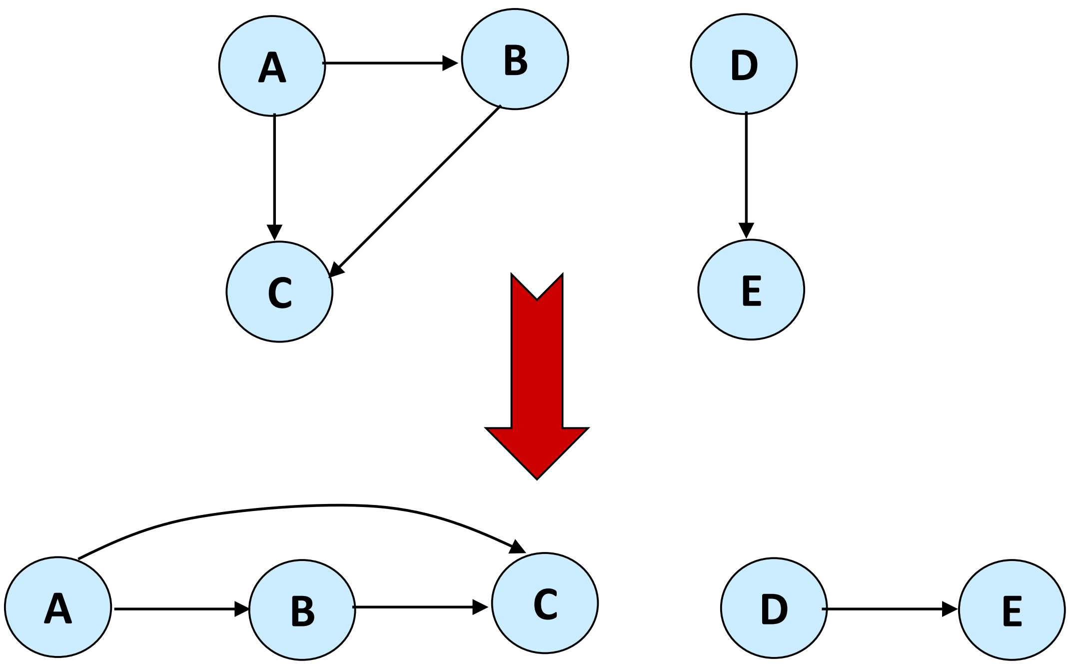
Graph Algorithms

Topological Sort

Topological Sort

- Want to “sort” a directed acyclic graph (DAG).



- Think of original DAG as a **partial order**.
- Want a **total order** that extends this partial order.

Topological Sort

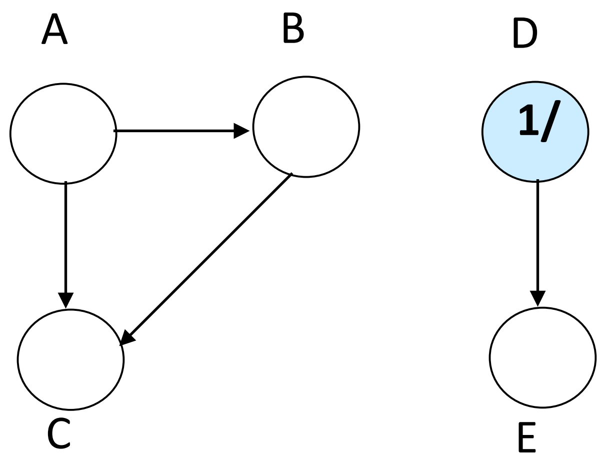
- Performed on a DAG.
- Linear ordering of the vertices of G such that if $(u, v) \in E$, then u appears somewhere before v.

Topological-Sort (G)

1. call DFS(G) to compute finishing times $f[V]$ for all $v \in V$
2. as each vertex is finished, insert it onto the front of a linked list
3. **return** the linked list of vertices

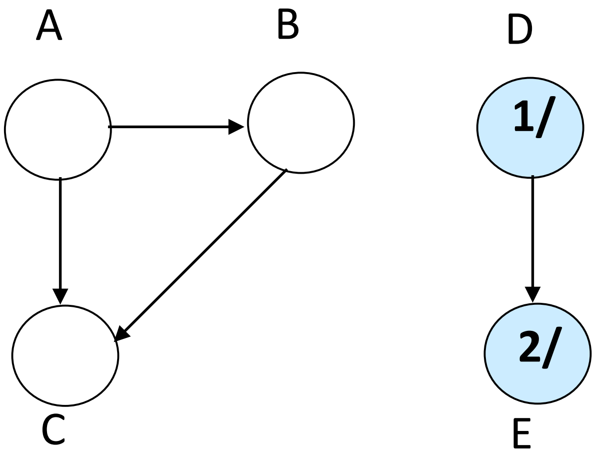
Time: $\Theta(V + E)$.

Example



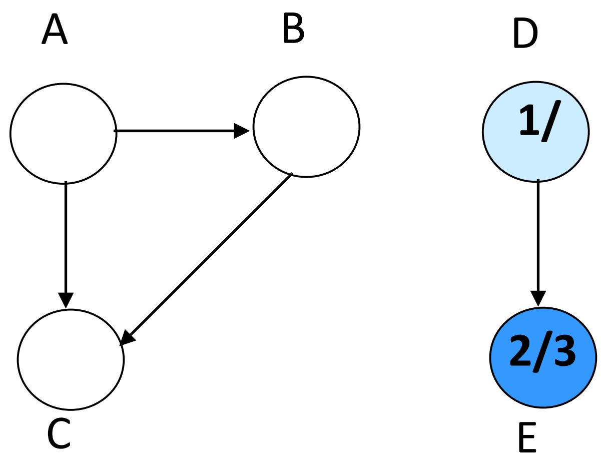
Linked List:

Example

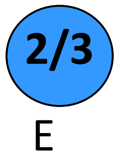


Linked List:

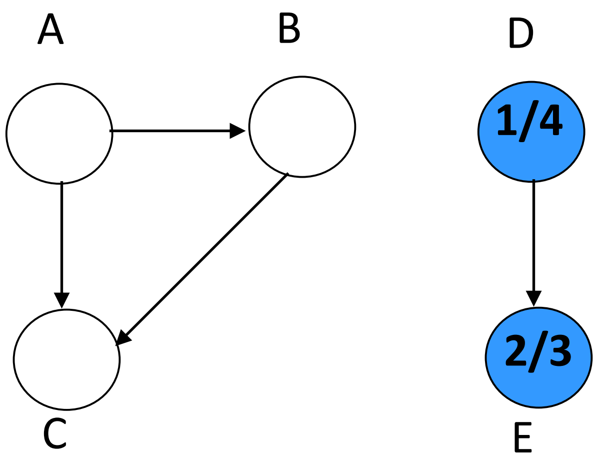
Example



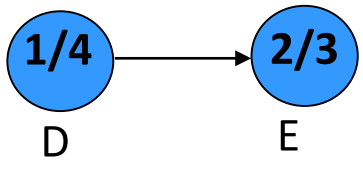
Linked List:



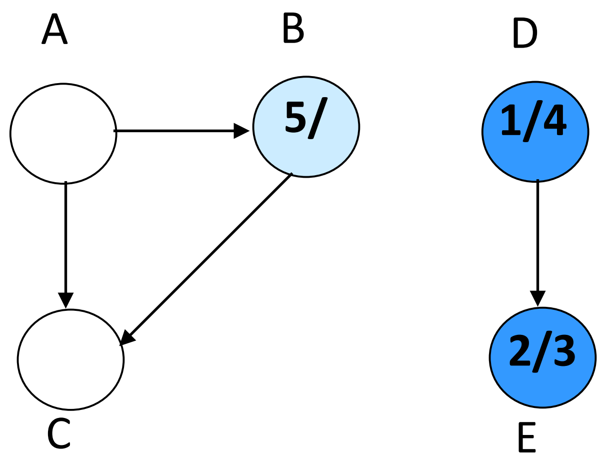
Example



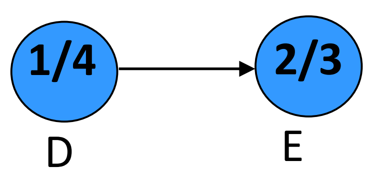
Linked List:



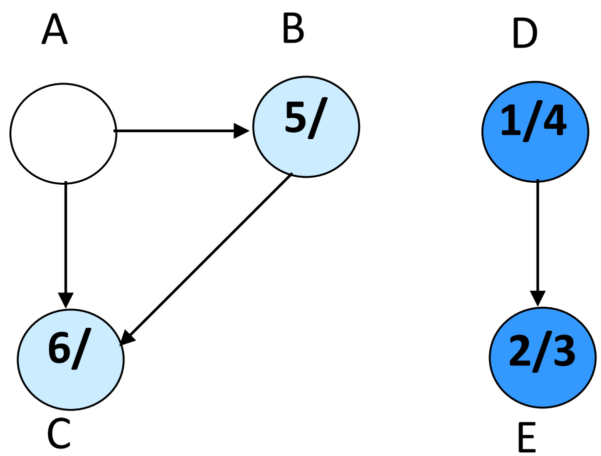
Example



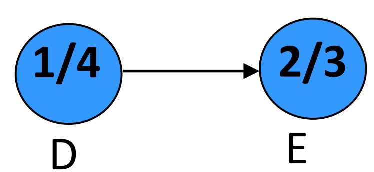
Linked List:



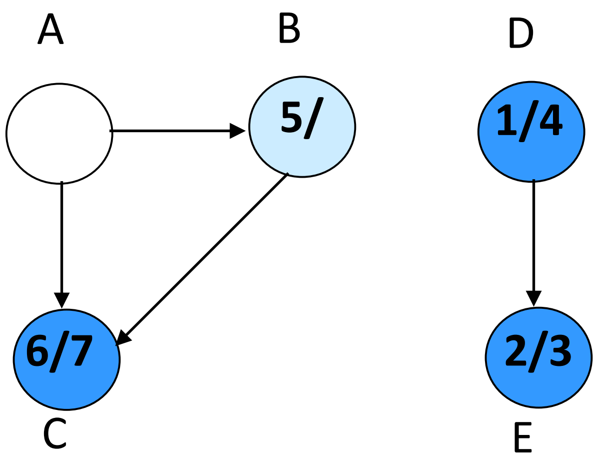
Example



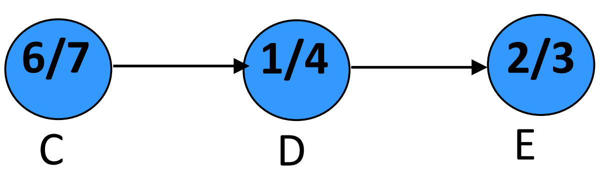
Linked List:



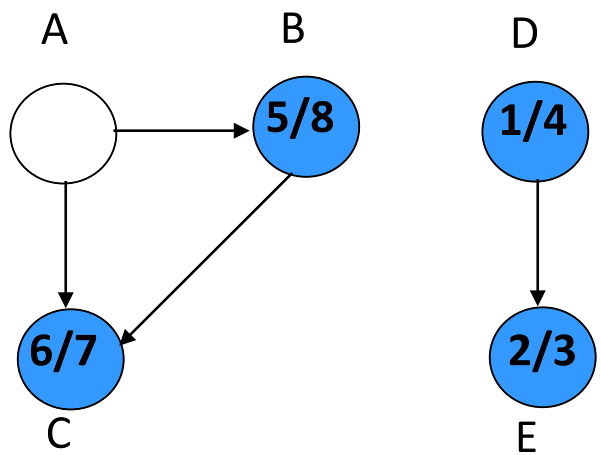
Example



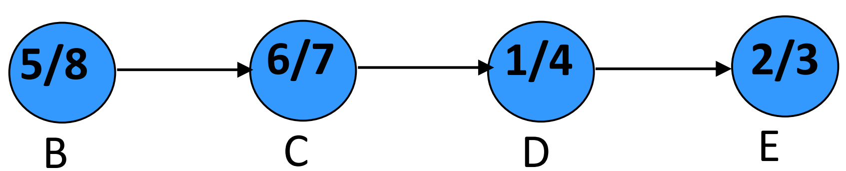
Linked List:



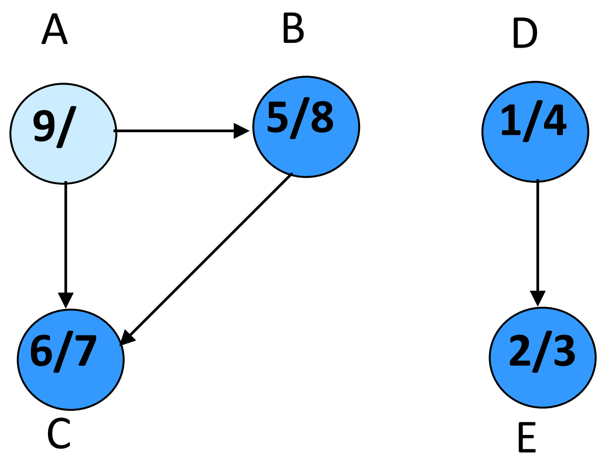
Example



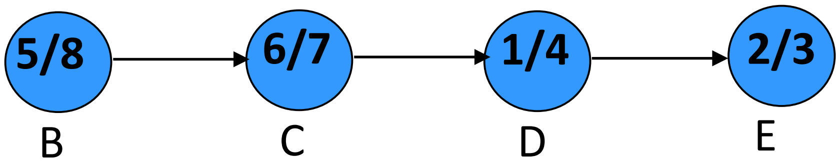
Linked List:



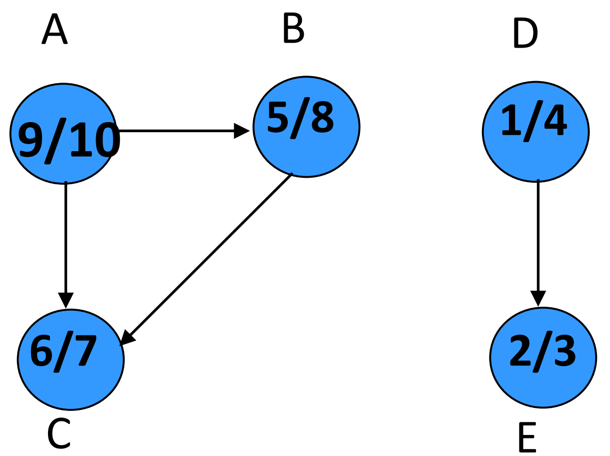
Example



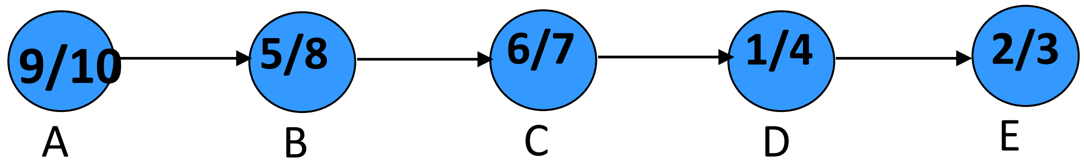
Linked List:



Example



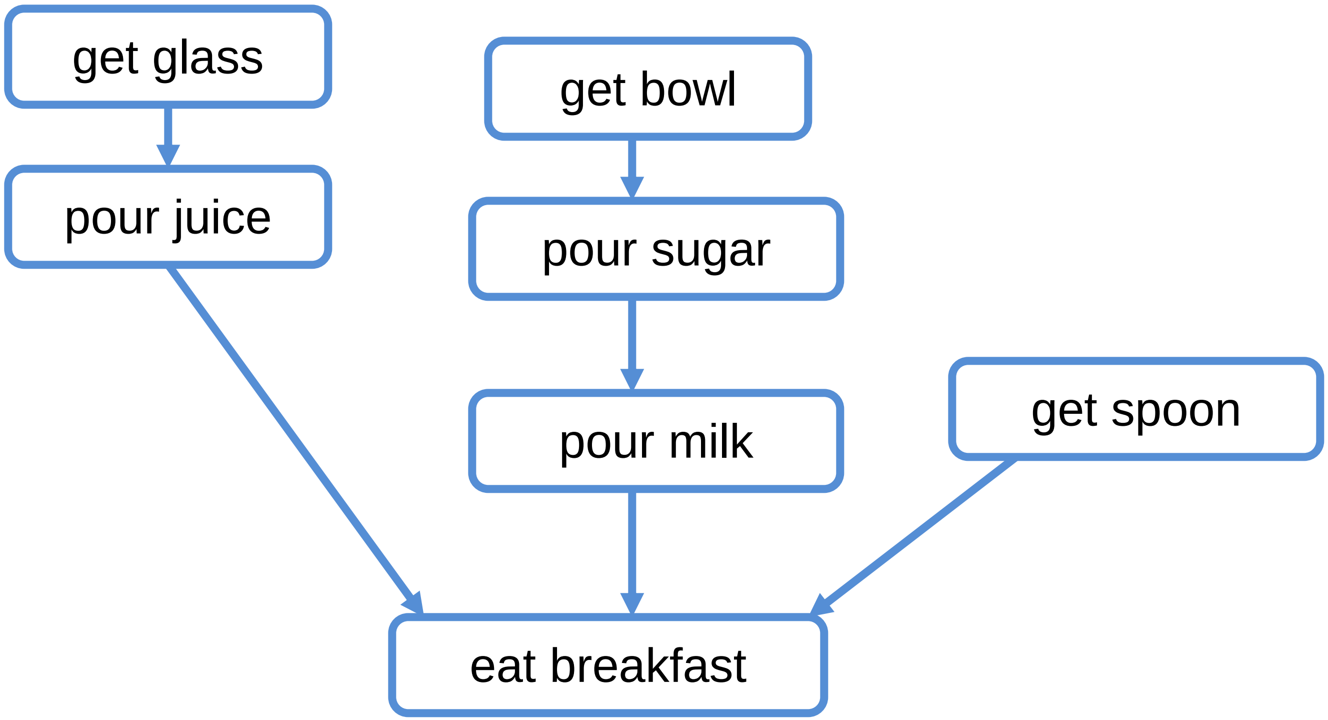
Linked List:



Precedence Example

- Tasks that have to be done to eat breakfast:
  - get glass, pour juice, get bowl, pour sugar, pour milk, get spoon, eat.
- Certain events must happen in a certain order (ex: get bowl before pouring milk)
- For other events, it doesn't matter (ex: get bowl and get spoon)

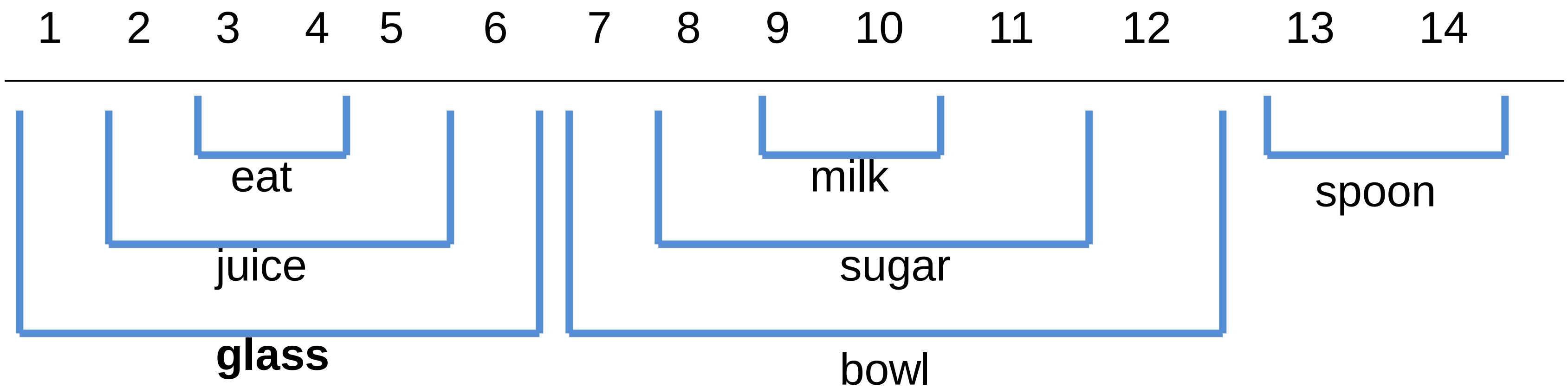
Precedence Example



Order: glass, juice, bowl, sugar, milk, spoon, eat.

