Graph Theory

– Course 3 –

Chapter 3:TREES AND ROOTED TREES (1/1)

Section 1:
Definitions

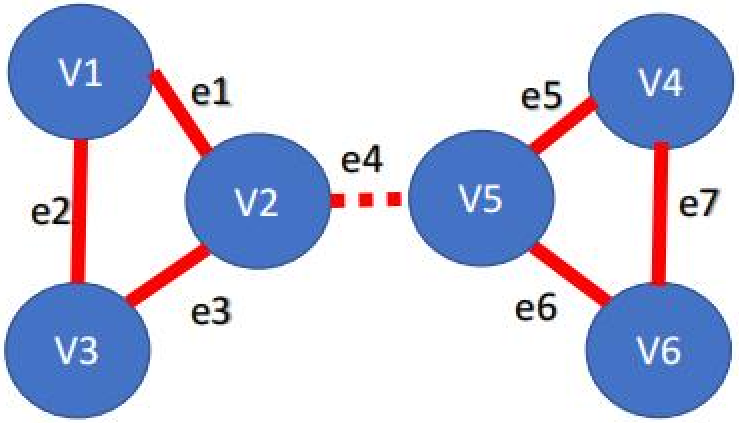
Isthmus

- Isthmus (*Bridge*)

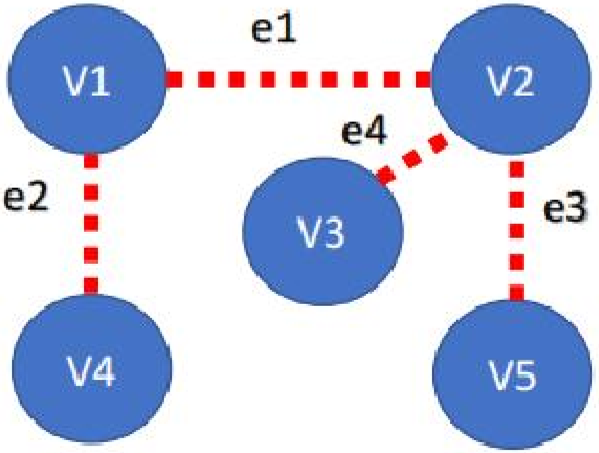
An edge is an isthmus if its deletion increases the number of connected components of a graph.

In graph G1 only edge e4 is an isthmus.
In graph G2 all edges are isthmuses.

G1=({V1,V2,V3,V4,V5,V6},{e1,e2,e3,e4,e5,e6,e7})



G2 =({V1,V2,V3,V4,V5}, {e1,e2,e3,e4})



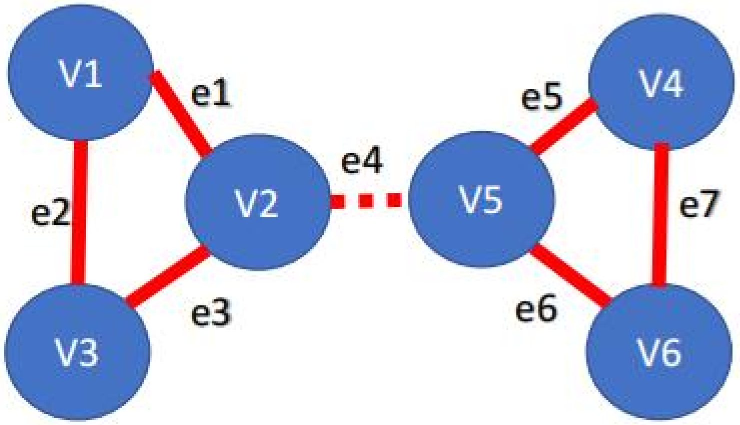
Isthmus

- Isthmus (*Bridge*)

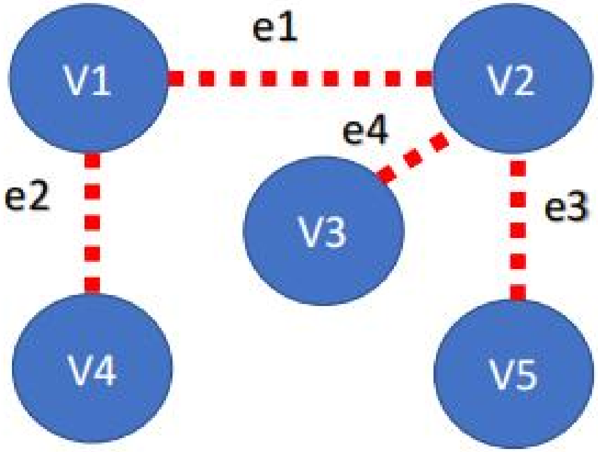
An edge of a graph is an isthmus if it does not belong to any cycle.

In graph G1 only edge e4 is an isthmus (e4 does not belong to any cycle).
In graph G2 all edges are isthmuses (all edges in graph G2 do not belong to any cycle).

G1=({V1,V2,V3,V4,V5,V6},{e1,e2,e3,e4,e5,e6,e7})



G2 =({V1,V2,V3,V4,V5}, {e1,e2,e3,e4})

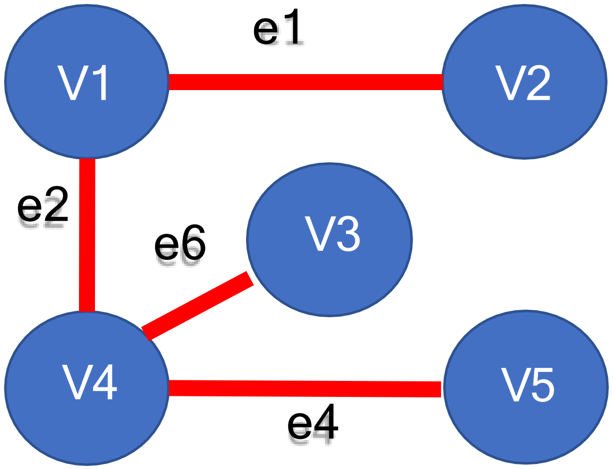


Tree

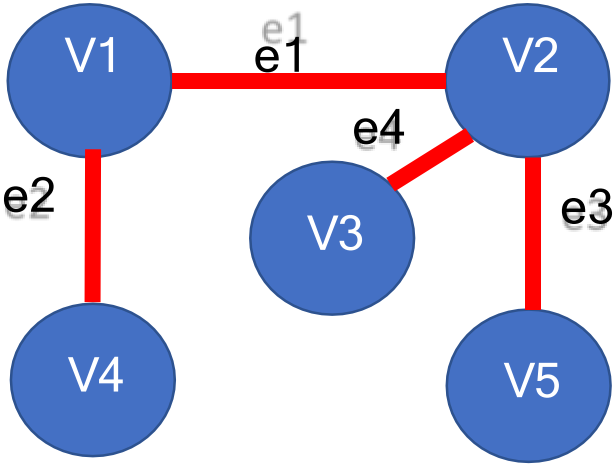
- TREE

A tree is a connected graph containing no cycles. All the edges of a tree are isthmuses.

G1=({V1,V2,V3,V4,V5},{e1,e2,e4,e6}) is a tree



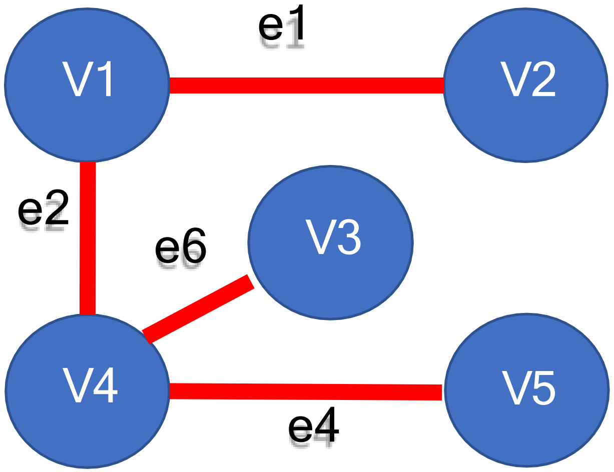
G2 =({V1,V2,V3,V4,V5}, {e1,e2,e5,e3}) is a tree



Tree (five properties)

- TREE : property 1

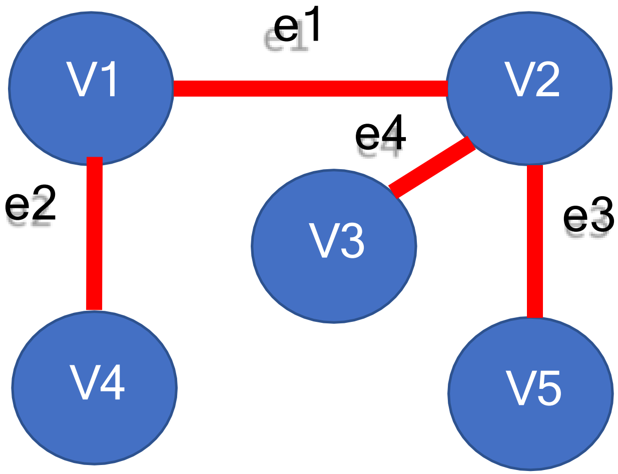
G1=({V1,V2,V3,V4,V5},{e1,e2,e4,e6}) is a tree



Let G=(X,U) be a graph with |X|=n≥2 .

The following properties are equivalent and characterize a tree:

G2 =({V1,V2,V3,V4,V5}, {e1,e2,e5,e3}) is a tree



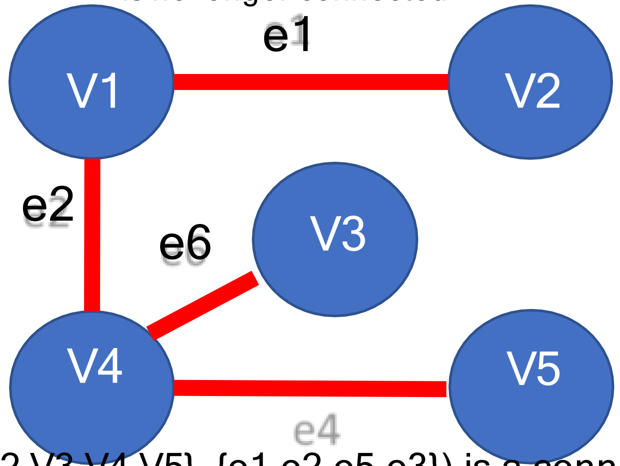
Property 1

G is connected and cycle-free.

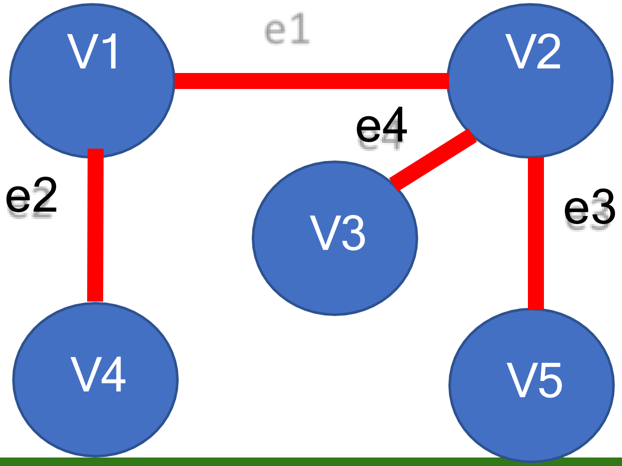
Tree (five properties)

- TREE : property 2

G1=({V1,V2,V3,V4,V5},{e1,e2,e4,e6}) is a connected and cycle-free tree, if we remove an edge it is no longer connected



G2 =({V1,V2,V3,V4,V5}, {e1,e2,e5,e3}) is a connected tree and without cycle, if we delete an edge it is no longer connected



Property 2

G is connected and is minimal for property 1. (If we remove an edge from G, then G is no longer connected).

Tree (five properties)

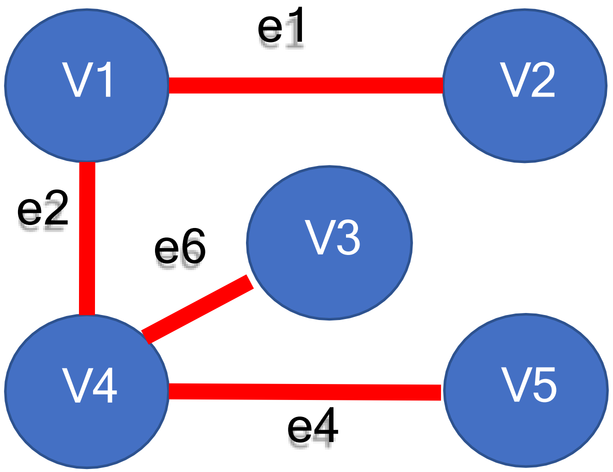
- TREE : property 3

G1=({V1,V2,V3,V4,V5},{e1,e2,e4,e6}) is a tree

connected and without cycle, having 5-1=4 edges

Property 3

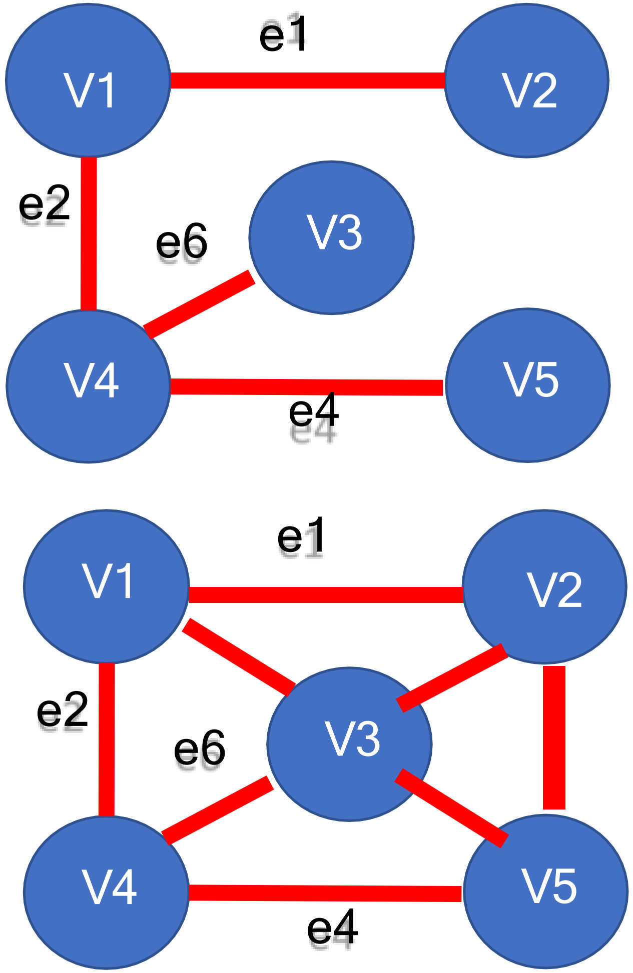
G is connected and has n-1 edges.



Tree (five properties)

- TREE : property 4

G1=({V1,V2,V3,V4,V5},{e1,e2,e4,e6}) is a connected and cycle-free tree, with 5-1=4 edges, if we add an edge, we form a cycle



Property 4

G is cycle-free and is maximal for property 3.
(If we add an arc to G, then it has a cycle).

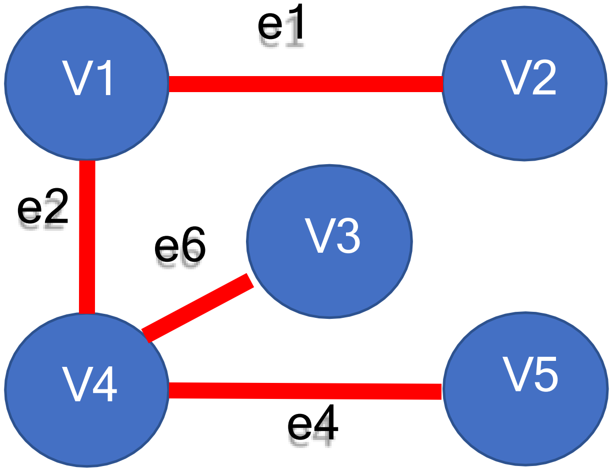
Tree (five properties)

- TREE : property 5

G1=({V1,V2,V3,V4,V5},{e1,e2,e4,e6}) is a connected and cycle-free graph, with 5-1=4 edges, in G, it exists one and only one chain joining any pair of vertices

Property 5

In G, it exists one and only one chain joining any pair of vertices.

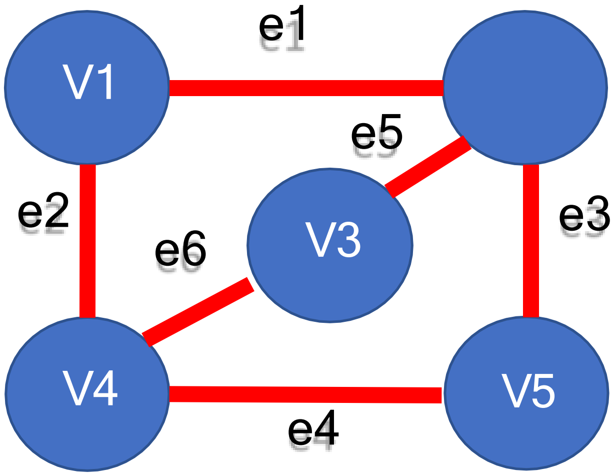


Spanning tree

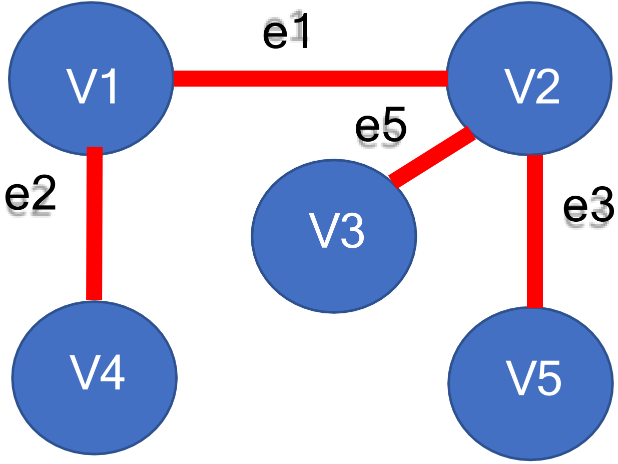
- *Spanning Tree*

A *Spanning Tree* is a maximal subgraph of a graph containing no cycles (which is also a tree).
Maximum means containing all vertices of the tree.

