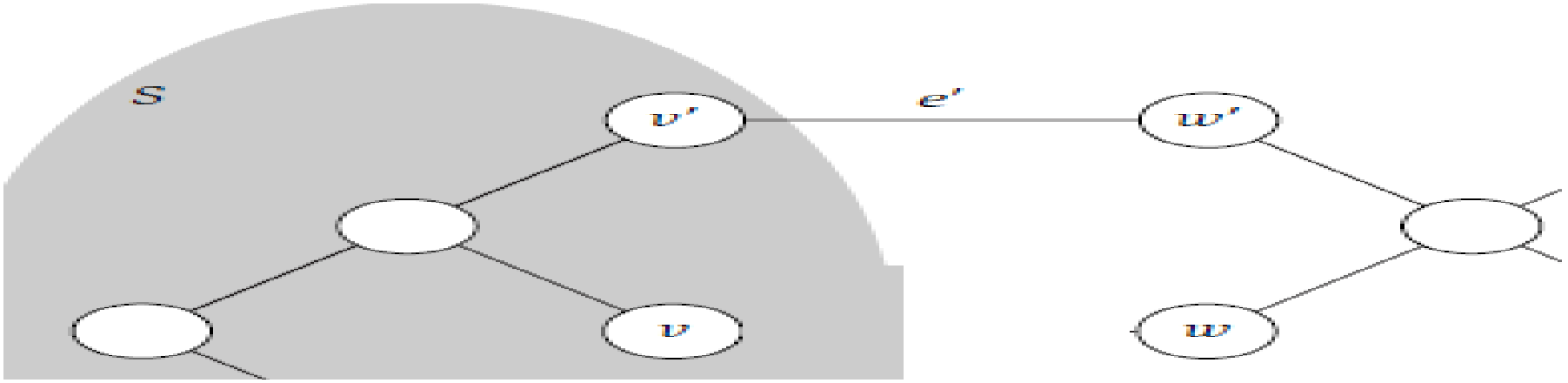
- Let e be any edge on C. We claim that $(V, T - \{e\})$ is still connected.
-
- Since any path that previously used the edge e can now go "the long way" around the remainder of the cycle C instead.

- It follows that $(V, T - \{e\})$ is also a valid solution to the problem, and it is cheaper—a contradiction.

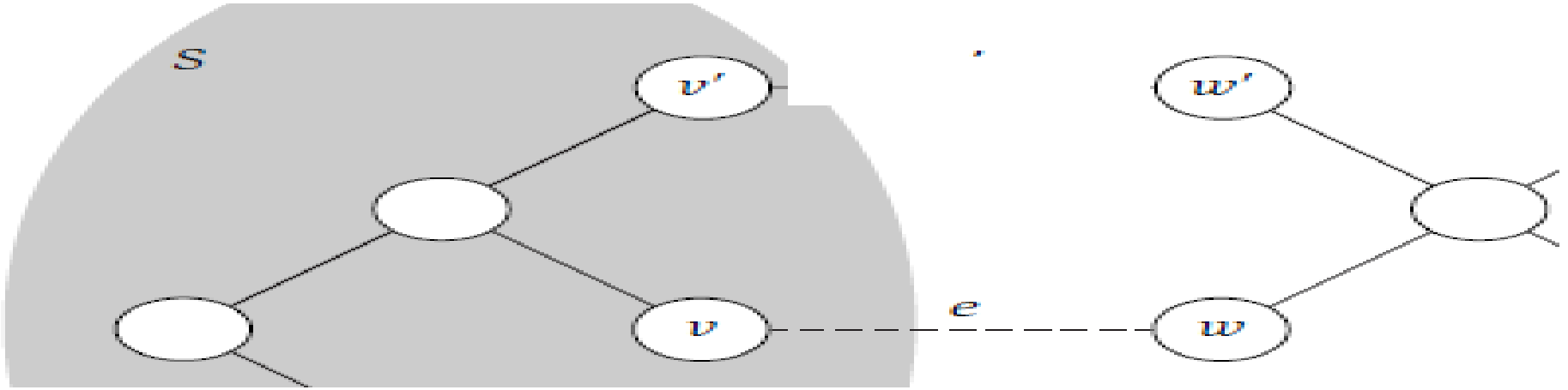
- ***Assume that all edge costs are distinct. Let S be any subset of nodes that is neither empty nor equal to all of V, and let edge e = (v, w) be the minimum cost edge with one end in S and the other in V – S. Then every minimum spanning tree contains the edge e.***


S
v′
e′
w′
v
e
w
f
h
e can be swapped for e′.