Precedence Example

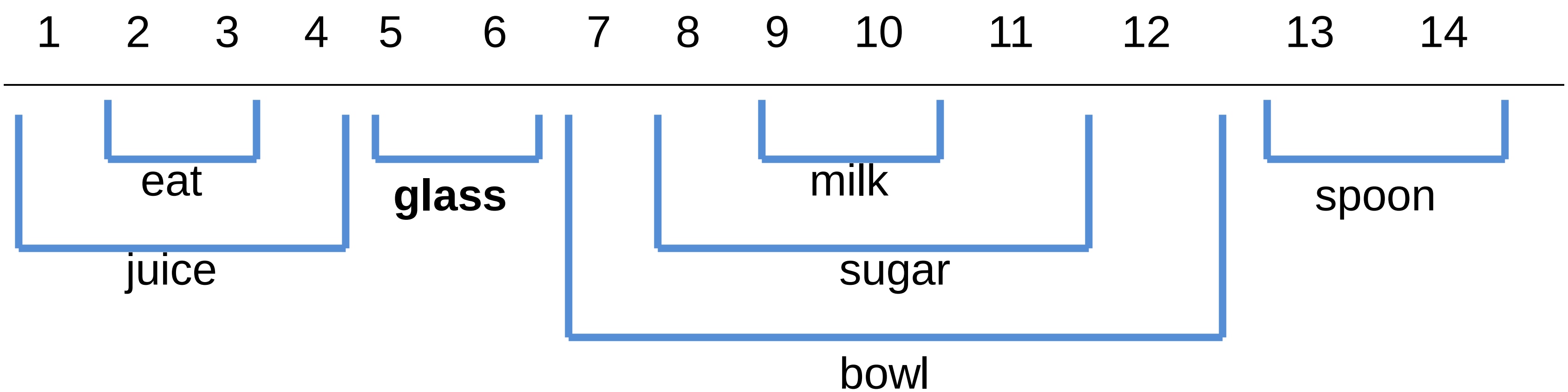
- Topological Sort



consider reverse order of finishing times:
spoon, bowl, sugar, milk, glass, juice, eat

Precedence Example

- What if we started with *juice?*



consider reverse order of finishing times:
spoon, bowl, sugar, milk, glass, juice, eat

Graph Algorithms

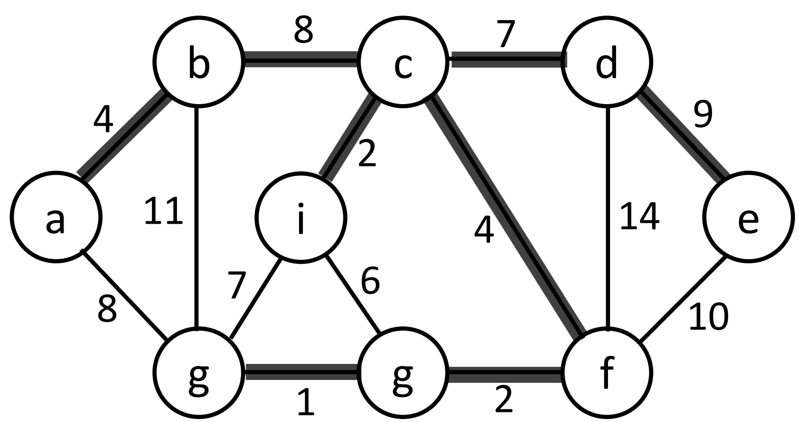
Minimum Spanning Tree

Definition of MST

- Let *G=(V,E)* be a connected, undirected graph.
- For each edge *(u,v)* in *E*, we have a weight *w(u,v)* specifying the cost (length of edge) to connect u and v.
- We wish to find a (acyclic) subset *T* of *E* that connects all of the vertices in *V* and whose total weight is minimized.
- Since the total weight is minimized, the subset T must be acyclic.
- Thus, *T* is a tree. We call it a minimum spanning tree.
- The problem of determining the tree T is called the minimum-spanning-tree problem.

Minimum Spanning Trees

- Spanning Tree
  - A tree (i.e., connected, acyclic graph) which contains all the vertices of the graph
- Minimum Spanning Tree
  - Spanning tree with the **minimum sum of weights**

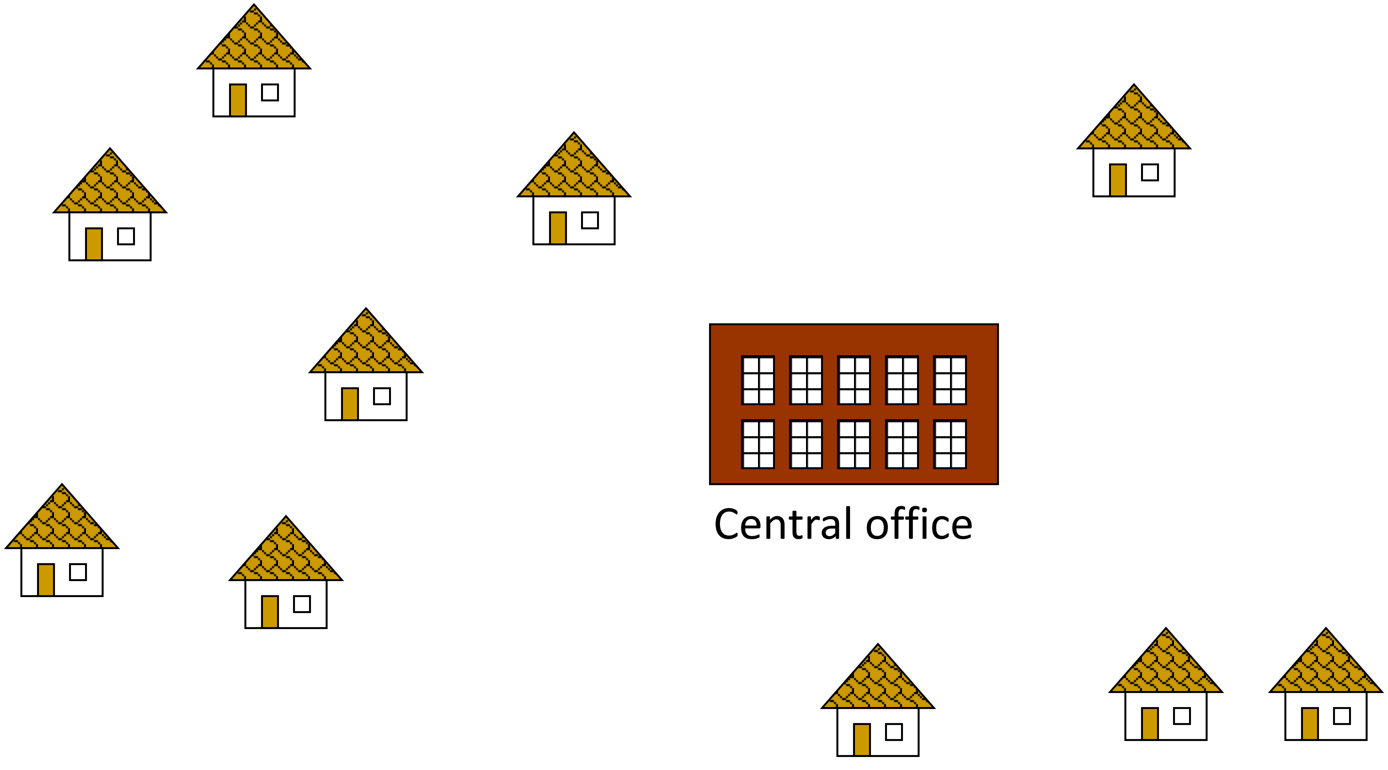


- Spanning forest
  - If a graph is not connected, then there is a spanning tree for each connected component of the graph

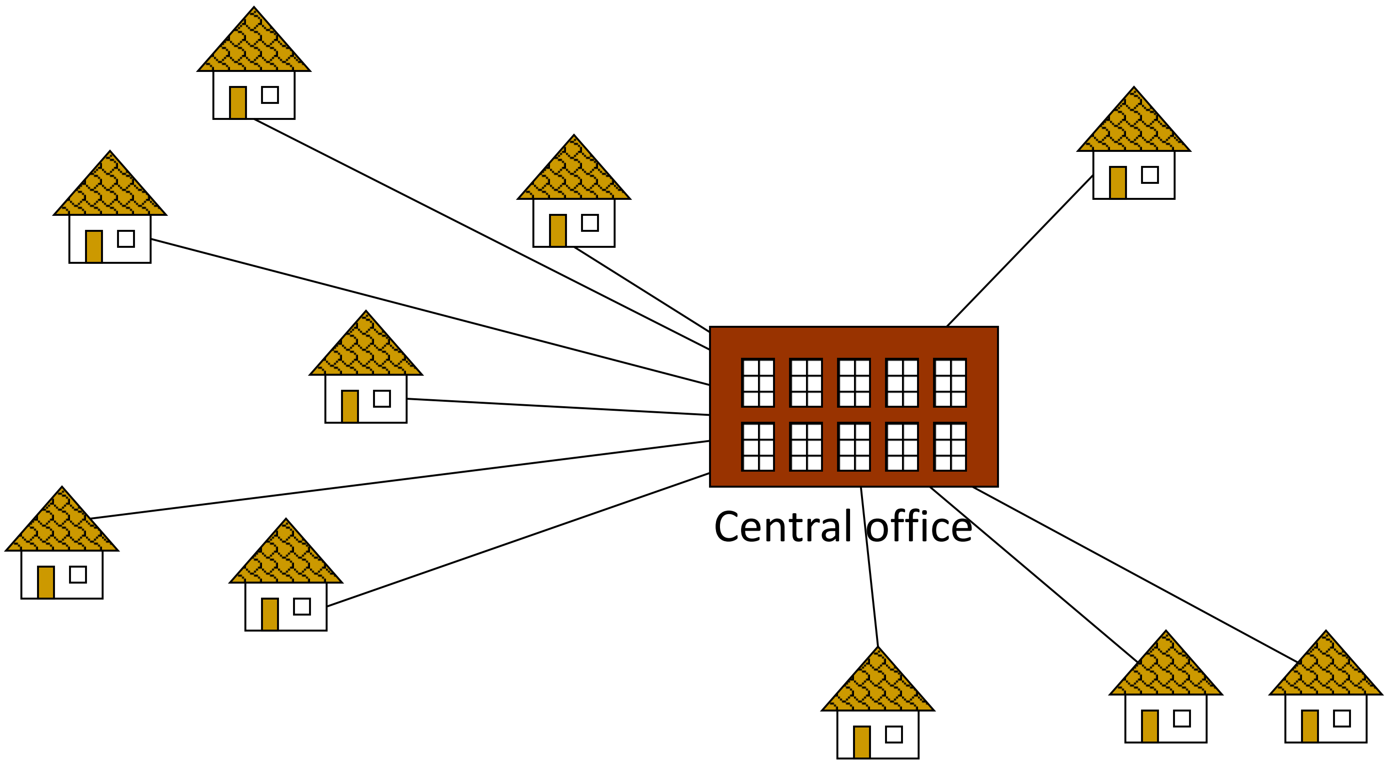
Application of MST: an example

- In the design of electronic circuitry, it is often necessary to make a set of pins electrically equivalent by wiring them together.
- Connecting Telephone wires to a set of houses. What’s the least amount of wire needed to still connect all the houses?

Problem: Laying Telephone Wire

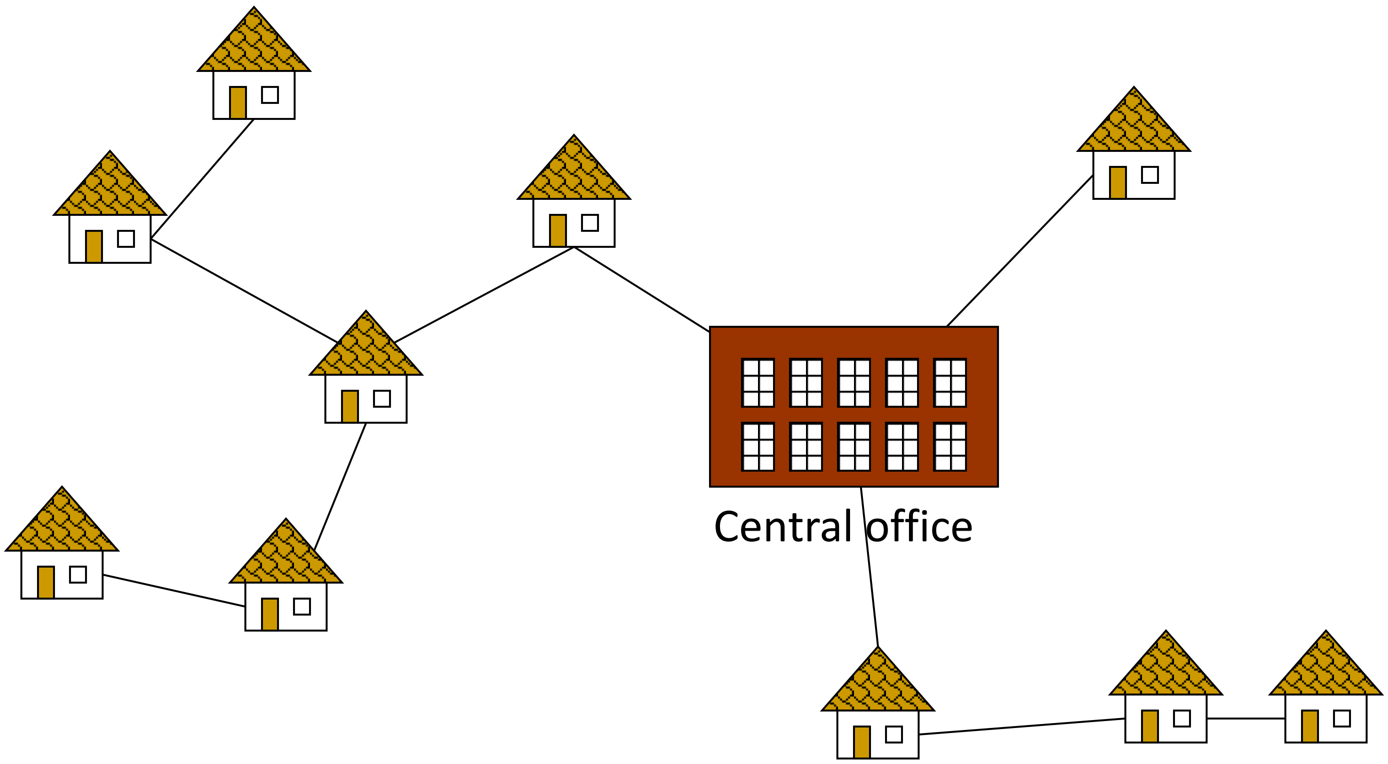


Wiring: Naïve Approach



Expensive!

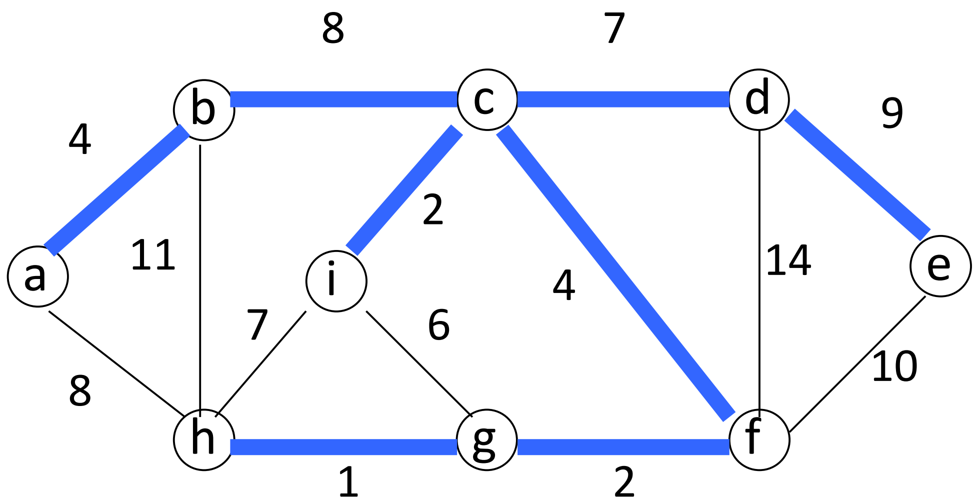
Wiring: Better Approach



Minimize the total length of wire connecting the customers

Example of MST

- Here is an example of a connected graph and its minimum spanning tree:



- Notice that the tree is not unique: replacing (b,c) with (a,h) yields another spanning tree with the same minimum weight.

Generic Algorithm

```
GENERIC_MST(G,w)
    A:={}
    while A does not form a spanning tree do
        find an edge (u,v) that is safe for A
        A:=A ∪ {(u,v)}
    return A
```

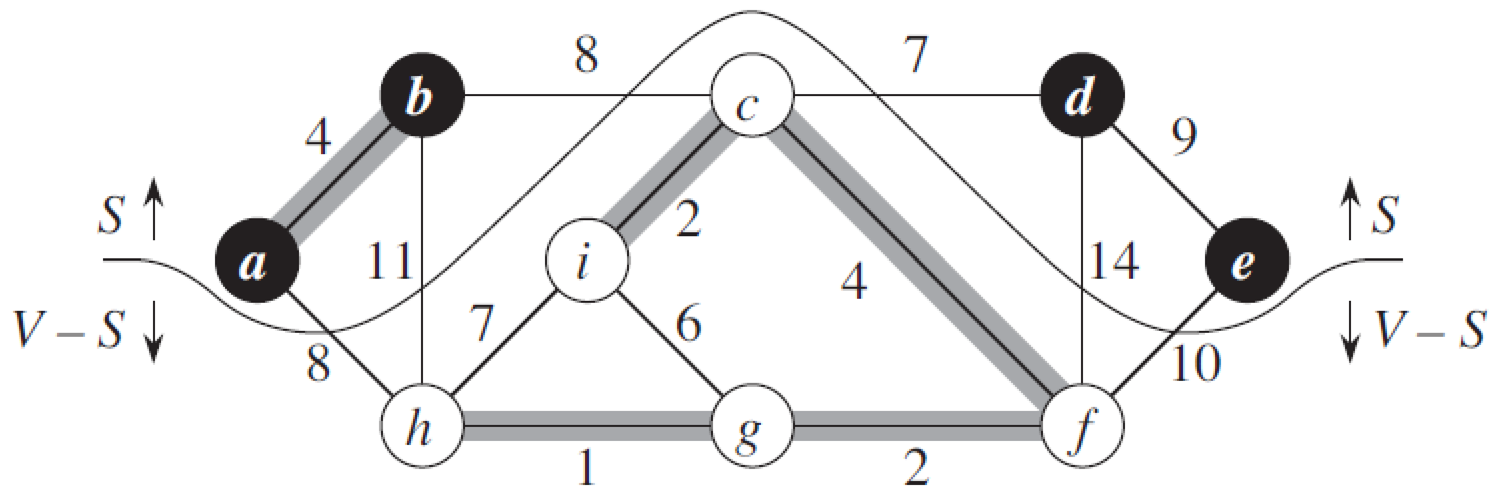
- Set A is always a subset of some minimum spanning tree.
- An edge (u,v) is a safe edge for A if by adding (u,v) to the subset A, we still have a minimum spanning tree.

How to find a safe edge

We need some definitions and a theorem.

- A cut (S,V-S) of an undirected graph G=(V,E) is a partition of V.
- An edge crosses the cut (S,V-S) if one of its endpoints is in S and the other is in V-S.
- An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut.

How to find a safe edge



- This figure shows a cut (S,V-S) of the graph.
- The edge (d,c) is the unique light edge crossing the cut.

Algorithms of Kruskal and Prim

- The two algorithms are elaborations of the generic algorithm.
- They each use a specific rule to determine a safe edge in the GENERIC_MST.
- In Kruskal's algorithm,
  - The set A is a forest.
  - The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.
- In Prim's algorithm,
  - The set A forms a single tree.
  - The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

Kruskal’s algorithm

- Basic idea:
  - Grow many small trees
  - Find two trees that are closest (i.e., connected with the lightest edge), join them with the lightest edge
  - Terminate when a single tree forms

Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

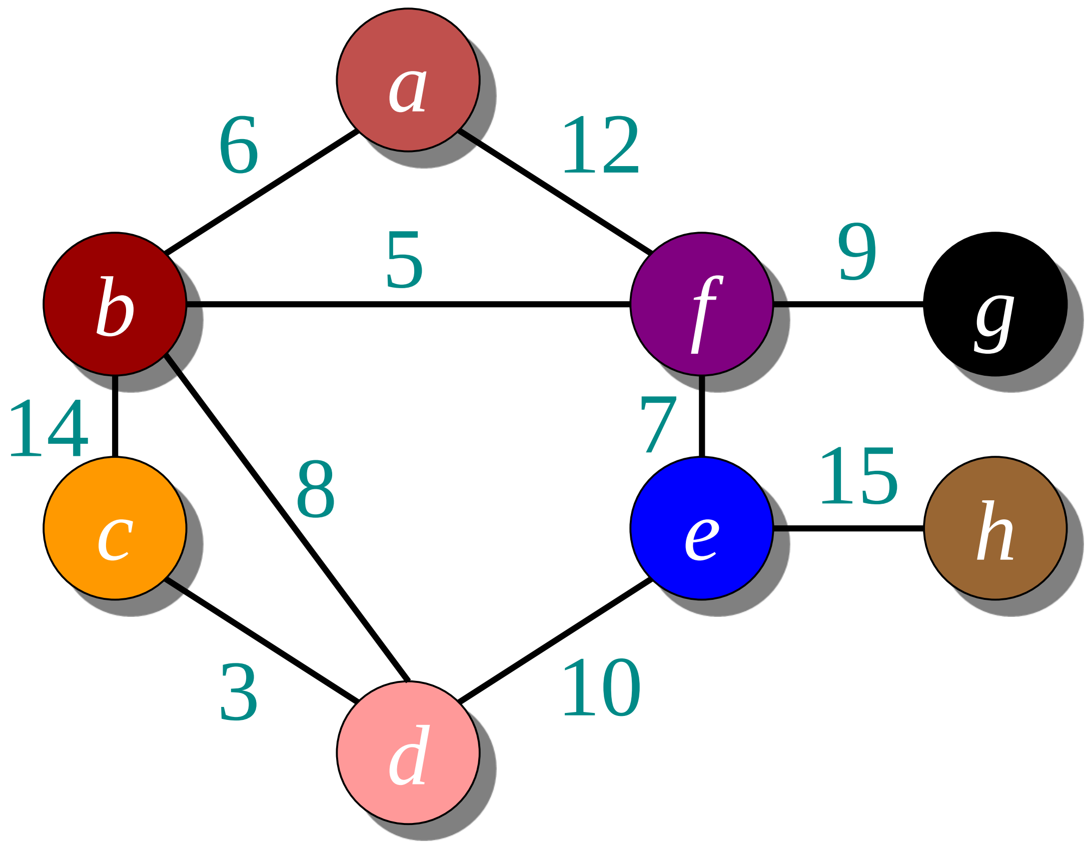
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