G=({V1,V2,V3,V4,V5},{e1,e2,e3,e4,e5,e6})



Spanning tree of G treeG =({V1,V2,V3,V4,V5},
{e1,e2,e5,e3})



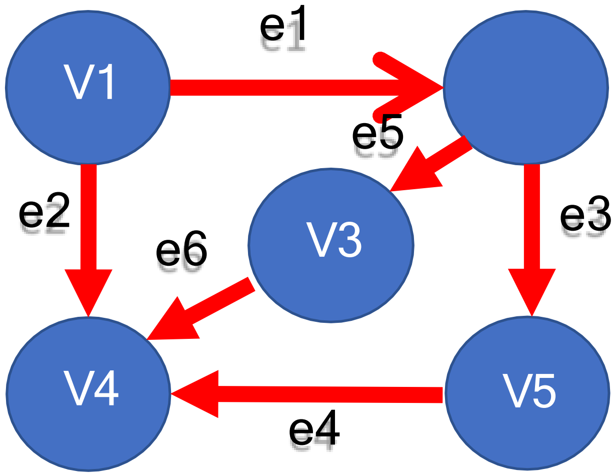
Rooted Tree

- *Rooted tree*

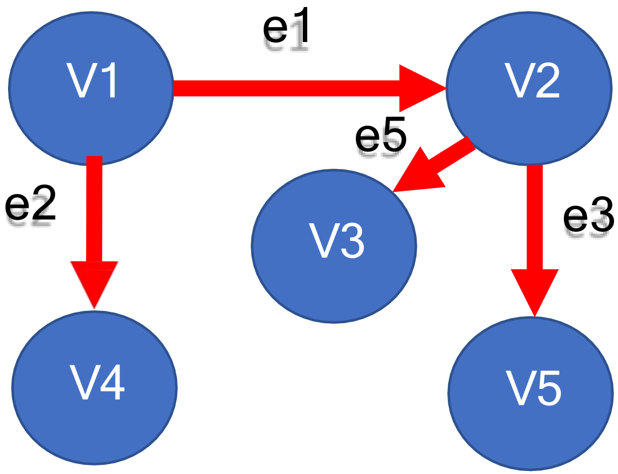
A *Rooted tree* is a tree with distinguished or privileged vertex r (*Root*).

Example: RootedG is a tree with vertex V1 as root, because RootedG is a tree + there is one and only one path from V1 to all vertices.

G=({V1,V2,V3,V4,V5},{e1,e2,e3,e4,e5,e6})



RootedG =({V1,V2,V3,V4,V5},
{e1,e2,e5,e3})

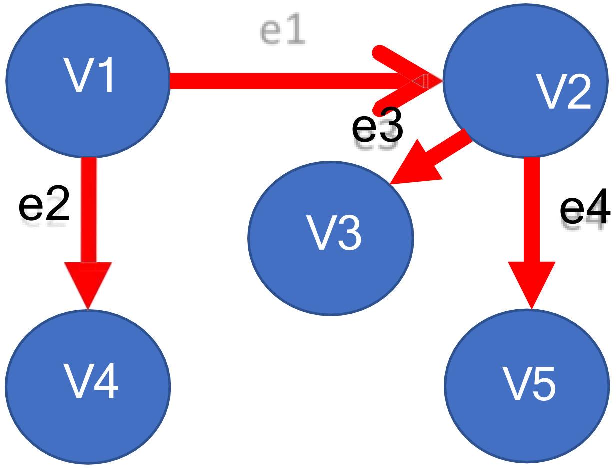


Forest

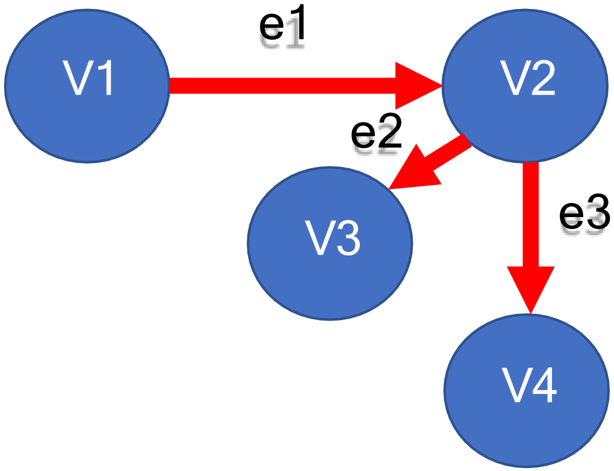
- *Forest*

A forest is a set of trees.
A forest is a graph that does not contain cycles.
The components of the forest are trees.
Example: Tree 1 and Tree 2 are two components of a forest.

Tree1=({V1,V2,V3,V4,V5},{e1,e2,e3,e4})



Tree2 =({V1,V2,V3,V4}, {e1,e2,e3})



Section 2:
Problem of min/max spanning trees

Min/max spanning trees problem

The problem is posed as follows: given a graph with a number of vertices and a number of edges having weights of values in the set of relative integers

- the minimum spanning tree consists of finding the set of edges allowing to join all the vertices without forming a cycle, and this, with a minimal cost

graph $\xrightarrow{\text{Spanning tree}}$ tree with all vertices and minimal cost

Min/max spanning trees problem

This problem finds various practical applications: it is directly applicable to

- optimization
- the design of various types of networks (electrical, internet, etc.).

Section 3:
Application of algorithms (Kruskal, Prim)

History of Spanning Tree Algorithms

- Kruskal's algorithm (1956)
- Prim's algorithm (1957)
  - Prim-dijkstra, Jarnik (1930)
- Boruvka's algorithm (1926)
  - Sollin (1961)

The most well-known minimum spanning tree algorithms are those of Kruskal and Prim

Principle of Kruskal's algorithm

- Sort edge weights in ascending order.
- Add edges and their vertices to the spanning tree if no cycle creation.
- Stop if the number of edges in the spanning tree reaches the number of vertices in the original tree minus 1.

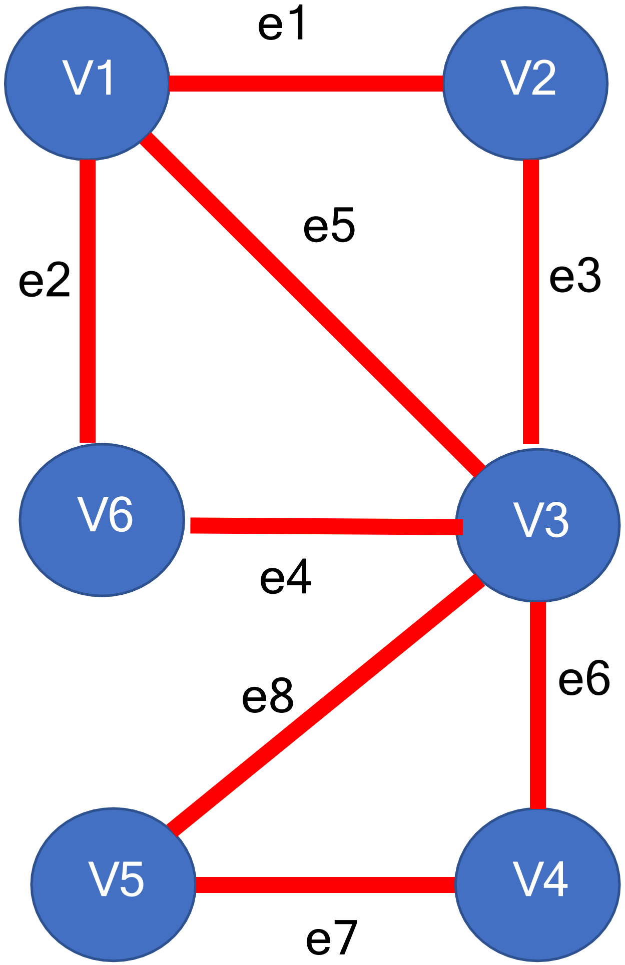
Kruskal maintains an acyclic tree at each edge addition

Kruskal's algorithm

- Building a minimal weight spanning tree

The problem data: A connected and weighted graph G = (V,E) defined by the finite set of vertices V = {v1, v2, . . . , vn} and by the finite set of valued edges

E={e1, e2, . . . , em}

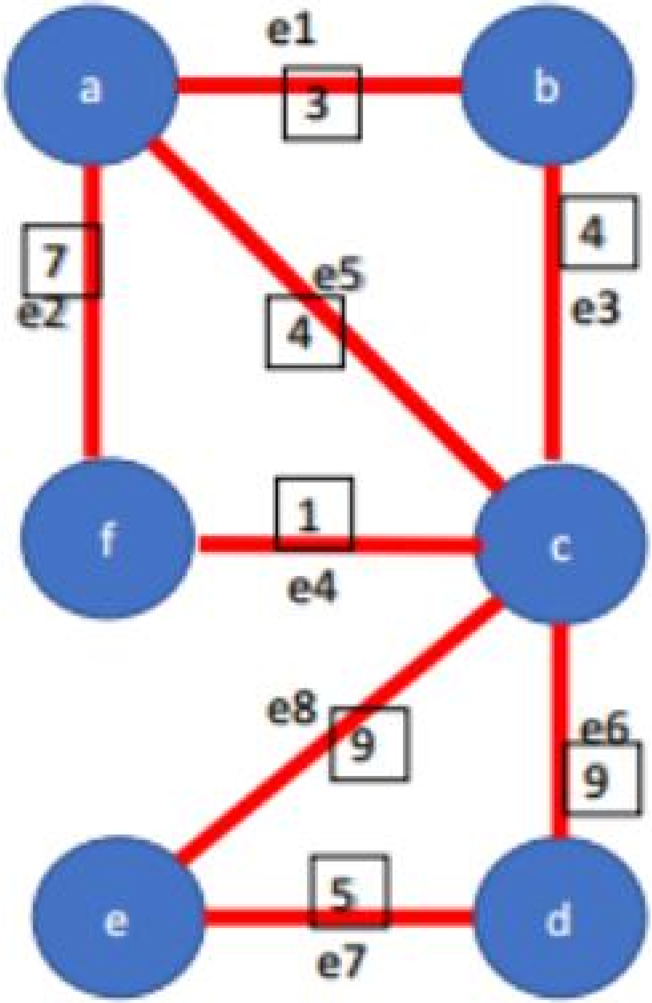


Kruskal's algorithm

- Building a minimal weight spanning tree

Example:

A connected and weighted graph G = (V,E) defined by the finite set of vertices V = {a,b,c,d,e,f} and by the finite set of valued edges E = {e1, e2,e3,e4,e5,e6,e7,e8} with the respective weights {3,7,4,1,4,9,5,9}



Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 1: Sort the edges by increasing weight

f,c : 1

a,b: 3

b,c: 4

a,c: 4

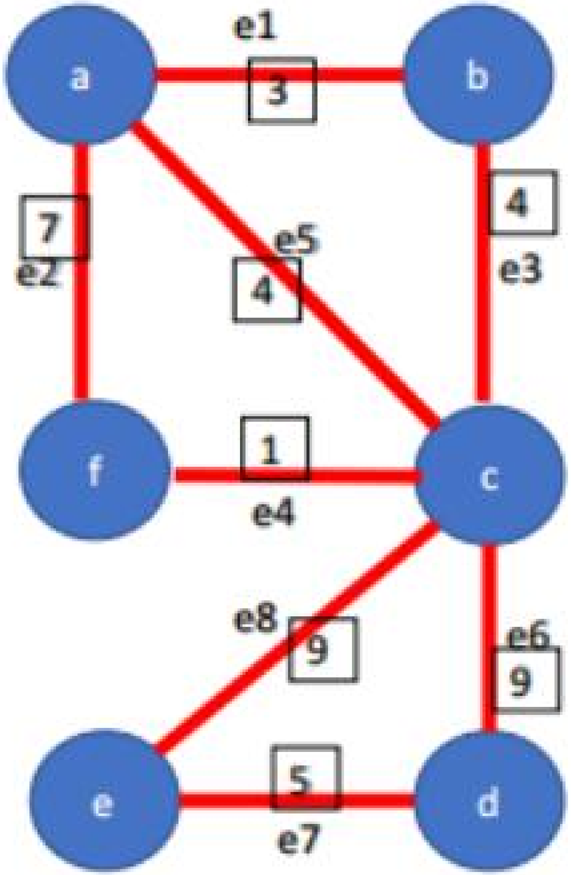
e,d: 5

a,f: 7

c,d: 9

c,e: 9

first execution



Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 1: Sort the edges by increasing weight

f,c : 1

a,b: 3

b,c: 4

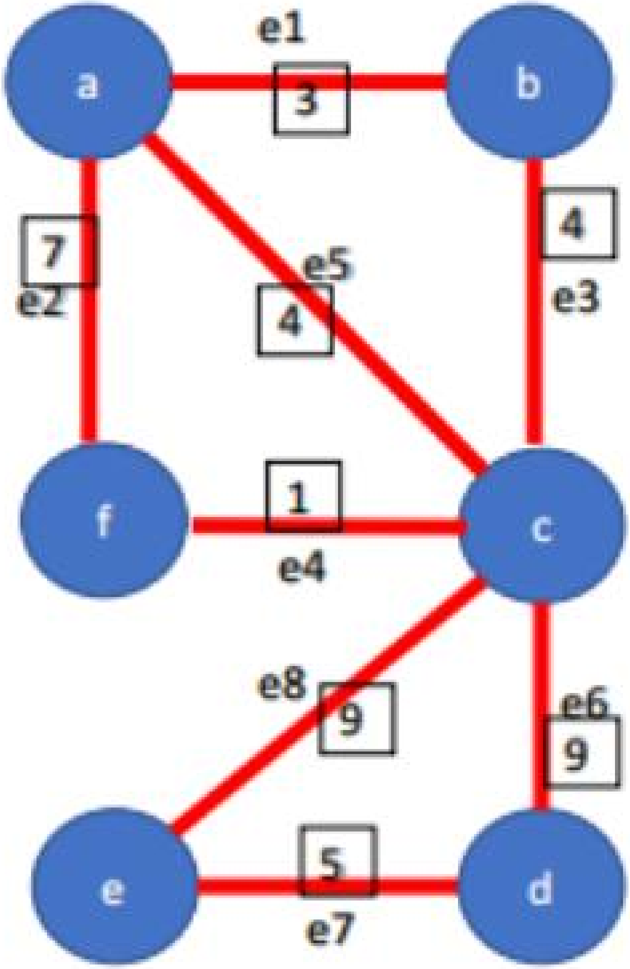
a,c: 4

e,d: 5

a,f: 7

c,d: 9

c,e: 9

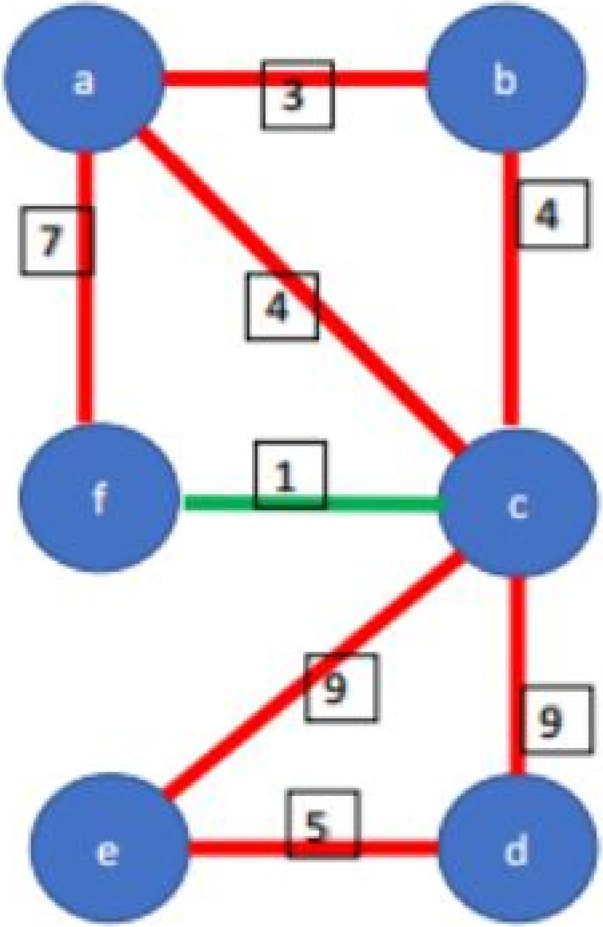


Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
b,c: 4
a,c: 4
e,d: 5
a,f: 7
c,d: 9
c,e: 9

f,c : 1 ✓



Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1

a,b: 3

b,c: 4

a,c: 4

e,d: 5

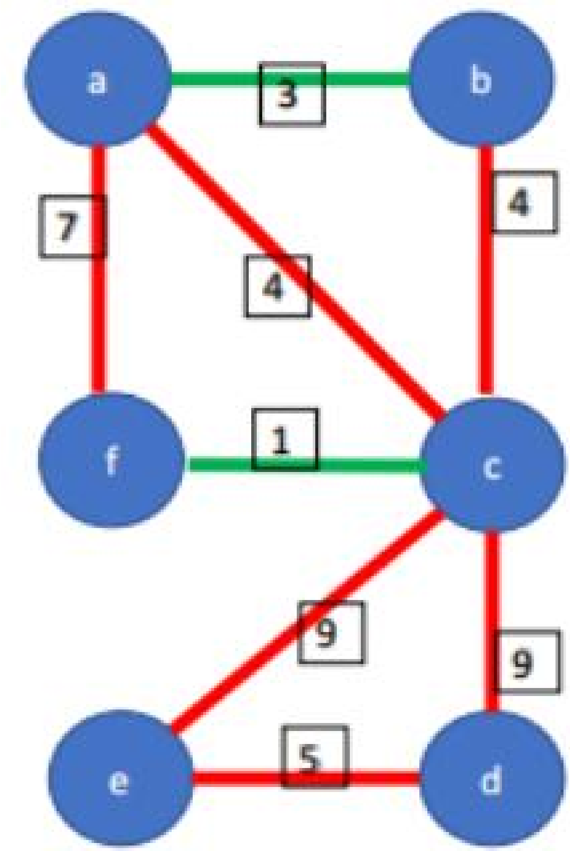
a,f: 7

c,d: 9

c,e: 9

f,c : 1 ✓

a,b: 3 ✓

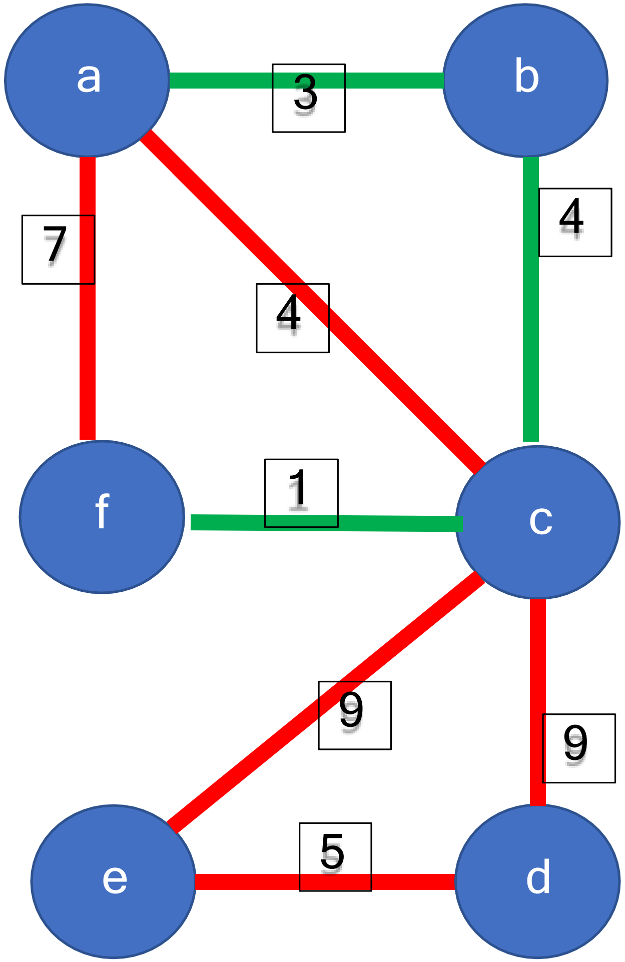


Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
b,c: 4
a,c: 4
e,d: 5
a,f: 7
c,d: 9
c,e: 9