- Let *T* be a spanning tree that does not contain *e*; we need to show that *T* does not have the minimum possible cost.



- we'll identify an edge *e'* in *T* that is more expensive than *e*, and with the property exchanging *e* for *e'* results in another spanning tree T'. This resulting spanning tree will then be cheaper than *T.*



- the ends of *e* are *v* and *w*. *T* is a spanning tree, so there must be a path *P* in *T* from *v* to *w*.
- Starting at *v*, suppose we follow the nodes of *P* in sequence; there is a first node *w'* on *P* that is in *V* – *S*.
- Let v' € S be the node just before w' on P, let e'=(v',w') be the edge joining them.

- Let *e* = *(v, w)* be the edge in P.
- *e* is an edge of *T* with one end in *S* and the other in *V* – *S.*
- If we exchange *e* for *e'*, we get a set of edges

  $T' = T - \{e'\} \cup \{e\}$.
- We claim that *T'* is a spanning tree.

- Clearly *(V, T')* is connected, since *(V, T)* is connected, and any path in *(V, T)* that used the edge *e' = (v', w')* can now be "rerouted" in *(V, T')* to follow the portion of *P* from *v'* to *v*,

- then the edge *e*, and then the portion of *P* from *w* to *w'*. To see that *(V, T')* is also acyclic.

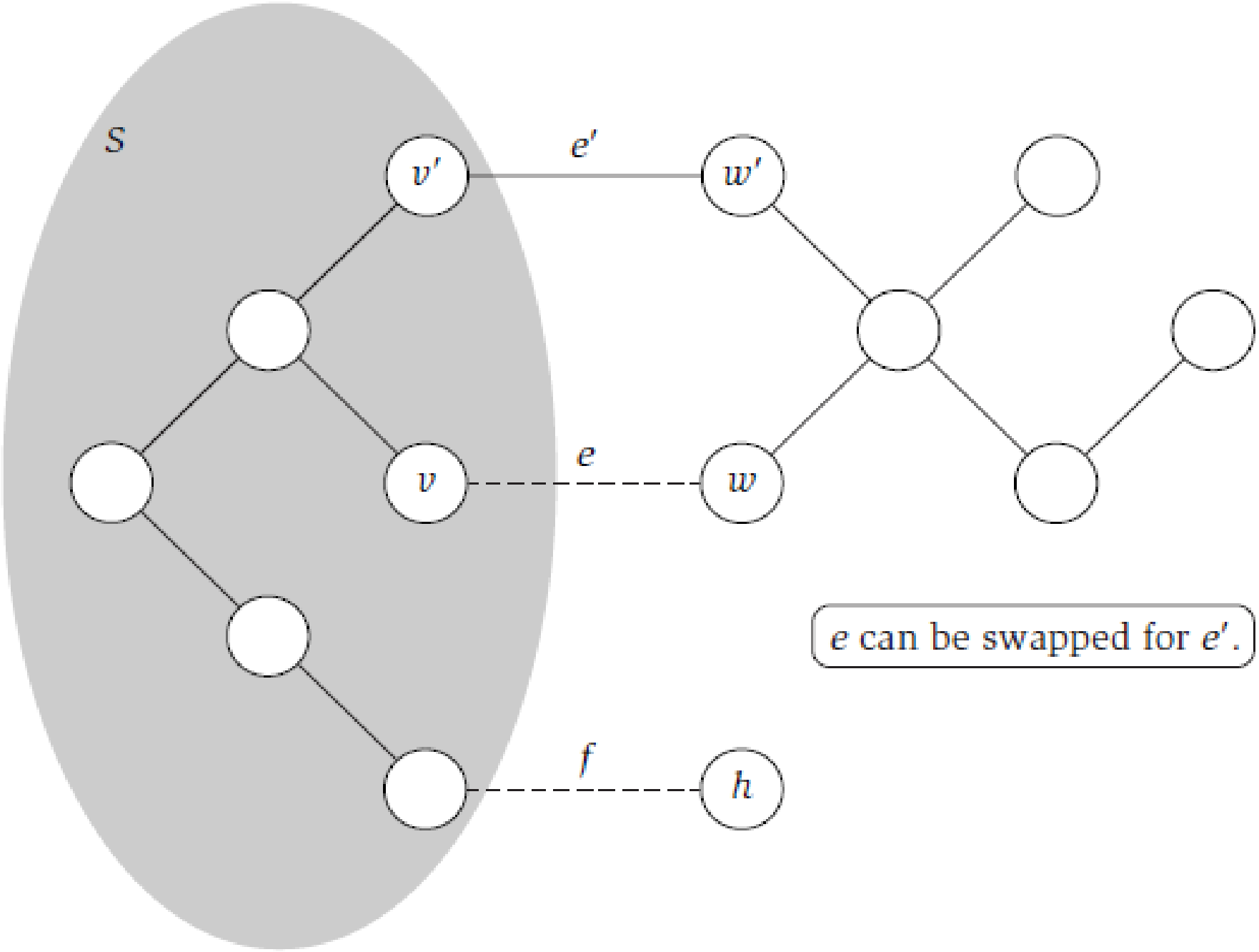
- note that the only cycle in *(V, T* ∪ {*e'*}*)* is the one composed of *e* and the path P