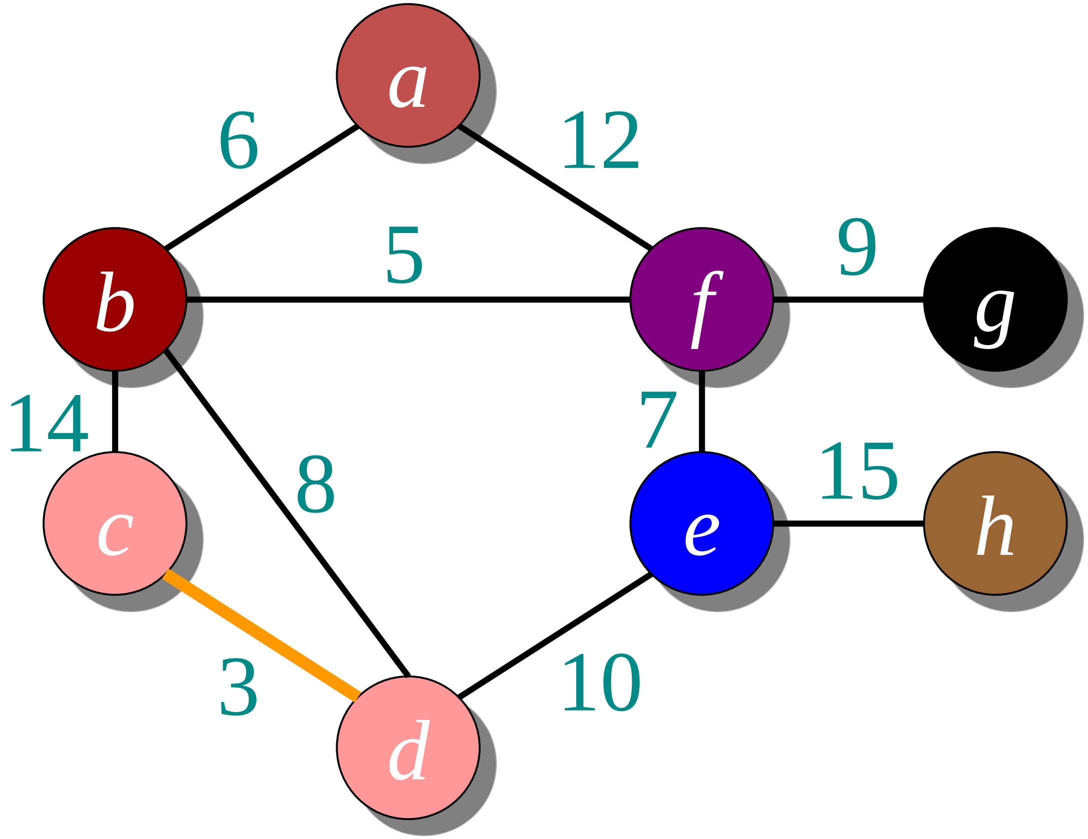
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

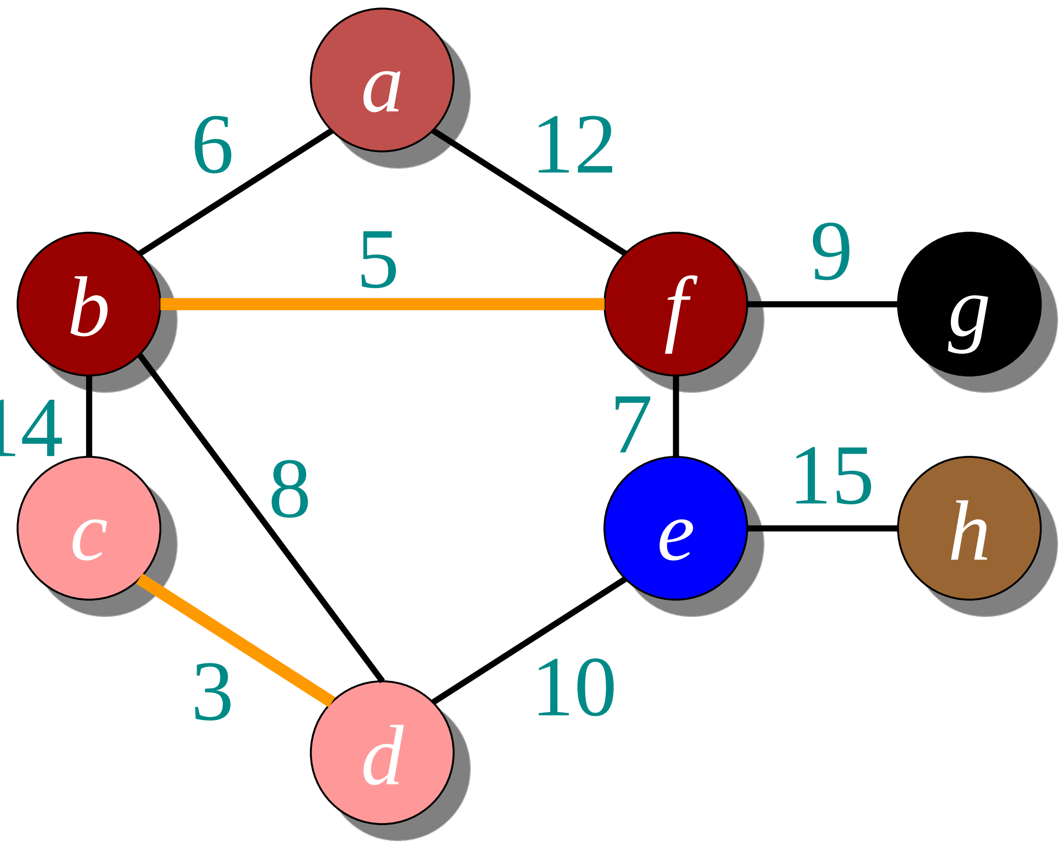
f-g: 9

d-e: 10

a-f: 12

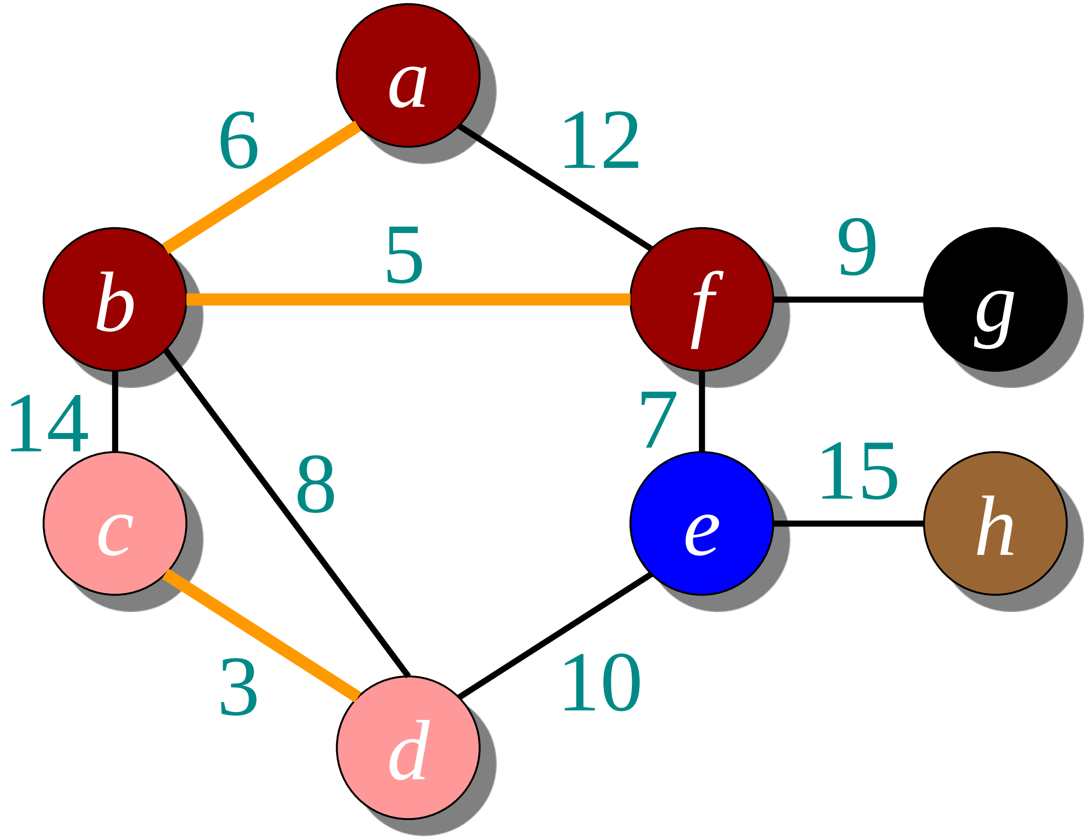
b-c: 14

e-h: 15



Example

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

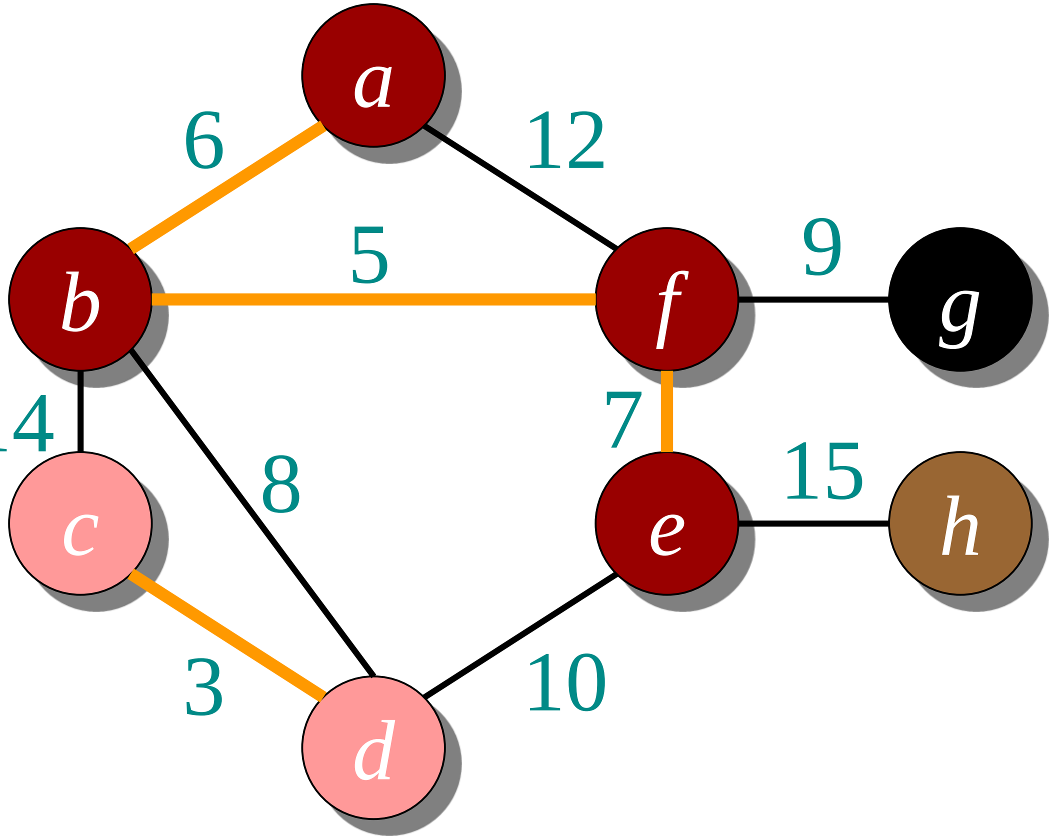
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

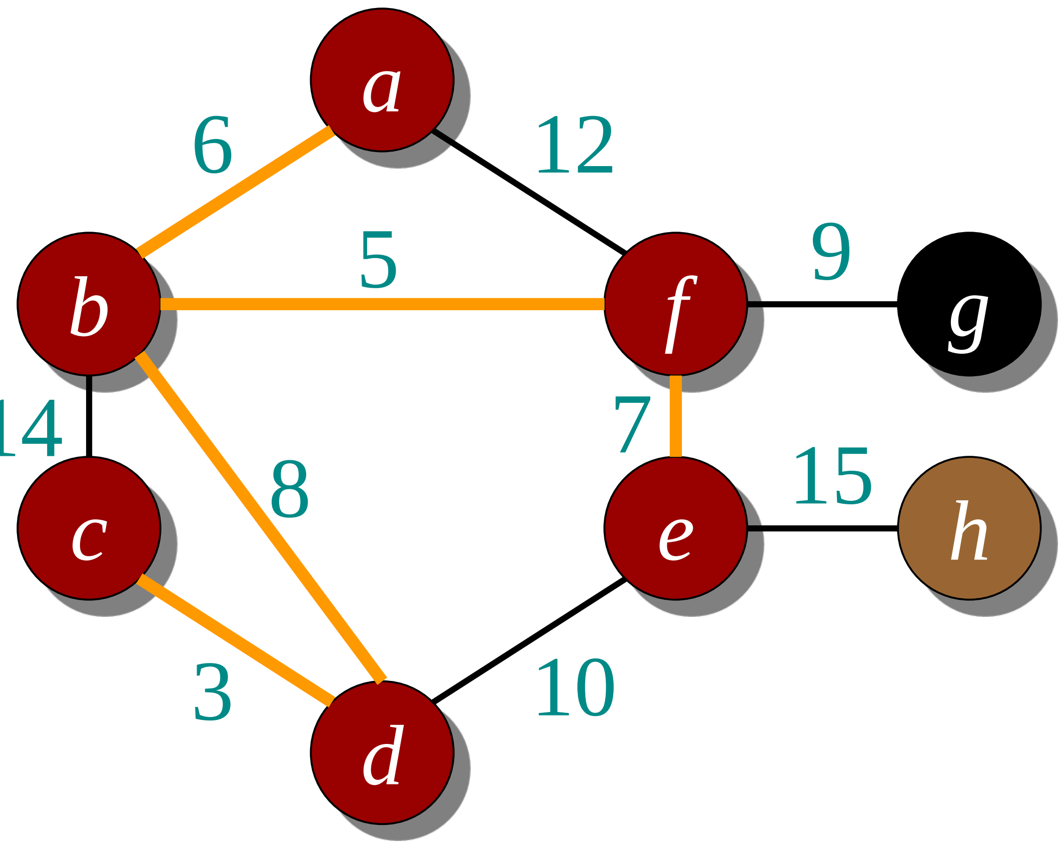
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

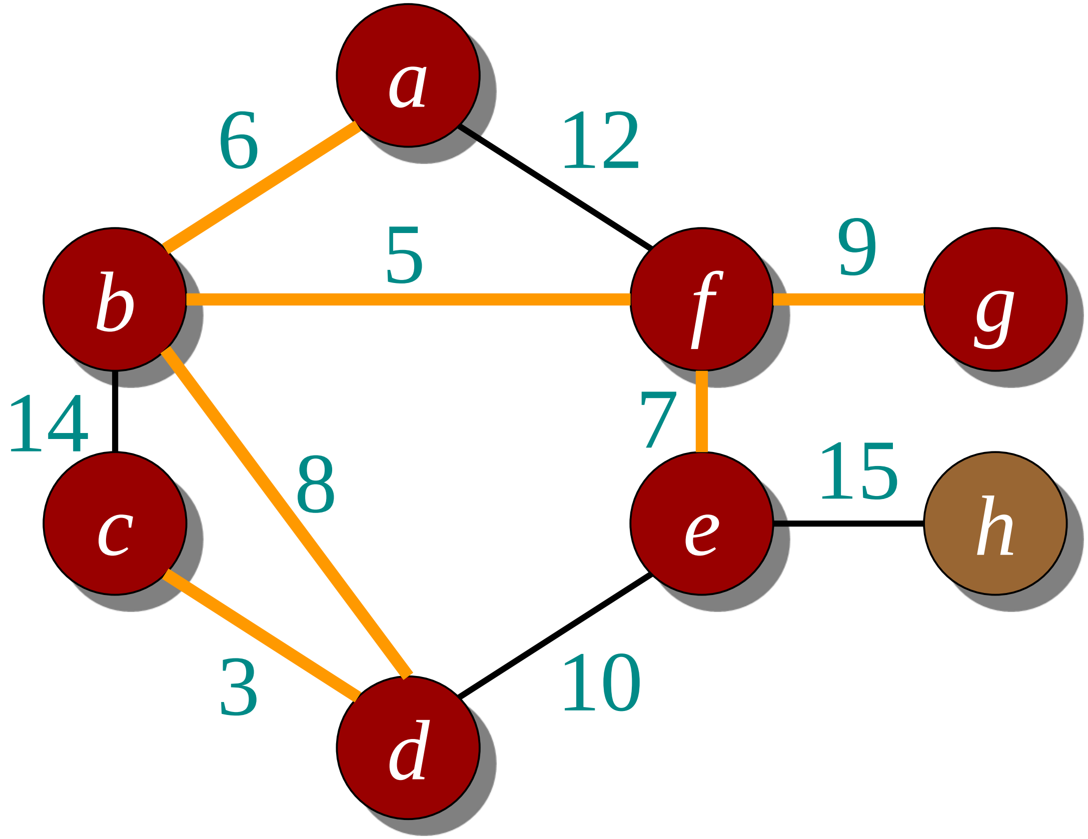
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

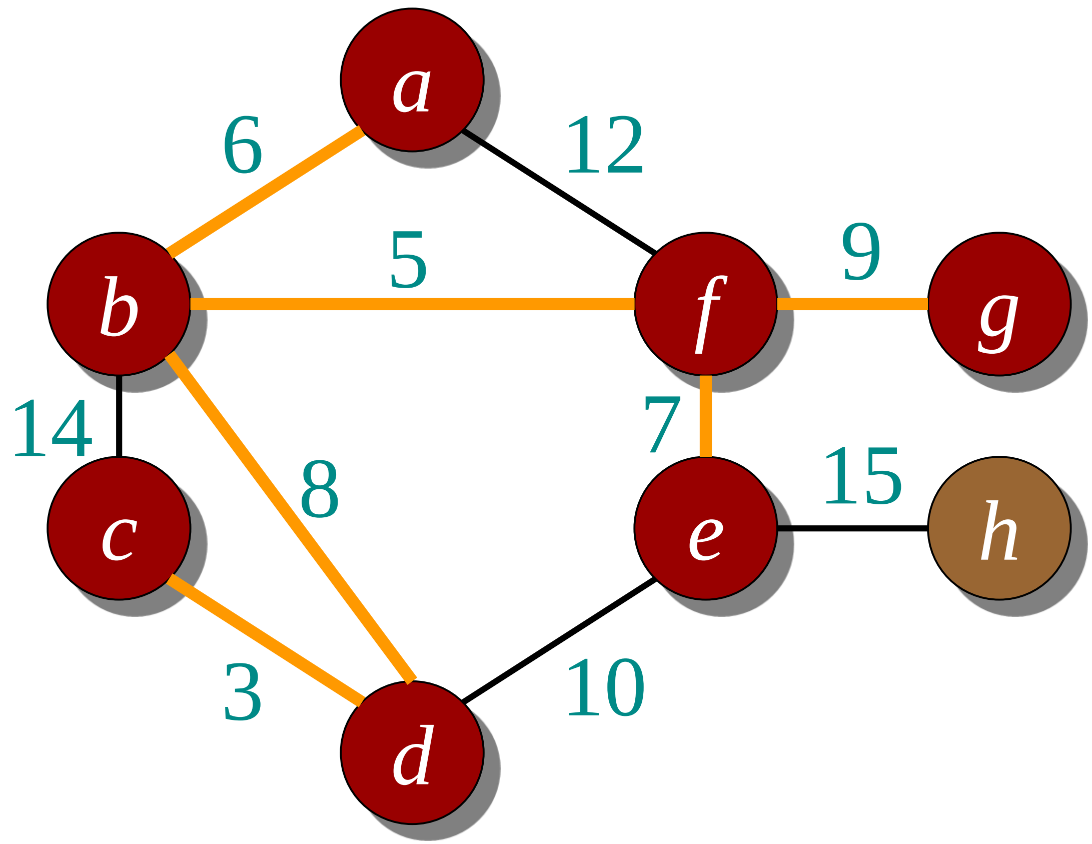
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

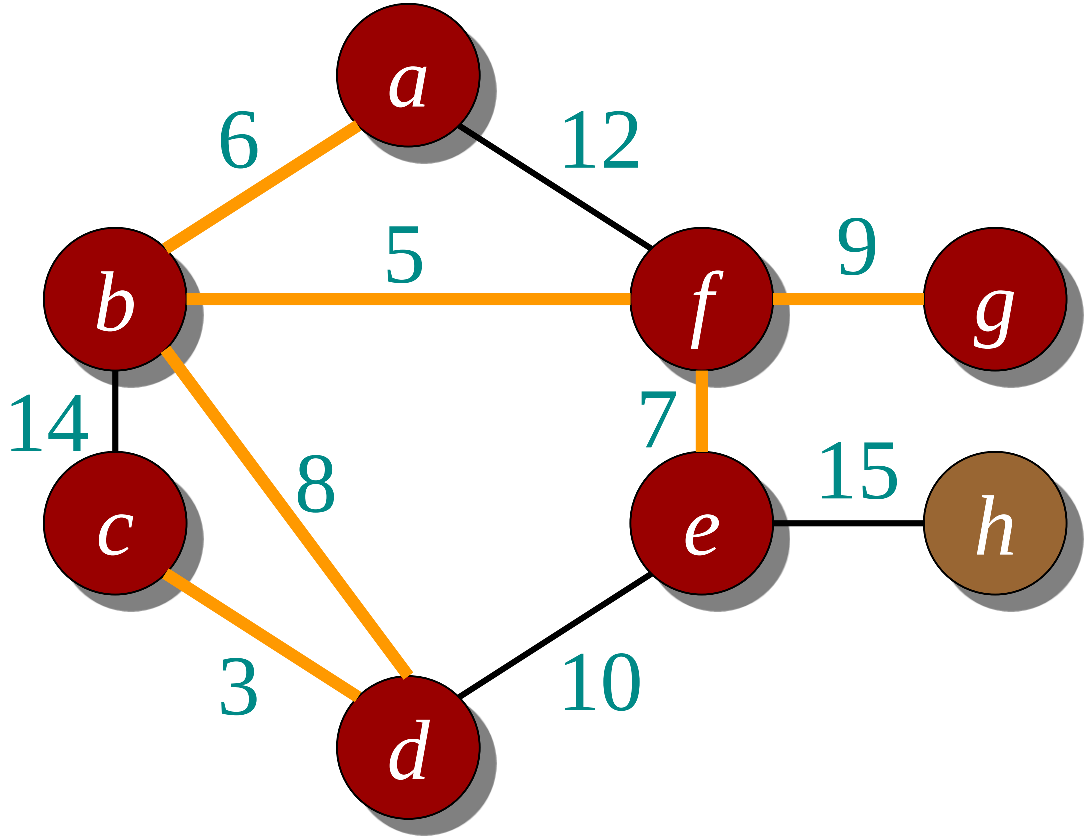
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