f,c : 1 ✓
a,b: 3 ✓
b,c: 4 ✓

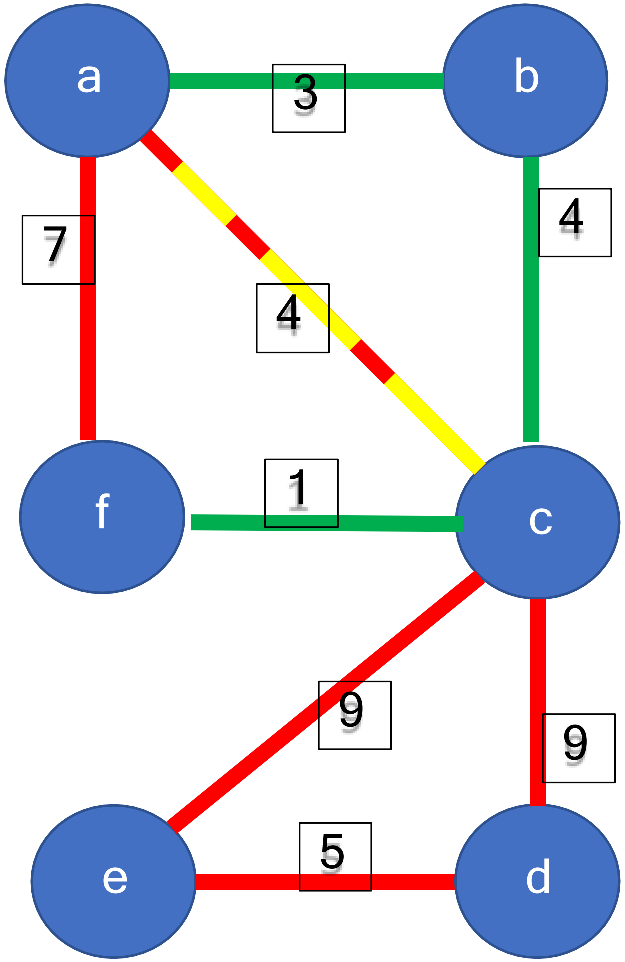


Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
b,c: 4
a,c: 4
e,d: 5
a,f: 7
c,d: 9
c,e: 9

f,c : 1 ✓
a,b: 3 ✓
b,c: 4 ✓
a,c: 4 ✗

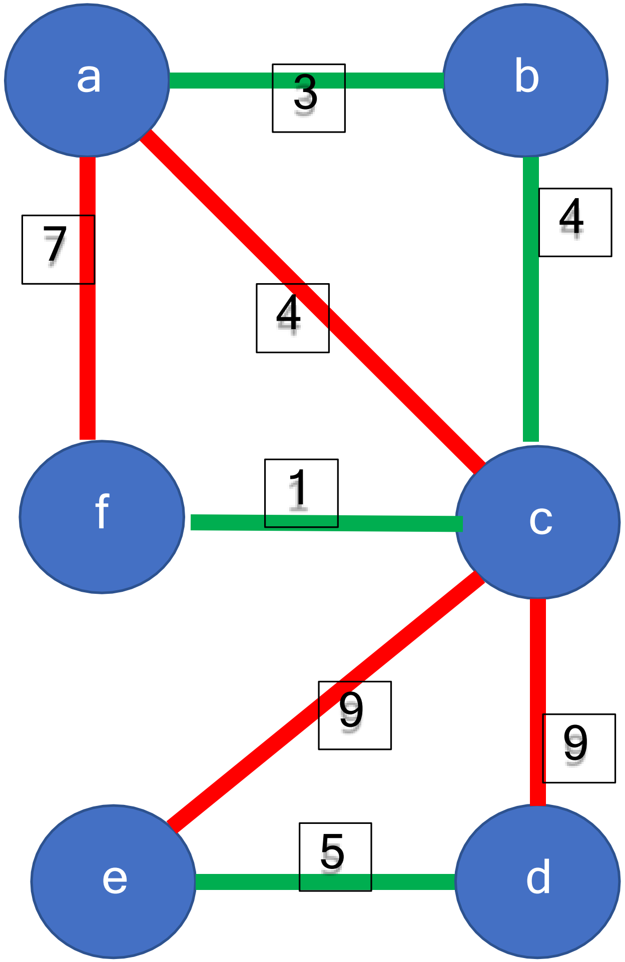


Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
b,c: 4
a,c: 4
e,d: 5
a,f: 7
c,d: 9
c,e: 9

f,c : 1 ✓
a,b: 3 ✓
b,c: 4 ✓
a,c: 4 ✗
e,d: 5 ✓



Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1

a,b: 3

b,c: 4

a,c: 4

e,d: 5

a,f: 7

c,d: 9

c,e: 9

f,c : 1 ✓

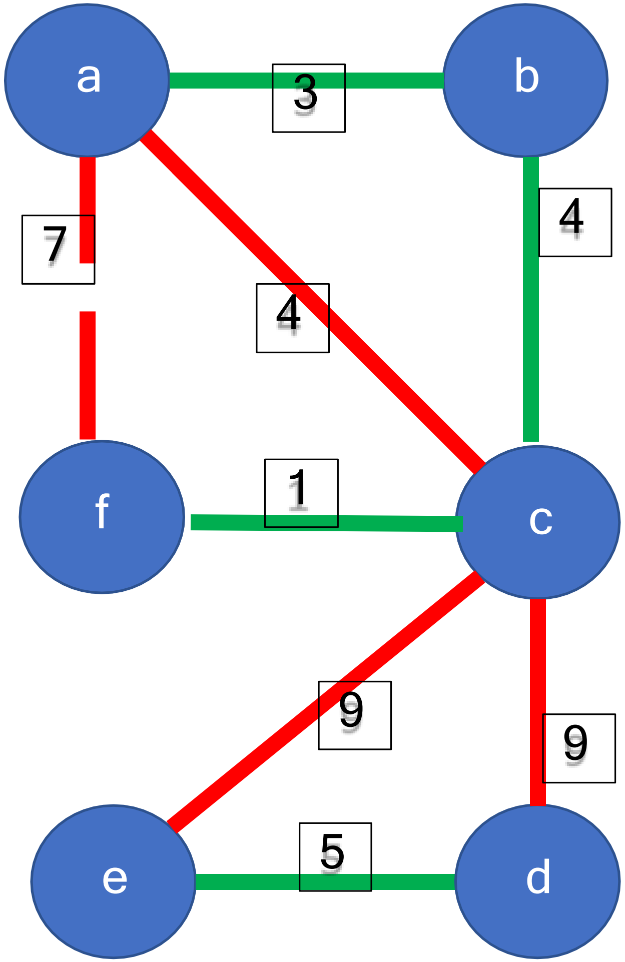
a,b: 3 ✓

b,c: 4 ✓

a,c: 4 ✗

e,d: 5 ✓

a,f: 7 ✗



Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1

a,b: 3

b,c: 4

a,c: 4

e,d: 5

a,f: 7

c,d: 9

c,e: 9

f,c : 1 ✓

a,b: 3 ✓

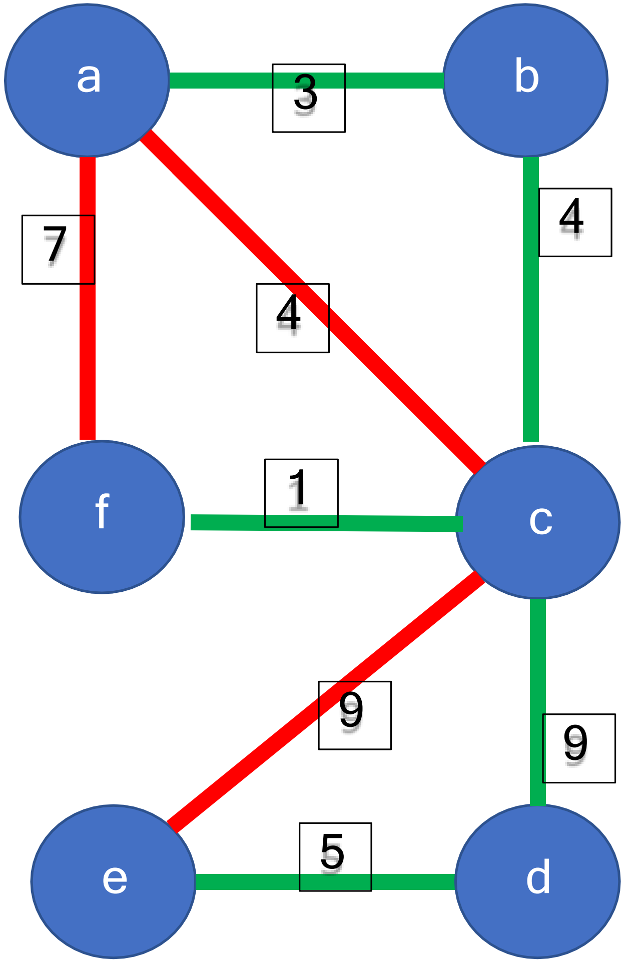
b,c: 4 ✓

a,c: 4 ✗

e,d: 5 ✓

a,f: 7 ✗

c,d: 9 ✓



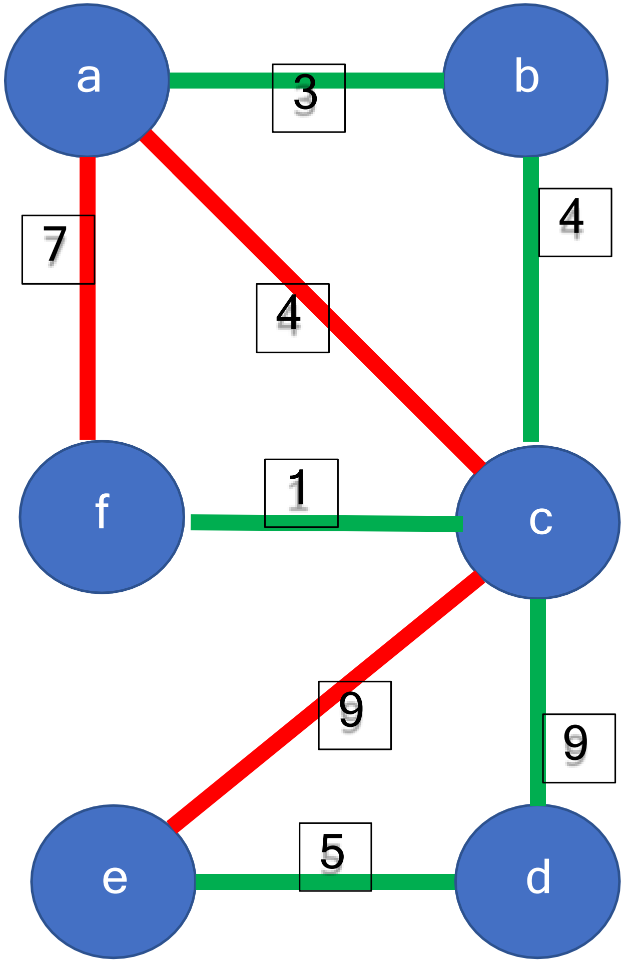
Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
b,c: 4
a,c: 4
e,d: 5
a,f: 7
c,d: 9
c,e: 9

f,c : 1	✓
a,b: 3	✓
b,c: 4	✓
a,c: 4	✗
e,d: 5	✓
a,f: 7	✗
c,d: 9	✓

We stop because the tree obtained covers all the vertices



Kruskal's algorithm (first run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1

a,b: 3

b,c: 4

a,c: 4

e,d: 5

a,f: 7

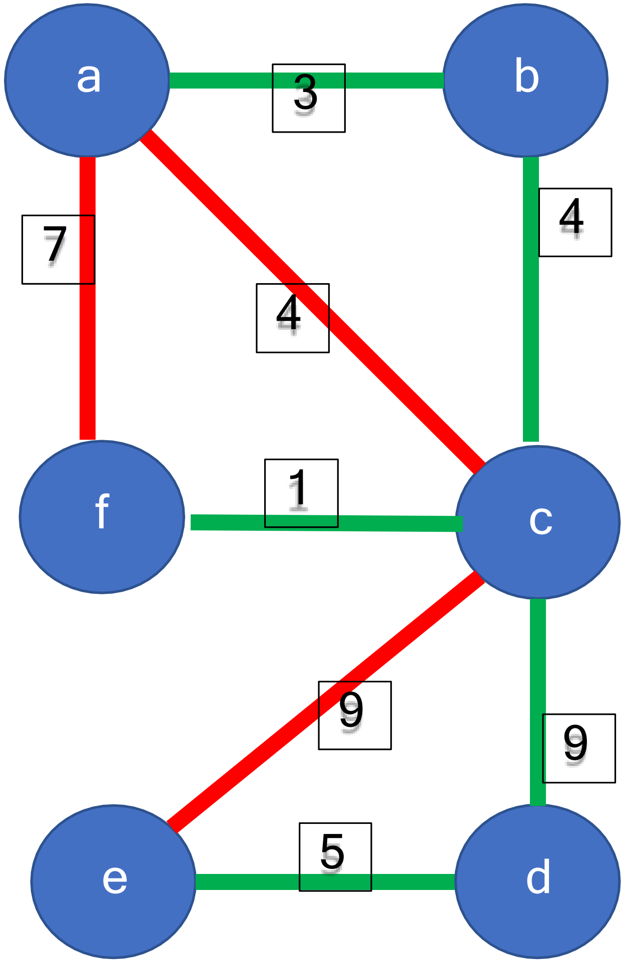
c,d: 9

c,e: 9

f,c : 1	✓
a,b: 3	✓
b,c: 4	✓
a,c: 4	✗
e,d: 5	✓
a,f: 7	✗
c,d: 9	✓

Weight of this tree =

1+3+4+5+9=22



Kruskal's algorithm (first run)

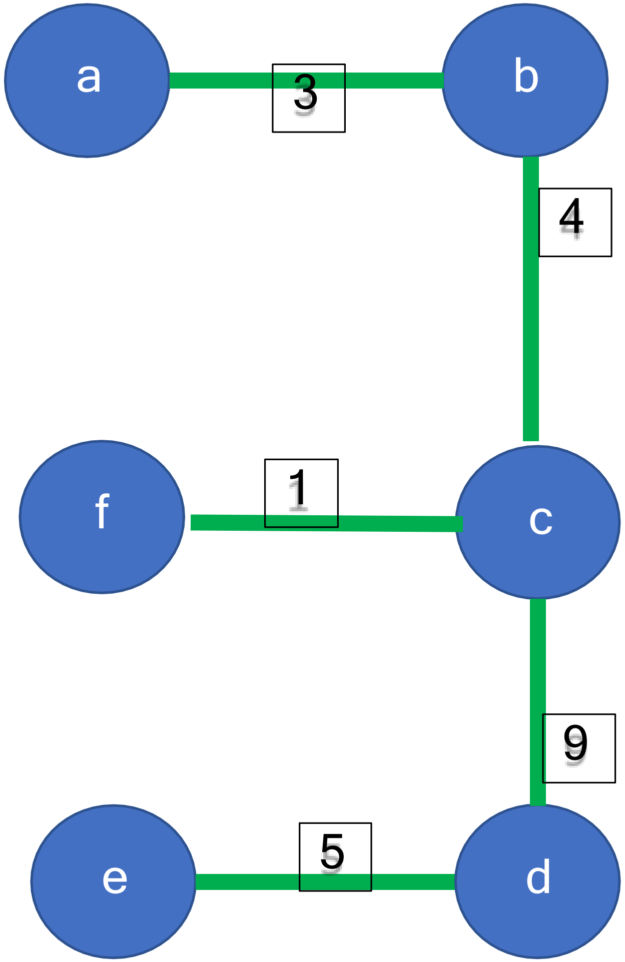
- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
b,c: 4
a,c: 4
e,d: 5
a,f: 7
c,d: 9
c,e: 9

f,c : 1	✓
a,b: 3	✓
b,c: 4	✓
a,c: 4	✗
e,d: 5	✓
a,f: 7	✗
c,d: 9	✓

Weight of this tree = 1+3+4+5+9=22

Connected graph without cycle containing the six original vertices



Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 1: Sort the edges by increasing weight

