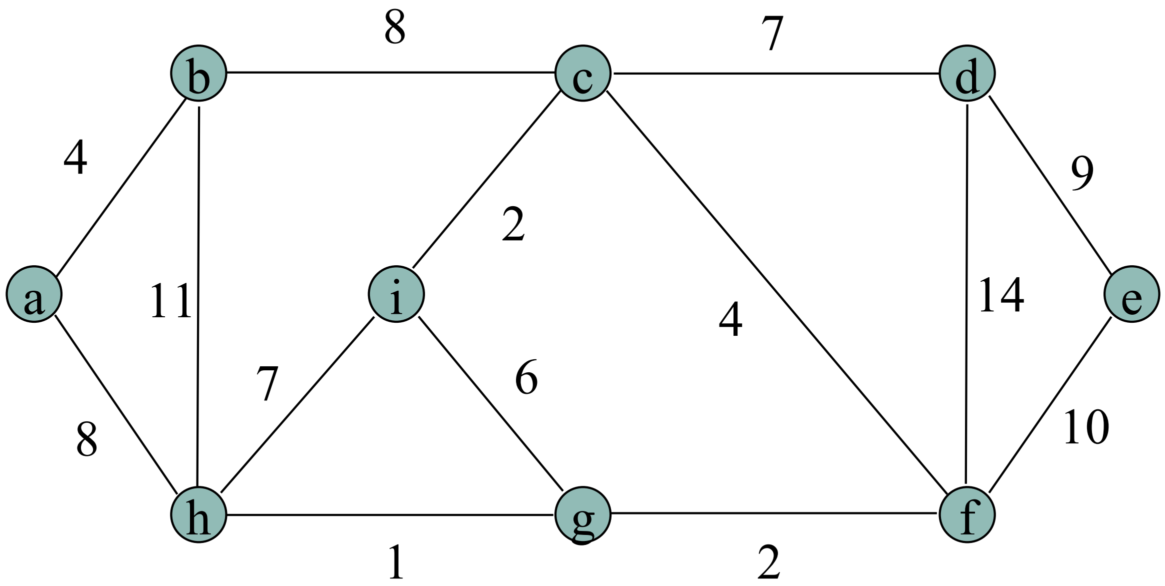
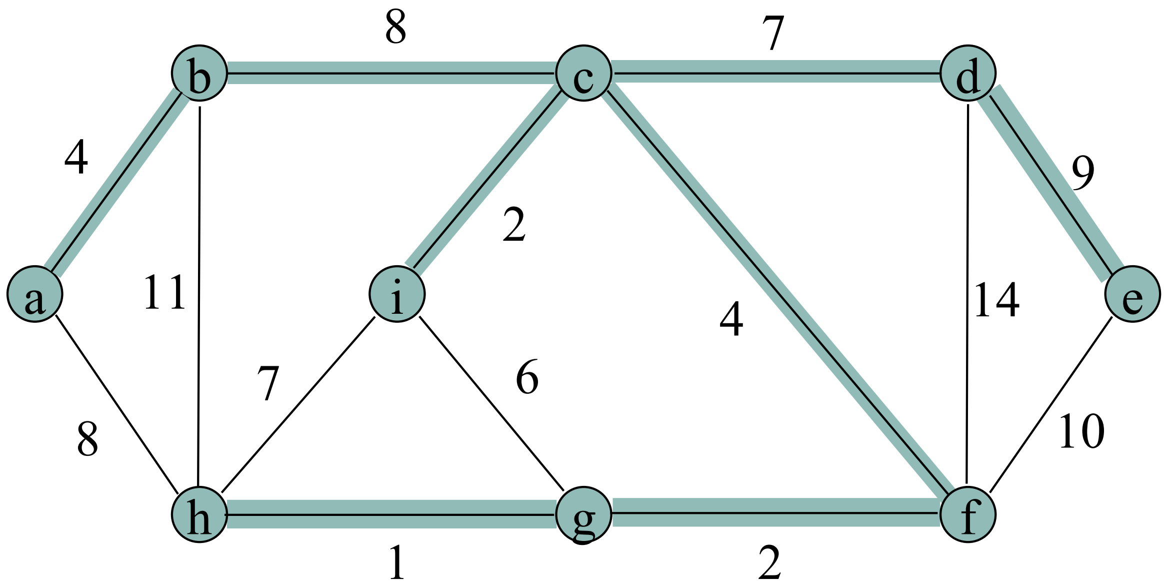
Minimum Spanning Trees

Slides from Heejin Park

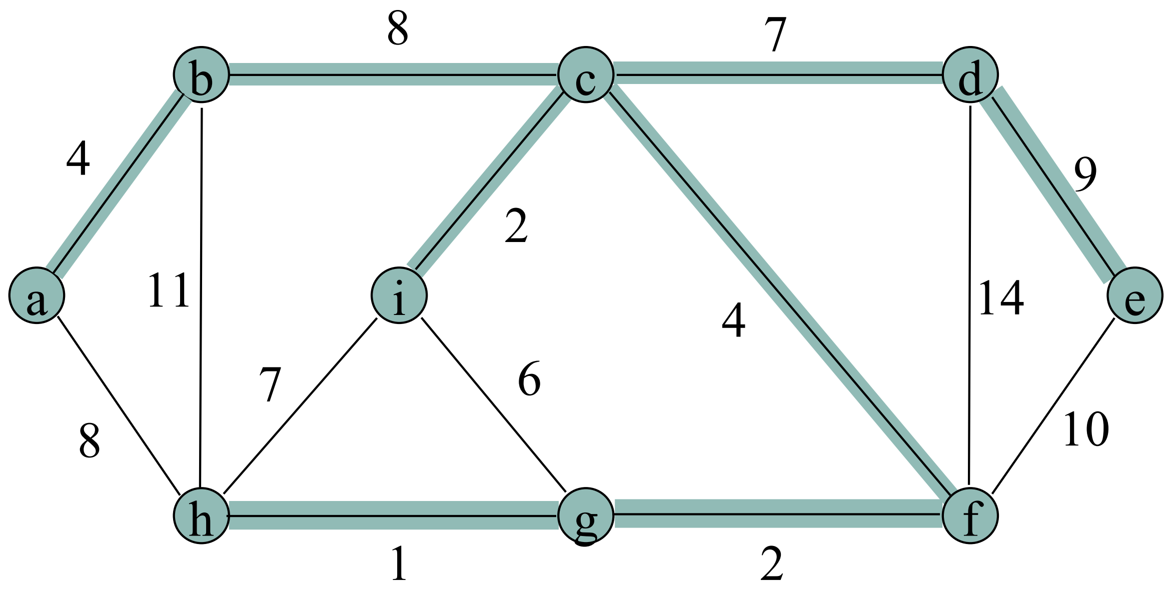
- Weighted undirected graph $G = (V, E)$
  - For each edge $(u, v) \in E$, we have a weight $w(u, v)$.

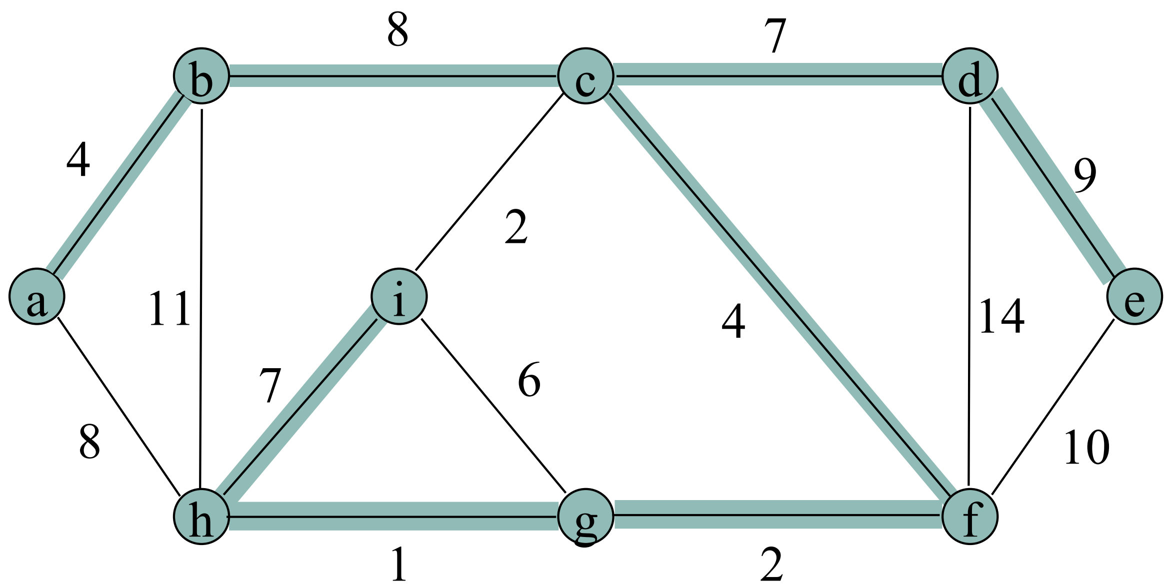


- **A *spanning tree*** for G.
  - A tree containing all of the vertices in G and edges of the tree are selected from the edges in G.



  - There are many spanning trees.


8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2


8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

- ***Cost of a spanning tree***

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

- ***Minimum-spanning-tree problem***
  - Finding a spanning tree whose cost is the smallest.
  - T is acyclic and connects all of the vertices → a tree

- GENERIC-MST

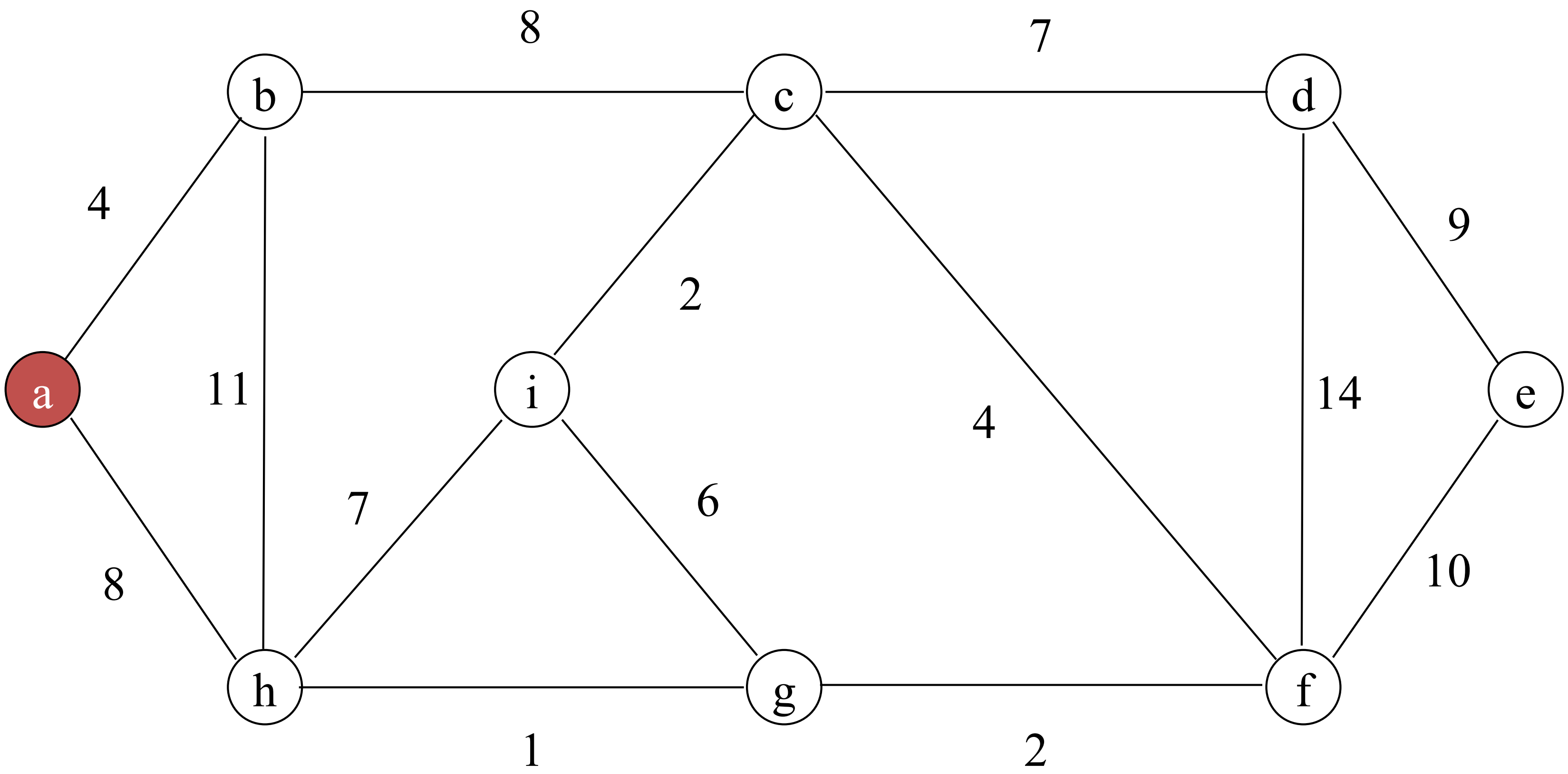
```
GENERIC-MST(G, w)
A = Ø
while A does not form a spanning tree
        do find an edge (u, v) that is safe for A
        A = A ∪ {(u, v)}
return A
```