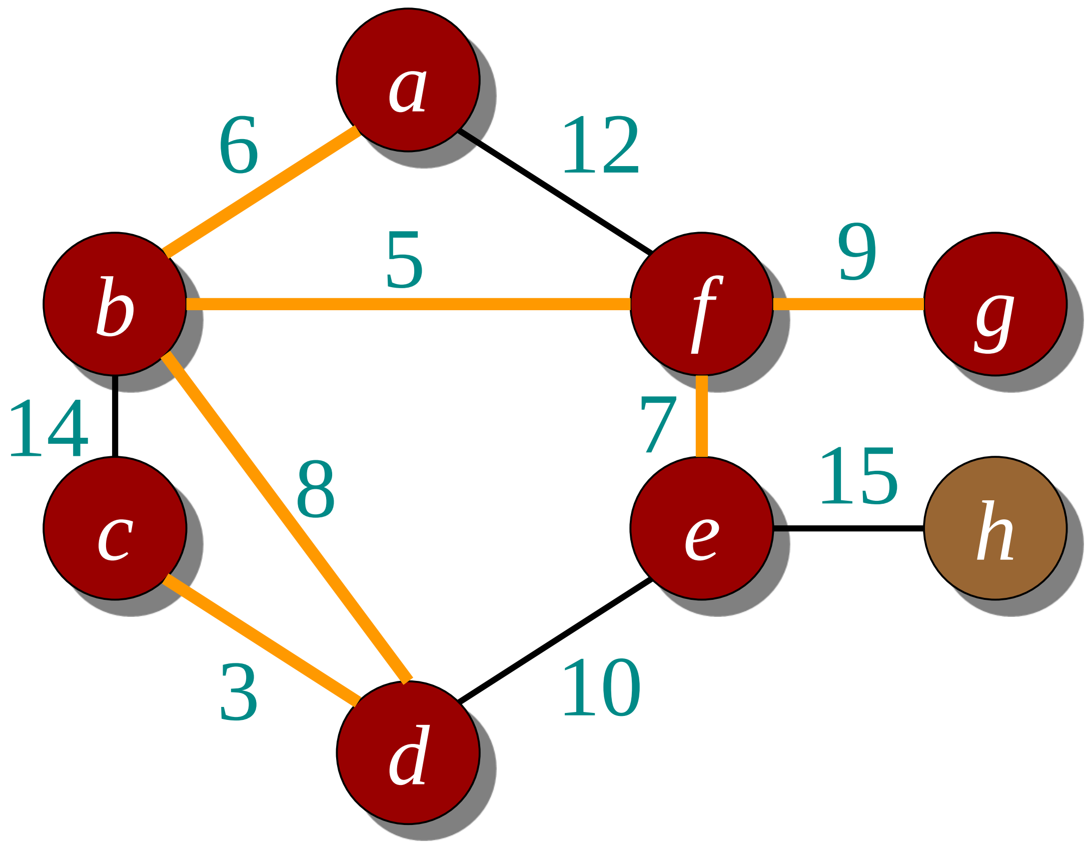
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

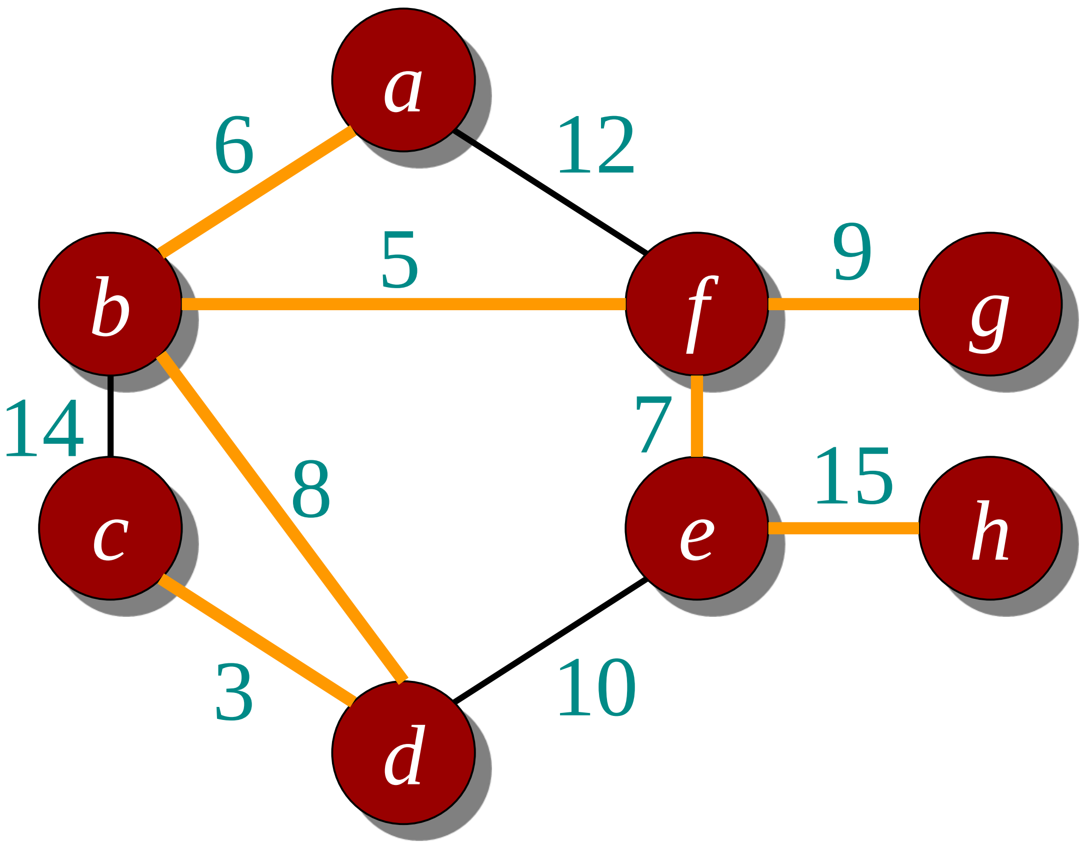
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Kruskal's algorithm in words

- Procedure:
  - Sort all edges into non-decreasing order
  - Initially each node is in its own tree
  - For each edge in the sorted list
    - If the edge connects two separate trees, then
      - join the two trees together with that edge

Disjoint-Set

- Keep a collection of sets S_1, S_2, .., S_k,
  - Each S_i is a set, e,g, $S_1=\{v_1, v_2, v_8\}$.
- Three operations
  - Make-Set(x)-creates a new set whose only member is x.
  - Union(x, y) –unites the sets that contain x and y, say, S_x and S_y, into a new set that is the union of the two sets.
  - Find-Set(x)-returns a pointer to the representative of the set containing x.

Algorithm for Disjoint-Set Forest

UNION(x,y)
1. LINK(FIND-SET(x),FIND-SET(y))

MAKE-SET(x)
1. $p[x] \leftarrow x$
2. $rank[x] \leftarrow 0$

LINK(x,y)
1. **if** $rank[x] > rank[y]$
2. **then** $p[y] \leftarrow x$
3. **else** $p[x] \leftarrow y$
4. **if** $rank[x] = rank[y]$
5. **then** $rank[y]$++

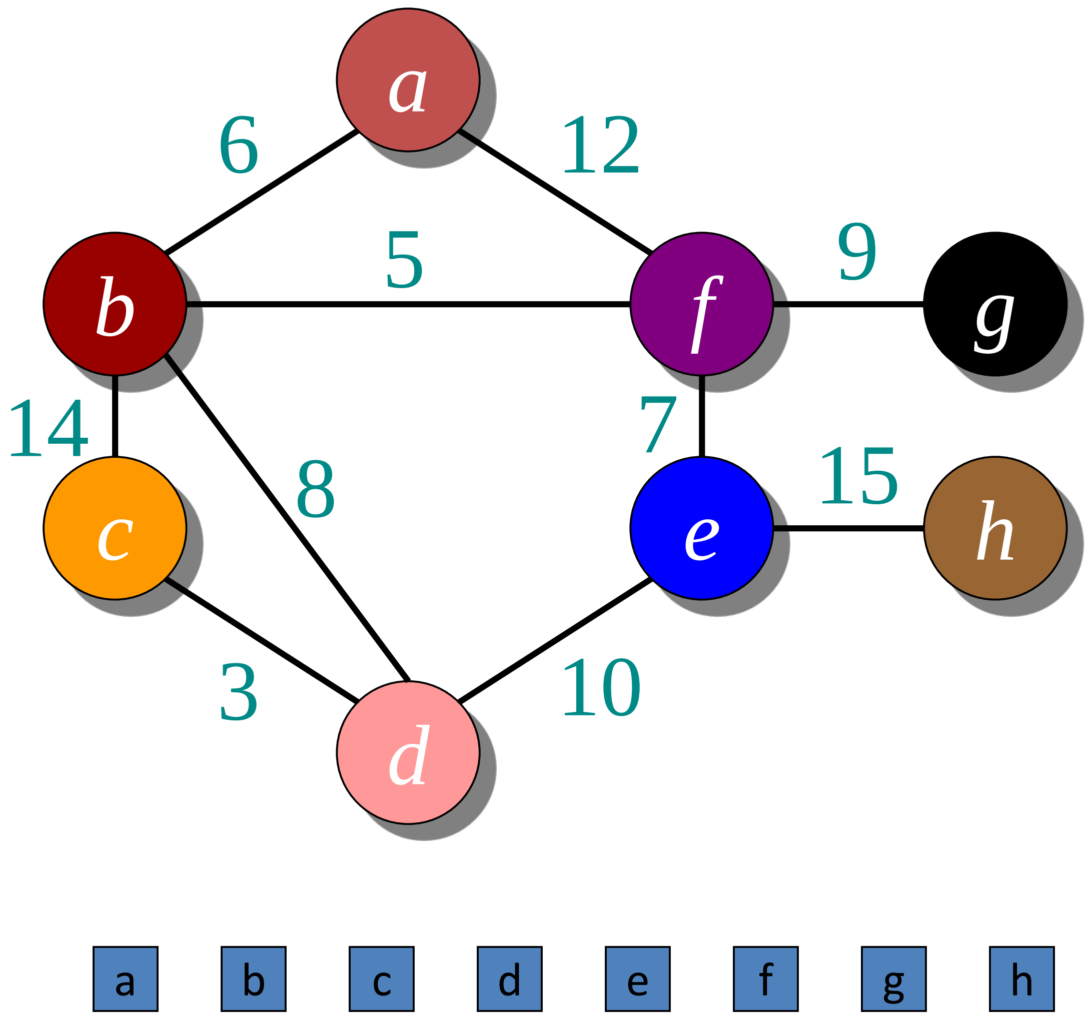
FIND-SET(x)
1. **if** $x \neq p[x]$
2. **then** $p[x] \leftarrow$FIND-SET($p[x]$)
3. **return** $p[x]$

Kruskal's Algorithm

```
MST-Kruskal(G,w)
A ← ∅
for each vertex v ∈ V[G] do
    Make-Set(v)  //creates set containing v (for initialization)
sort the edges of E
for each (u,v)∈E do
  if Find-Set(u) ≠ Find-Set(v) then // different component
      A ← A ∪ {(u,v)}
      Union(Set(u),Set(v)) // merge
return A
```

Example with disjoint set union

c-d: 3
b-f: 5
b-a: 6
f-e: 7
b-d: 8
f-g: 9
d-e: 10
a-f: 12
b-c: 14
e-h: 15



Example with disjoint set union

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

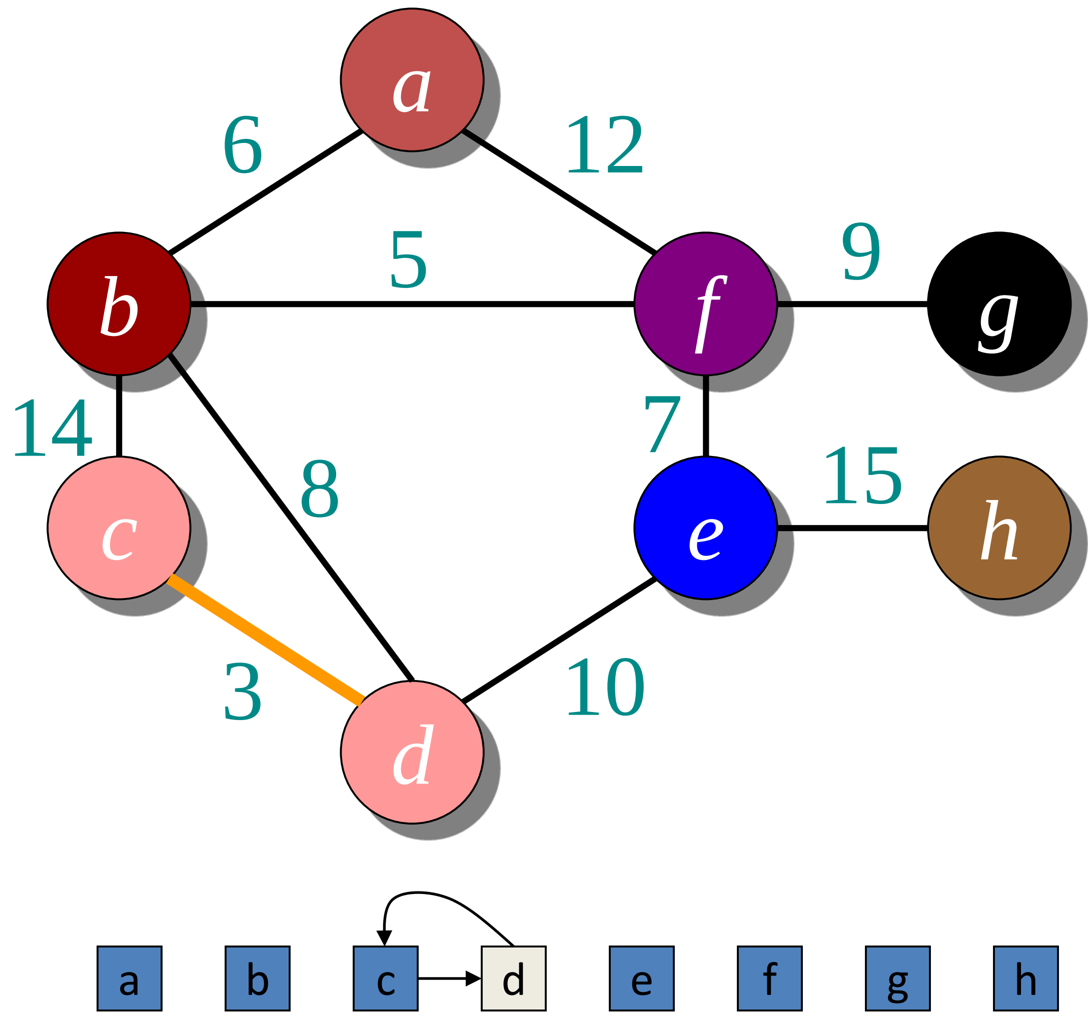
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example with disjoint set union