f,c : 1

a,b: 3

b,c: 4

a,c: 4

e,d: 5

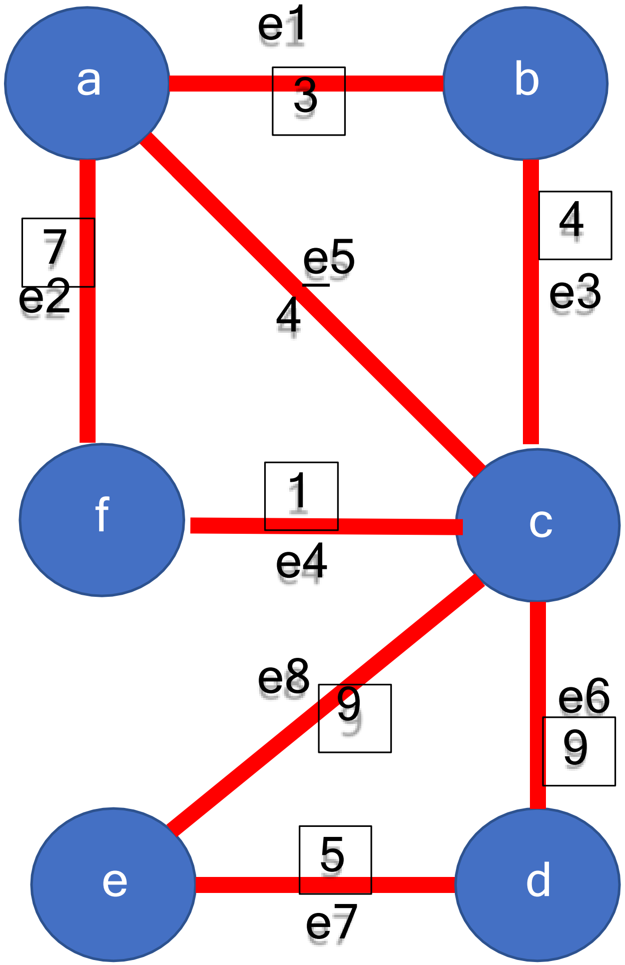
a,f: 7

c,d: 9

c,e: 9

second execution

reverse the order



Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 1: Sort the edges by increasing weight

f,c : 1

a,b: 3

a,c: 4

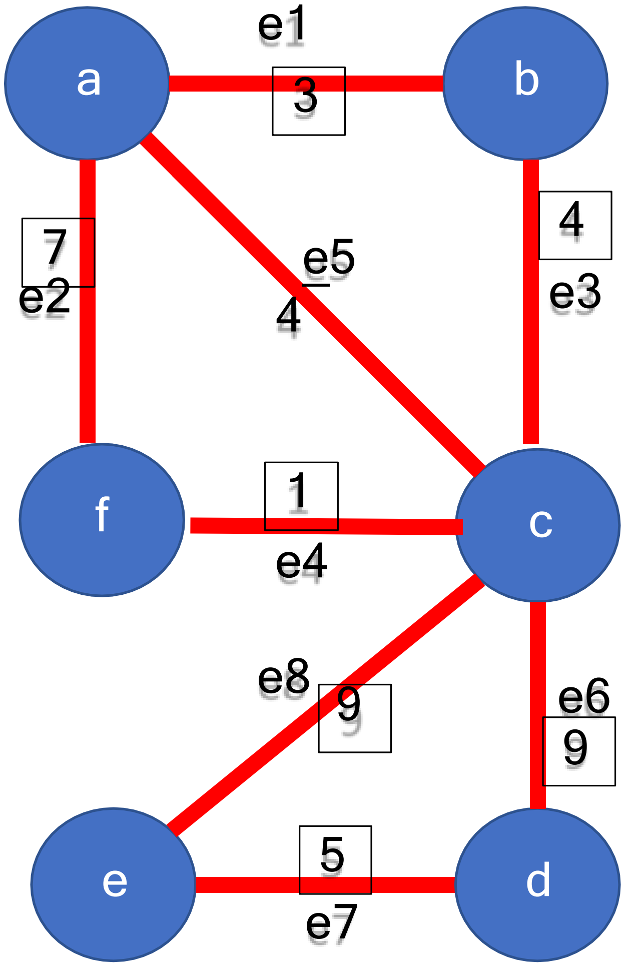
b,c: 4

e,d: 5

a,f: 7

c,e: 9

c,d: 9



Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1

a,b: 3

a,c: 4

b,c: 4

e,d: 5

a,f: 7

c,e: 9

c,d: 9

f,c : 1 ✓

a,b: 3

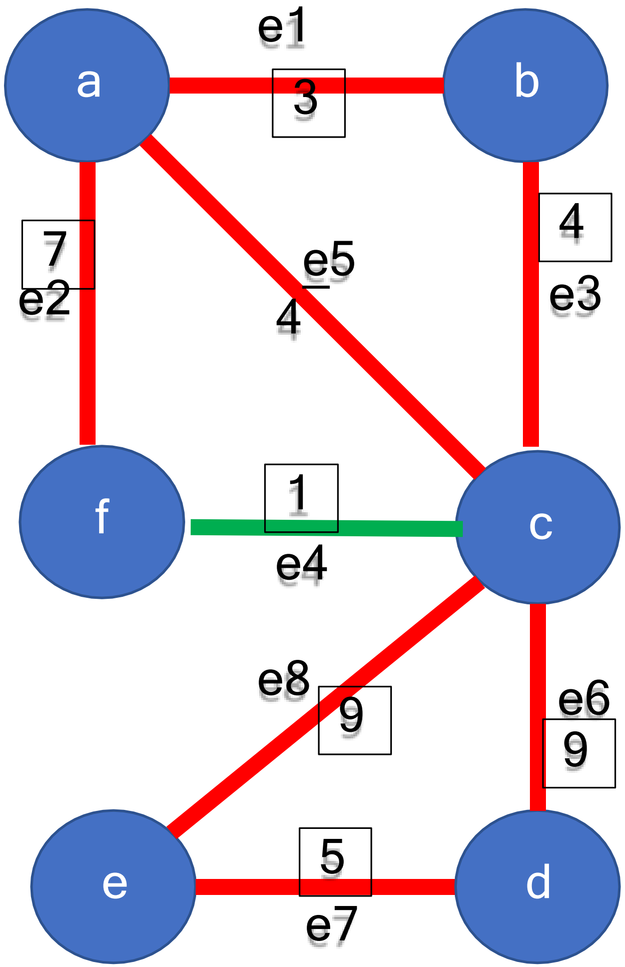
a,c: 4

b,c: 4

e,d: 5

a,f: 7

c,e: 9

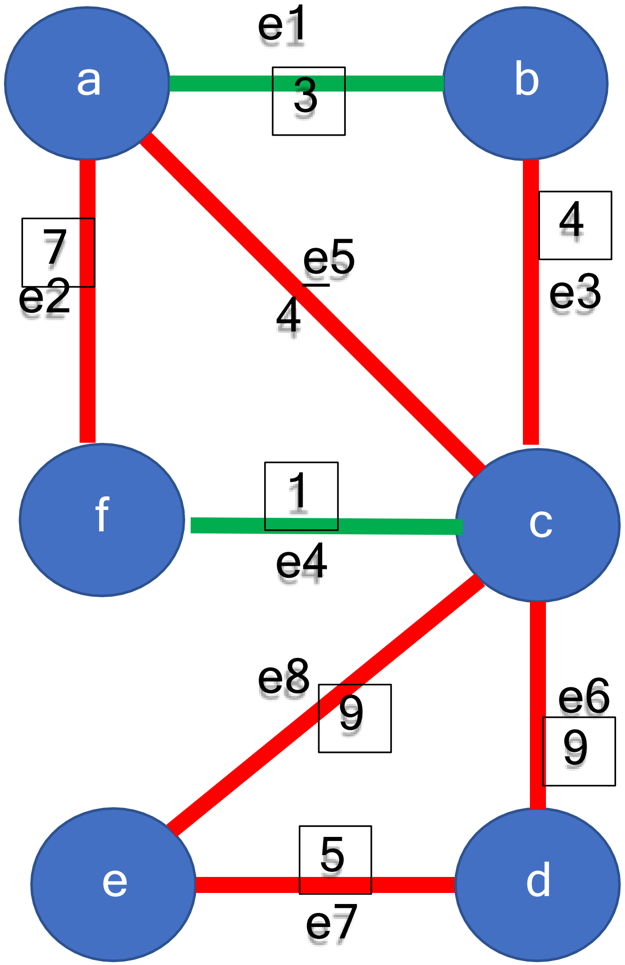


Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
a,c: 4
b,c: 4
e,d: 5
a,f: 7
c,e: 9
c,d: 9

f,c : 1 ✓
a,b: 3 ✓
a,c: 4
b,c: 4
e,d: 5
a,f: 7
c,e: 9

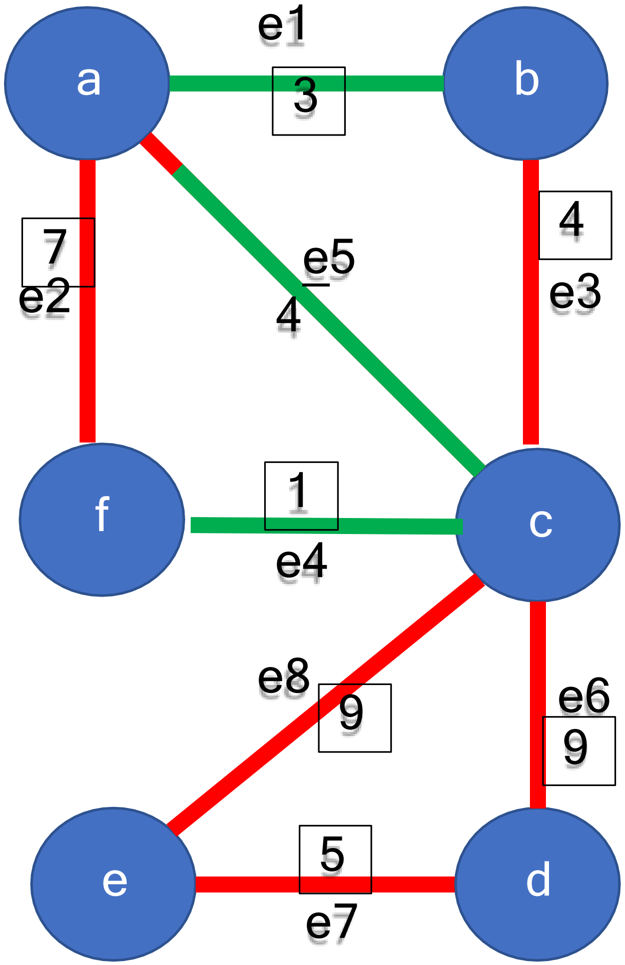


Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
a,c: 4
b,c: 4
e,d: 5
a,f: 7
c,e: 9
c,d: 9

f,c : 1 ✓
a,b: 3 ✓
a,c: 4 ✓
b,c: 4
e,d: 5
a,f: 7
c,e: 9

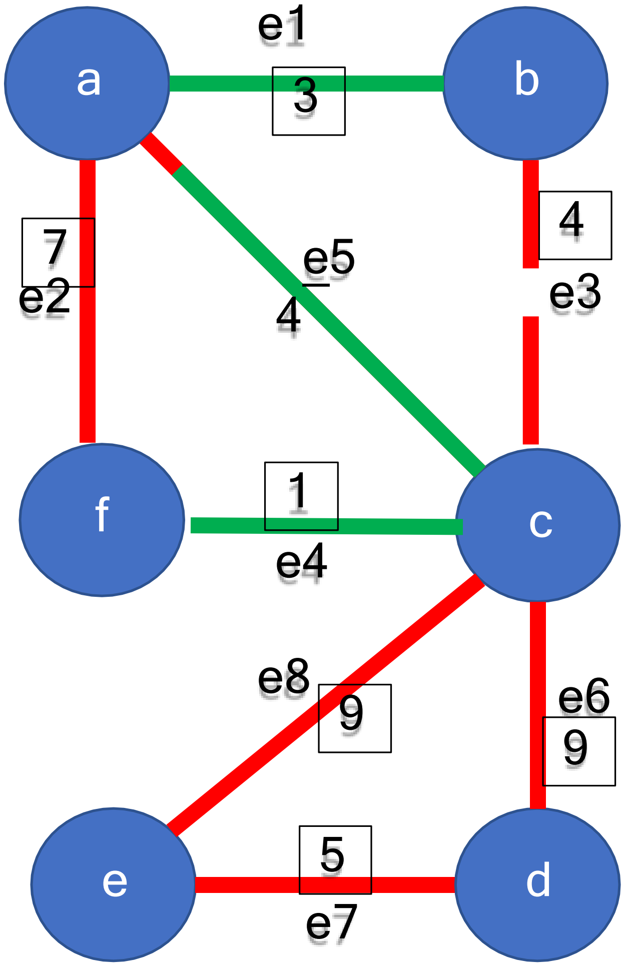


Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
a,c: 4
b,c: 4
e,d: 5
a,f: 7
c,e: 9
c,d: 9

f,c : 1 ✓
a,b: 3 ✓
a,c: 4 ✓
b,c: 4 ✗
e,d: 5
a,f: 7
c,e: 9



Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1

a,b: 3

a,c: 4

b,c: 4

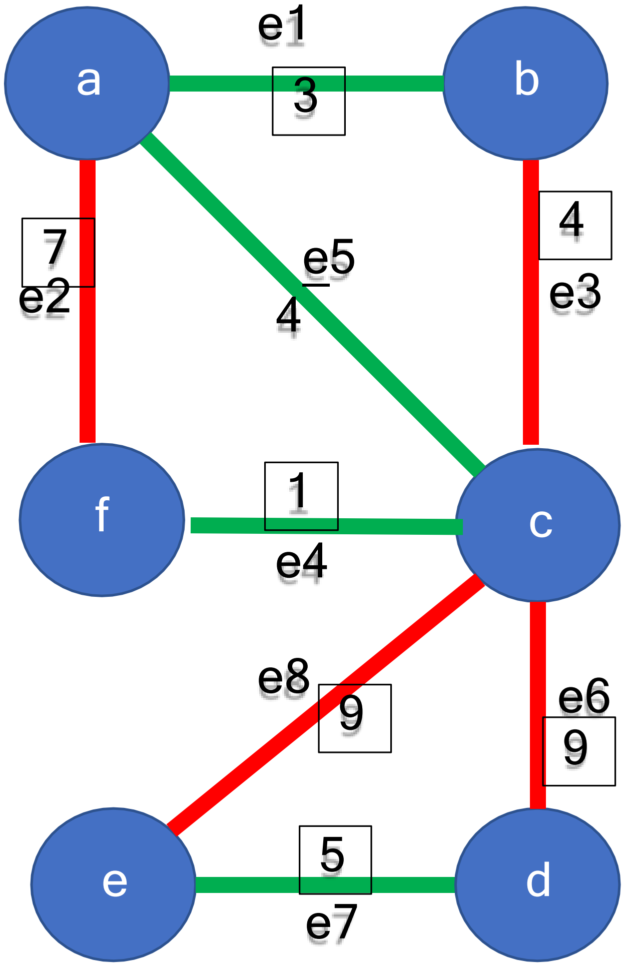
e,d: 5

a,f: 7

c,e: 9

c,d: 9

f,c : 1	✓
a,b: 3	✓
a,c: 4	✓
b,c: 4	✗
e,d: 5	✓
a,f: 7	
c,e: 9	

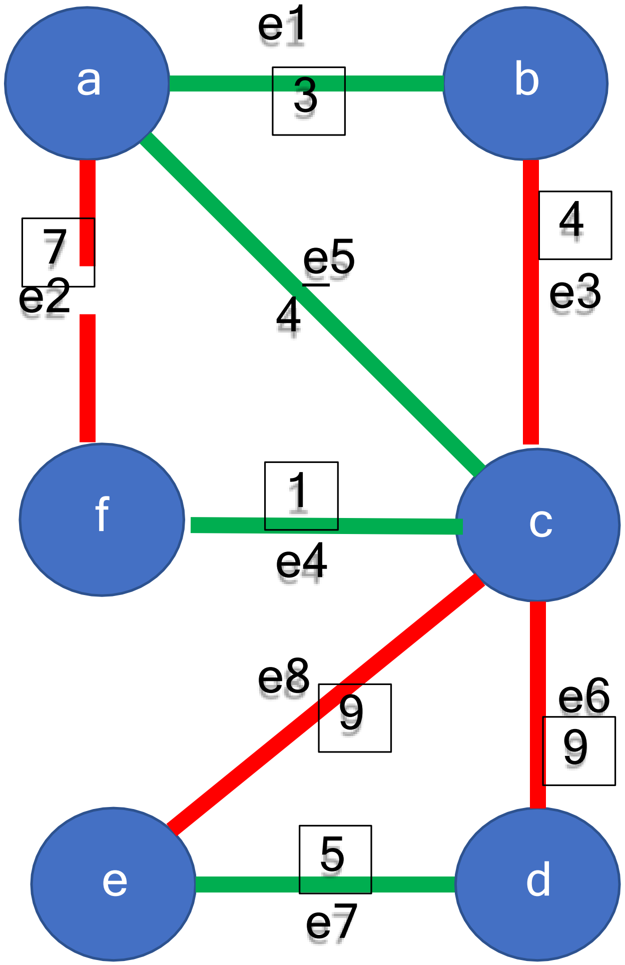


Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
a,c: 4
b,c: 4
e,d: 5
a,f: 7
c,e: 9
c,d: 9

f,c : 1	✓
a,b: 3	✓
a,c: 4	✓
b,c: 4	✗
e,d: 5	✓
a,f: 7	✗
c,e: 9	



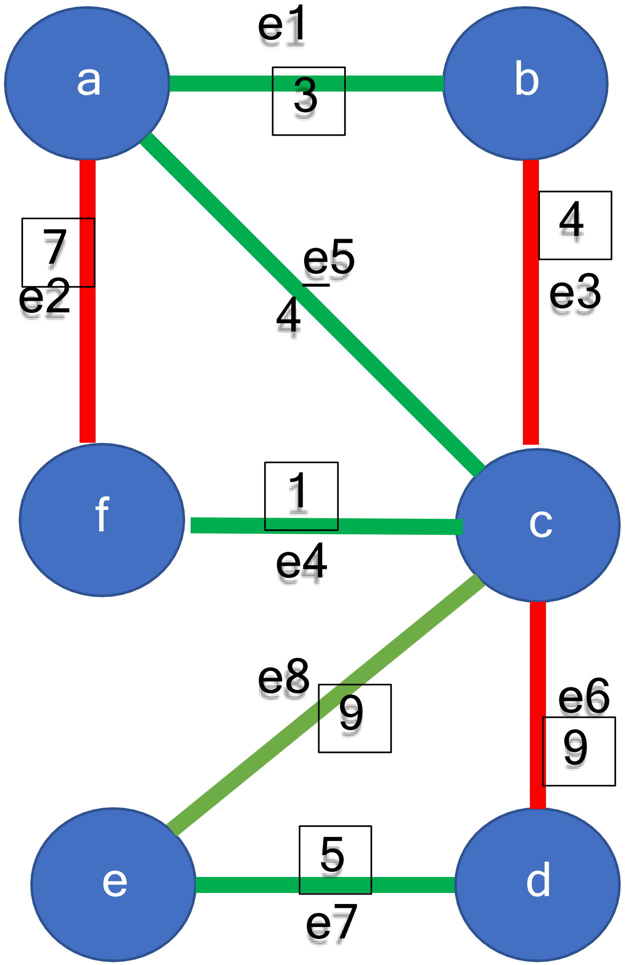
Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1
a,b: 3
a,c: 4
b,c: 4
e,d: 5
a,f: 7
c,e: 9
c,d: 9