– It grows the minimum spanning tree one edge at a time.

– It adds an edge (u, v) to A such that $A \cup \{(u, v)\}$ is also a subset of some minimum spanning tree.

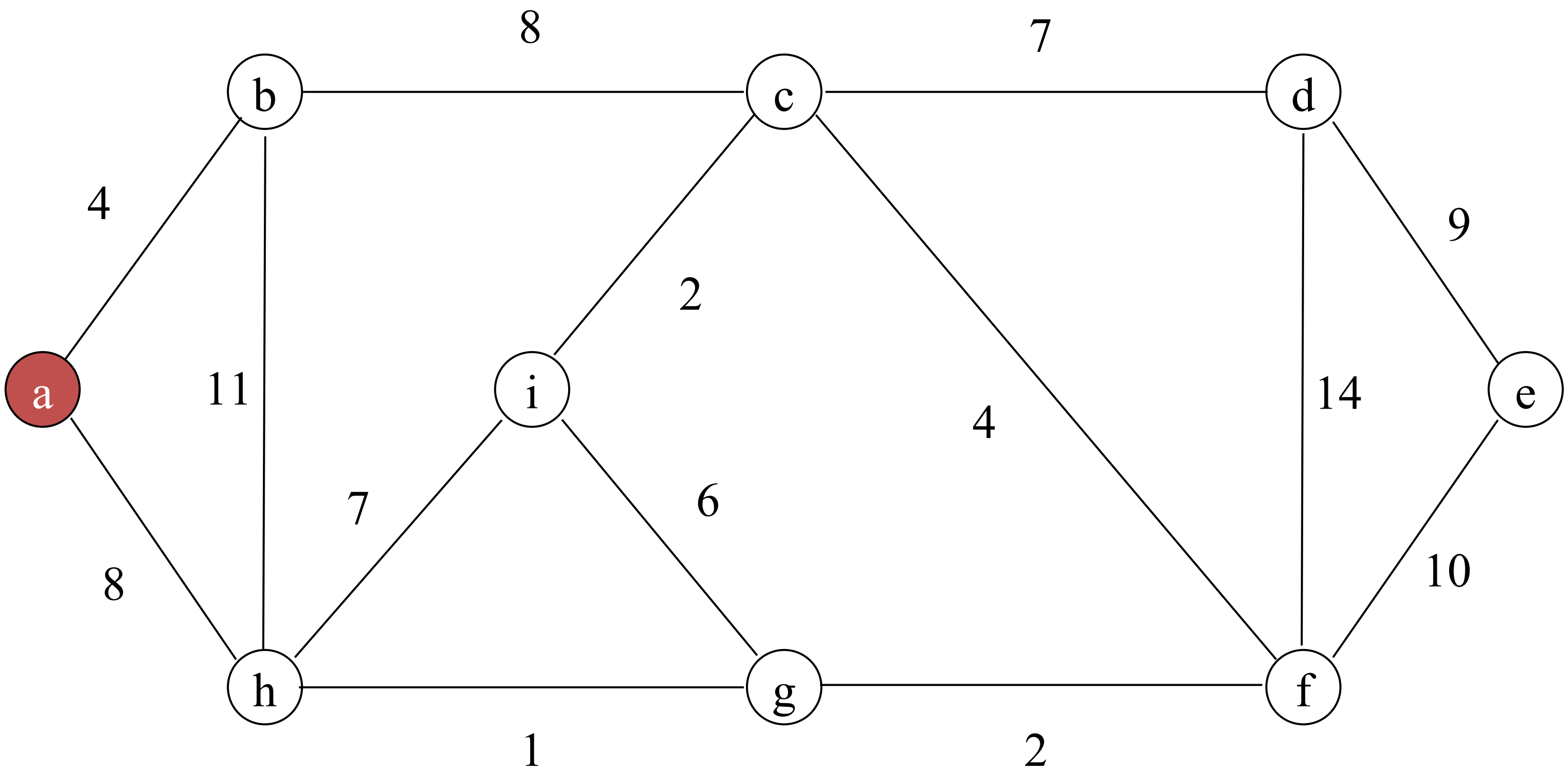
  - Call such an edge a ***safe edge*** for A.

Prim's Algorithm



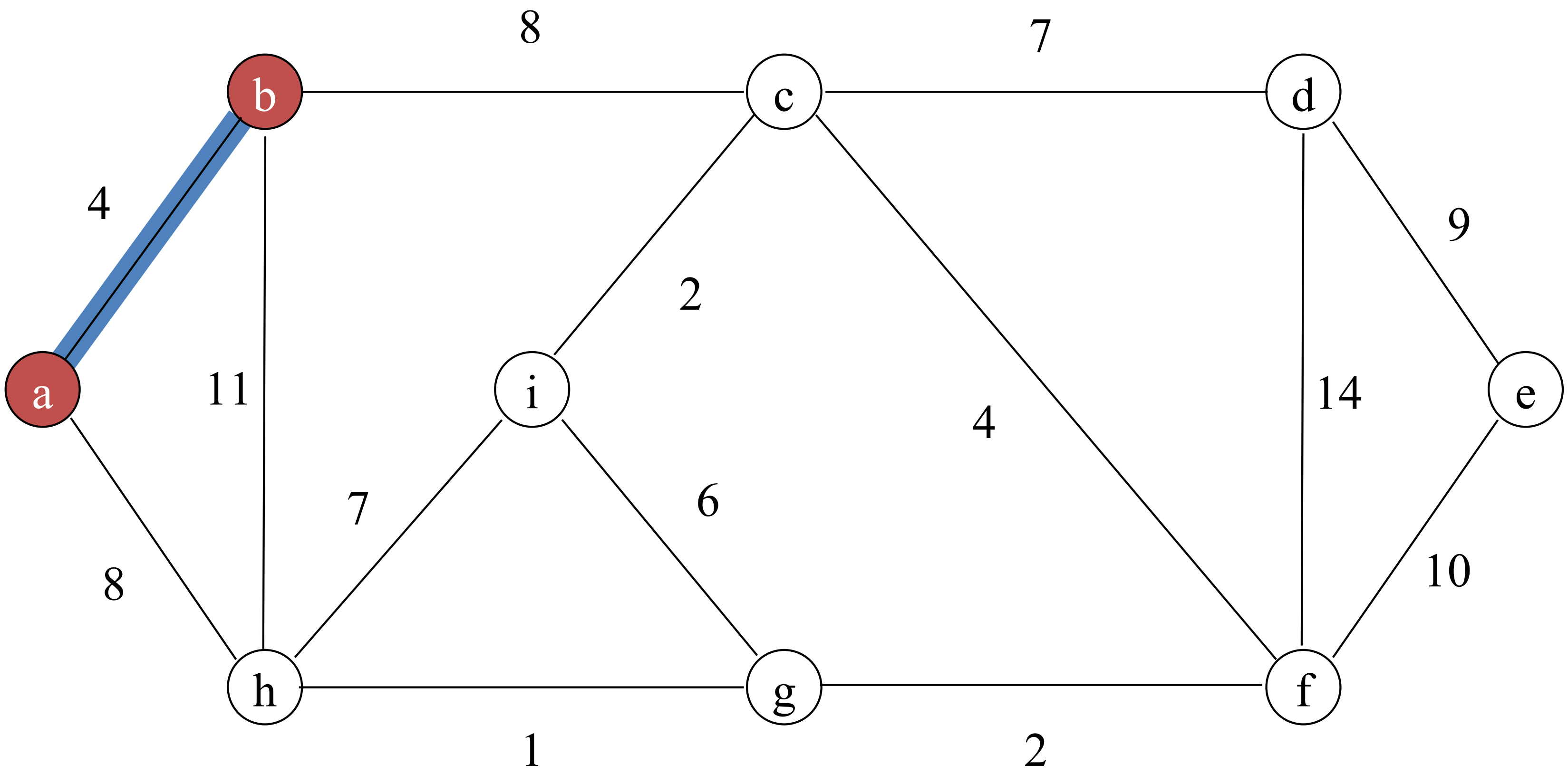
	a	b	c	d	e	f	g	h	i
cost	0	∞	∞	∞	∞	∞	∞	∞	∞
pre	a								