- and this cycle is not present in *(V, T')* due to the deletion of *e'*.

- We noted above that the edge *e'* has one end in *S* and the other in *V* – *S*.

- But *e* is the cheapest edge with this property, and so *ce* < *ce'*. (The inequality is strict since no two edges have the same cost.)

- Thus the total cost of *T'* is less than that of *T*, as desired.


S
v′
e′
w′
v
e
w
f
h
e can be swapped for e′.

- ***Kruskal's Algorithm produces a minimum spanning tree of G.***

Proof.

- Consider any edge $e = (v, w)$ added by Kruskal's Algorithm, and let

- S be the set of all nodes to which v has a path at the moment just before e is added.
- Clearly $v \in S$, but $w \notin S$, since adding e does not create a cycle.

- Moreover, no edge from S to $V - S$ has been encountered yet, since any such edge could have been added without creating a cycle, and hence would have been added by Kruskal's Algorithm.

- Thus e is the cheapest edge with one end in S and the other in $V - S$, and so by (4.17) it belongs to every minimum spanning tree.

- So if we can show that the output *(V, T)* of Kruskal's Algorithm is in fact a spanning tree of *G*, then we will be done.

- Clearly *(V, T)* contains no cycles, since the algorithm is explicitly designed to avoid creating cycles.

- Further, if *(V, T)* were not connected, then there would exist a nonempty subset of nodes *S* such that there is no edge from *S* to $V - S$.
- But this contradicts the behavior of the algorithm: we know that since *G* is connected, there is at least one edge between *S* and $V - S$, and the algorithm will add the first of these that it encounters.

- ***Prim's Algorithm produces a minimum spanning tree of G.***

Proof.

- For Prim's Algorithm, it is also very easy to show that it only adds edges belonging to every minimum spanning tree.
- Indeed, in each iteration of the algorithm, there is a set $S \subseteq V$ on which a partial spanning tree has been constructed, and a node v and edge e are added that minimize the quantity
- $\min_{e=(u,v):u \in S} c_e$.
- By definition, e is the cheapest edge with one end in S and the other end in $V - S$, and so by the Cut Property (4.17) it is in every minimum spanning tree.

- It is also straightforward to show that Prim's Algorithm produces a spanning tree of G, and hence it produces a minimum spanning tree.