f,c : 1	✓
a,b: 3	✓
a,c: 4	✓
b,c: 4	✗
e,d: 5	✓
a,f: 7	✗
c,e: 9	✓

We stop because the tree covers all the vertices



Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1

a,b: 3

a,c: 4

b,c: 4

e,d: 5

a,f: 7

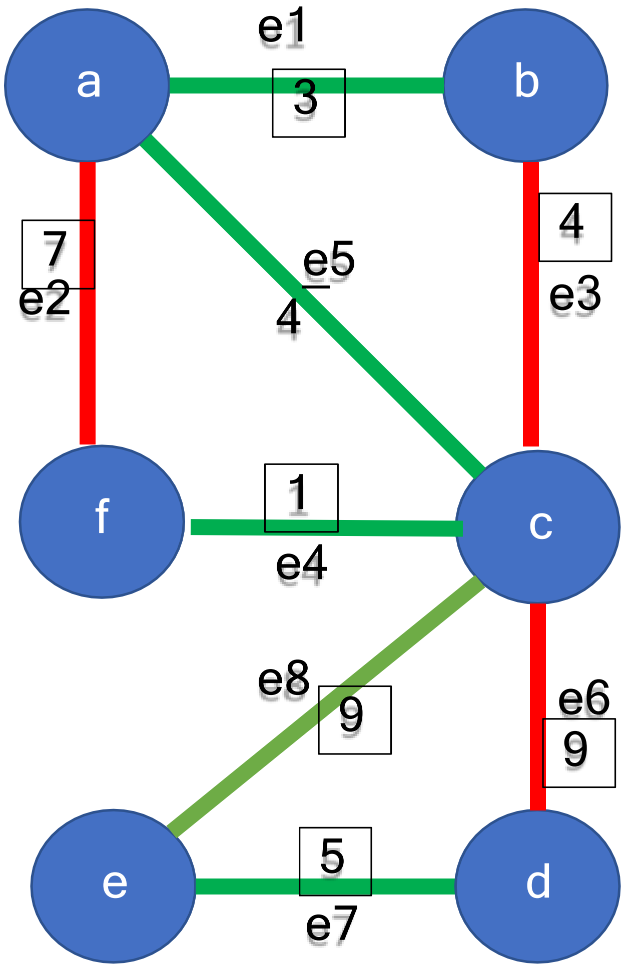
c,e: 9

c,d: 9

f,c : 1	✓
a,b: 3	✓
a,c: 4	✓
b,c: 4	✗
e,d: 5	✓
a,f: 7	✗
c,e: 9	✓

Weight of spanning tree

=1+3+4+5+9 =22



Kruskal's algorithm (second run)

- Building a minimal weight spanning tree
  - Step 2: Add edges one by one if no cycle creation

f,c : 1

a,b: 3

a,c: 4

b,c: 4

e,d: 5

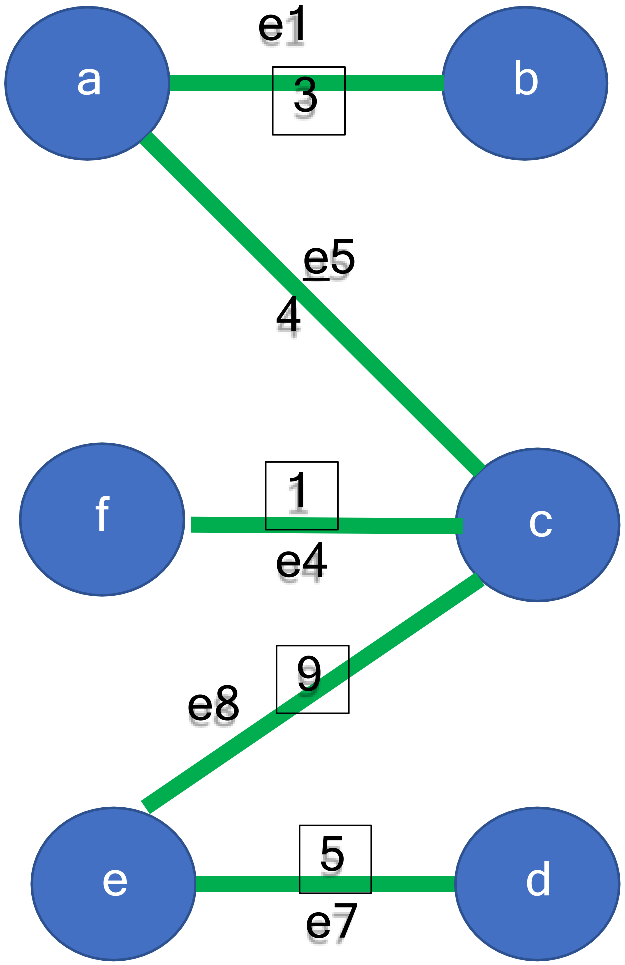
a,f: 7

c,e: 9

c,d: 9

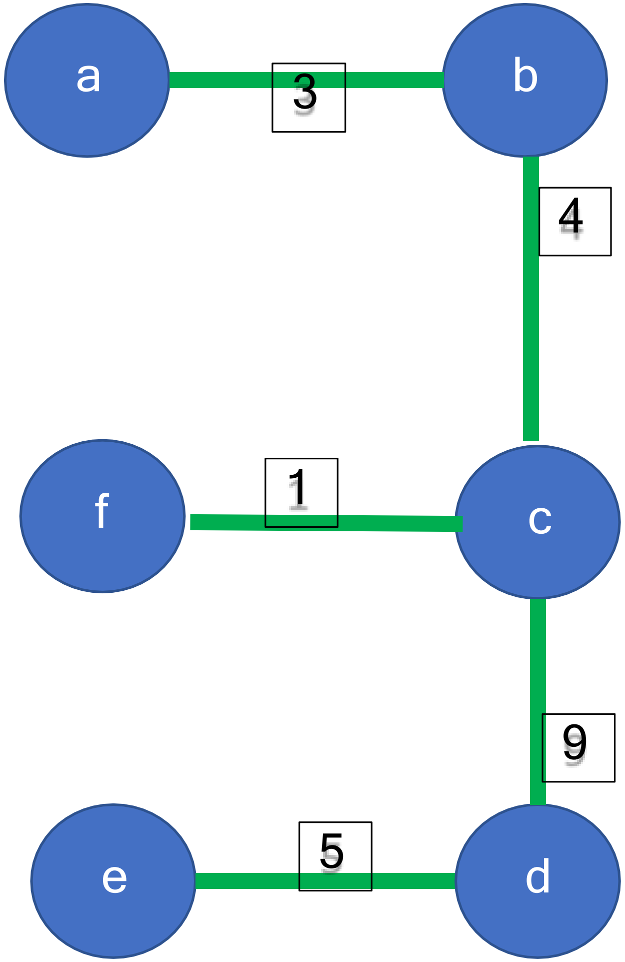
Weight of spanning tree =1+3+4+5+9=22

Connected graph without cycle containing the six original vertices



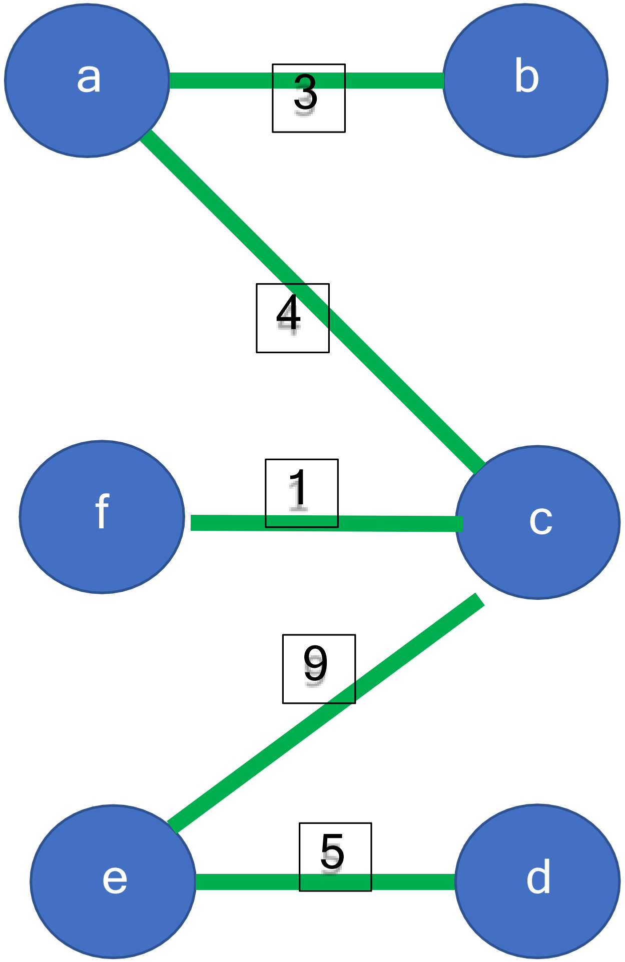
Kruskal's algorithm (compare the two executions)

Connected graph without cycle containing the six original vertices



Weight of spanning tree = 1+3+4+5+9=22

Connected graph without cycle containing the six original vertices

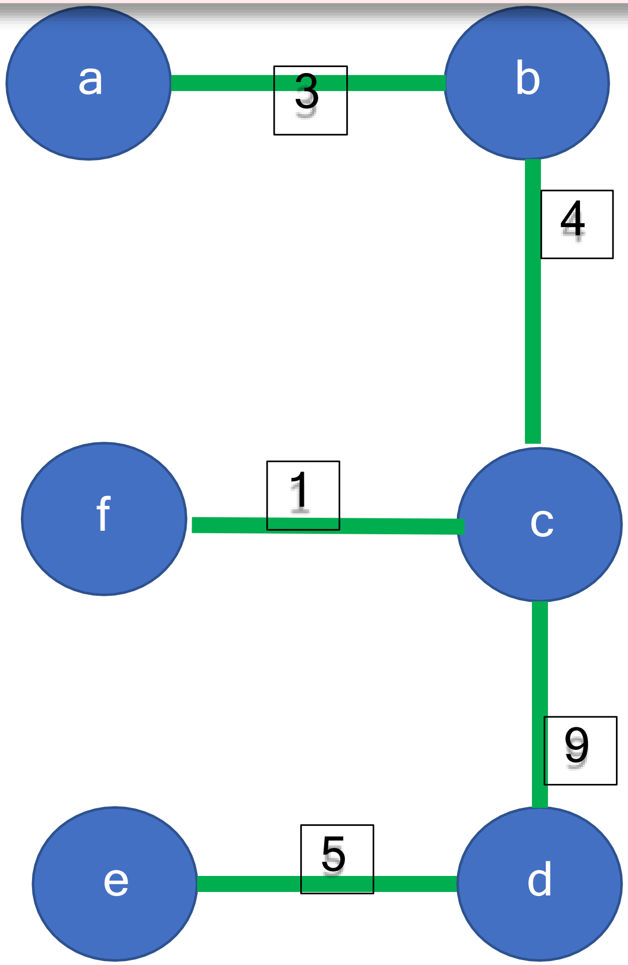


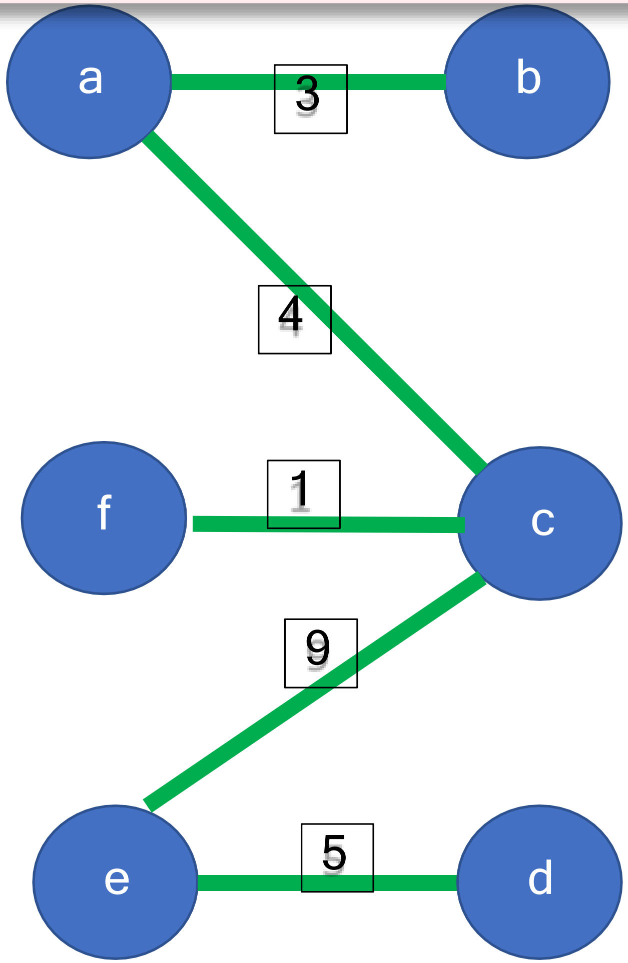
Weight of spanning tree =1+3+4+5+9=22

Kruskal's algorithm (compare the two executions)

Conclusion

The weight of the spanning tree is always the same but the spanning trees are different.



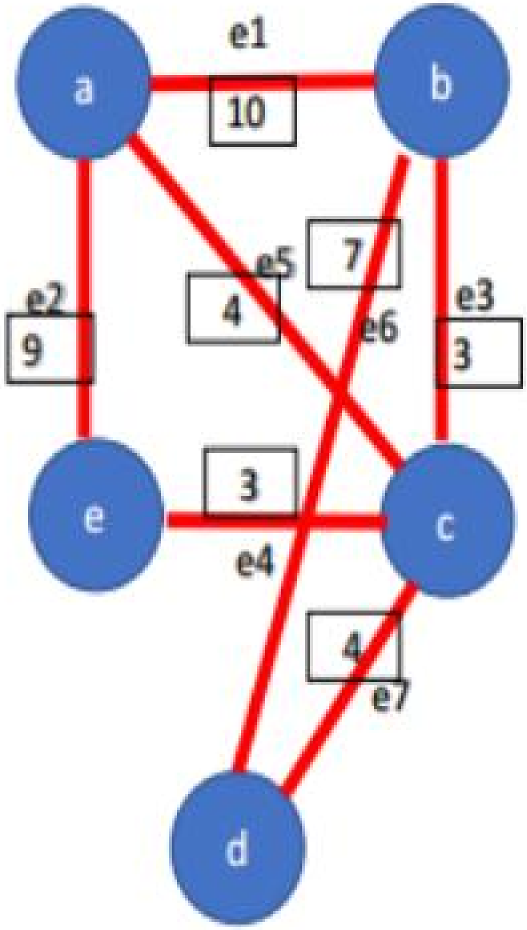


Kruskal's algorithm (exercise)

- Building a minimal weight spanning tree

Example:

A connected and weighted graph G = (V,E) defined by the finite set of vertices V = {a,b,c,d,e} and by the finite set of valued edges E = {e1, e2,e3,e4,e5,e6,e7} with the respective weights {10,9,3,3,4,7,4}



Principle of Prim's algorithm

- Choose any vertex.
- Choose the edge of minimum weight among the edges that come out of the chosen vertex.
- Add vertices and their incident edges to the spanning tree if no cycle creation.
- Stop if the number of vertices in the spanning tree reaches the number of vertices in the original tree.

Prim maintains an acyclic tree at each vertex addition

...