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

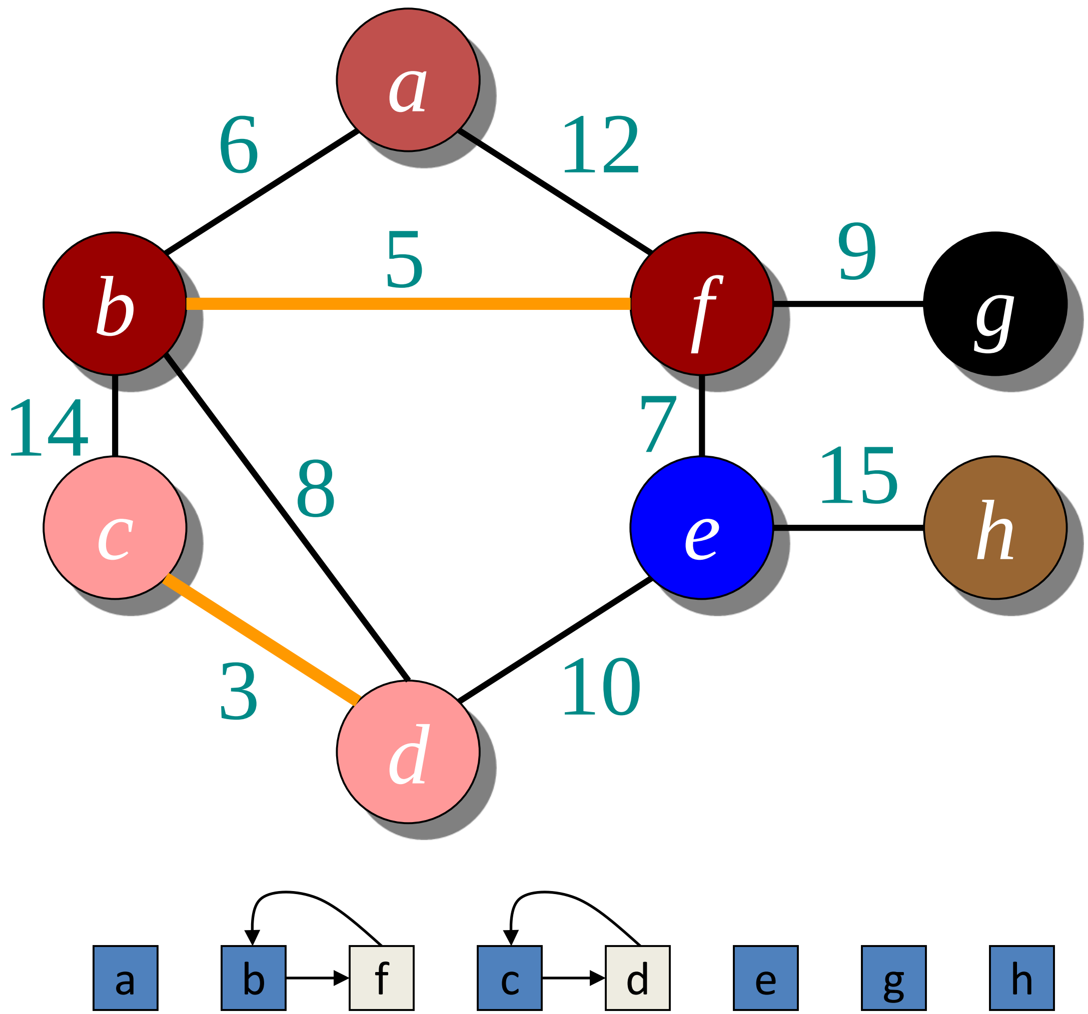
f-g: 9

d-e: 10

a-f: 12

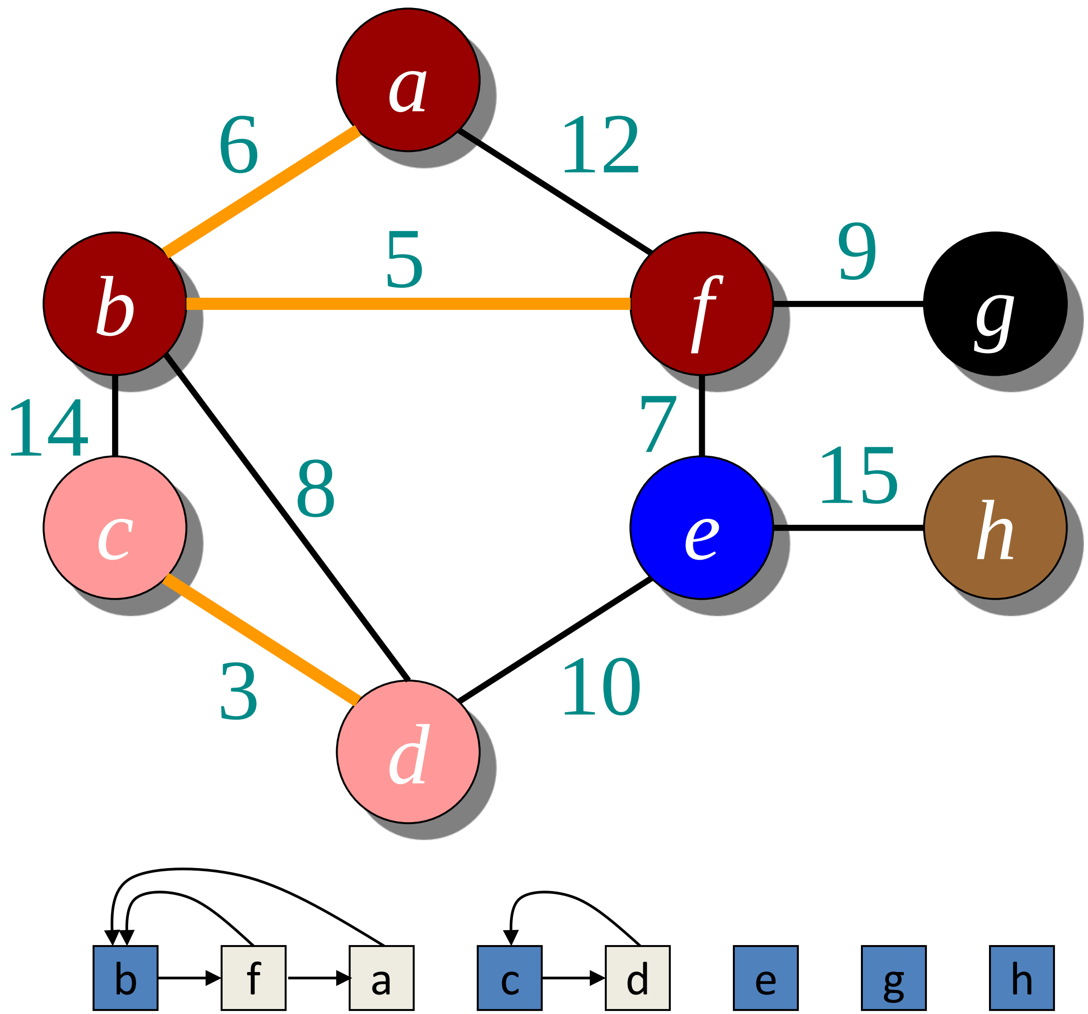
b-c: 14

e-h: 15



Example with disjoint set union

c-d: 3
b-f: 5
b-a: 6
f-e: 7
b-d: 8
f-g: 9
d-e: 10
a-f: 12
b-c: 14
e-h: 15



Example with disjoint set union

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

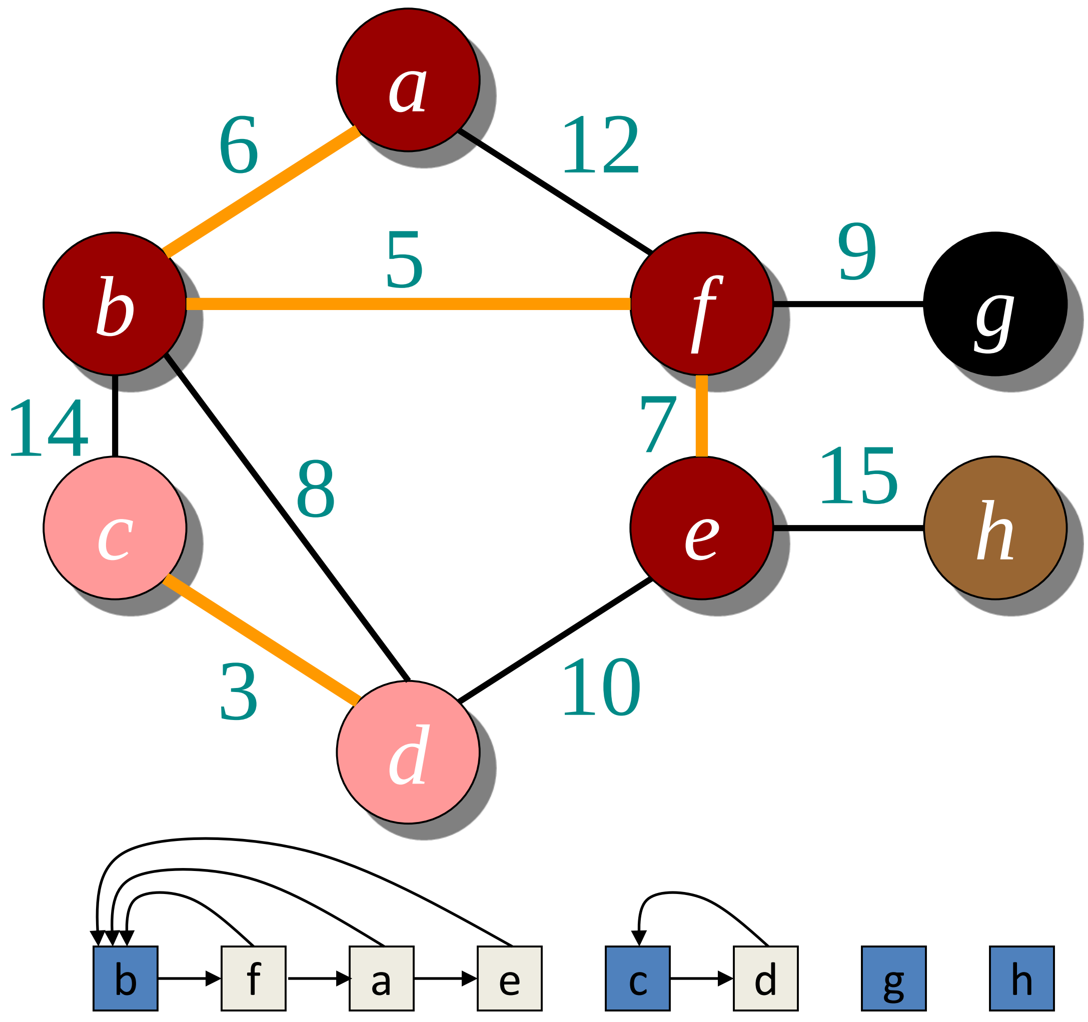
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



Example with disjoint set union

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

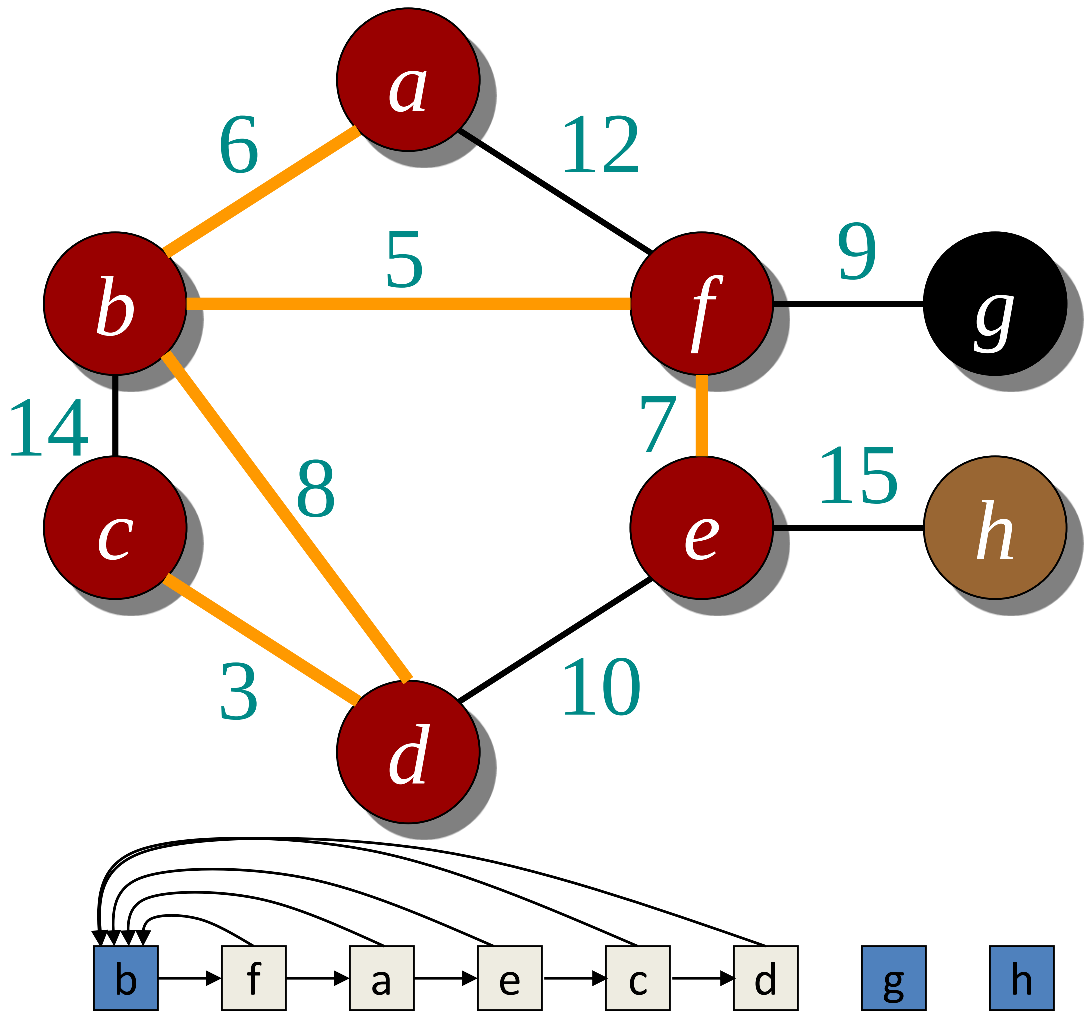
f-g: 9

d-e: 10

a-f: 12

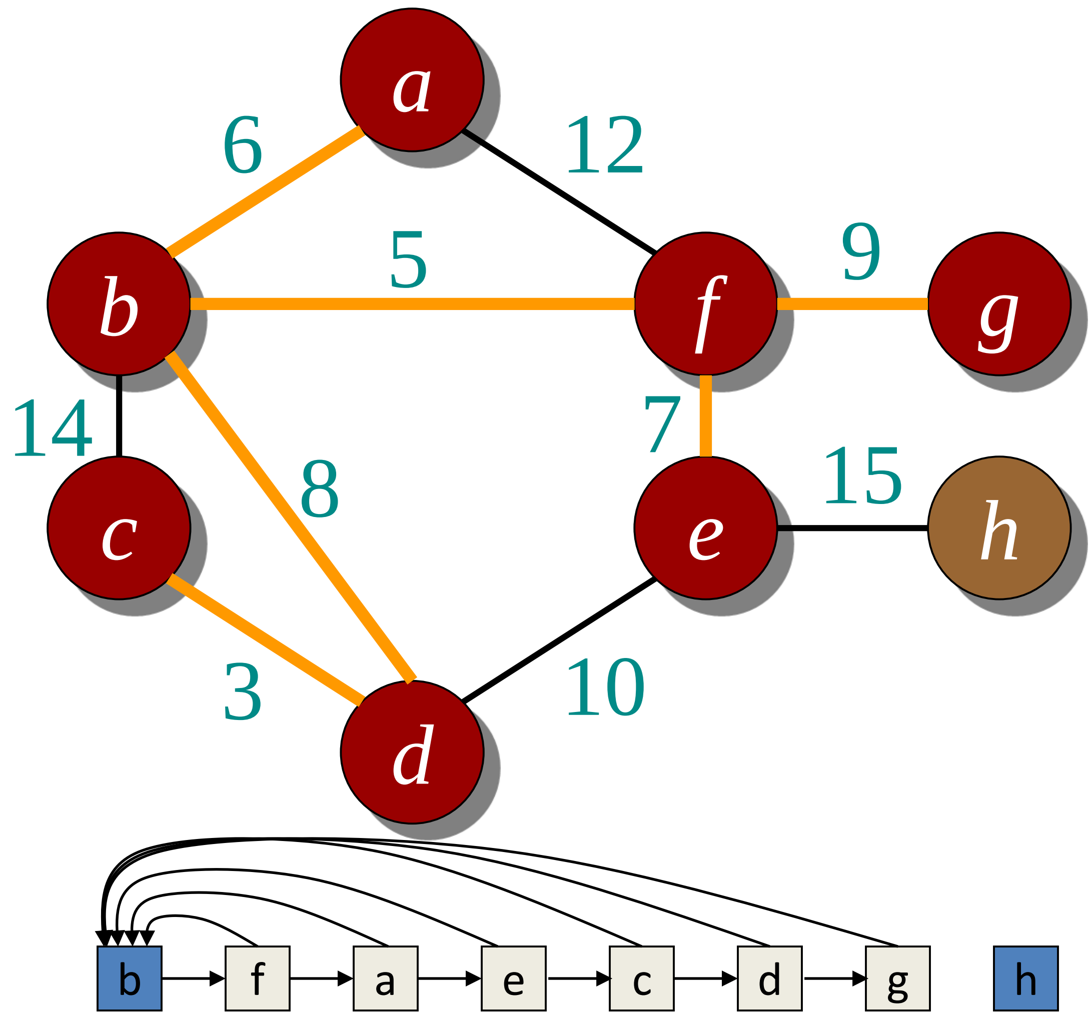
b-c: 14

e-h: 15



Example with disjoint set union

c-d:	3
b-f:	5
b-a:	6
f-e:	7
b-d:	8
f-g:	9
d-e:	10
a-f:	12
b-c:	14
e-h:	15



Example with disjoint set union

c-d: 3

b-f: 5

b-a: 6

f-e: 7

b-d: 8

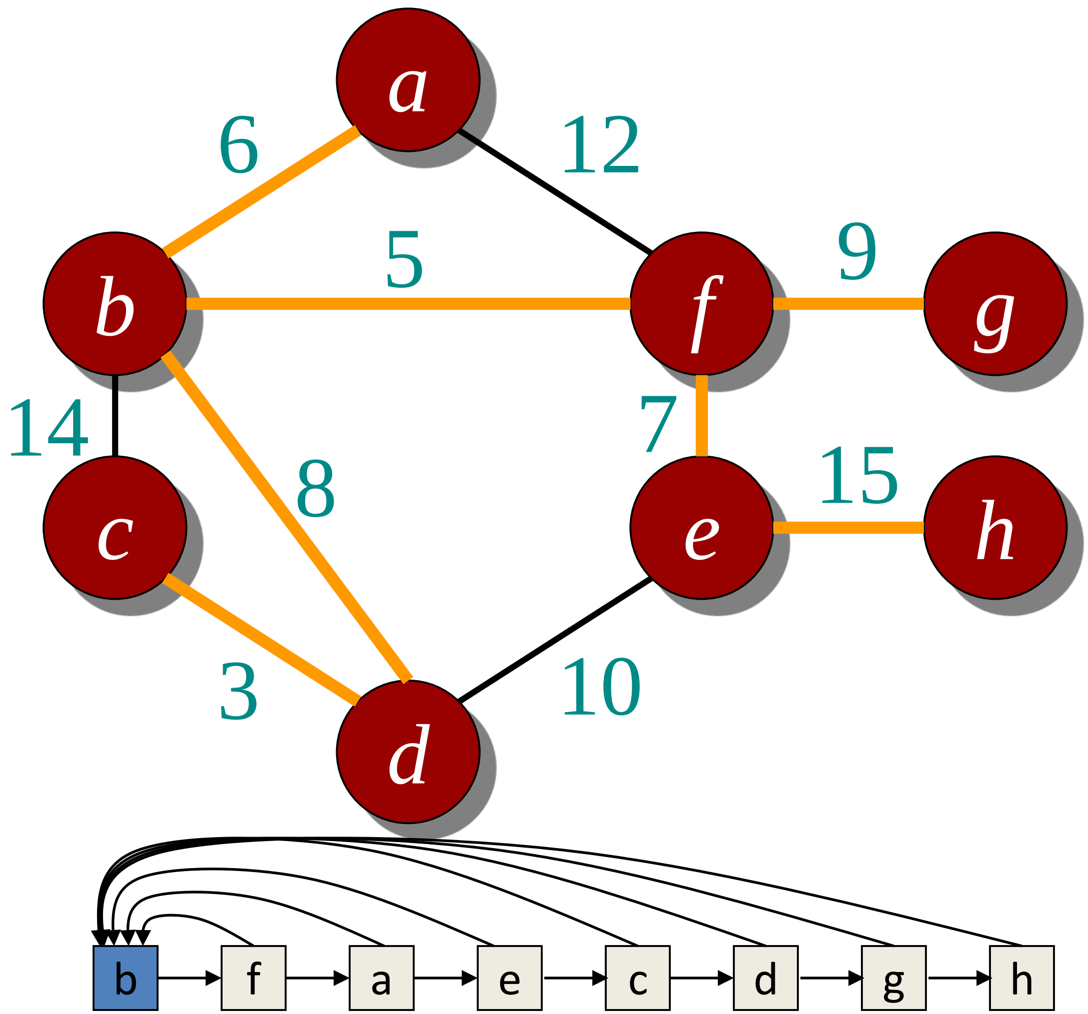
f-g: 9

d-e: 10

a-f: 12

b-c: 14

e-h: 15



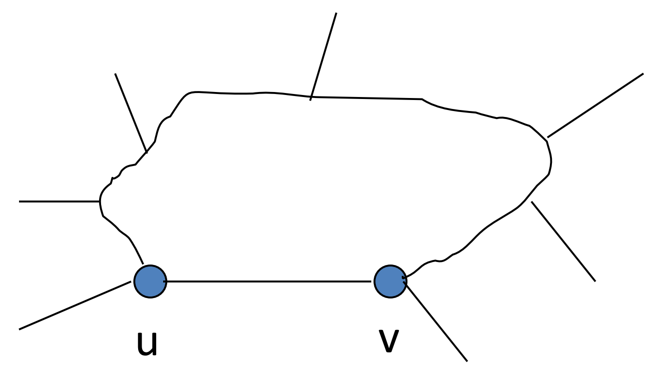
Running Time of Kruskal's Algorithm

- Kruskal's Algorithm consists of two stages.
  - Initializing the set A in line 1 takes O(1) time.
  - Sorting the edges by weight in line 4.
    - takes $O(E \lg E)$
  - Performing
    - $|V|$ MakeSet() operations — for loop in lines 2-3.
    - $|E|$ FindSet(), — for loop in lines 5-8.
    - $|V|$ - 1 Union(), — for loop in lines 5-8.
    - which takes $O(E \lg E)$
- The total running time is
  - $O(E \lg E)$
  - Observing that $|E| < |V|^2$, we have $\lg |E| = O(\lg V)$
  - So total running time becomes $O(E \lg V)$.

Claim

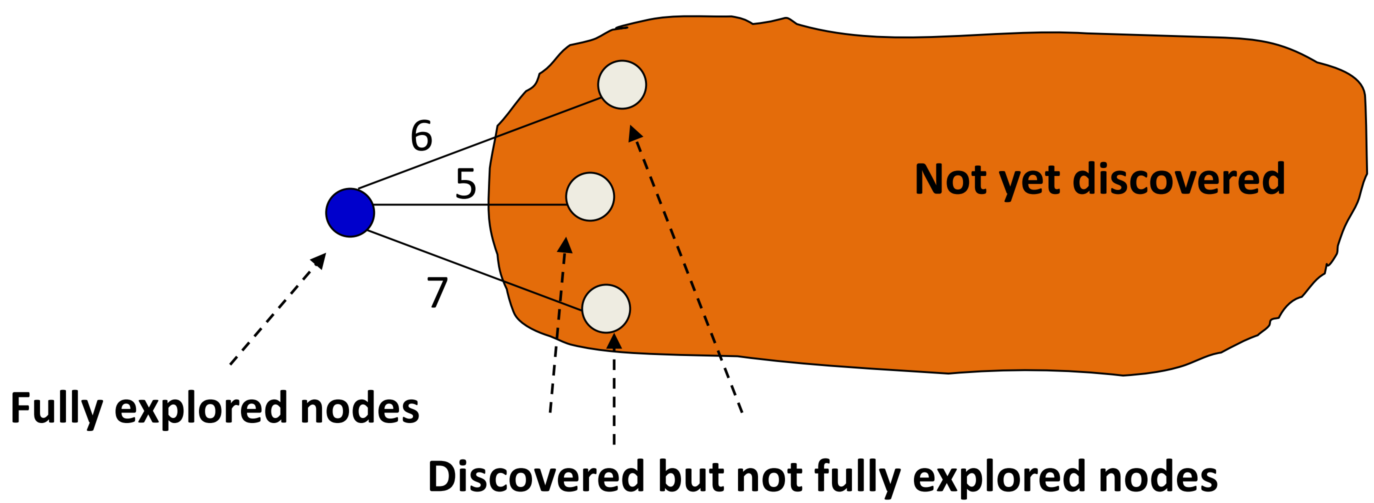
- If edge (u, v) is the lightest among all edges, (u, v) is in a MST
- Proof by contradiction:
  - Suppose that (u, v) is not in any MST
  - Given a MST T, if we connect (u, v), we create a cycle
  - Remove an edge in the cycle, have a new tree T’
  - W(T’) < W(T)

By the same argument, the second, third, …, lightest edges, if they do not create a cycle, must be in MST