Prim's Algorithm



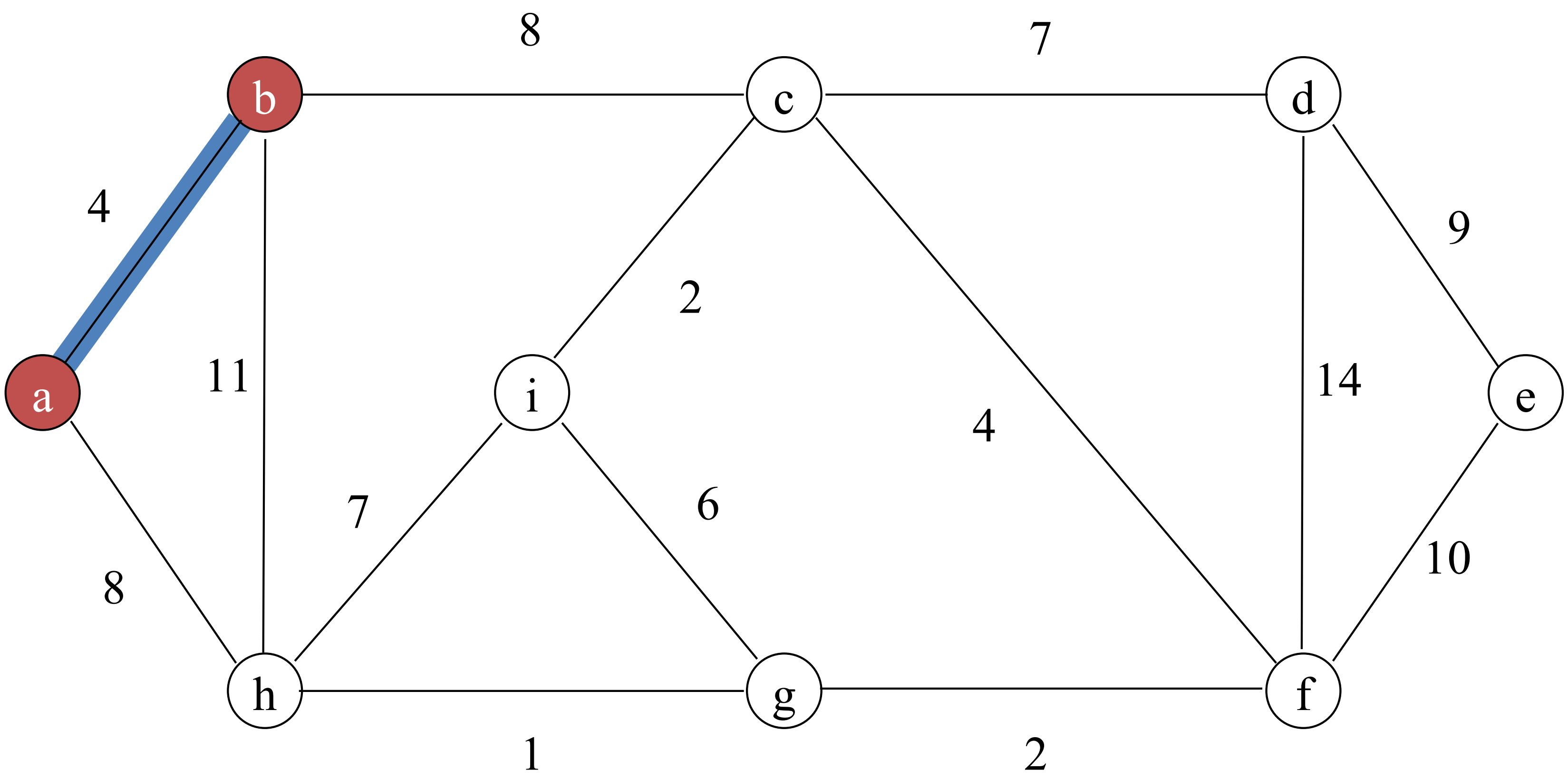
	a	b	c	d	e	f	g	h	i
cost	0	4	∞	∞	∞	∞	∞	8	∞
pre	a	a						a	

Prim's Algorithm



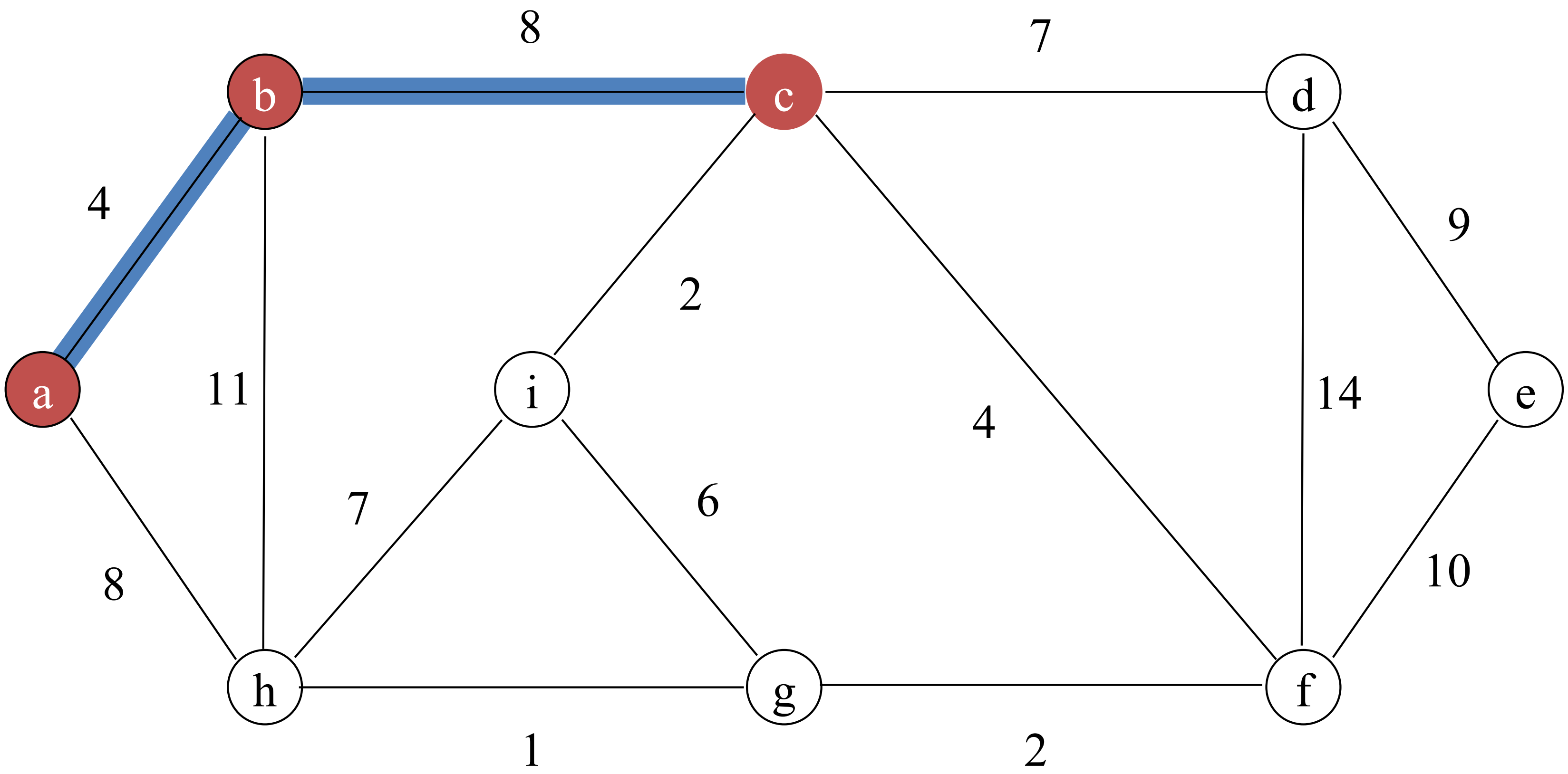
	a	b	c	d	e	f	g	h	i
cost	0	4	∞	∞	∞	∞	∞	8	∞
pre	a	a						a	