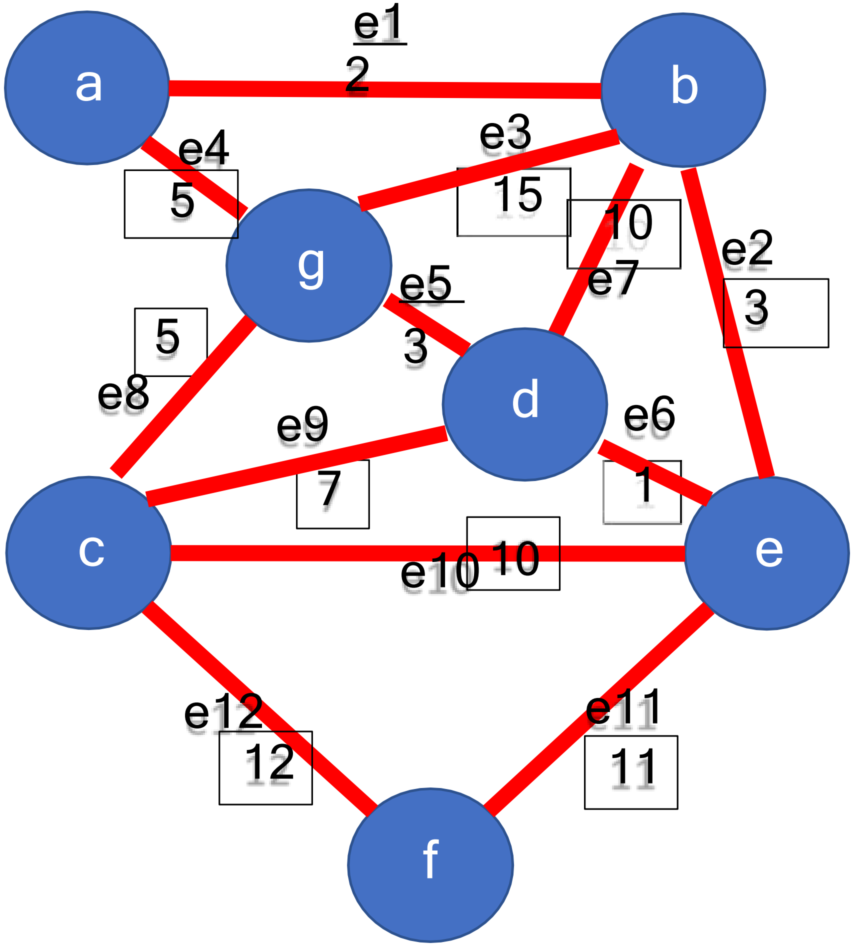
Prim's Algorithm

- Building a minimal weight spanning tree

given:

A connected and weighted graph G = (V,E) defined by the finite set of vertices V = {a,b,c,d,e,f,g} and by the finite set of valued edges E = {e1,

e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12} with the respective weights {2,3,15,5,3,1,10,5,7,10,11,12}



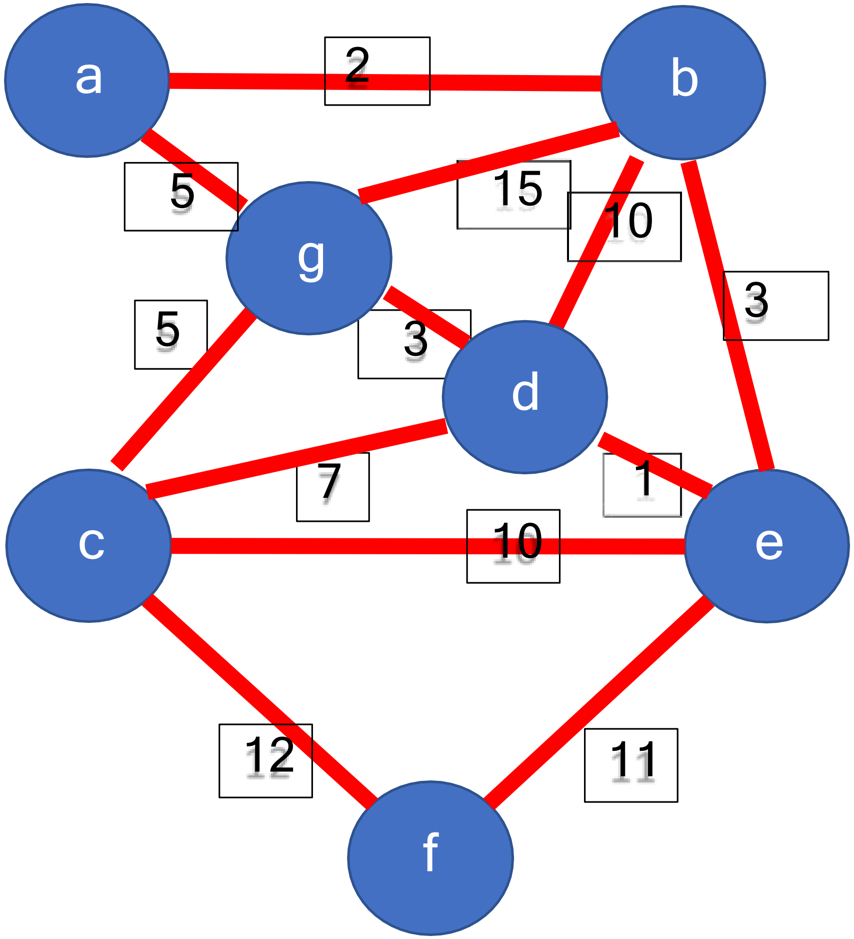
Prim's Algorithm

- Building a minimal weight spanning tree

given:

A connected and weighted graph G = (V,E) defined by the finite set of vertices V = {a,b,c,d,e,f,g} and by the finite set of valued edges E = {e1,

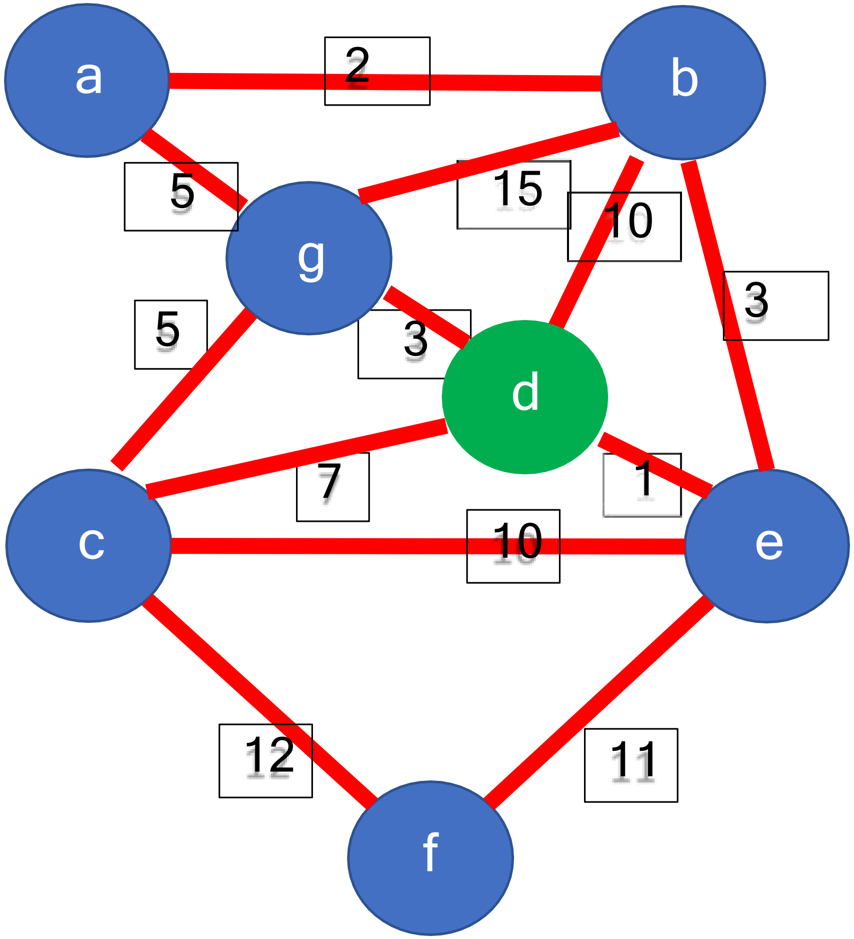
e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12} with the respective weights {2,3,15,5,3,1,10,5,7,10,11,12}



Prim's Algorithm

- Building a minimal weight spanning tree

Step 1:Choose any vertex
We chose the vertex «d»

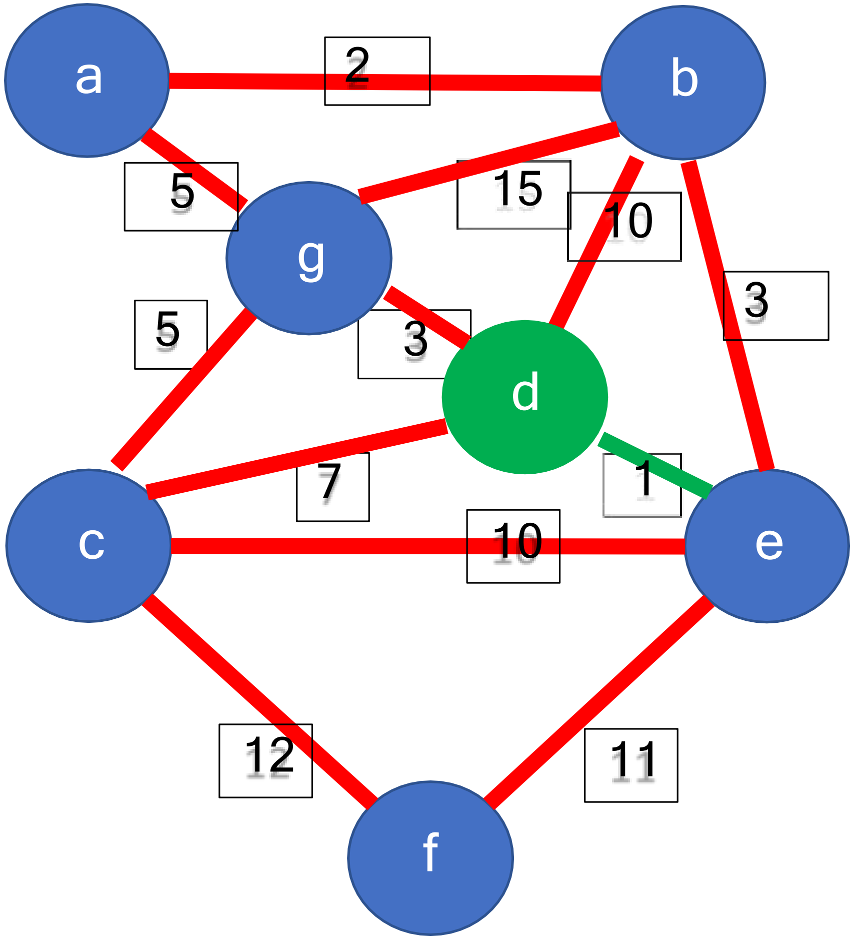


Prim's Algorithm

- Building a minimal weight spanning tree

Step 2:
Choose the minimum weight edge among the edges that come out of the chosen vertex

We choose the edge with weight = 1

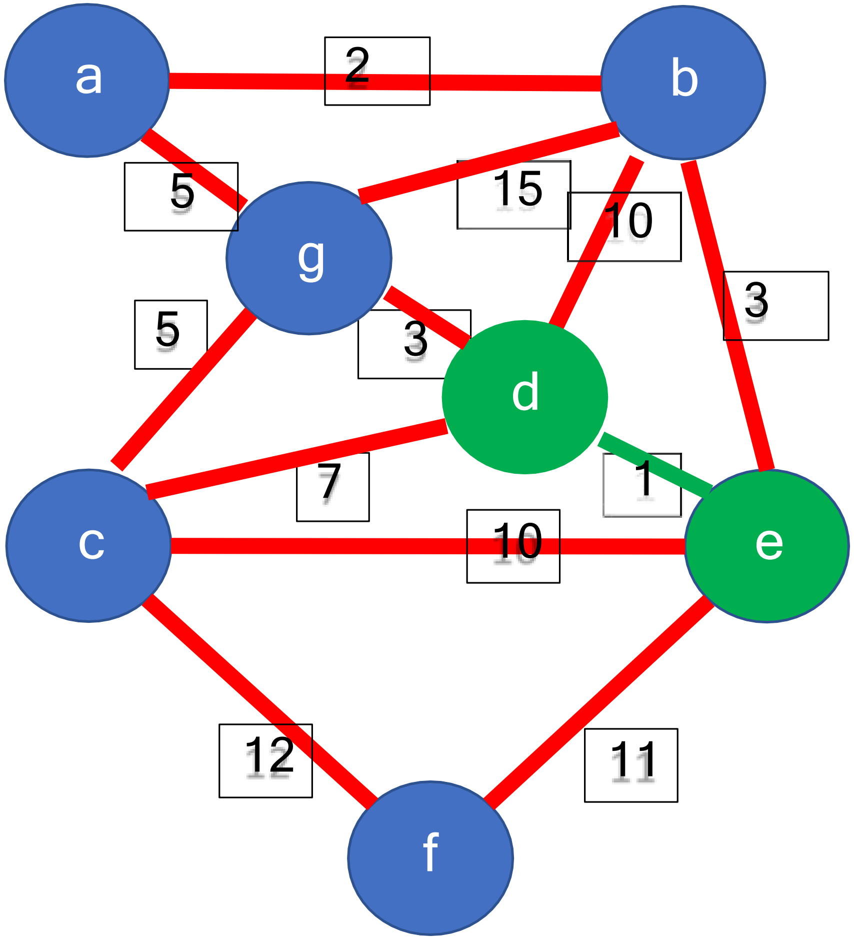


Prim's Algorithm

- Building a minimal weight spanning tree

Step 3:
Add the incident vertex to the chosen edge

We add the vertex **e**

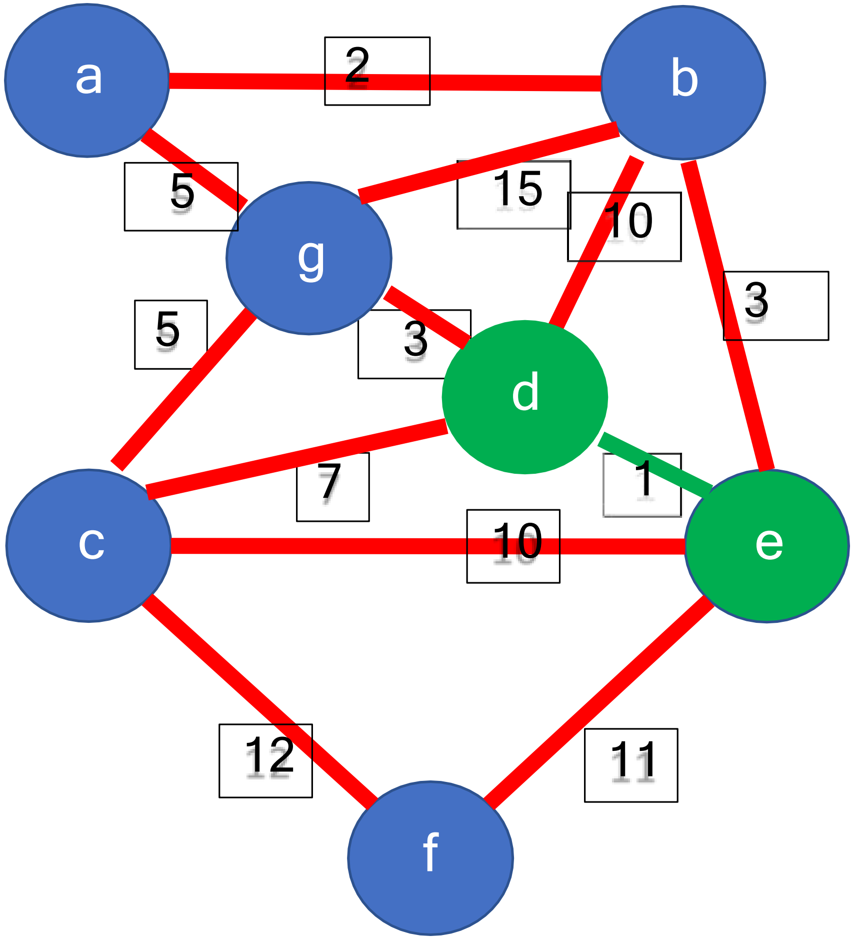


Prim's Algorithm

- Building a minimal weight spanning tree

Step 3:
Add the incident vertex to the chosen edge

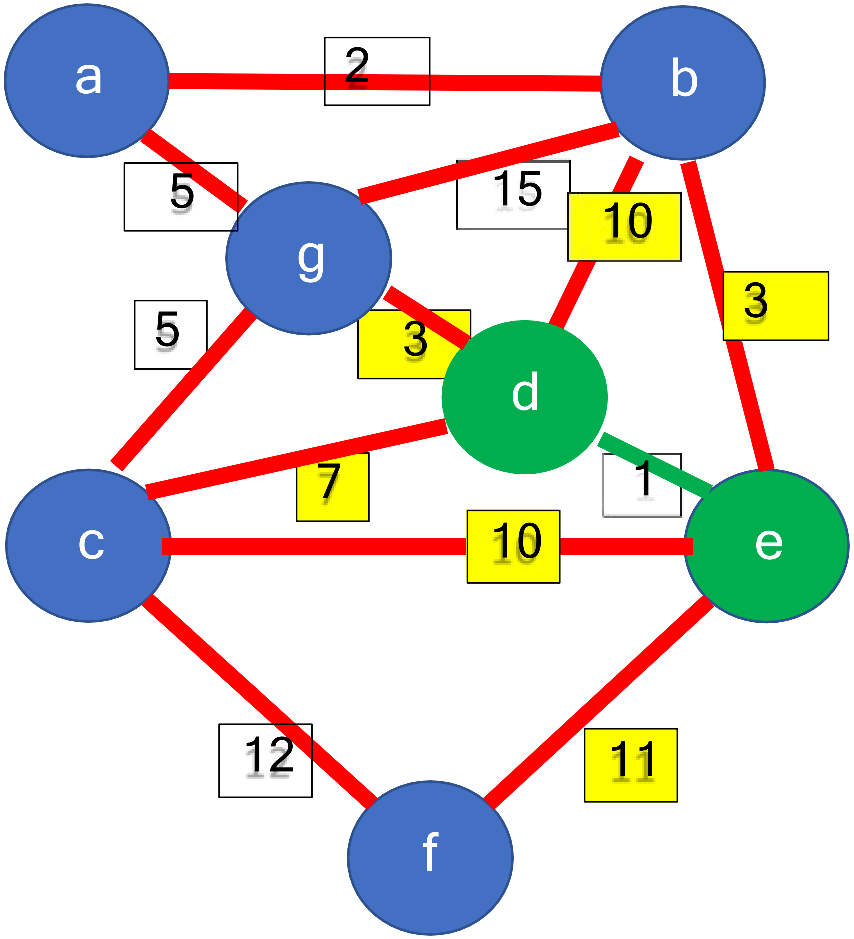
We obtain a partial graph containing two vertices **d** and **e**



Prim's Algorithm

- Building a minimal weight spanning tree

Step 4:
Consider all edges incident to the resulting tree and choose the one with the minimum weight

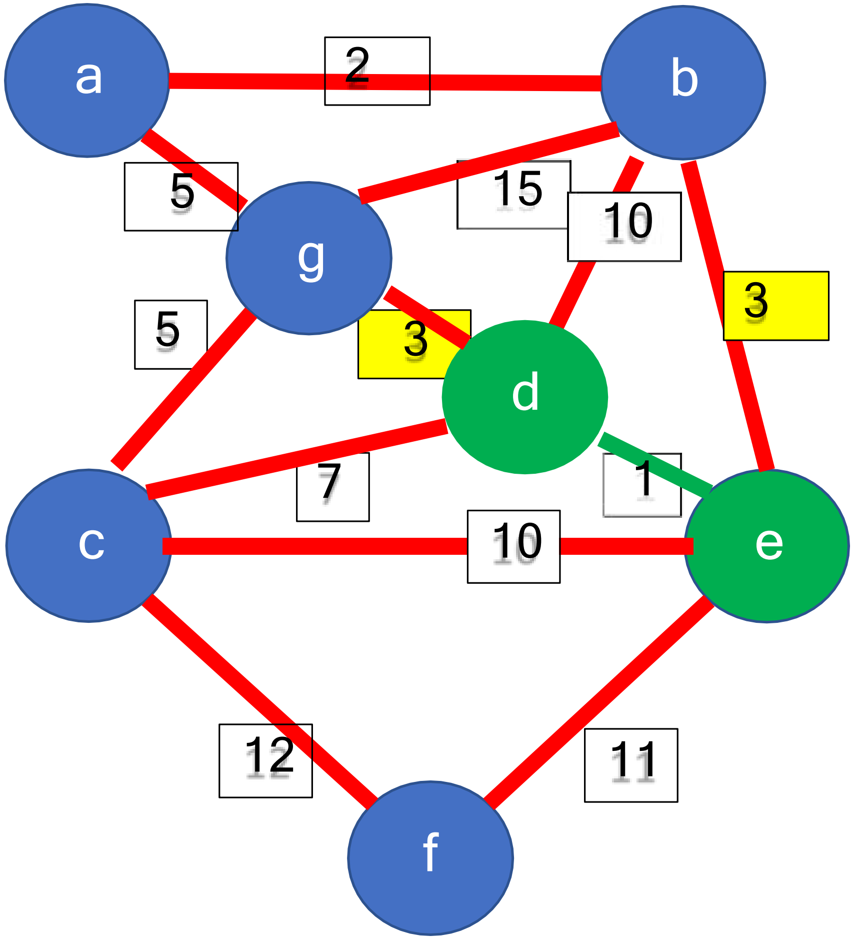


Prim's Algorithm

- Building a minimal weight spanning tree

Step 4:
Consider all edges incident to the resulting tree and choose the one with the minimum weight

There are two edges of minimum weight **be** and **gd**, we choose the one we want.