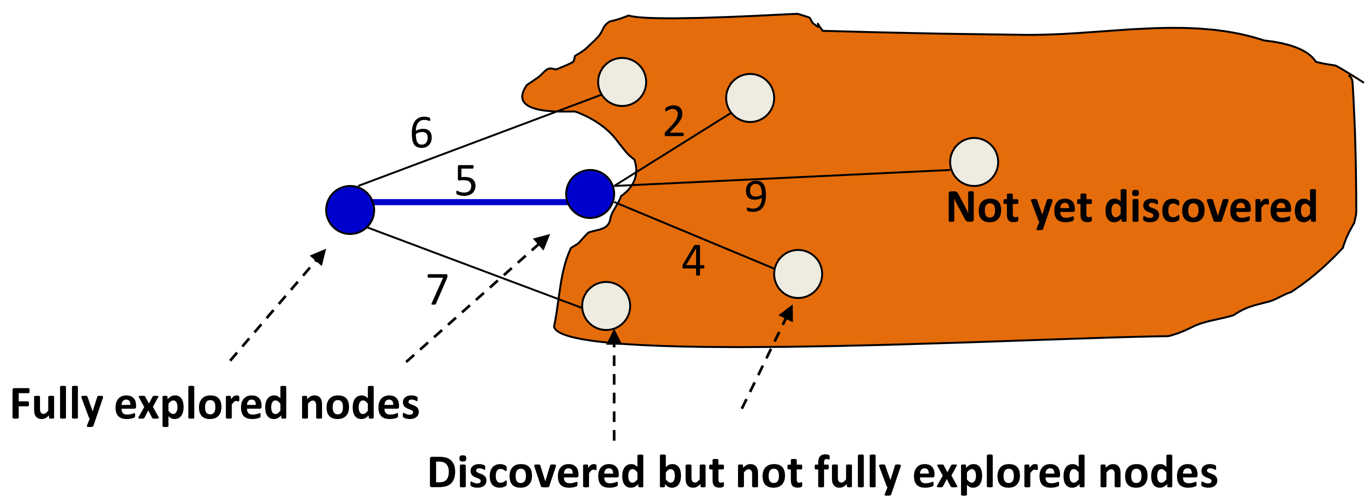
Prim's algorithm

- Basic idea:
  - Start from an arbitrary single node
    - A MST for a single node has no edge
  - Gradually build up a single larger and larger MST

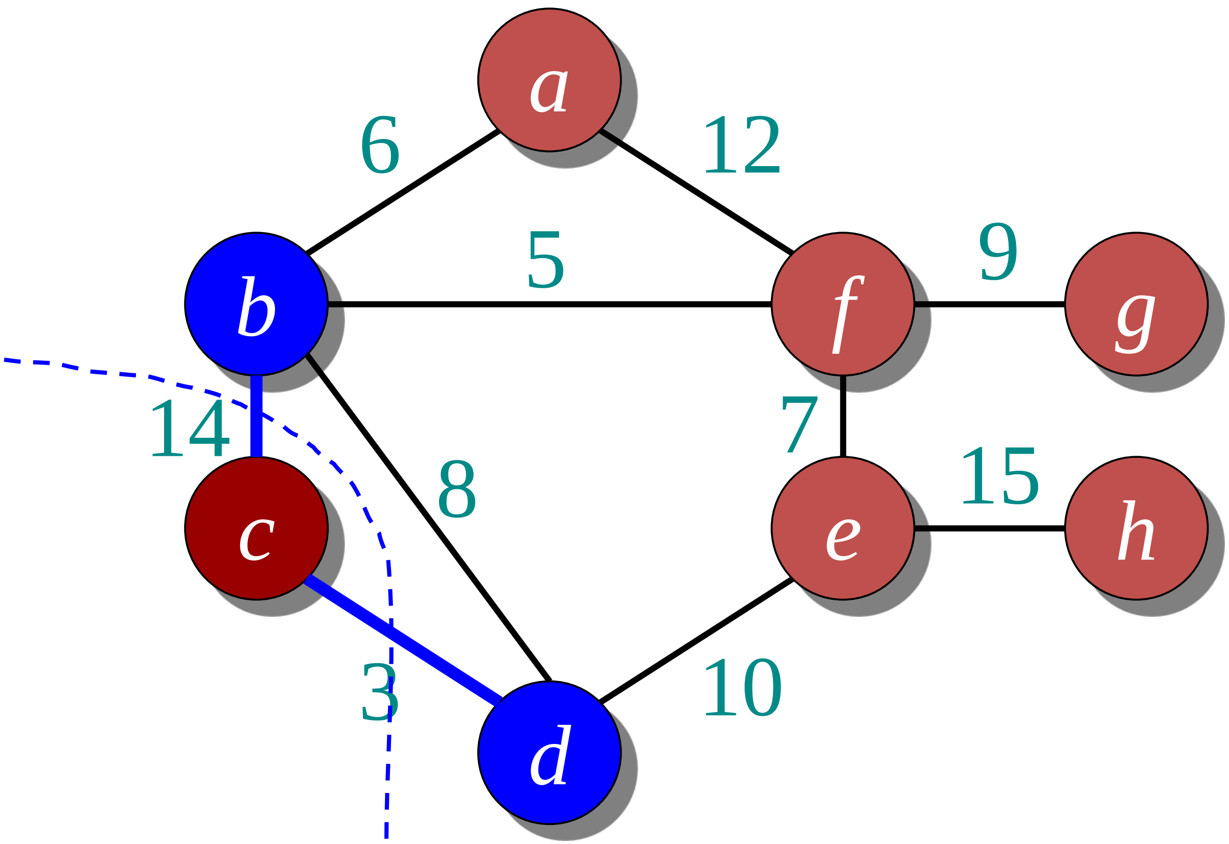


Prim’s algorithm

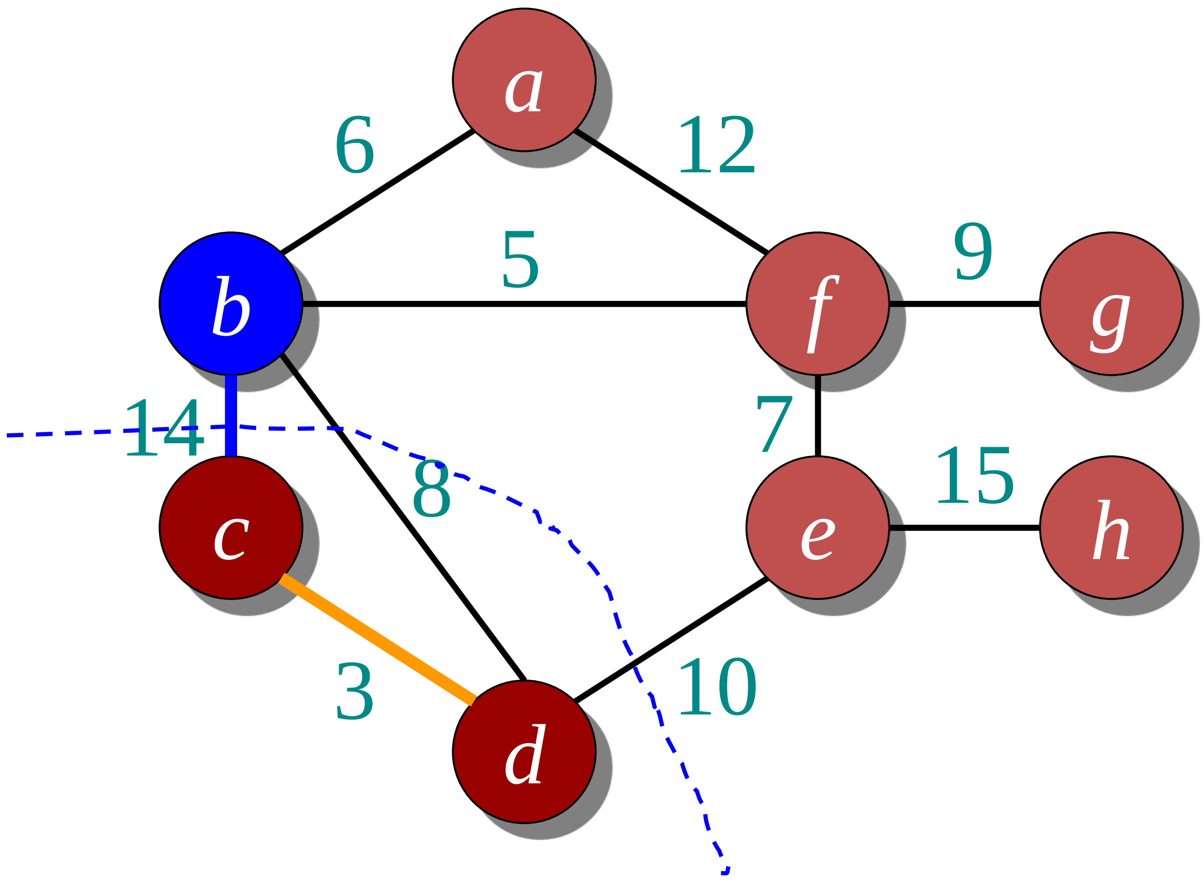
- Basic idea:
  - Start from an arbitrary single node
    - A MST for a single node has no edge
  - Gradually build up a single larger and larger MST