Prim's Algorithm

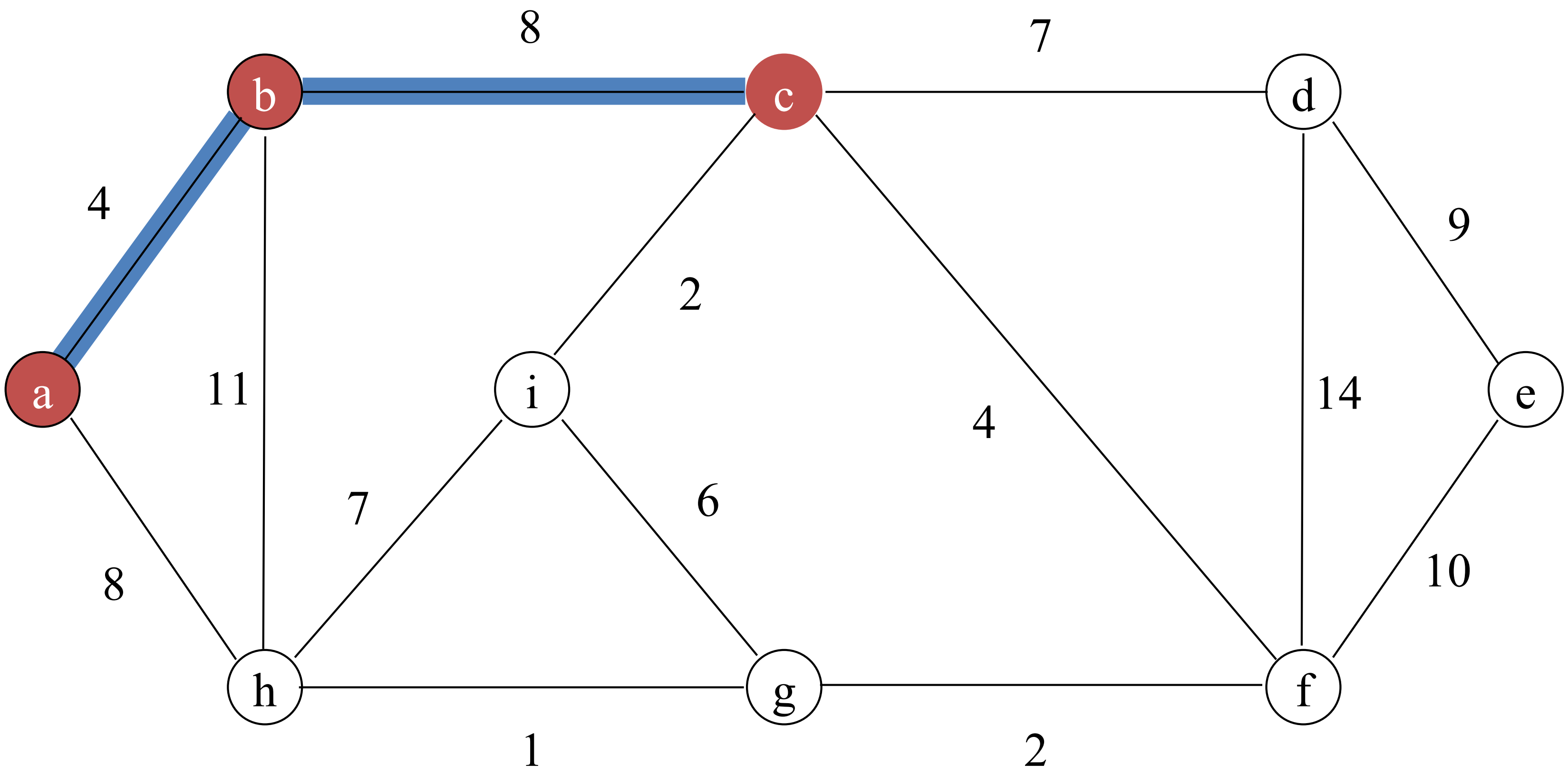


	a	b	c	d	e	f	g	h	i
cost	0	4	8	∞	∞	∞	∞	8	∞
pre	a	a	b					a	

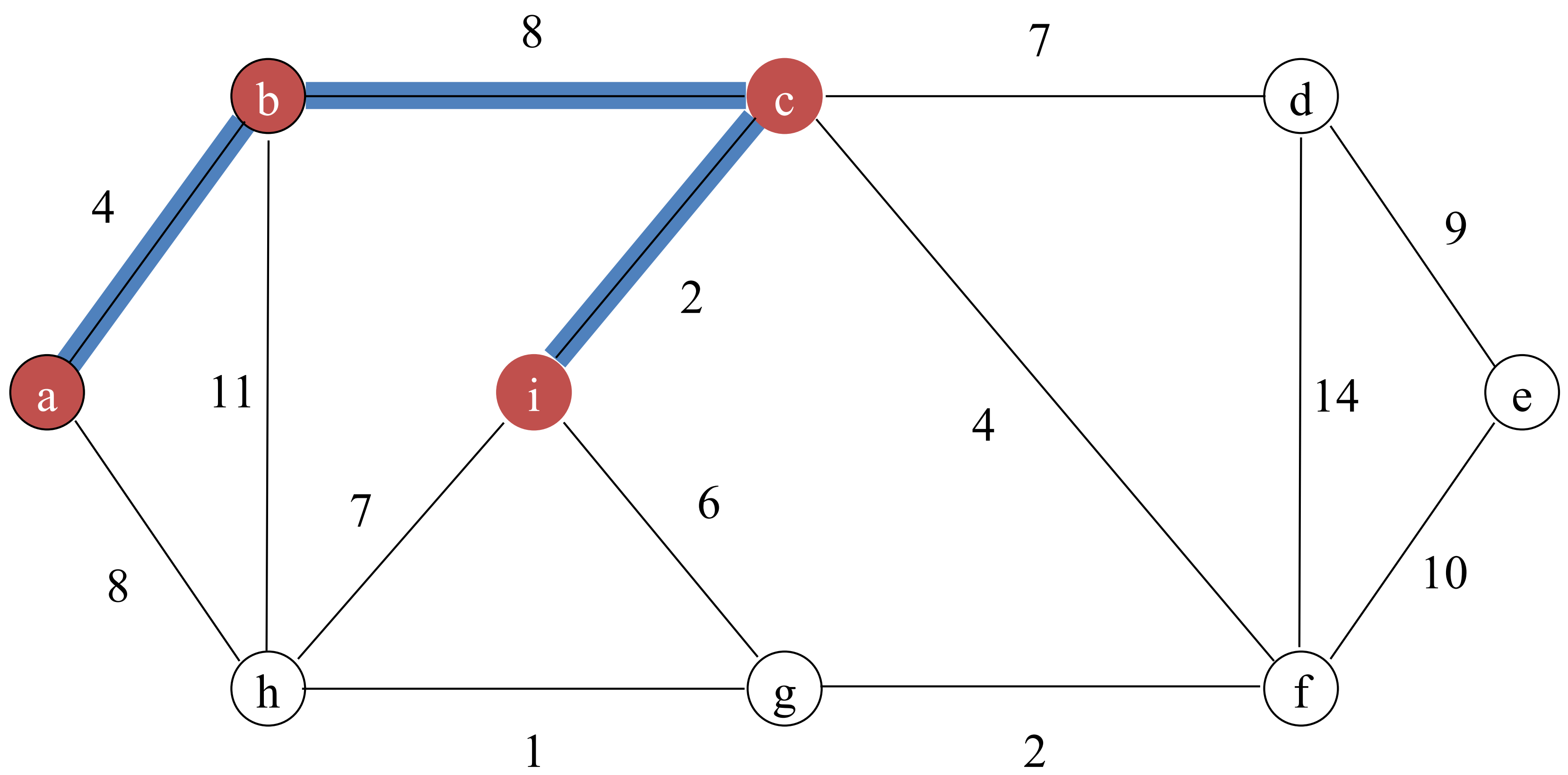
Prim's Algorithm



	a	b	c	d	e	f	g	h	i
cost	0	4	8	∞	∞	∞	∞	8	∞
pre	a	a	b					a	

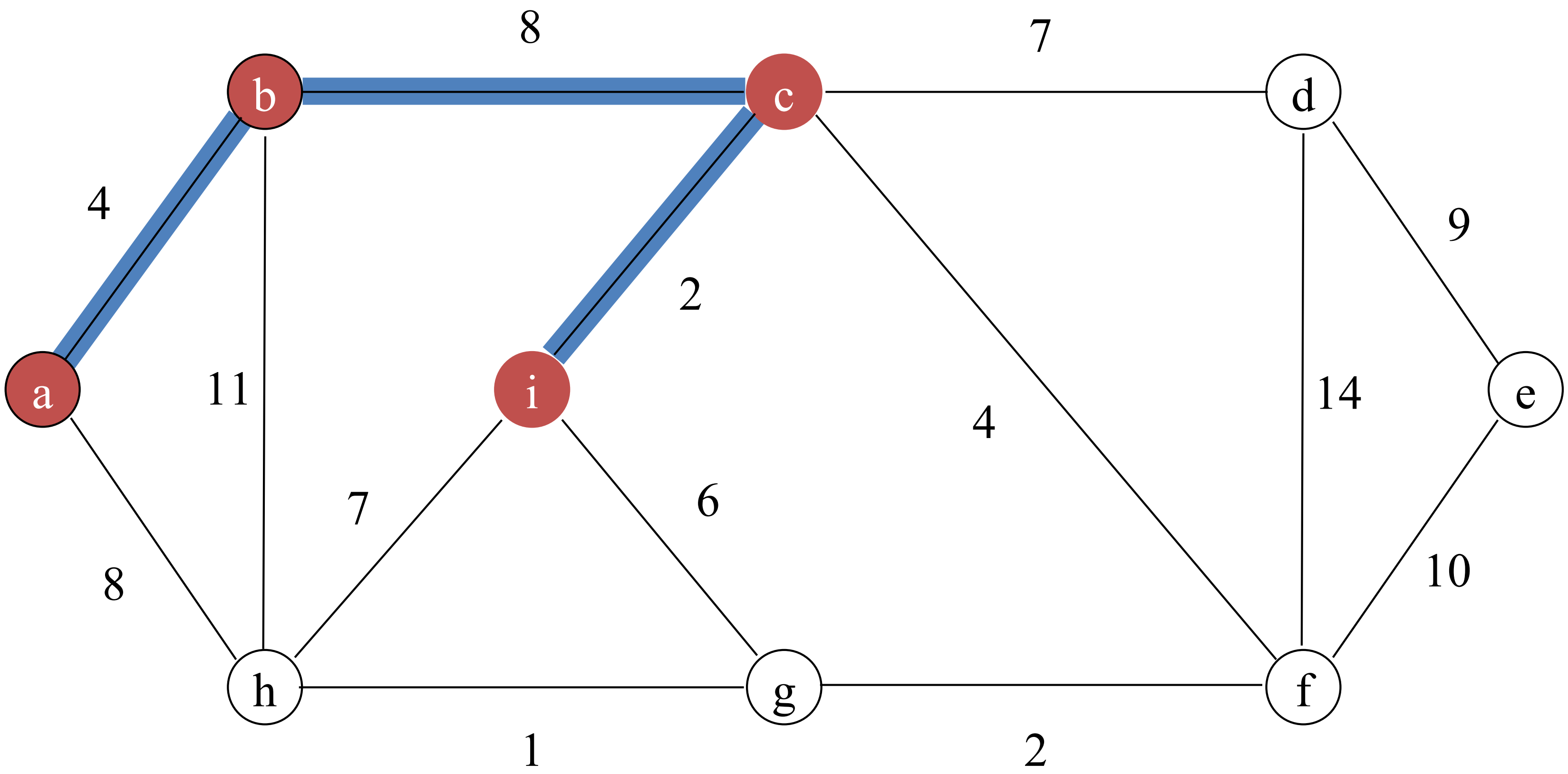


	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	∞	4	∞	8	2
pre	a	a	b	c		c		a	c

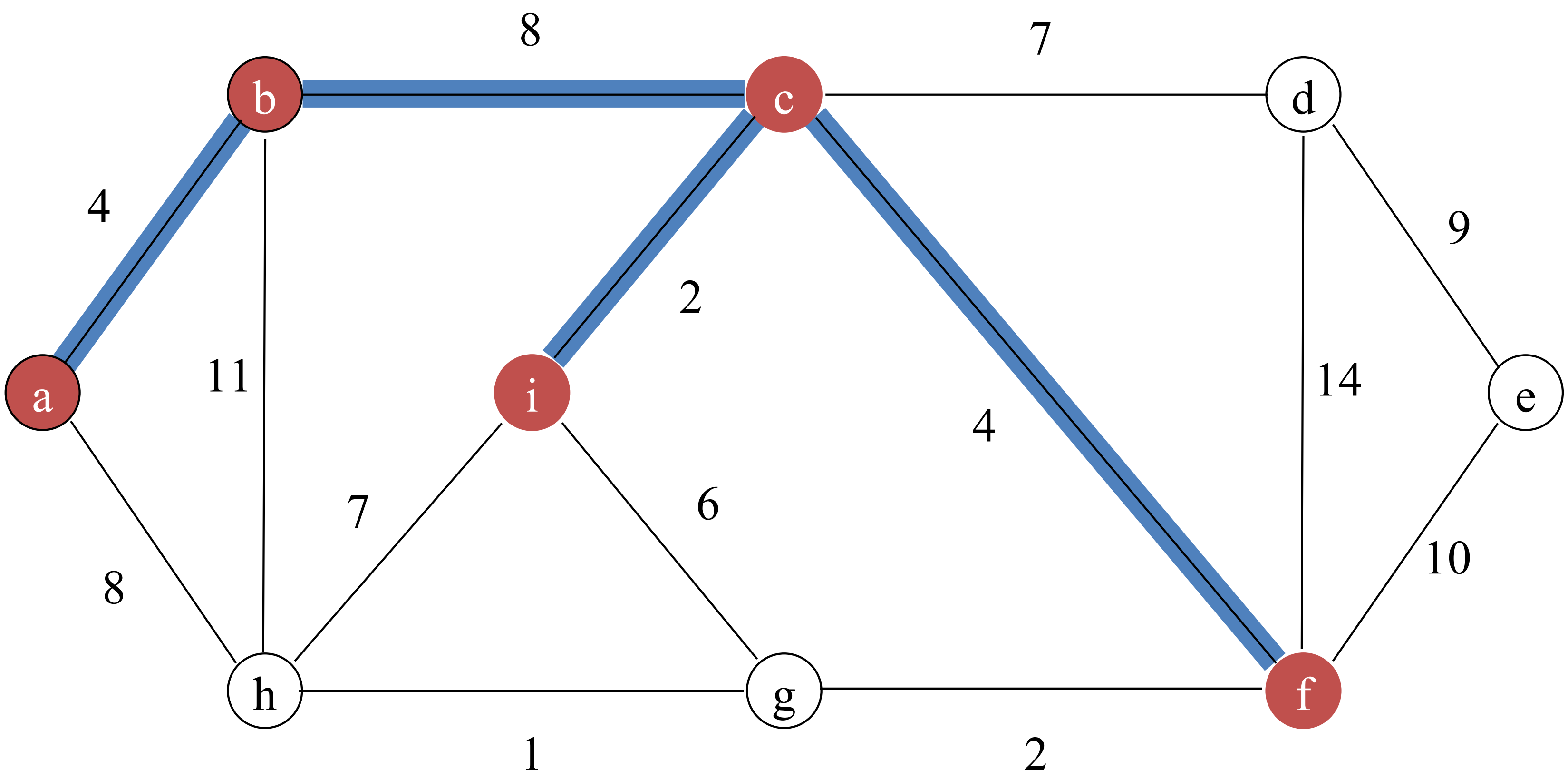


	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	∞	4	∞	8	2
pre	a	a	b	c		c		a	c