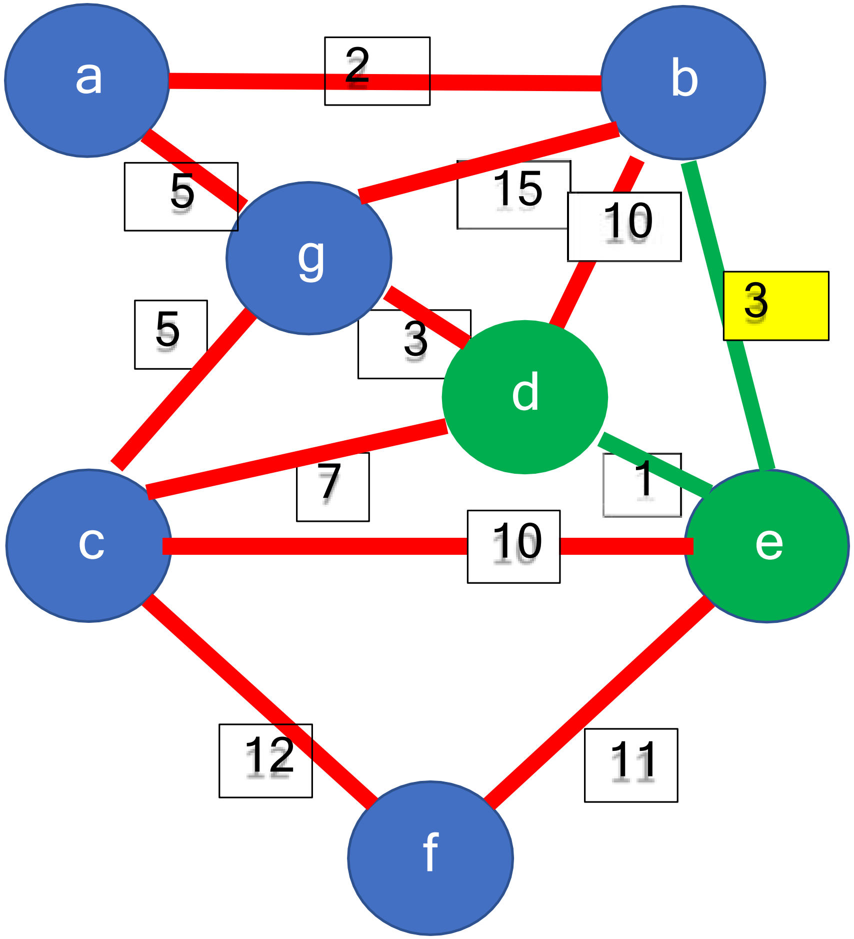


Prim's Algorithm

- Building a minimal weight spanning tree

Step 4:
Consider all edges incident to the resulting tree and choose the one with the minimum weight

For example, we choose the edge b e.



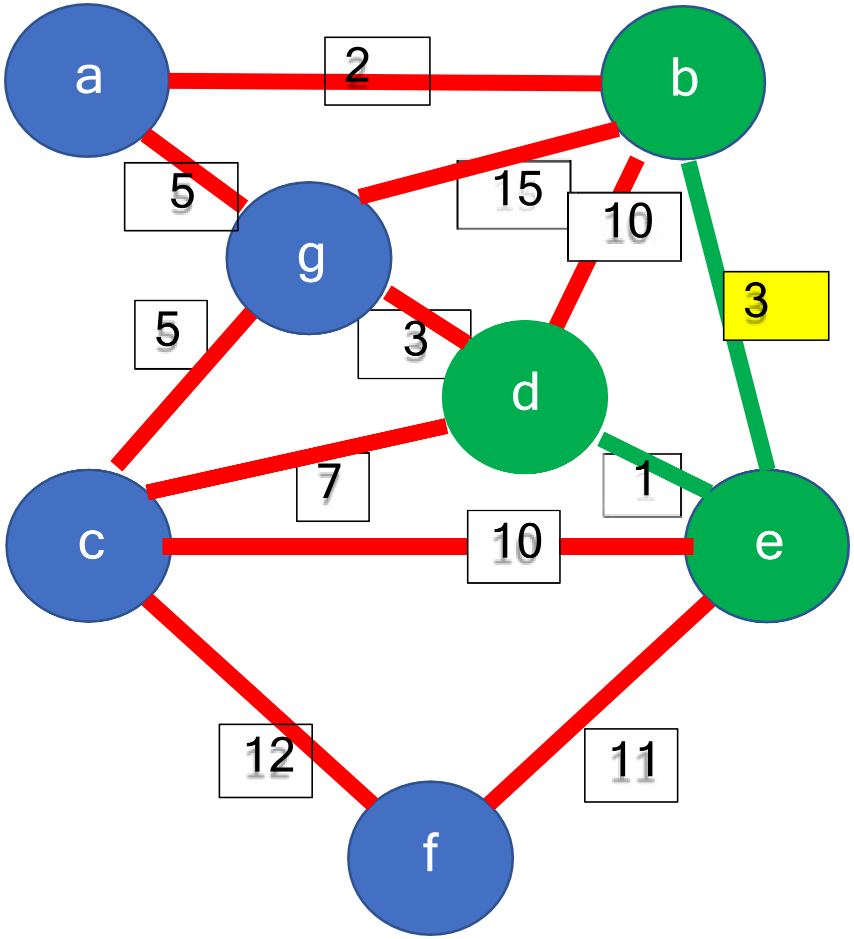
Prim's Algorithm

- Building a minimal weight spanning tree

Step 5:

Add the incident vertex to the chosen edge

We add vertex b.



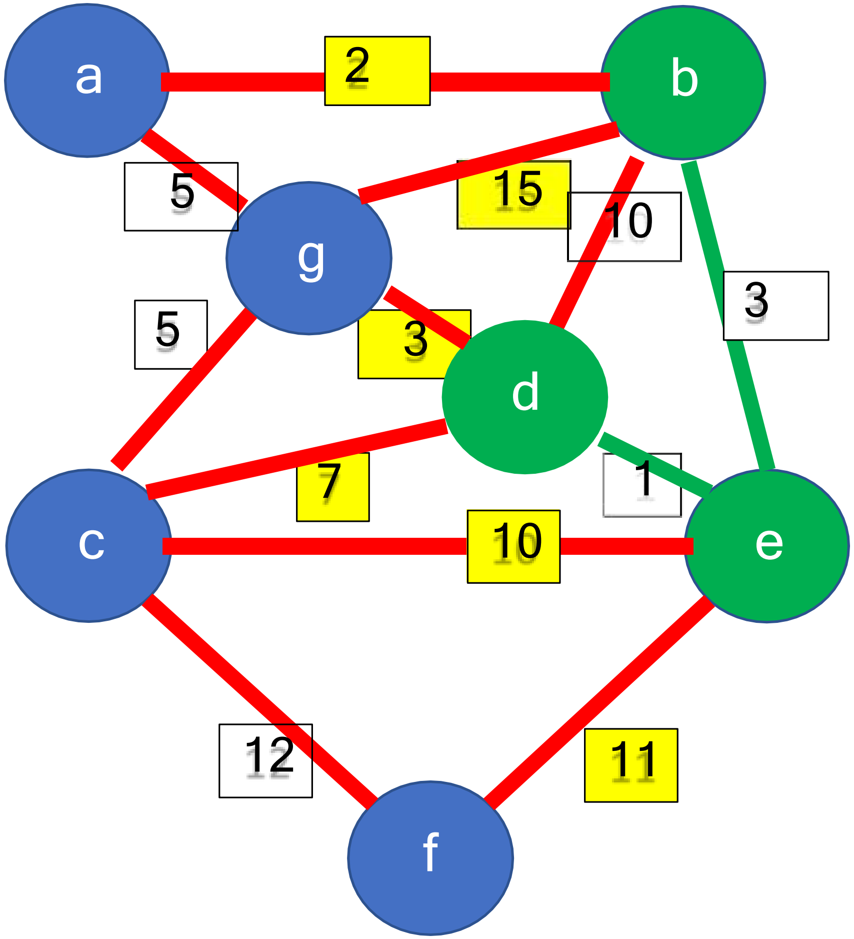
Prim's Algorithm

- Building a minimal weight spanning tree

Step 6: Repeat step 4 and 5 Consider all edges incident to the resulting tree that have one vertex in the tree and one vertex outside the tree, then choose the one with the minimum weight

Edge **bd** is not in the list of edges incident to the resulting tree because its two vertices are inside the tree.

We choose the edge **ab** of weight 2.



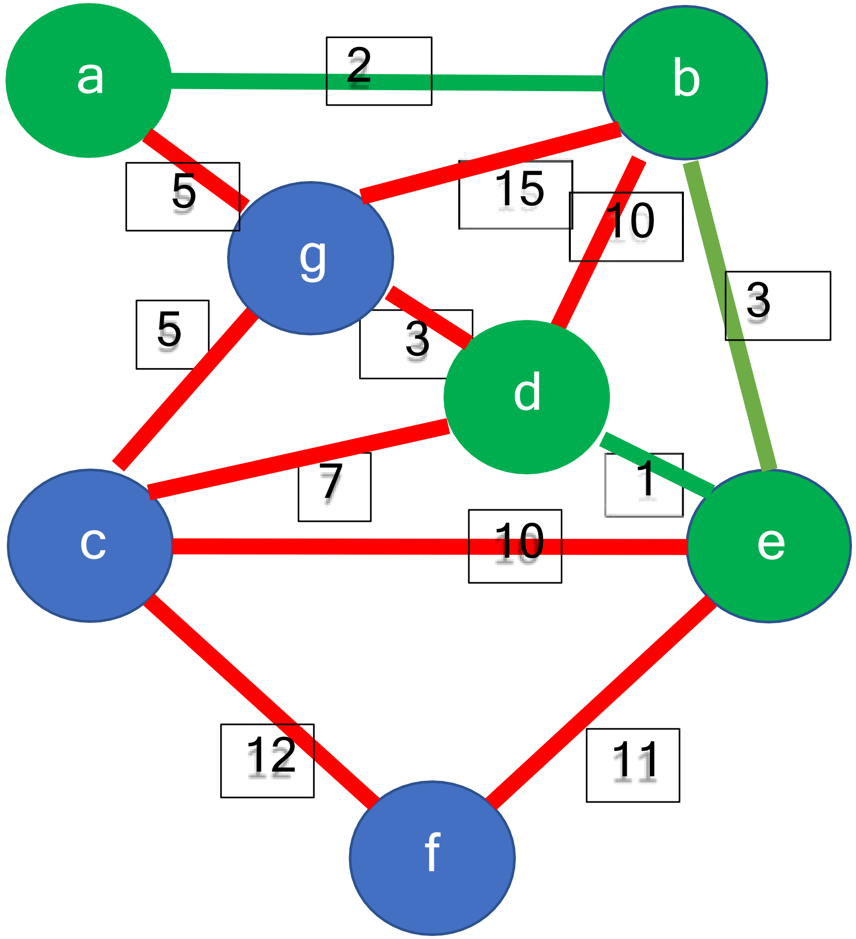
Prim's Algorithm

- Building a minimal weight spanning tree

Step 6:

Add the incident vertex to the chosen edge

We obtain a partial graph containing four vertices **d, e, b and a**

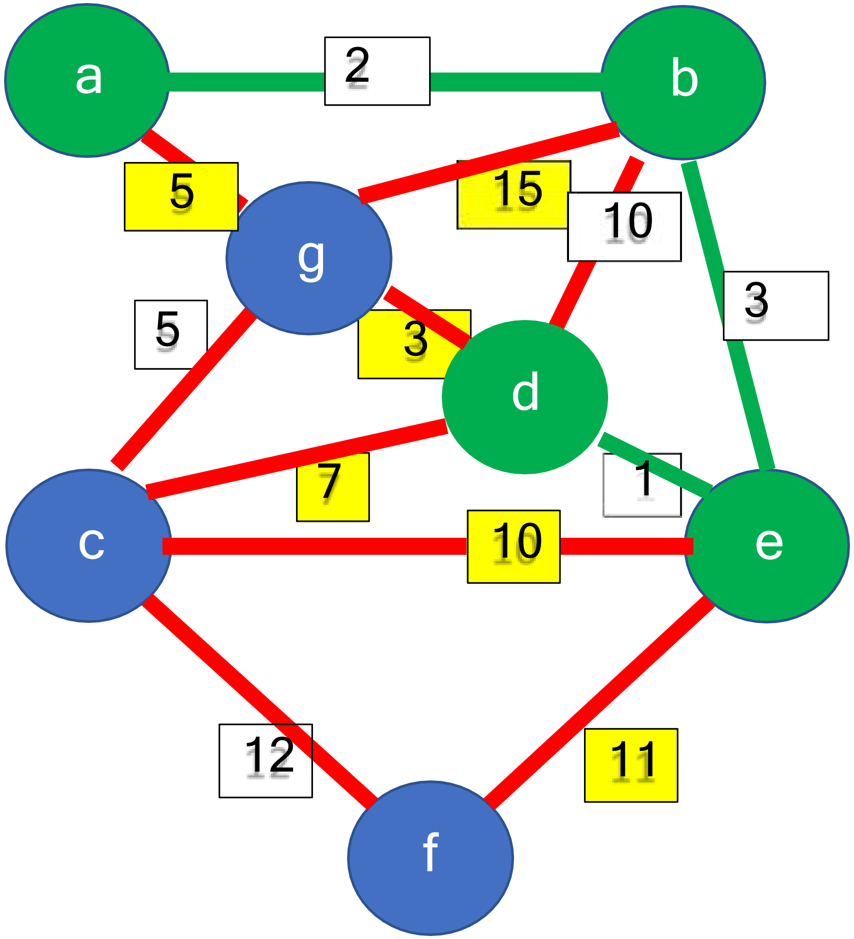


Prim's Algorithm

- Building a minimal weight spanning tree

Step 6: Repeat step 4 and 5
Consider all edges incident to the resulting tree that have one vertex in the tree and one vertex outside the tree, then choose the one with the minimum weight

We choose the edge **gd** of weight 3 and add its incident vertex to the resulting tree.

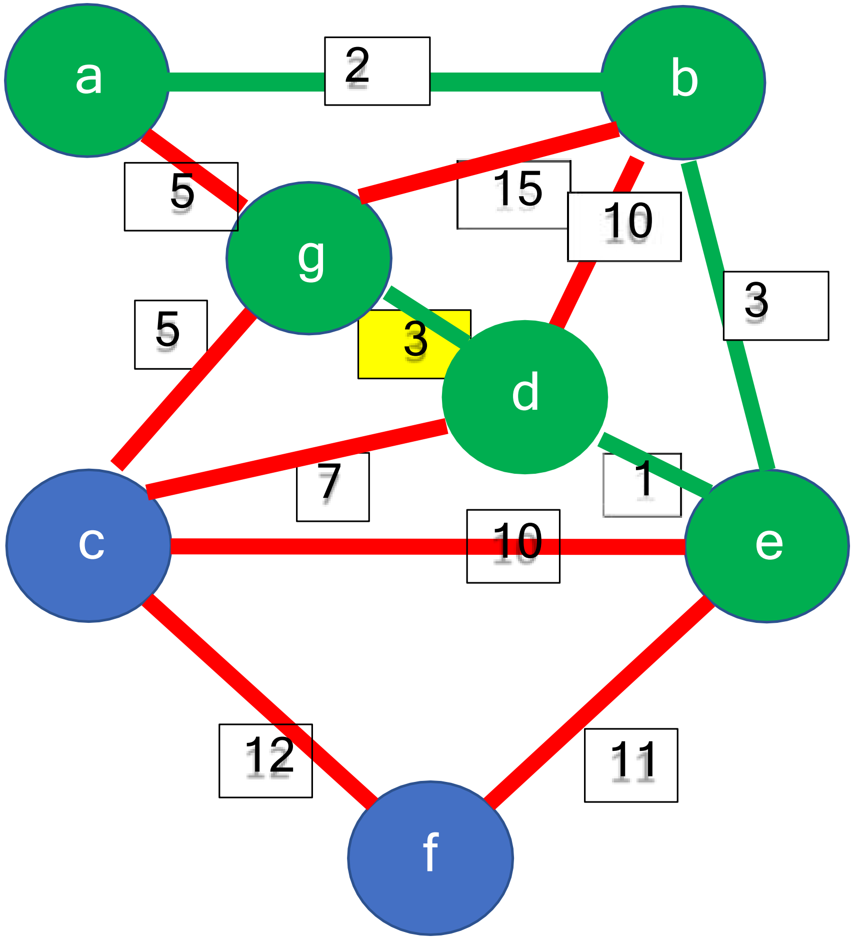


Prim's Algorithm

- Building a minimal weight spanning tree

Step 6: Repeat step 4 and 5
Consider all edges incident to the resulting tree that have one vertex in the tree and one vertex outside the tree, then choose the one with the minimum weight

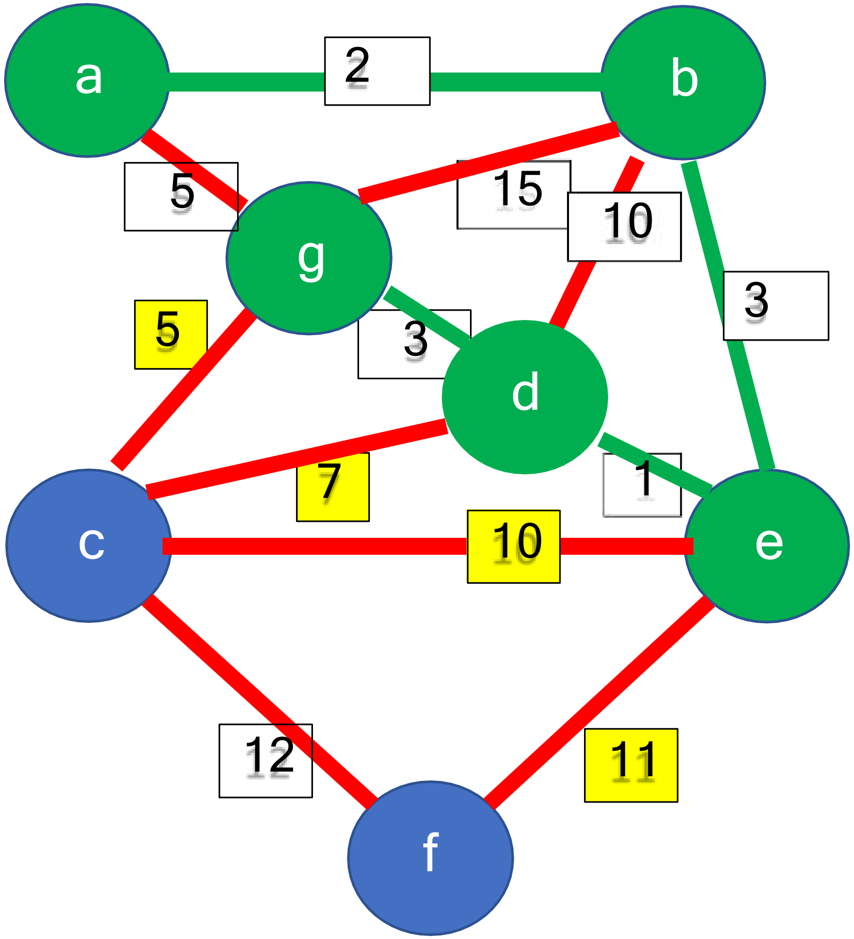
We choose the edge **gd** of weight 3 and add its incident vertex to the resulting tree.