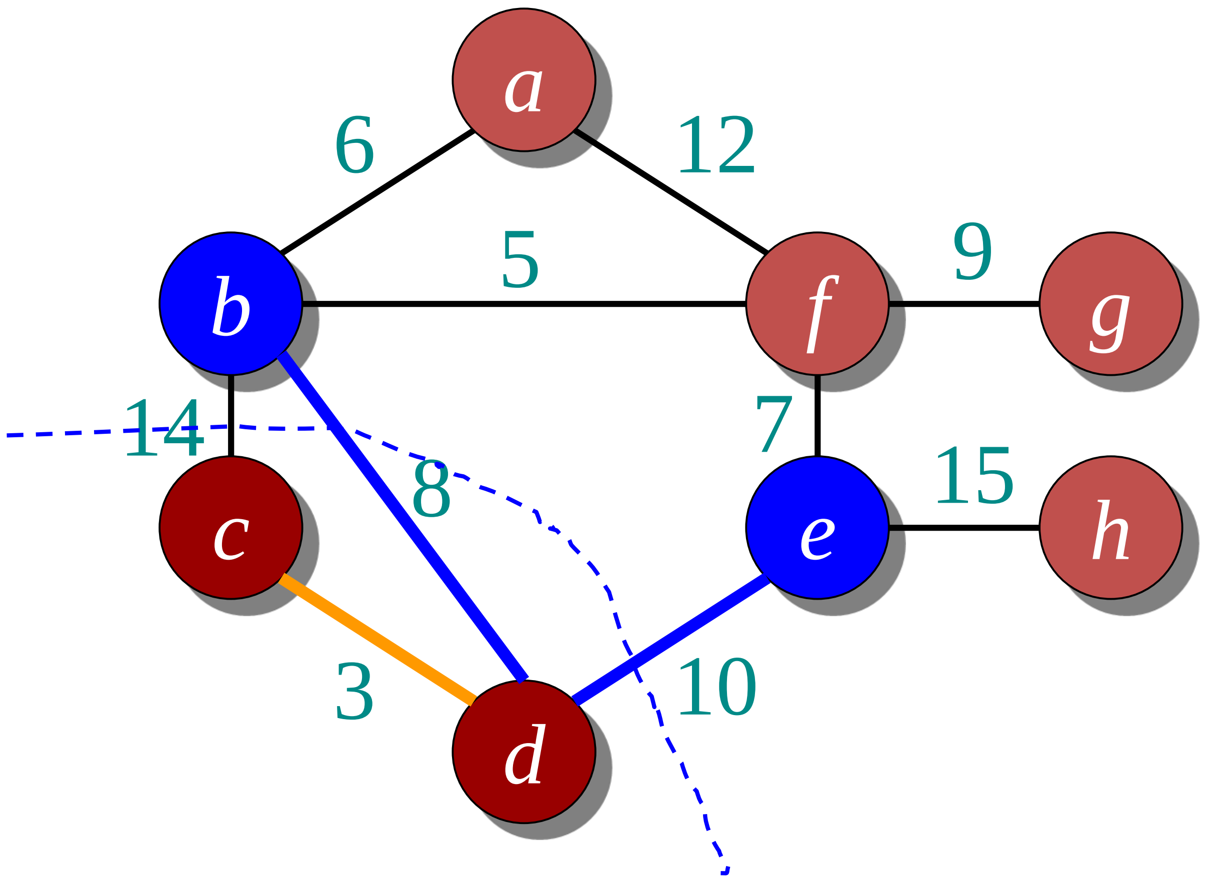
Example



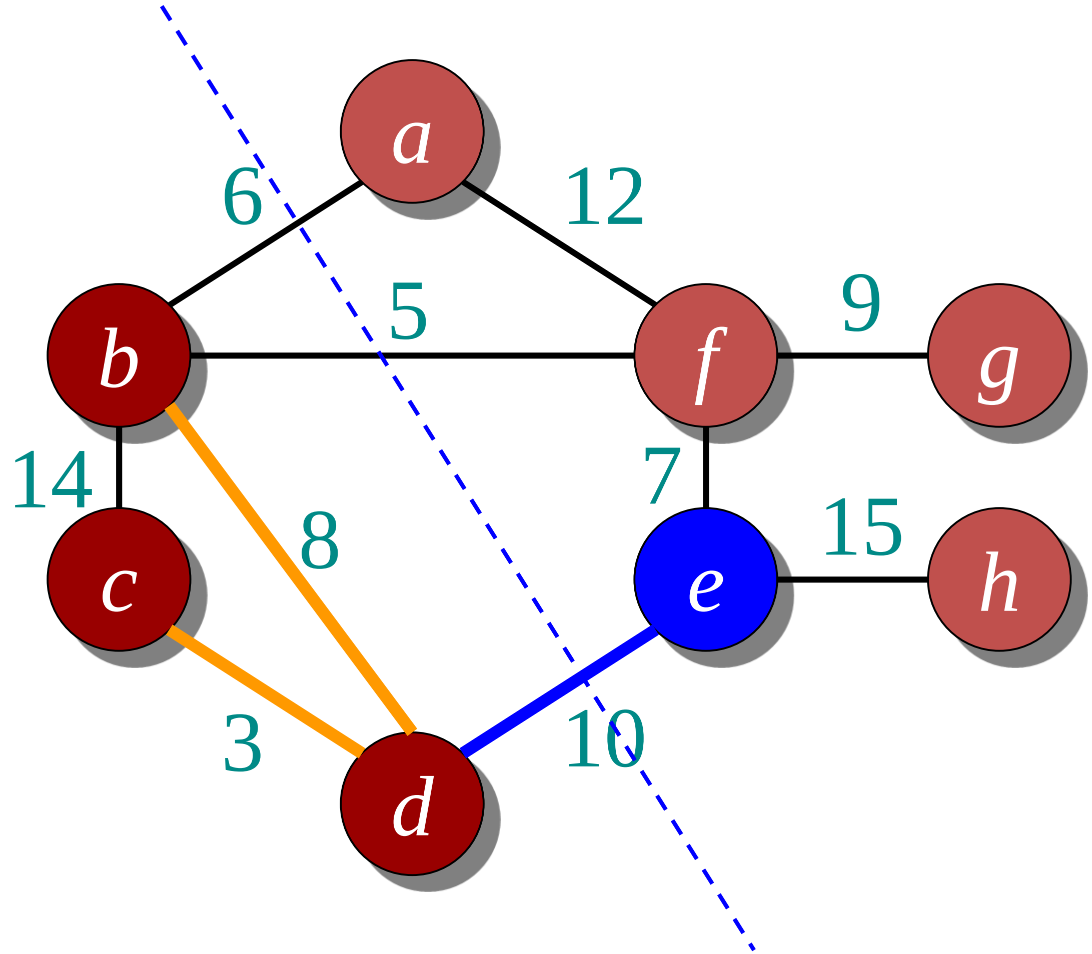
Example



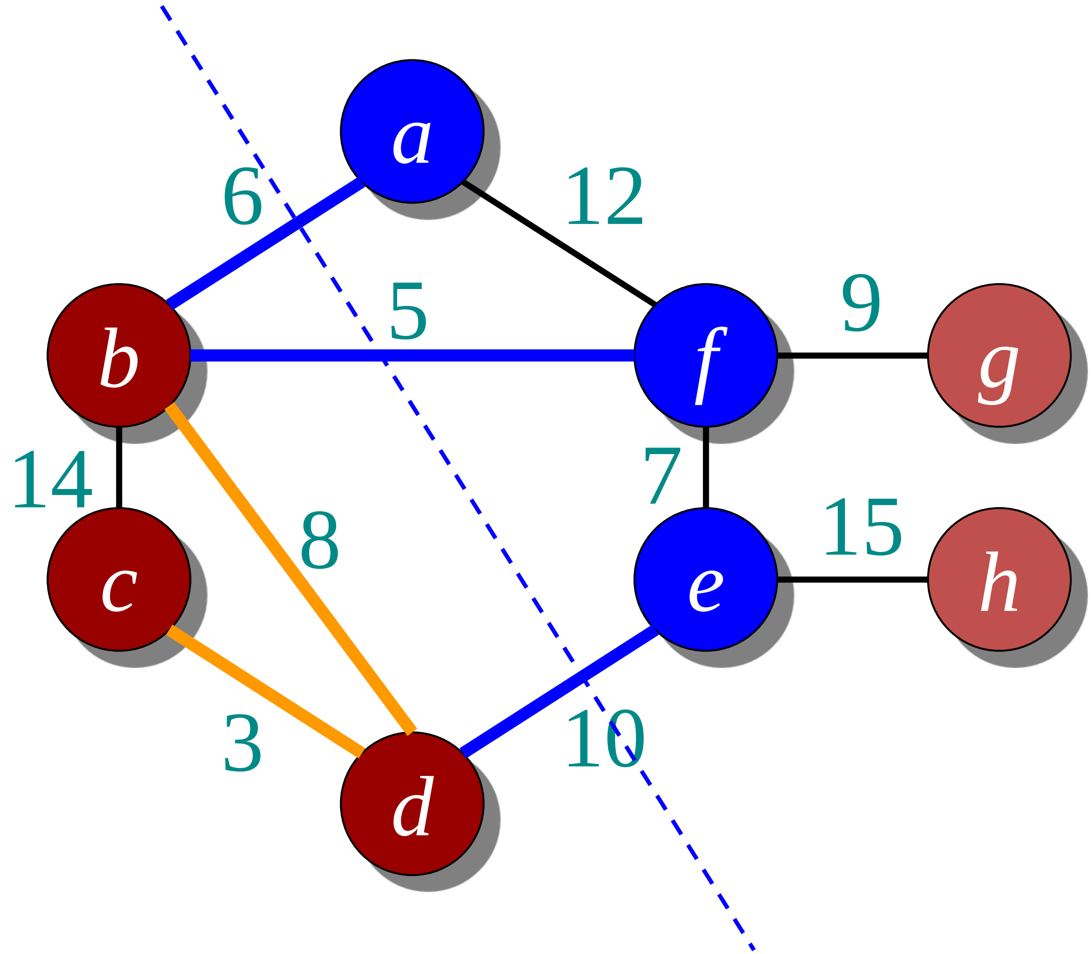
Example



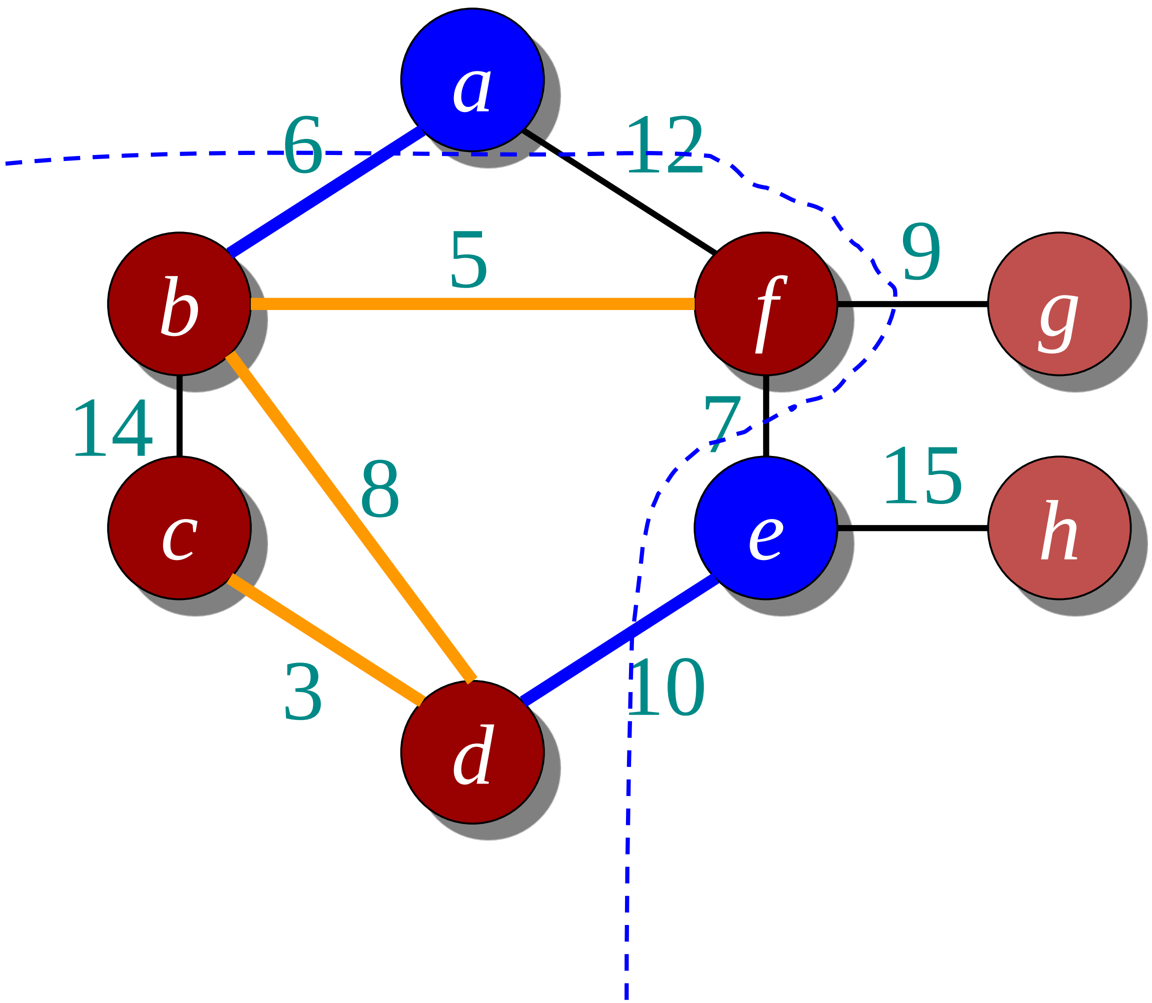
Example



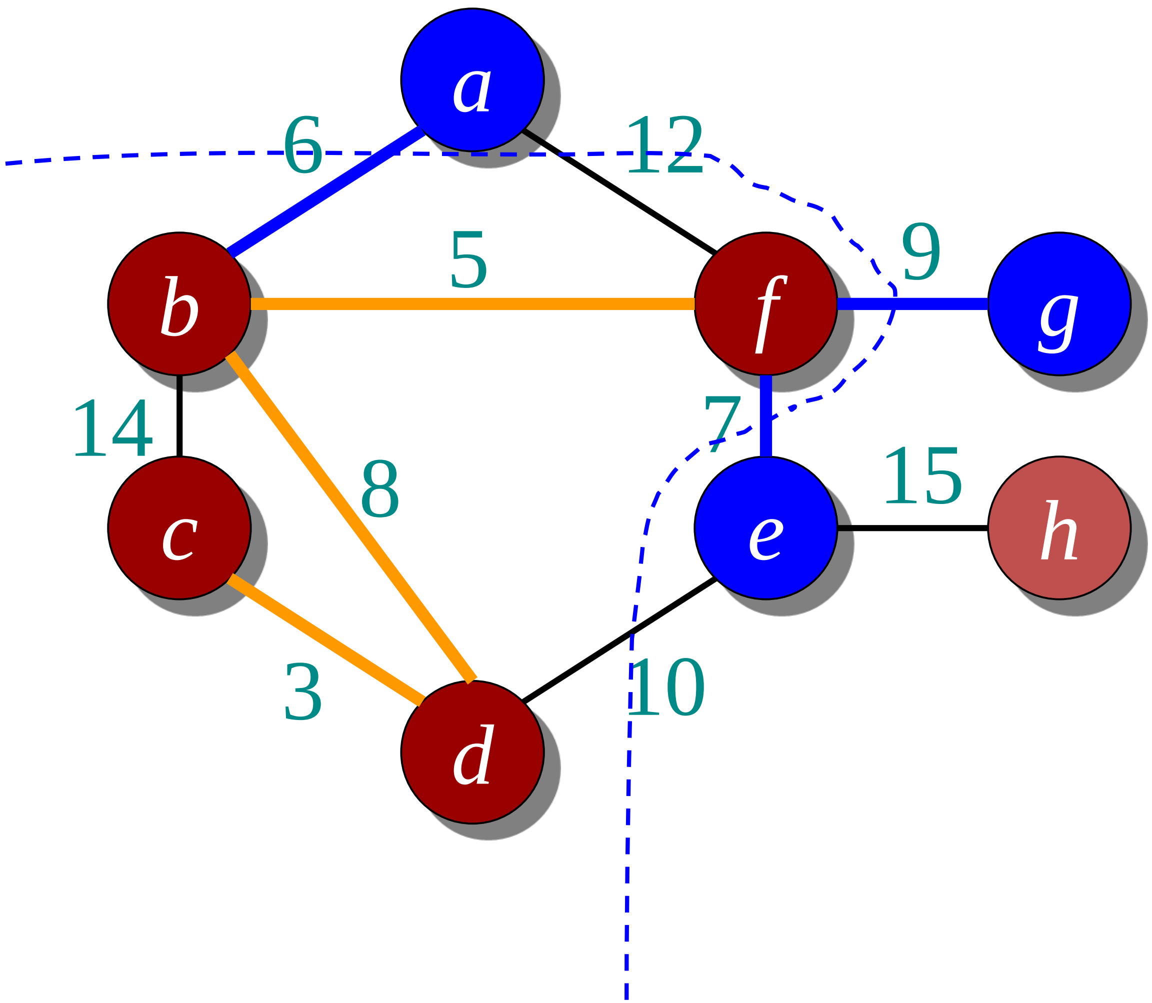
Example



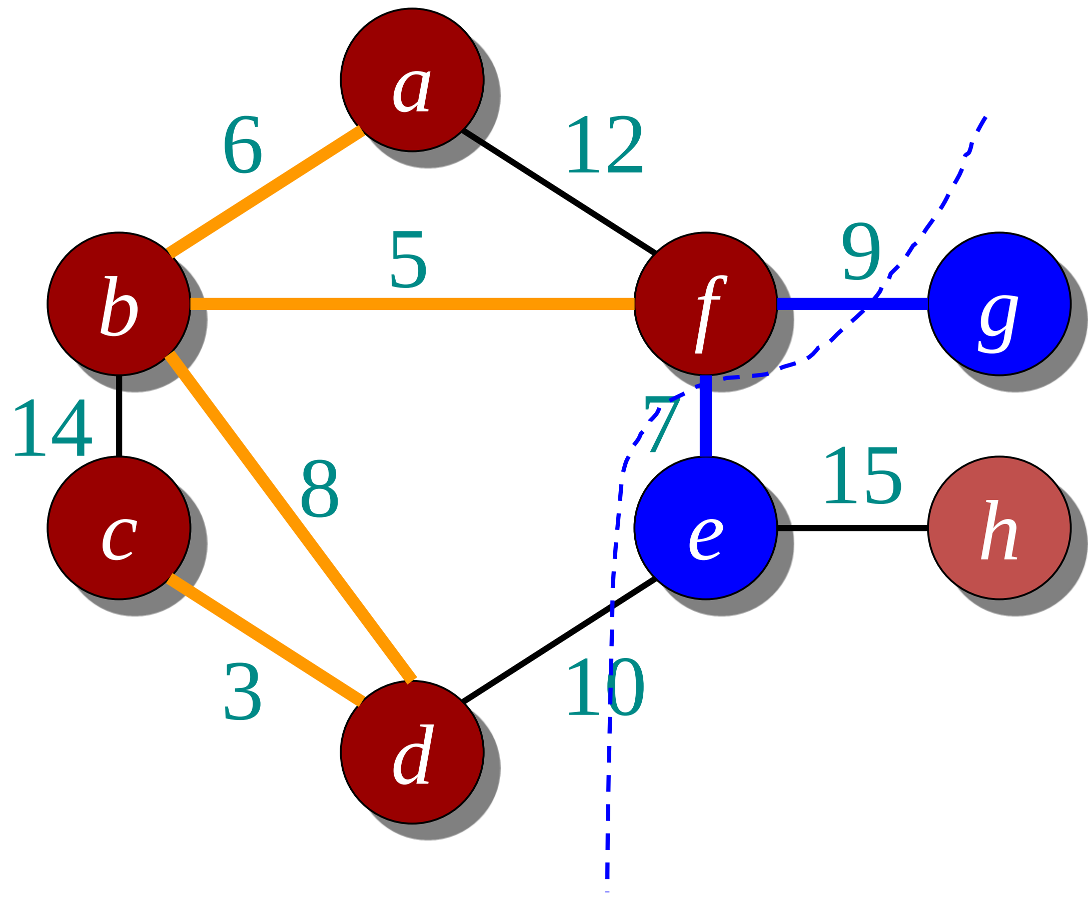
Example



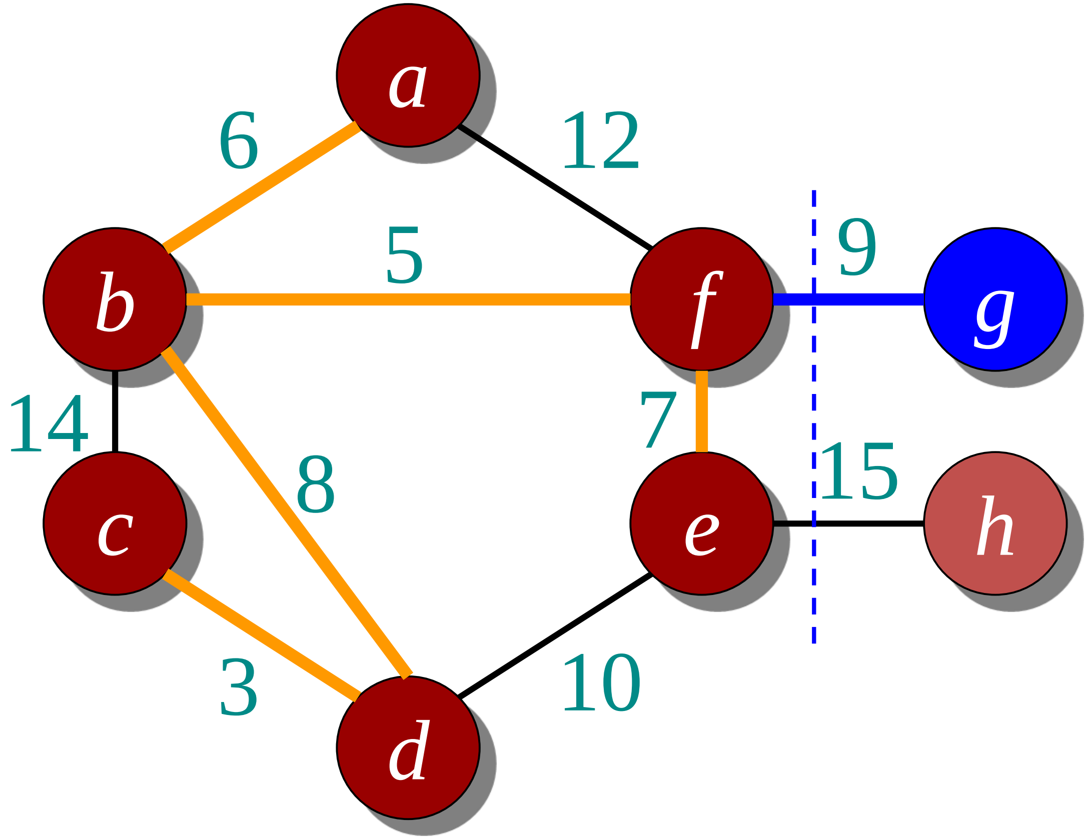
Example



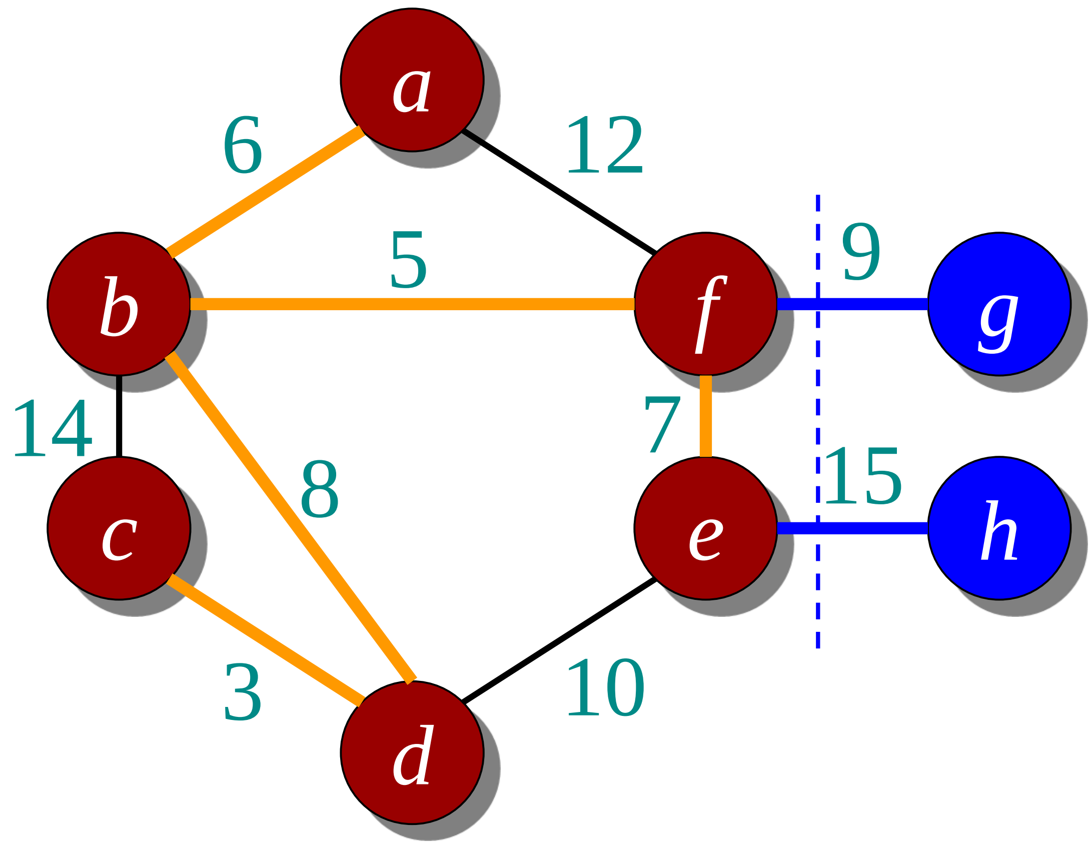
Example



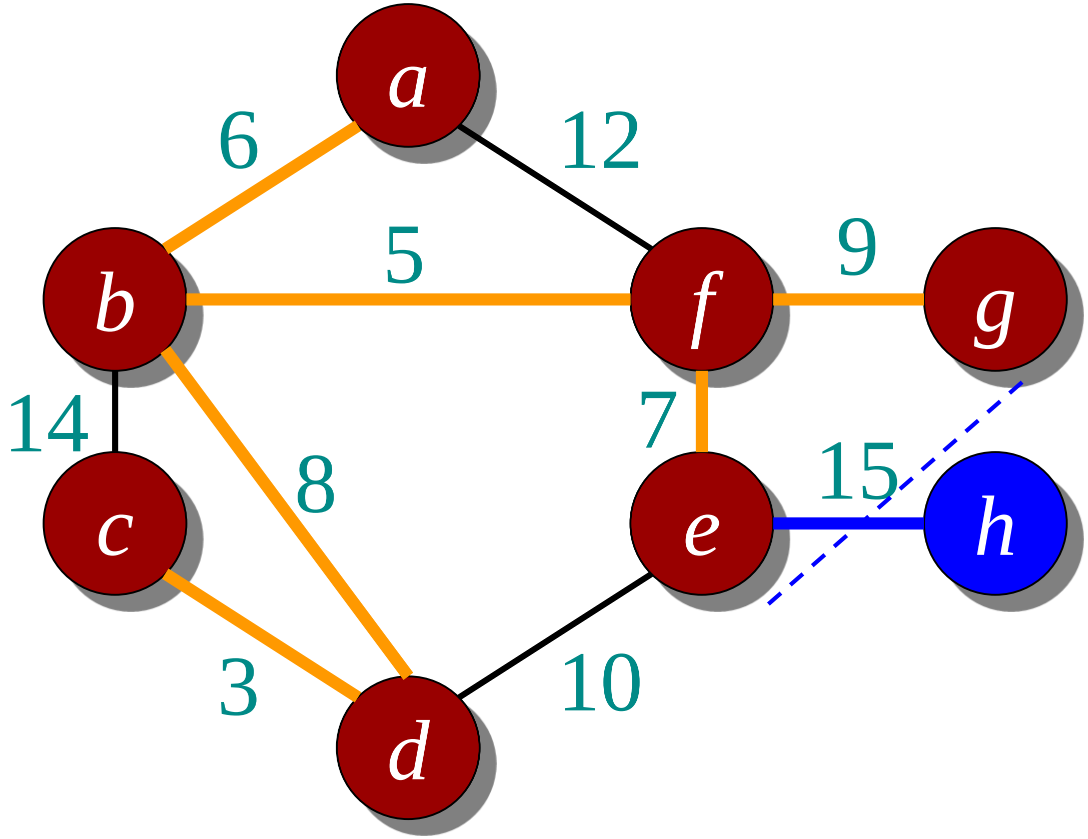
Example



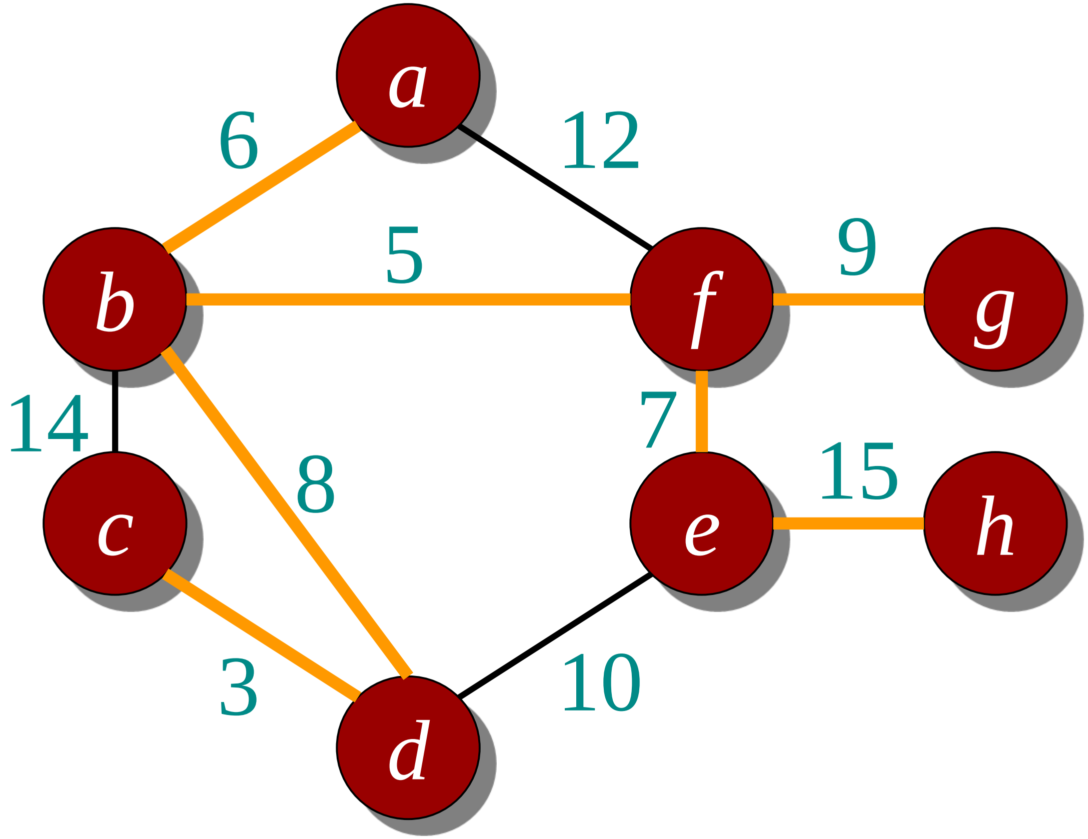
Example



Example



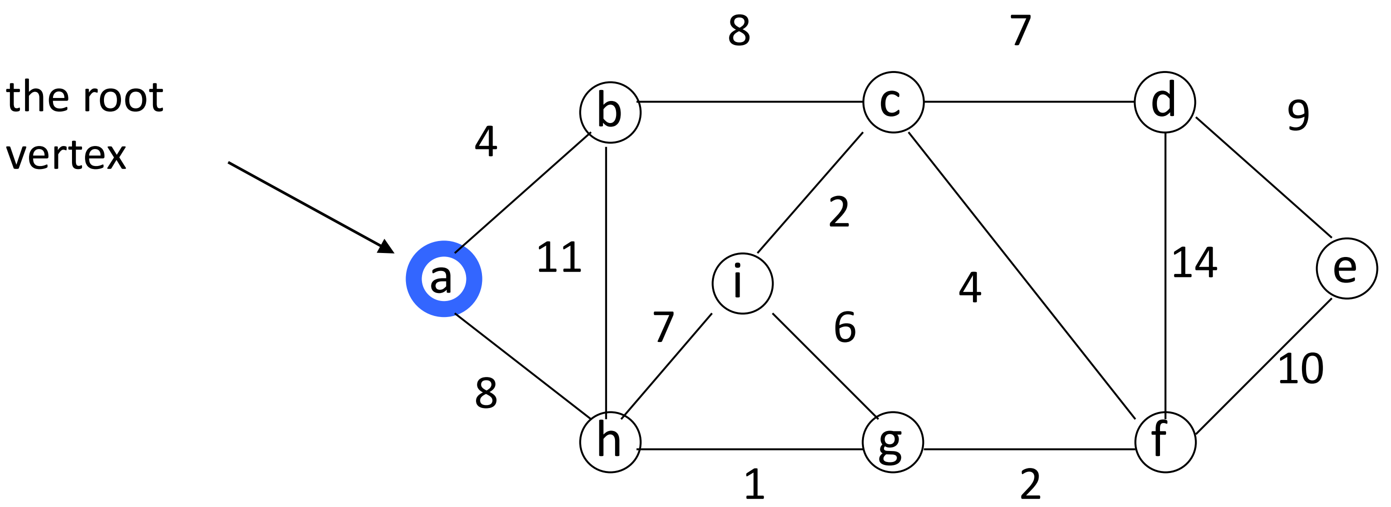
Example



Prim's Algorithm

```
MST-Prim(G,r)
Q ← V[G]
for each u ∈ Q
   key[u] ← ∞
key[r] ← 0
π[r] ← NIL
while Q ≠ ∅ do
  u ← ExtractMin(Q)
     for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
           π[v] ← u
           key[v] ← w(u,v)
```