Prim's Algorithm

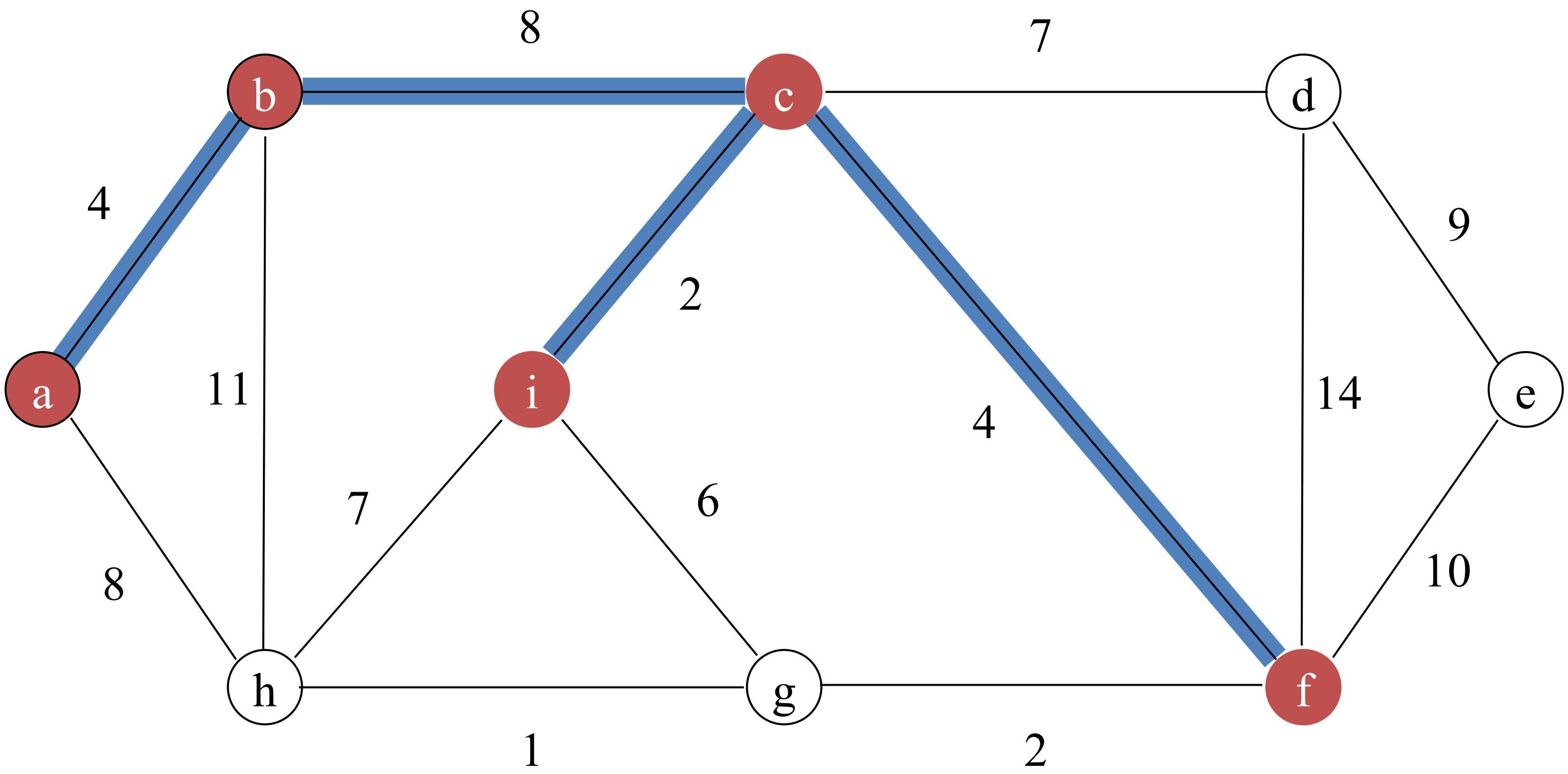


	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	∞	4	6	7	2
pre	a	a	b	c		c	i	i	c



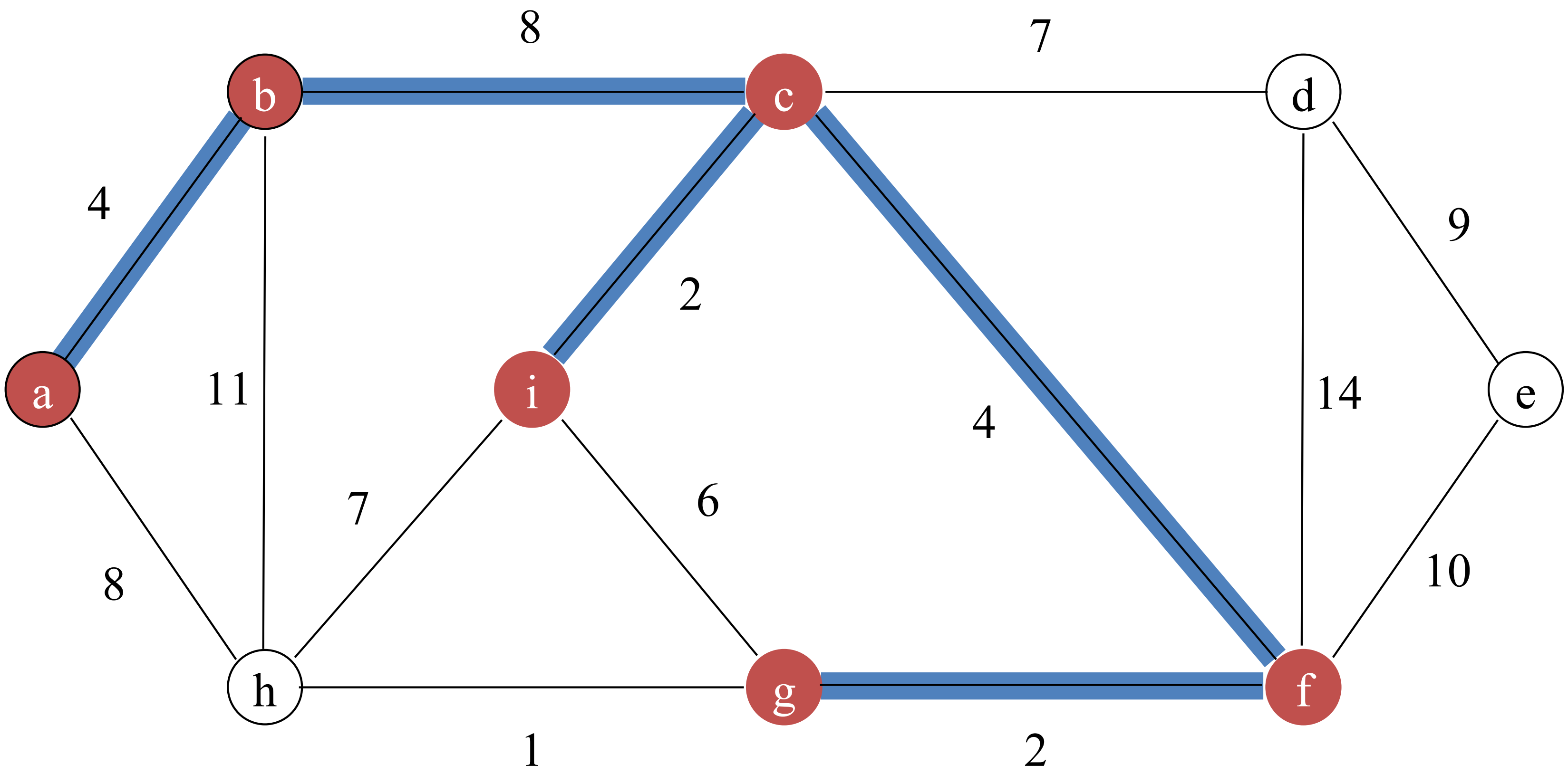
	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	∞	4	6	7	2
pre	a	a	b	c		c	i	i	c

Prim's Algorithm



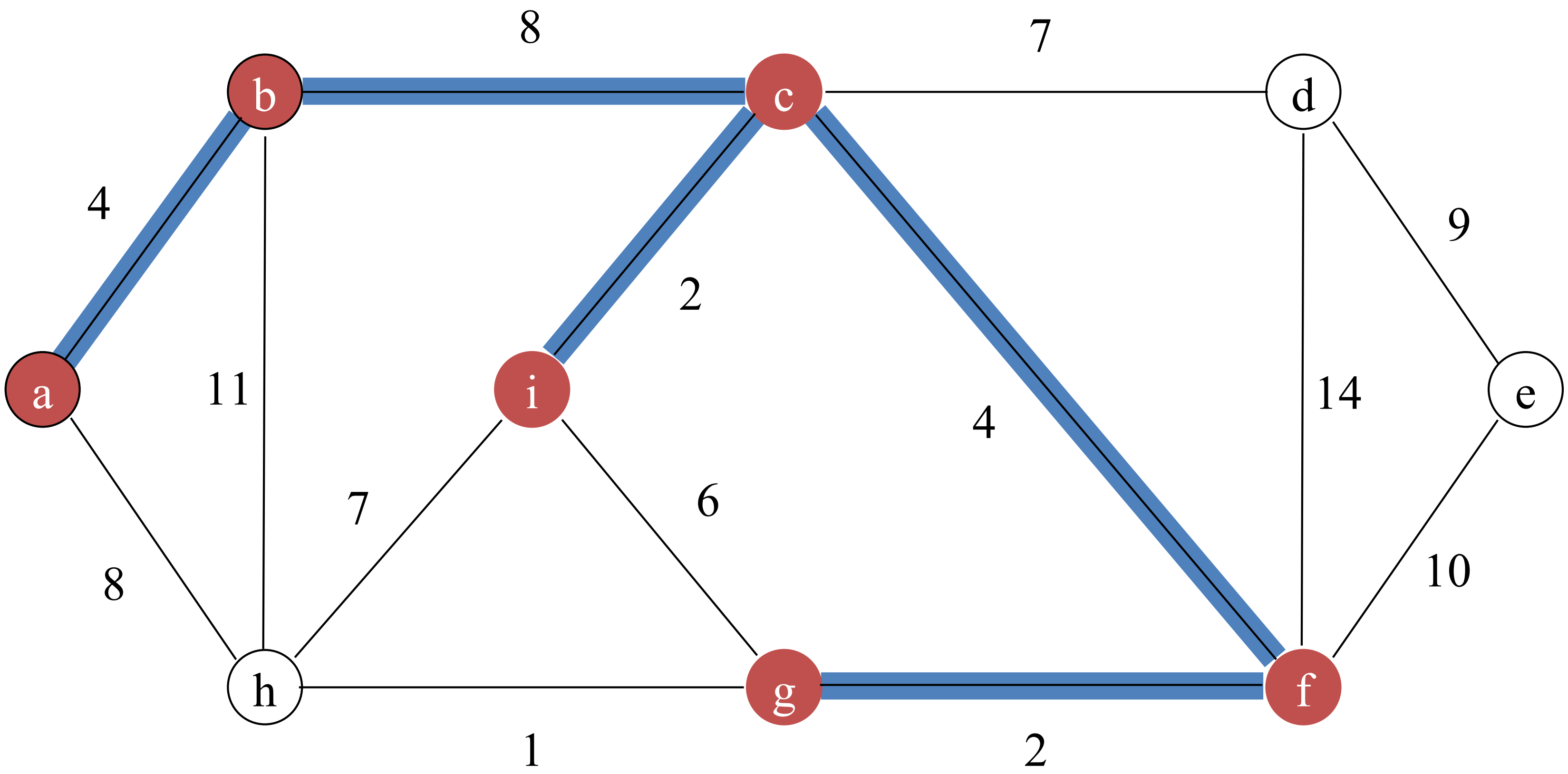
	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	10	4	2	7	2
pre	a	a	b	c	f	c	f	i	c