Prim's Algorithm

- Building a minimal weight spanning tree

Step 6: Repeat step 4 and 5 Consider all edges incident to the resulting tree that have one vertex in the tree and one vertex outside the tree, then choose the one with the minimum weight

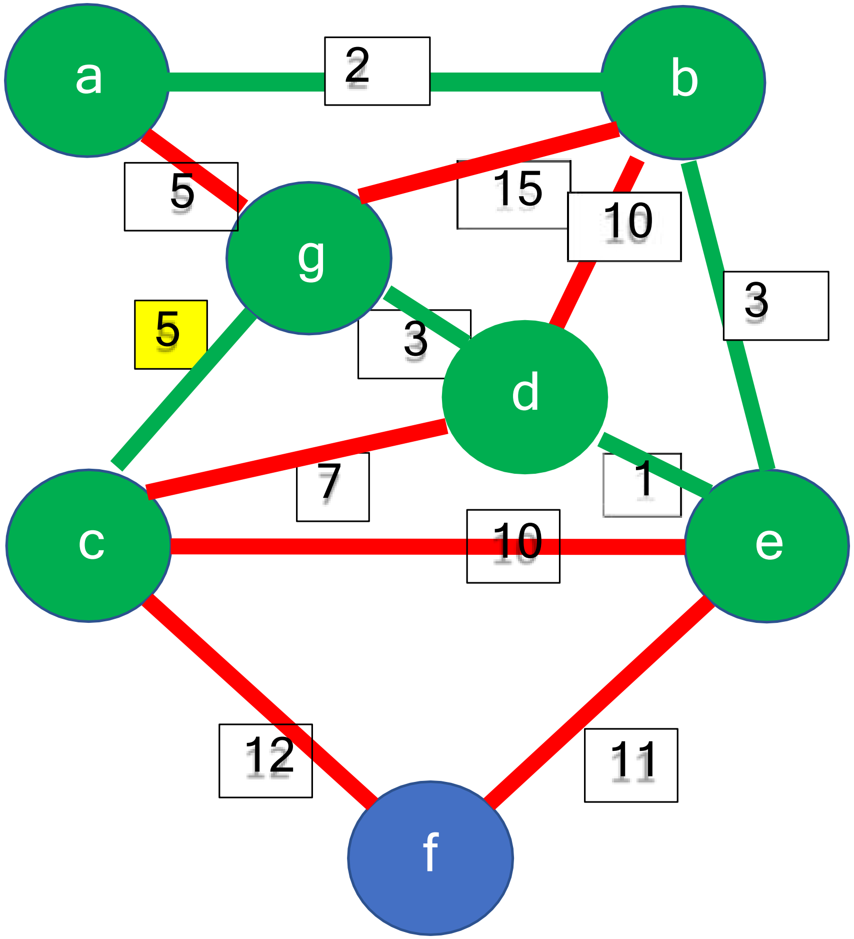


Prim's Algorithm

- Building a minimal weight spanning tree

Step 6: Repeat step 4 and 5 Consider all edges incident to the resulting tree that have one vertex in the tree and one vertex outside the tree, then choose the one with the minimum weight

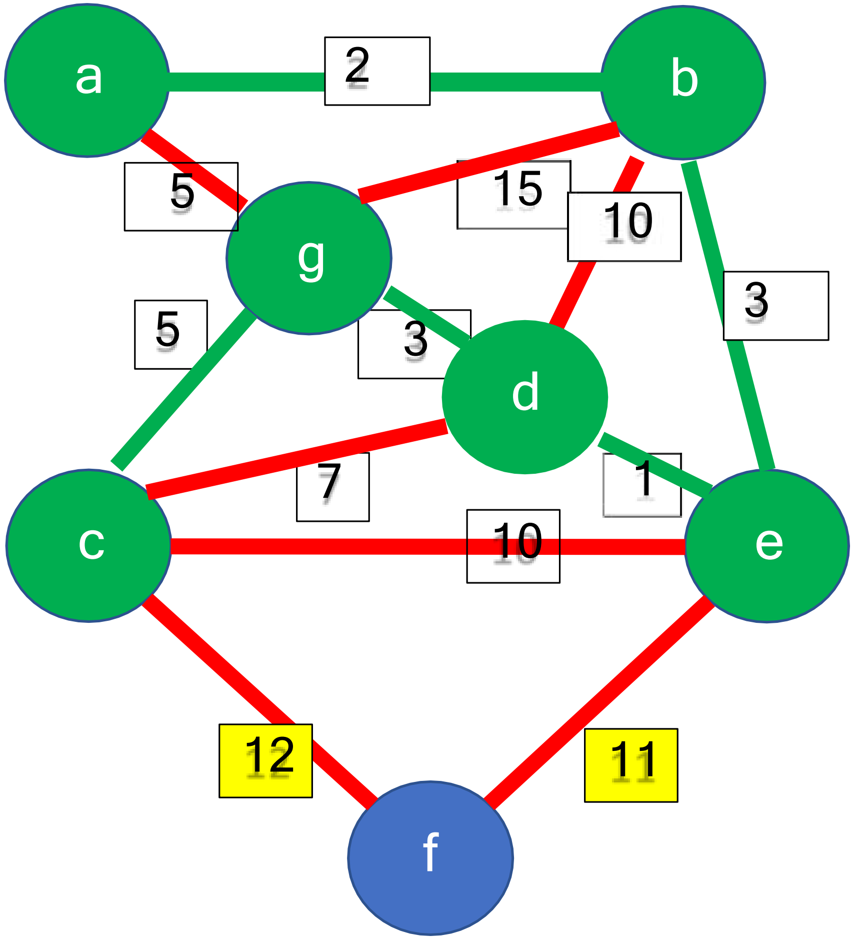
We choose the edge **gc** of weight 5 and add its incident vertex to the resulting tree.



Prim's Algorithm

- Building a minimal weight spanning tree

Step 6: Repeat step 4 and 5 Consider all edges incident to the resulting tree that have one vertex in the tree and one vertex outside the tree, then choose the one with the minimum weight

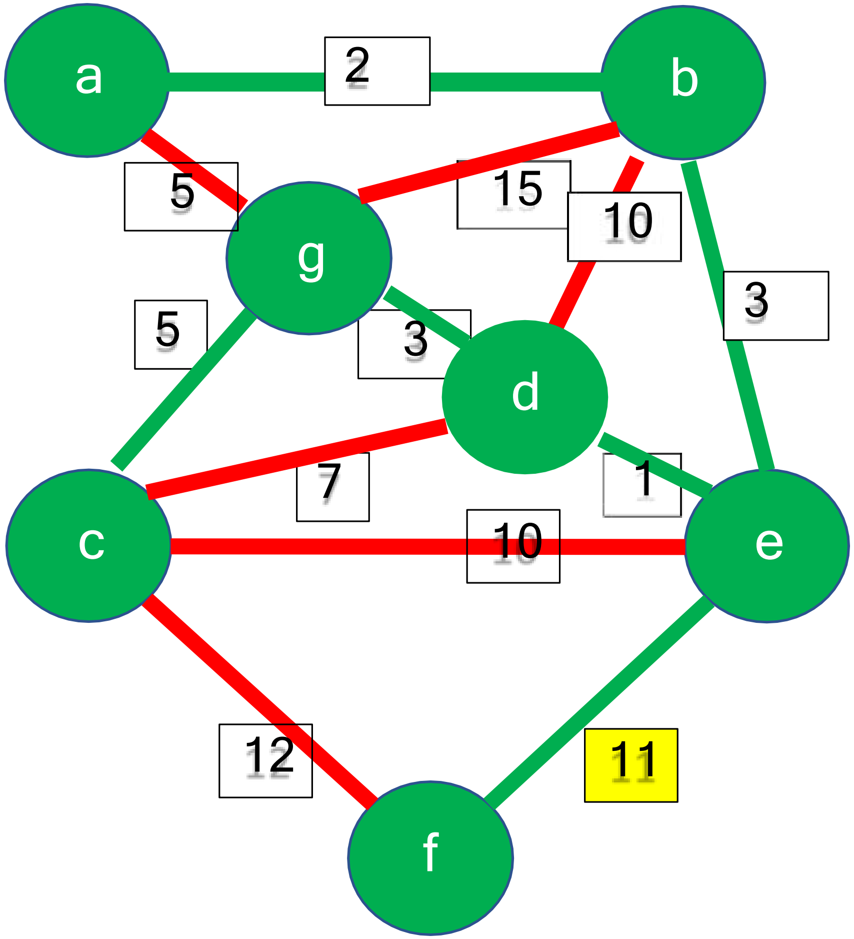


Prim's Algorithm

- Building a minimal weight spanning tree

Step 6: Repeat step 4 and 5
Consider all edges incident to the resulting tree that have one vertex in the tree and one vertex outside the tree, then choose the one with the minimum weight

We choose the edge **ef** of weight 11 and add its incident vertex to the resulting tree.

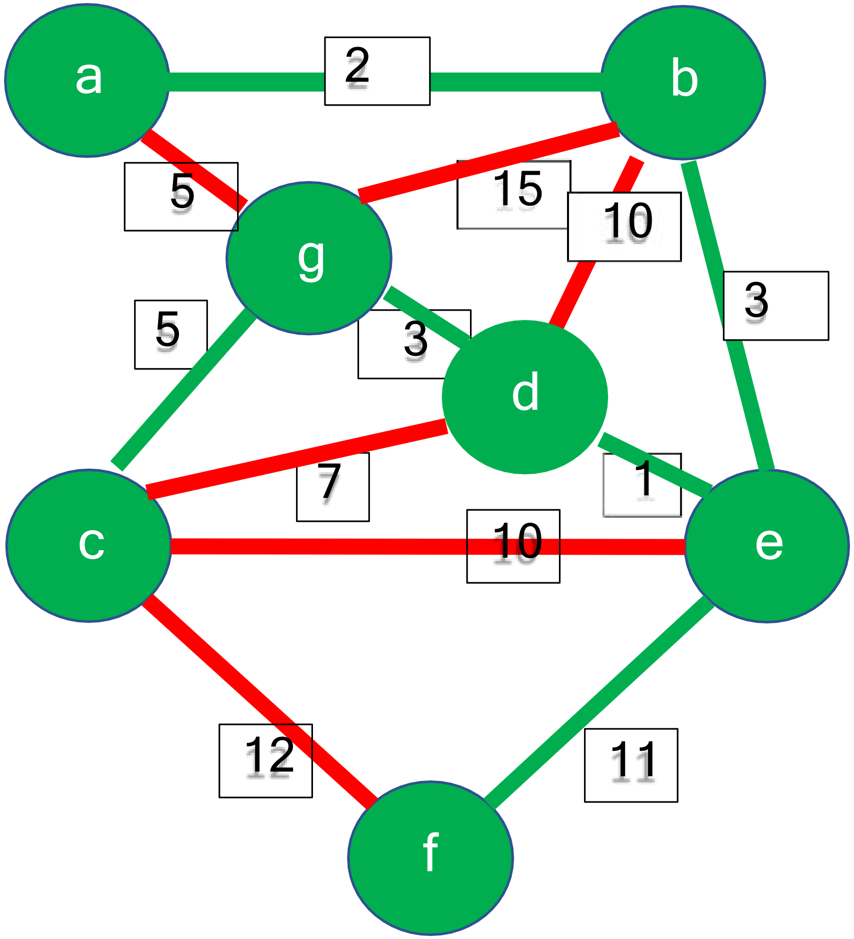


Prim's Algorithm

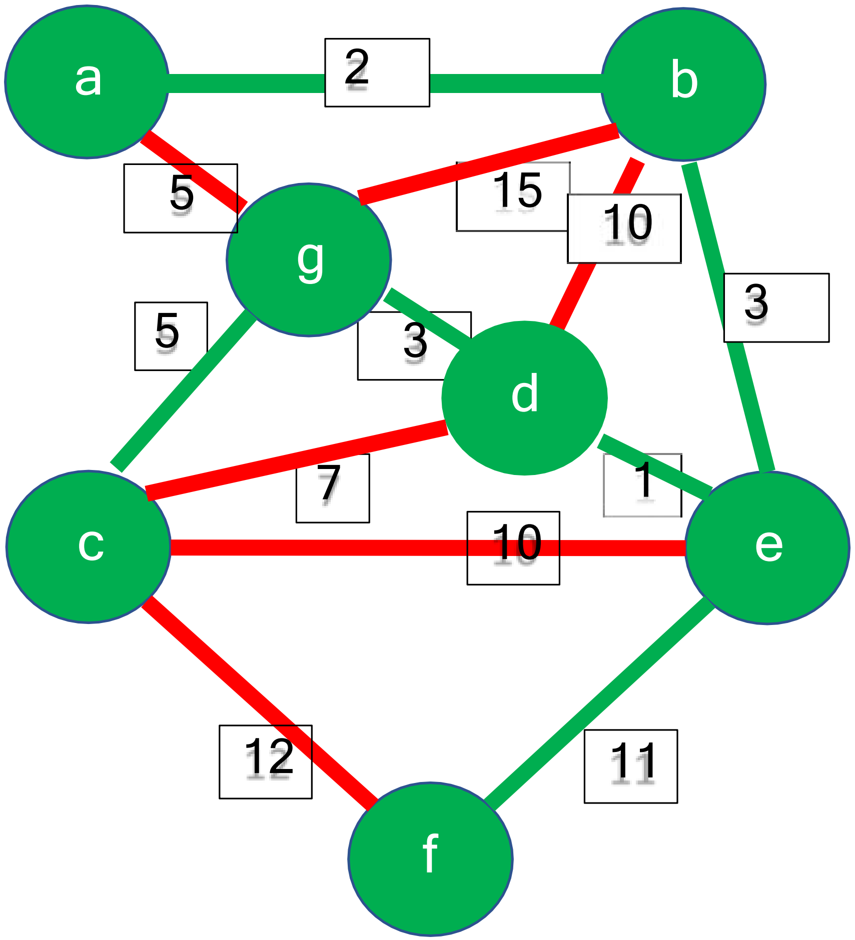
- Building a minimal weight spanning tree

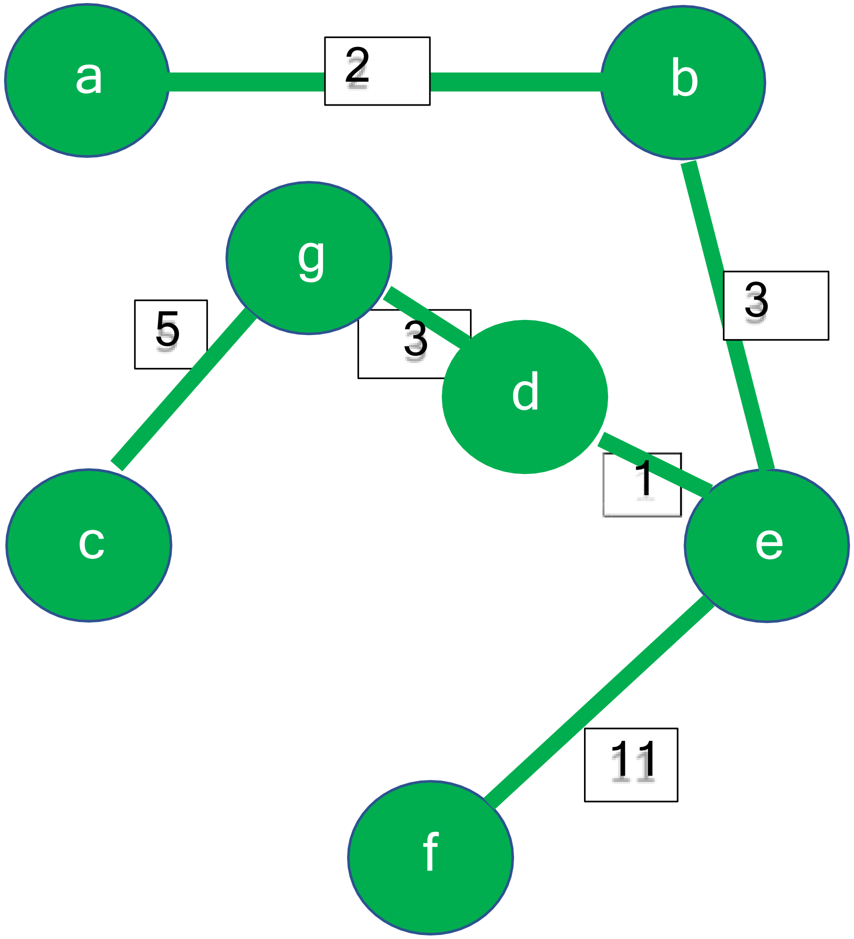
Step 7: Stop if all vertices are added

All the tree vertices have been added so we stop.



Prim's Algorithm





Here is the minimum weight spanning tree Weight = 2+3+1+3+5+11=25.