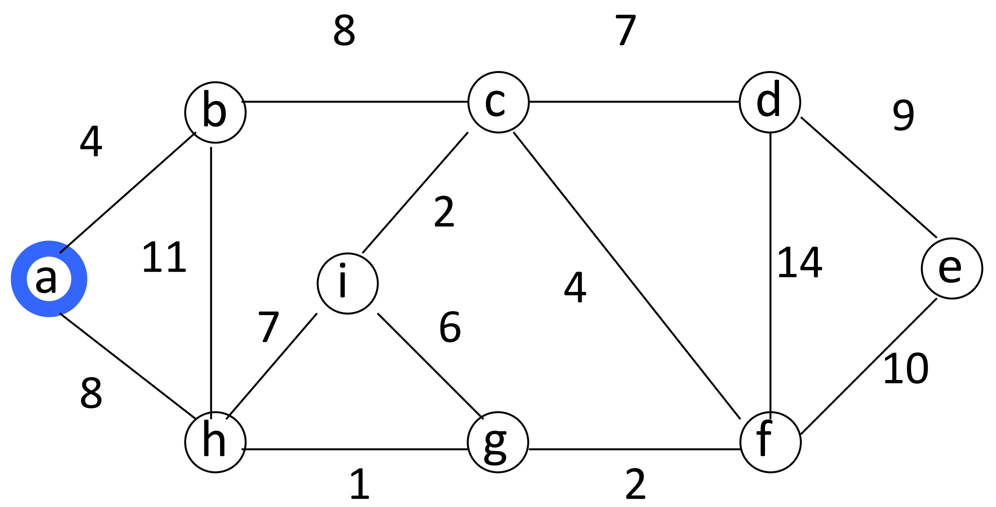
The Execution of Prim's Algorithm



V	a	b	c	d	e	f	g	h	i
T	1	0	0	0	0	0	0	0	0
Key	0	-	-	-	-	-	-	-	-
π	-1	-	-	-	-	-	-	-	-

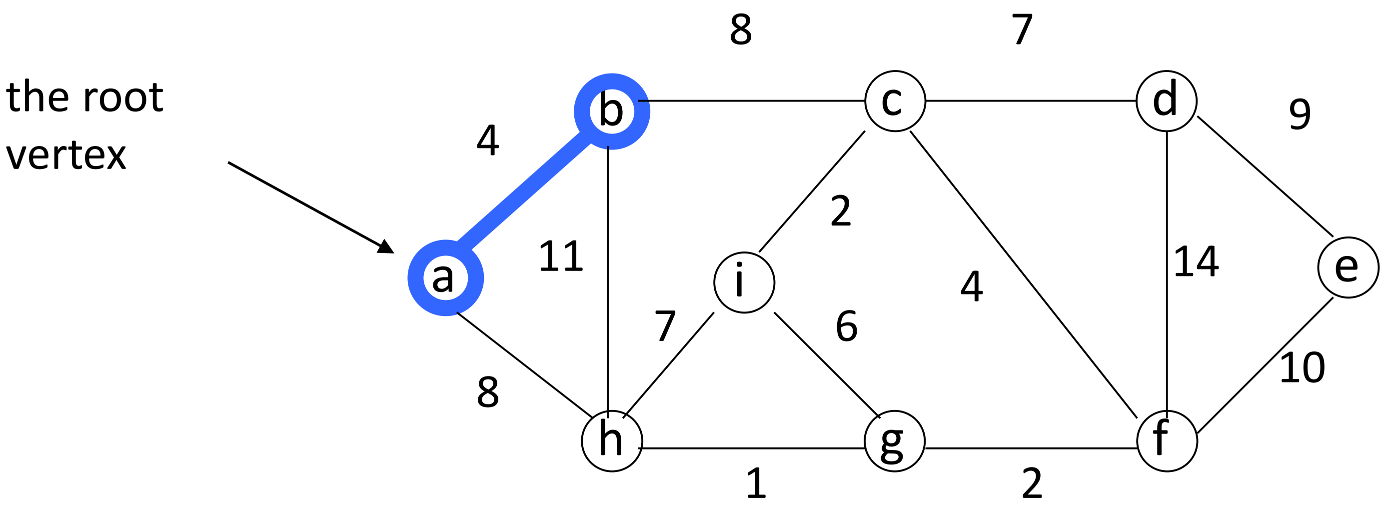
The Execution of Prim's Algorithm

the root vertex



V	a	b	c	d	e	f	g	h	i
T	1	**0**	0	0	0	0	0	0	0
Key	0	**4**	-	-	-	-	-	8	-
π	-1	**a**	-	-	-	-	-	a	-

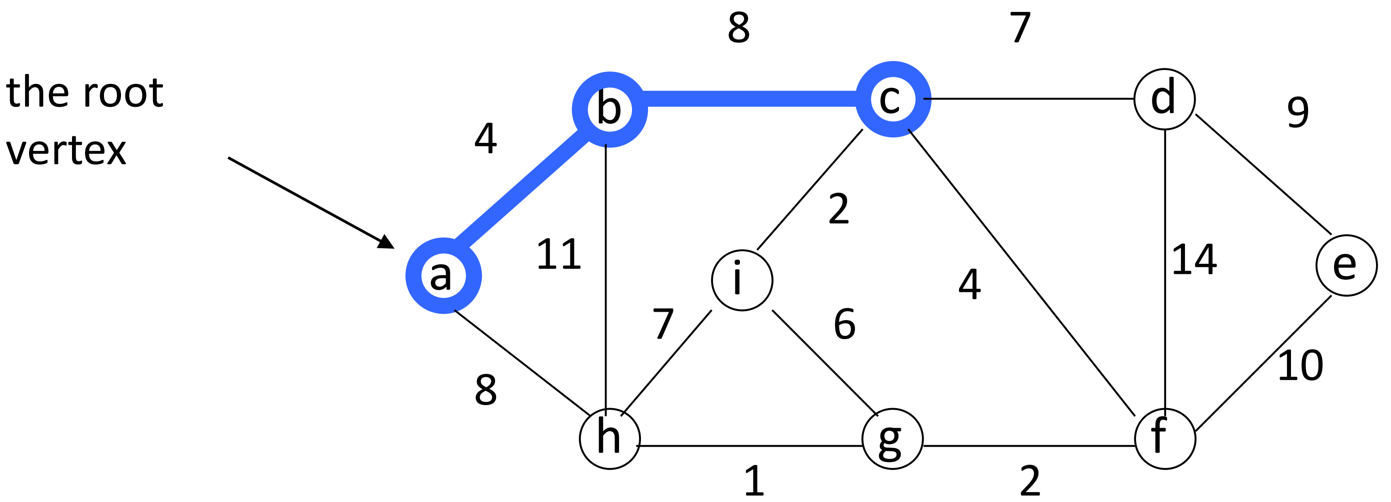
The Execution of Prim's Algorithm



Important: Update Key[v] only if T[v]==0

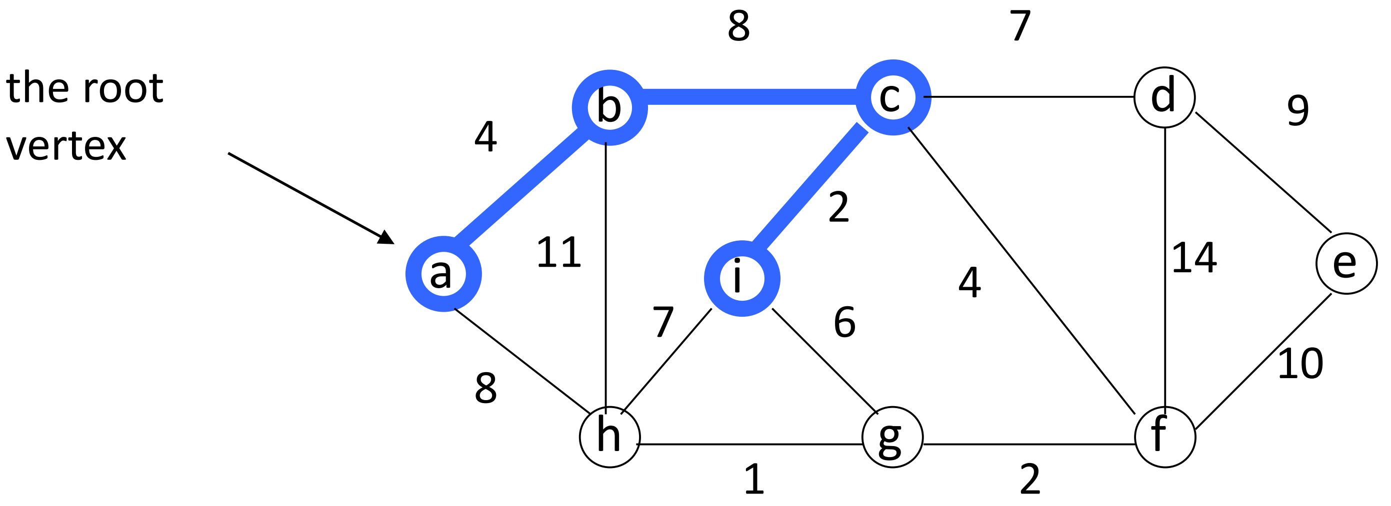
V	a	b	c	d	e	f	g	h	i
T	1	1	**0**	0	0	0	0	0	0
Key	0	4	**8**	-	-	-	-	8	-
π	-1	a	**b**	-	-	-	-	a	-

The Execution of Prim's Algorithm



V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	0	0	0	**0**
Key	0	4	8	7	-	4	-	8	**2**
π	-1	a	b	c	-	c	-	a	**c**

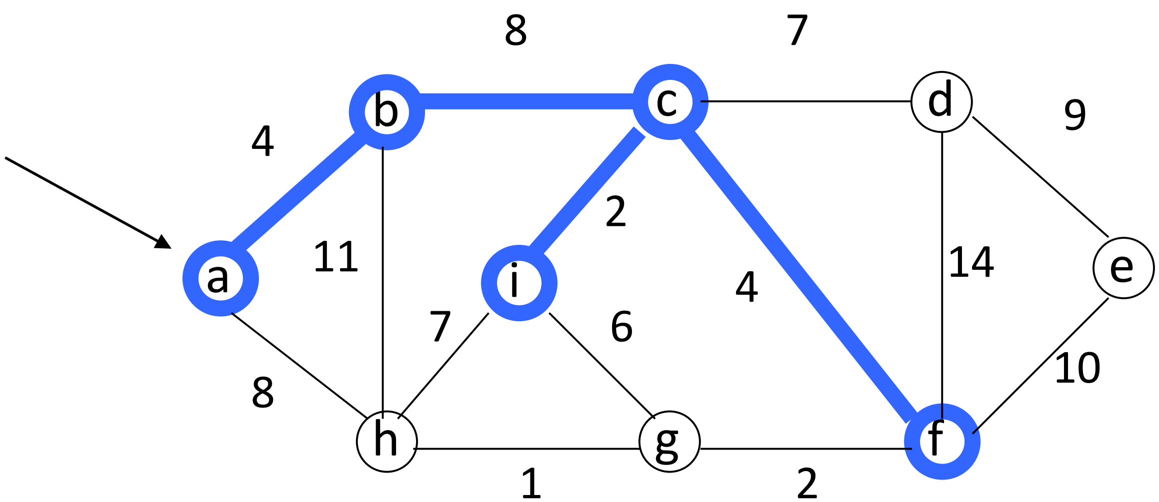
The Execution of Prim's Algorithm



V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	**0**	0	0	1
Key	0	4	8	7	-	**4**	6	7	2
π	-1	a	b	c	-	**c**	i	i	c

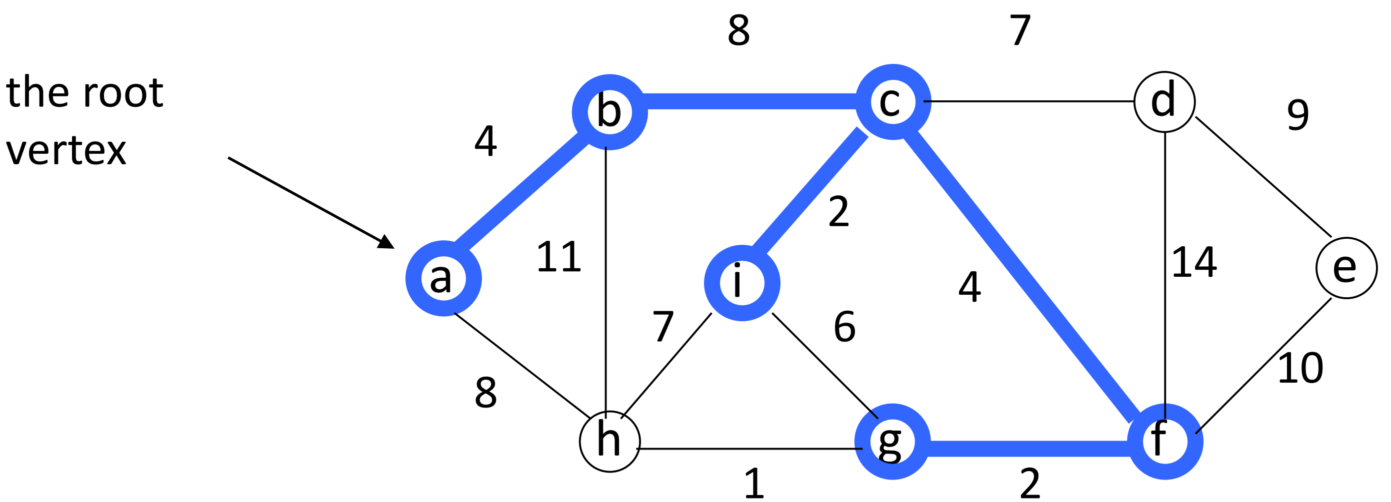
The Execution of Prim's Algorithm

the root vertex



V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	**1**	**0**	0	1
Key	0	4	8	7	10	**4**	**2**	7	2
π	-1	a	b	c	f	**c**	**f**	i	c

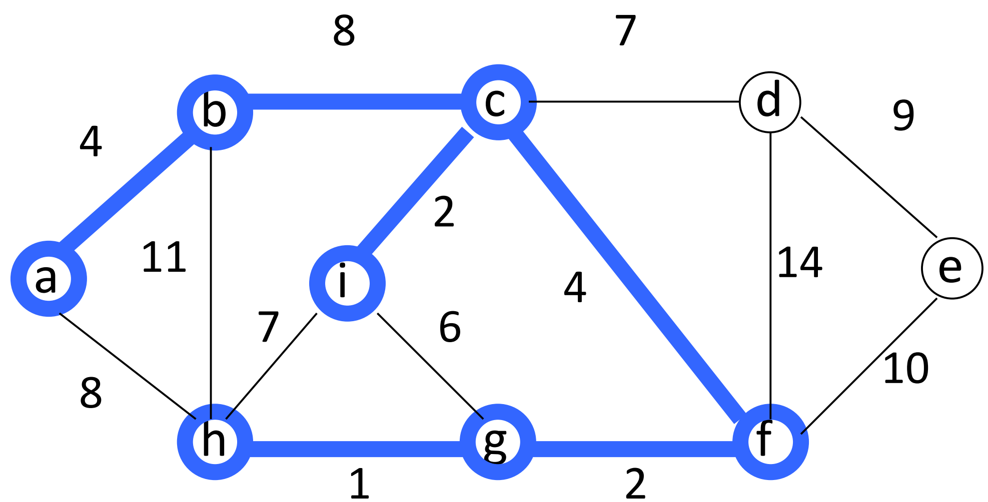
The Execution of Prim's Algorithm



V	a	b	c	d	e	f	g	h	i
T	1	1	1	0	0	1	**1**	**0**	1
Key	0	4	8	7	10	4	**2**	**1**	2
π	-1	a	b	c	f	c	**f**	**g**	c

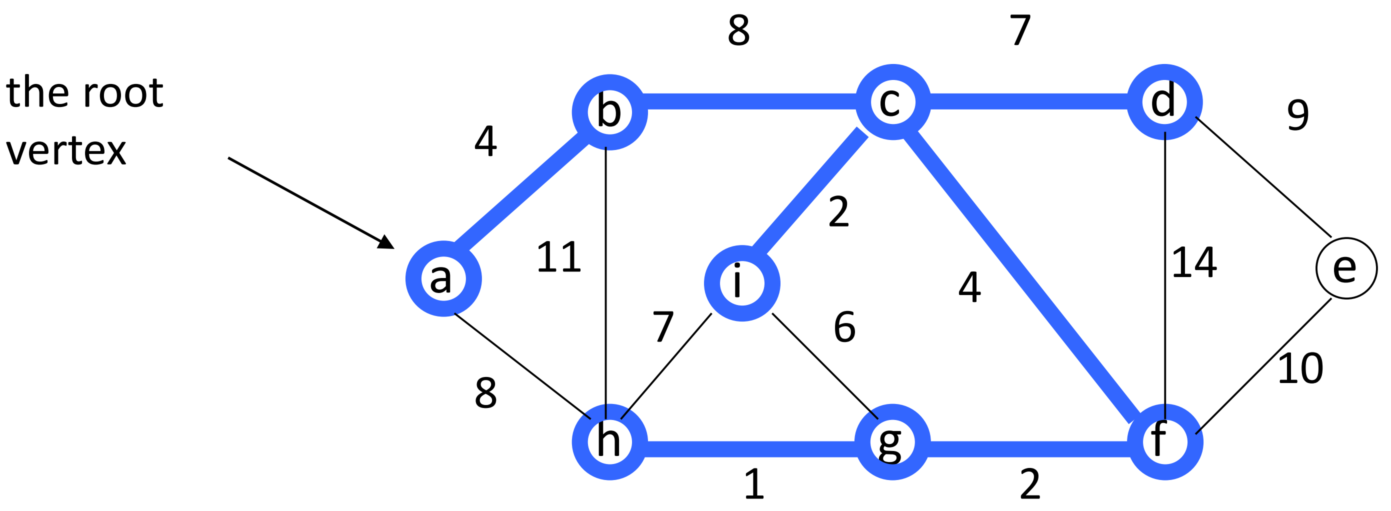
The Execution of Prim's Algorithm

the root vertex



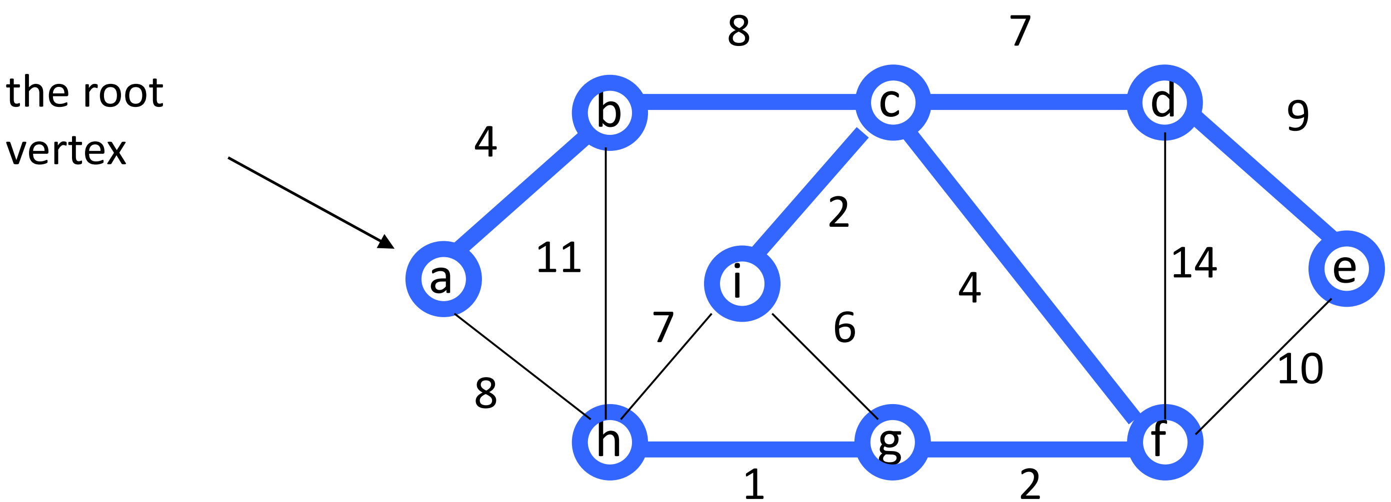
V	a	b	c	d	e	f	g	h	i
T	1	1	1	**0**	0	1	1	**1**	1
Key	0	4	8	**7**	10	4	2	**1**	2
π	-1	a	b	**c**	f	c	f	**g**	c

The Execution of Prim's Algorithm



V	a	b	c	d	e	f	g	h	i
T	1	1	1	**1**	**0**	1	1	1	1
Key	0	4	8	**7**	**9**	4	2	1	2
π	-1	a	b	**c**	**d**	c	f	g	c

The Execution of Prim's Algorithm



V	a	b	c	d	e	f	g	h	i
T	1	1	1	1	**1**	1	1	1	1
Key	0	4	8	7	**9**	4	2	1	2
π	-1	a	b	c	**d**	c	f	g	c

Complexity: Prim's Algorithm

```
MST-Prim(G,r)
Q ← V[G]
for each u ∈ Q                                        O(V)
   key[u] ← ∞
key[r] ← 0
π[r] ← NIL
while Q ≠ ∅ do                                        O(V)
  u ← ExtractMin(Q)                                   Heap: O(lgV)
     for each v ∈ Adj[u] do                           Overall: O(E)
        if v ∈ Q and w(u,v) < key[v] then
           π[v] ← u
           key[v] ← w(u,v)                            Decrease Key: O(lgV)
```

Overall complexity: O(***V***)+O(***V*** lg ***V***+***E*** lg ***V***) = O(***E*** lg ***V***)

Summary

Kruskal's algorithm

1. Select the shortest edge in a network
2. Select the next shortest edge which does not create a cycle
3. Repeat step 2 until all vertices have been connected

Prim's algorithm

1. Select any vertex
2. Select the shortest edge connected to that vertex
3. Select the shortest edge connected to any vertex already connected
4. Repeat step 3 until all vertices have been connected