Prim's Algorithm

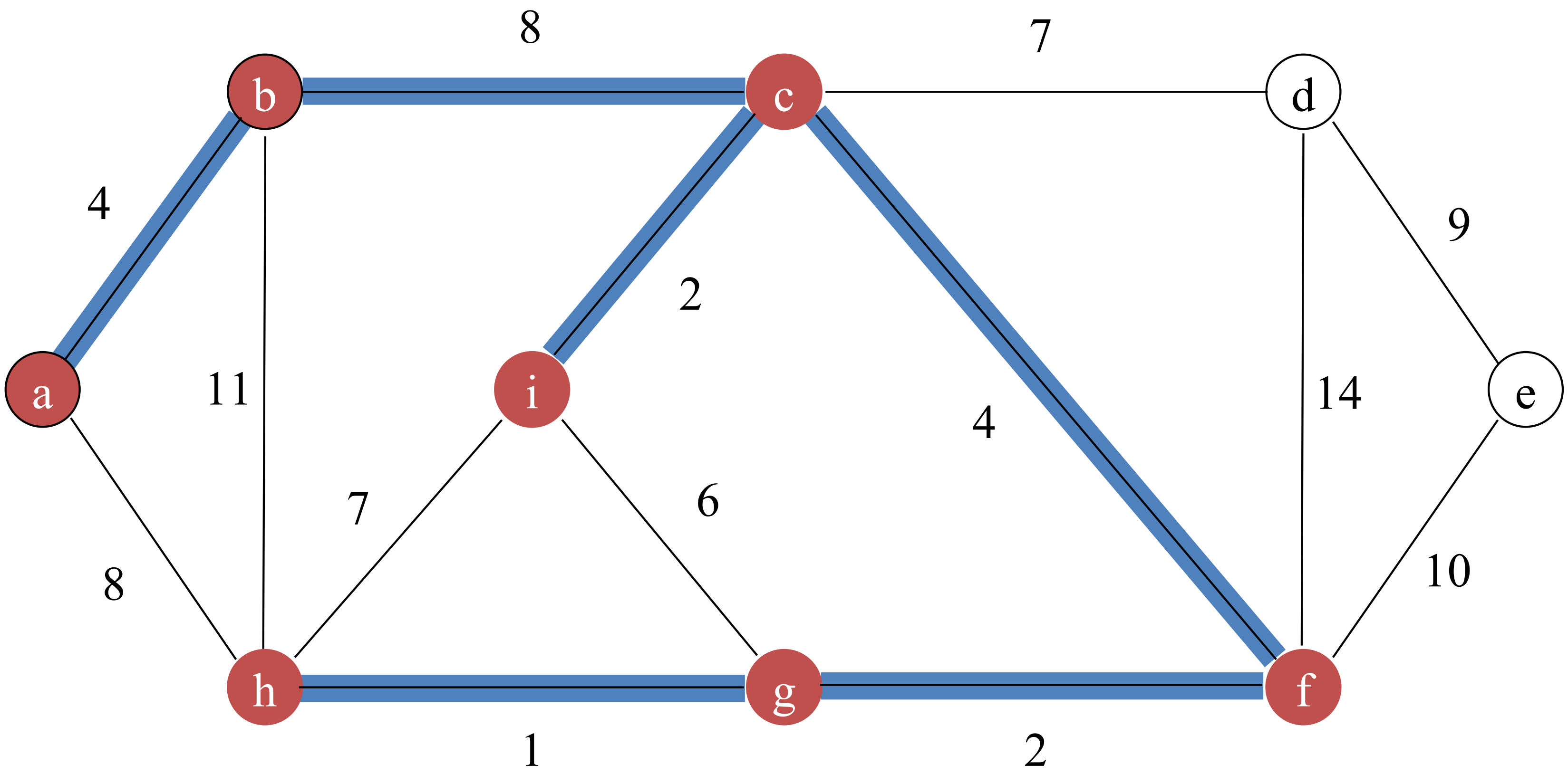


	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	10	4	2	7	2
pre	a	a	b	c	f	c	f	i	c

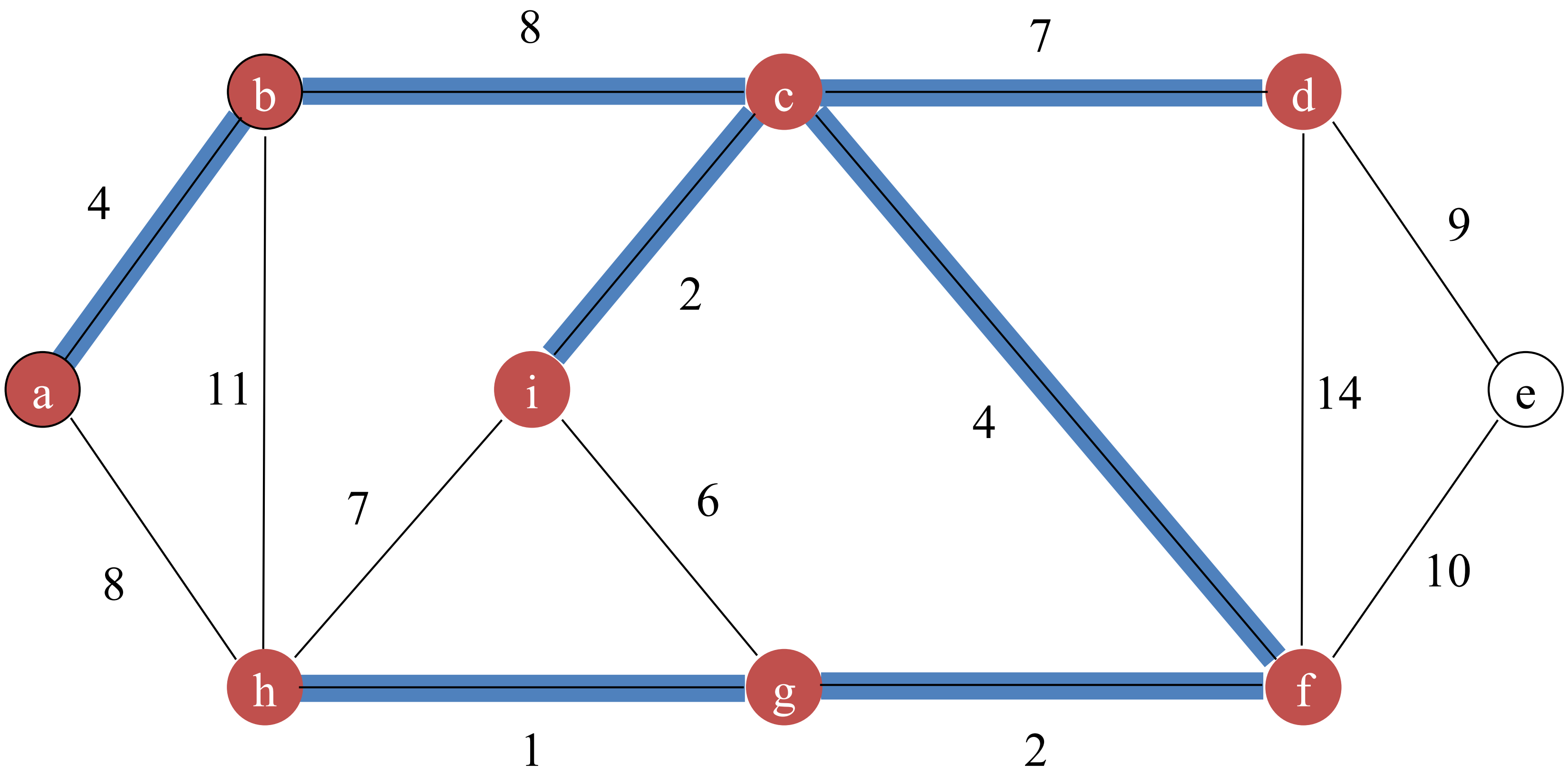
Prim's Algorithm



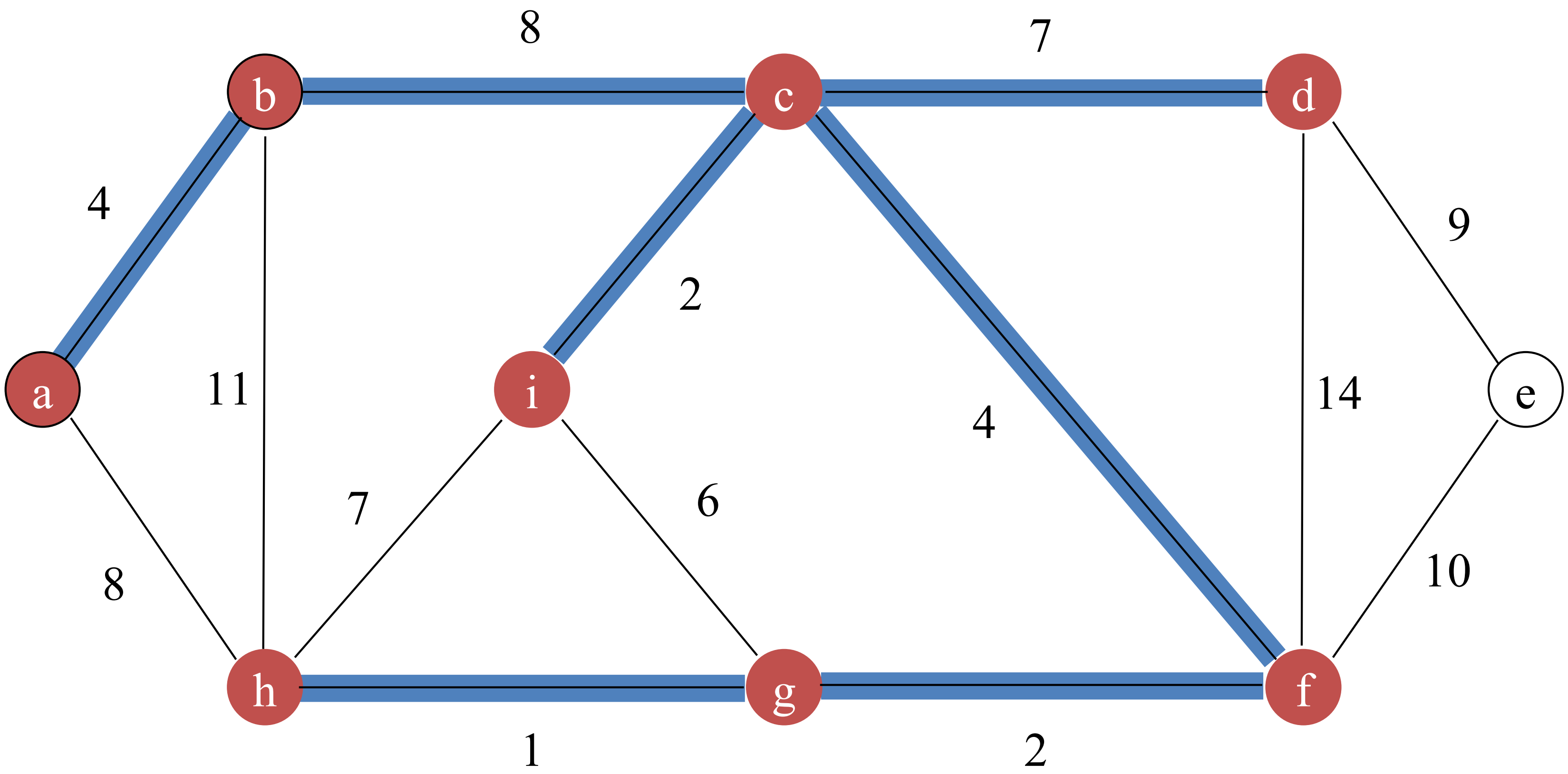
	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	10	4	2	1	2
pre	a	a	b	c	f	c	f	g	c



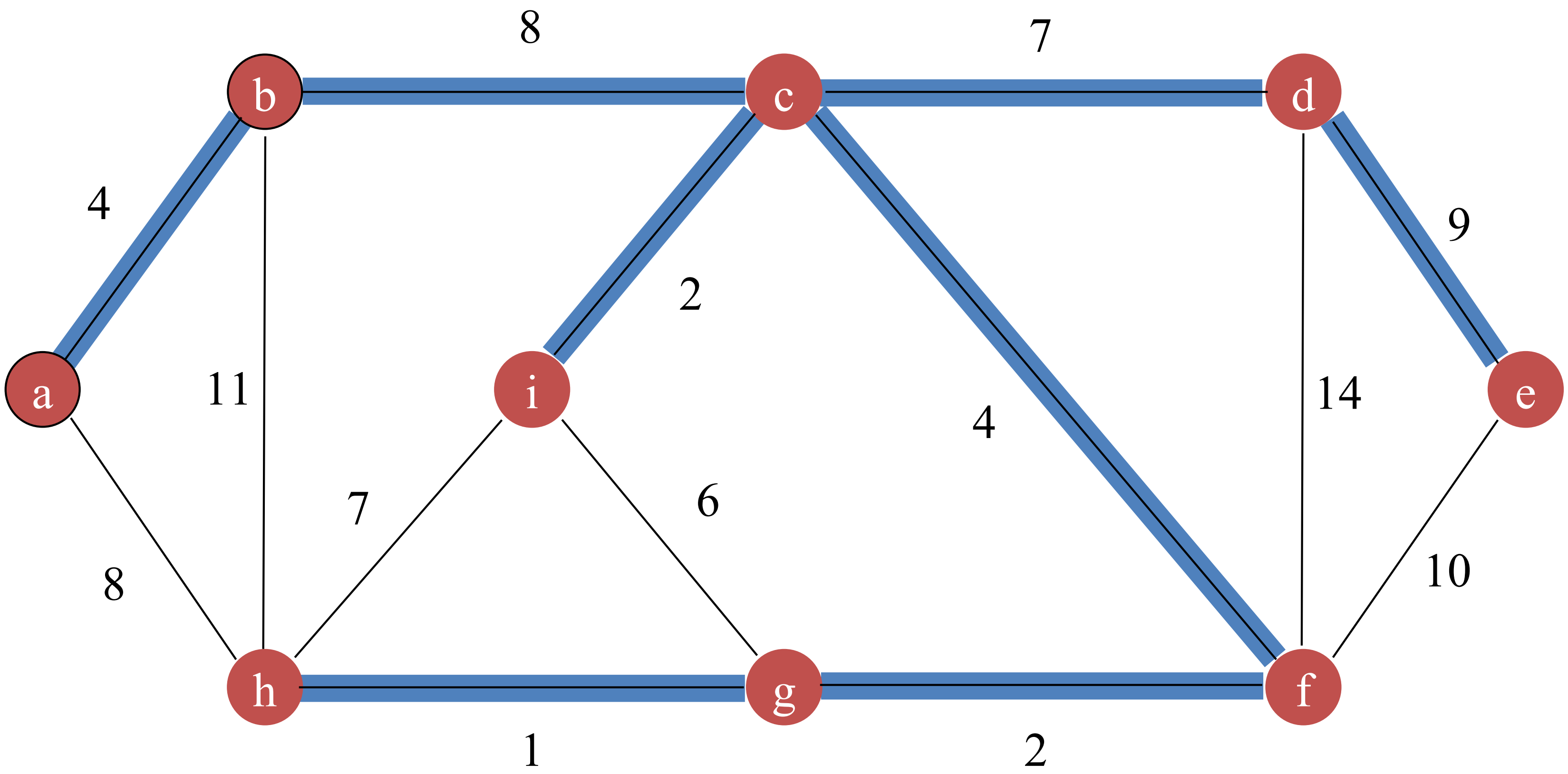
	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	10	4	2	1	2
pre	a	a	b	c	f	c	f	g	c



	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	10	4	2	1	2
pre	a	a	b	c	f	c	f	g	c

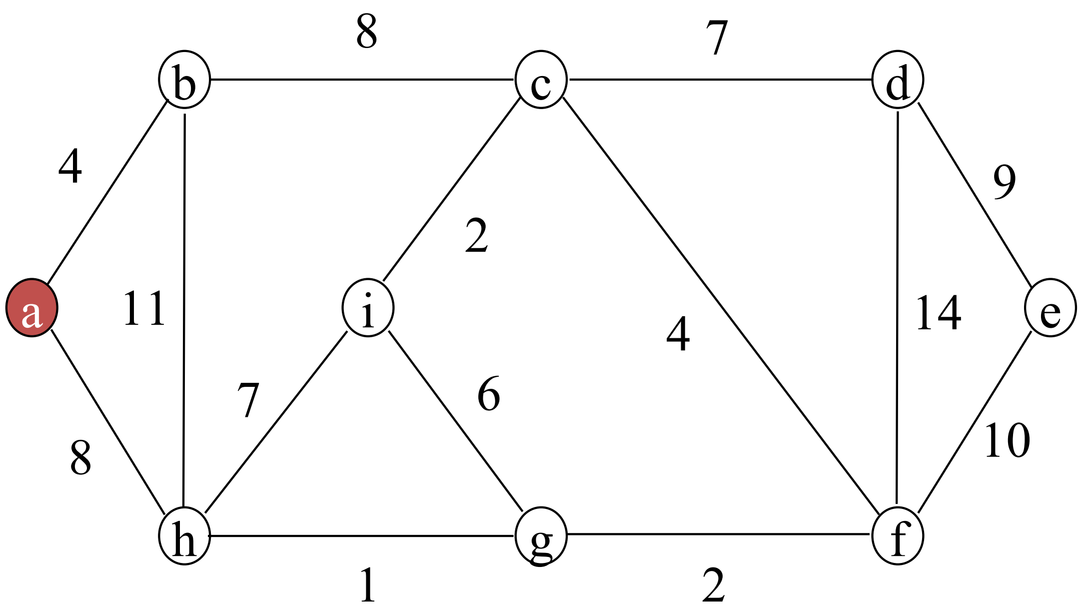


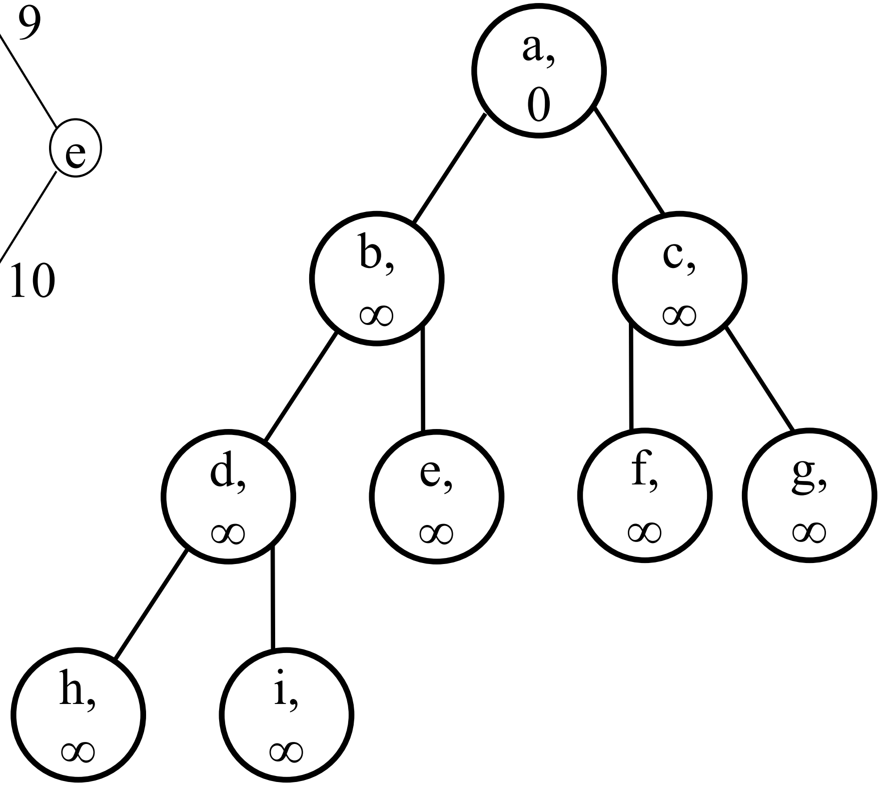
	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	9	4	2	1	2
pre	a	a	b	c	d	c	f	g	c



	a	b	c	d	e	f	g	h	i
cost	0	4	8	7	9	4	2	1	2
pre	a	a	b	c	d	c	f	g	c

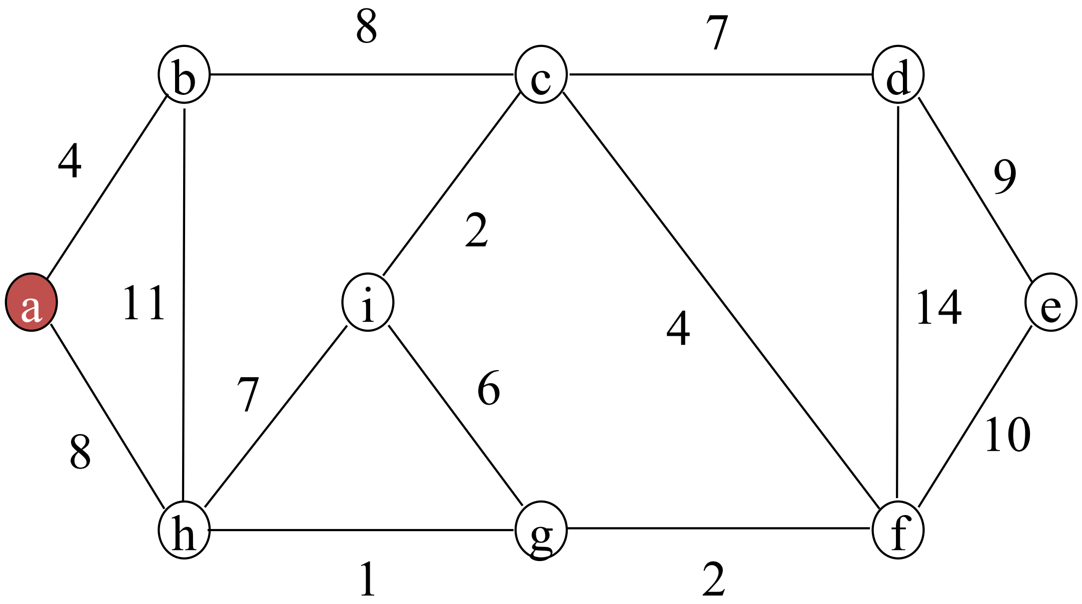
Prim's Algorithm

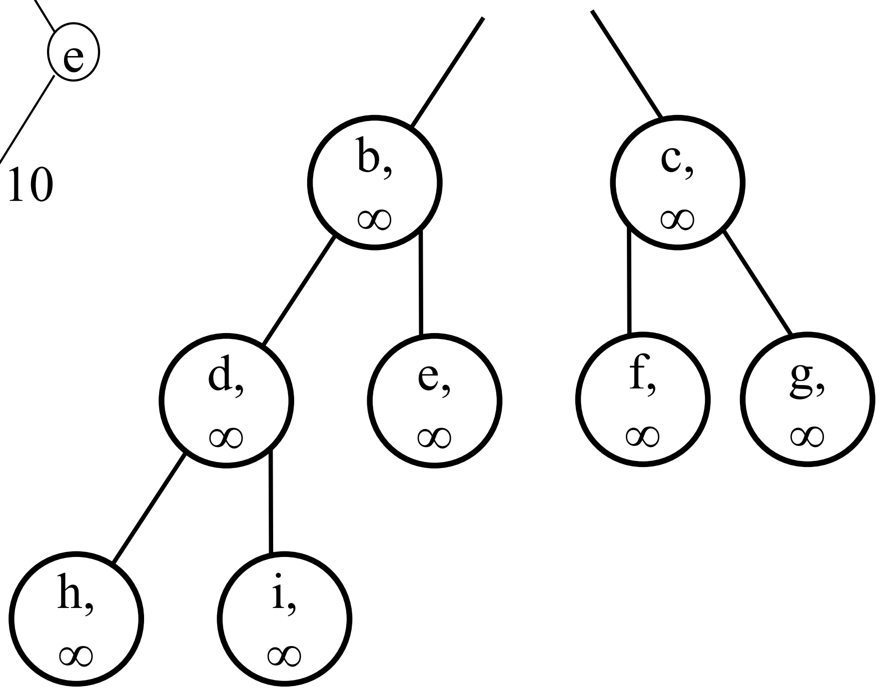




a	b	c	d	e	f	g	h	i

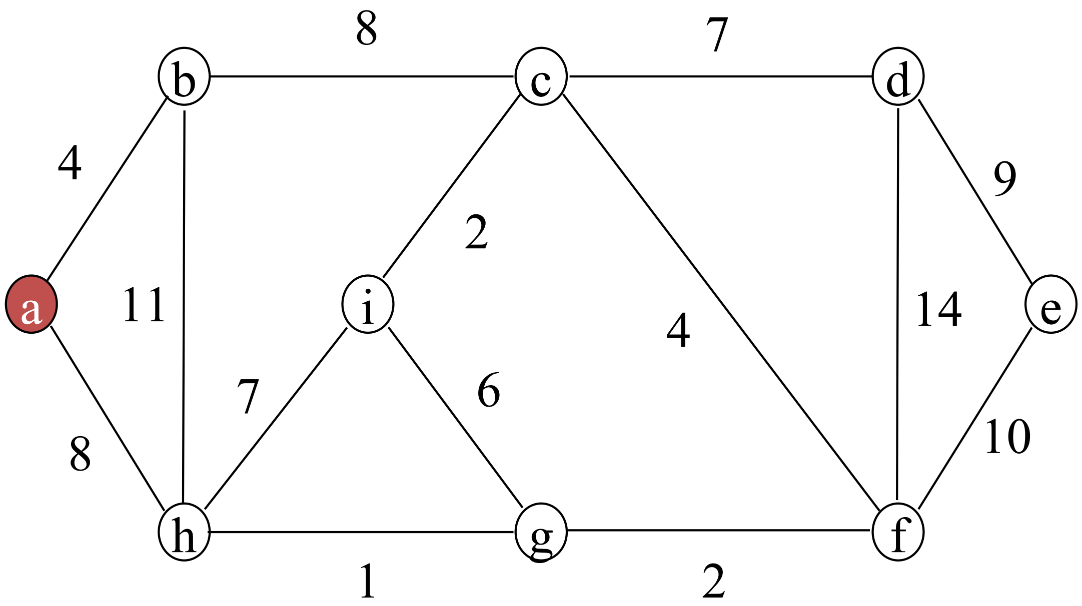
Prim's Algorithm

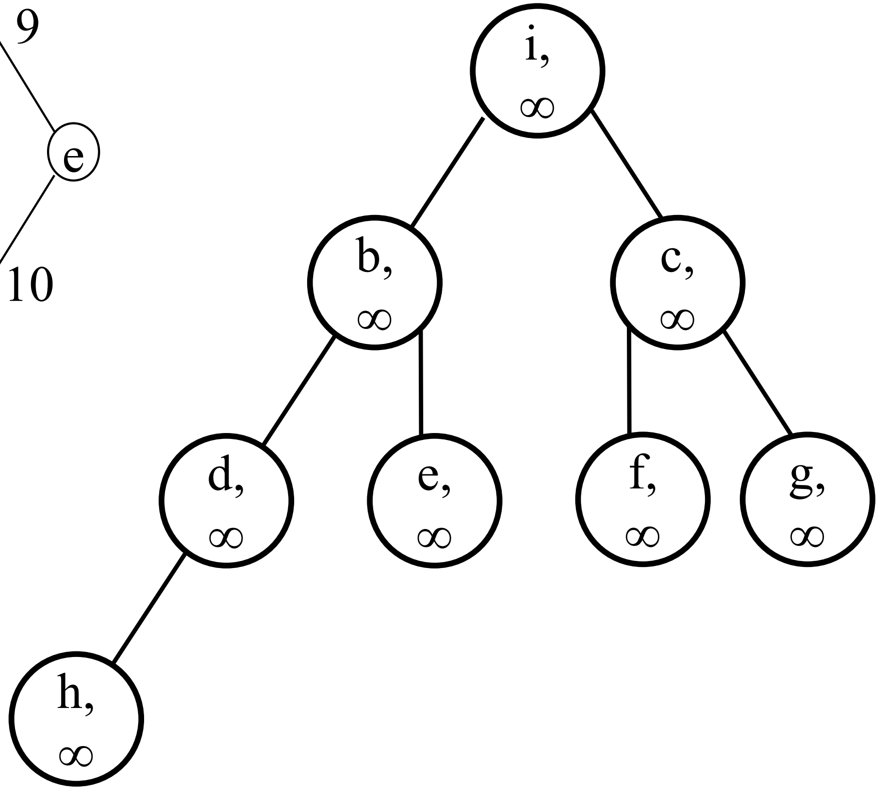




a	b	c	d	e	f	g	h	i

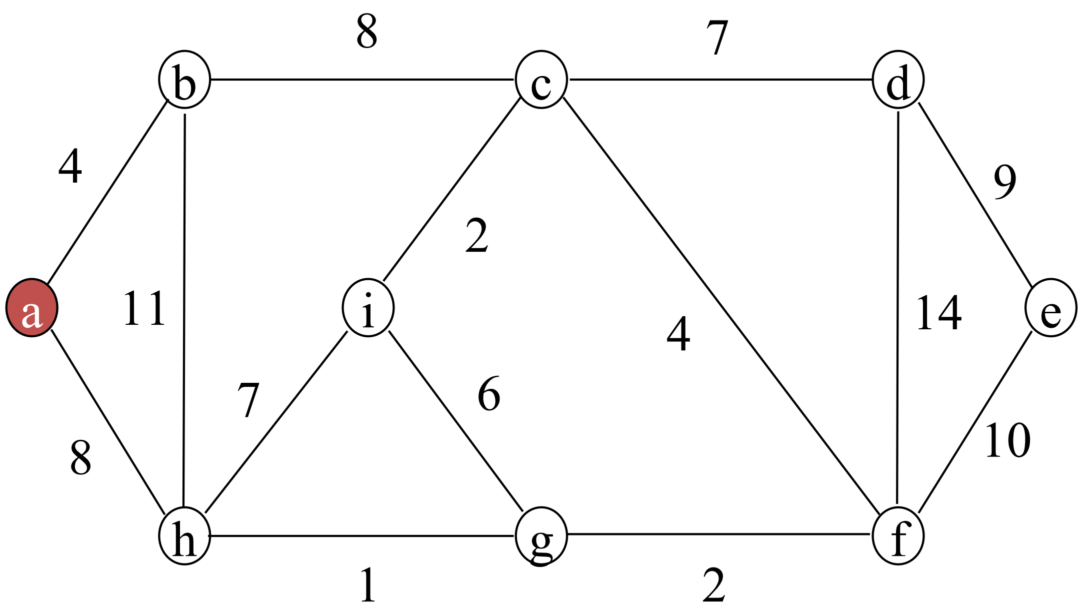
Prim's Algorithm

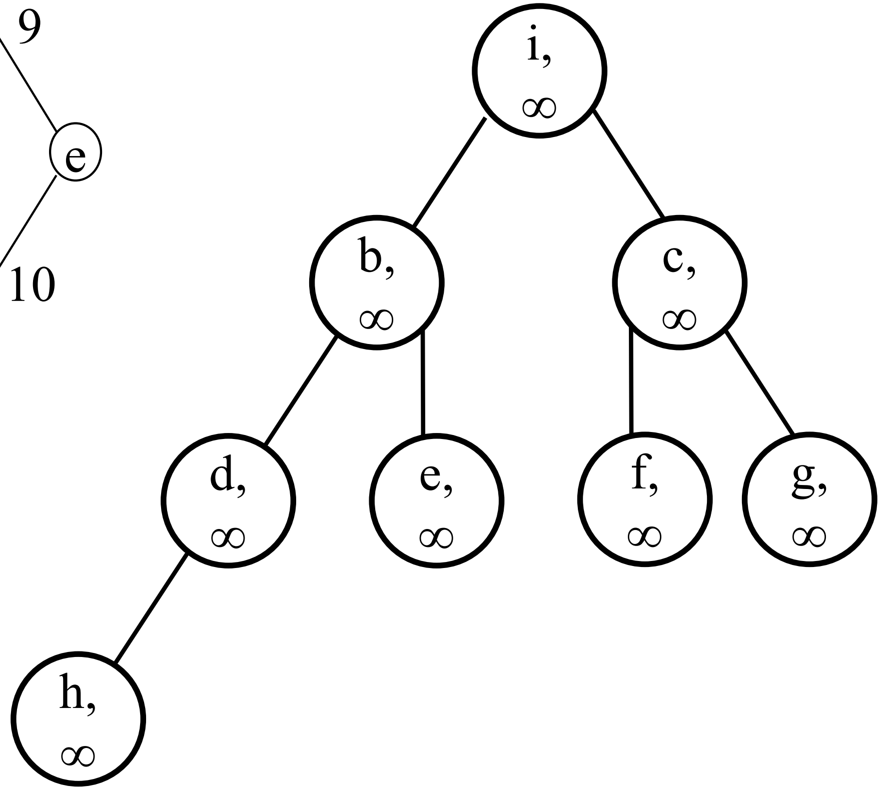




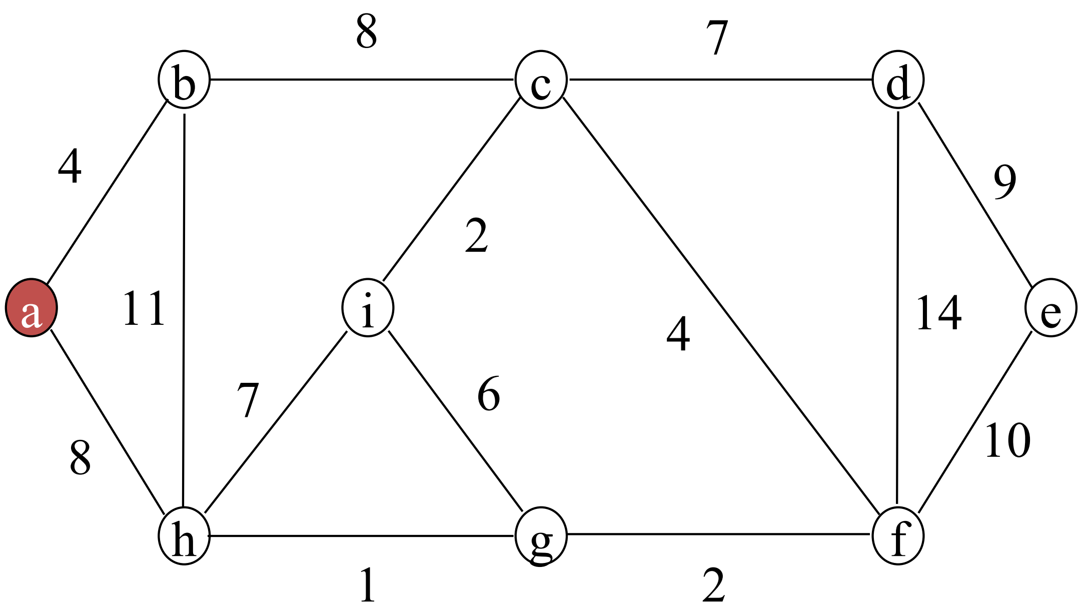
a	b	c	d	e	f	g	h	i

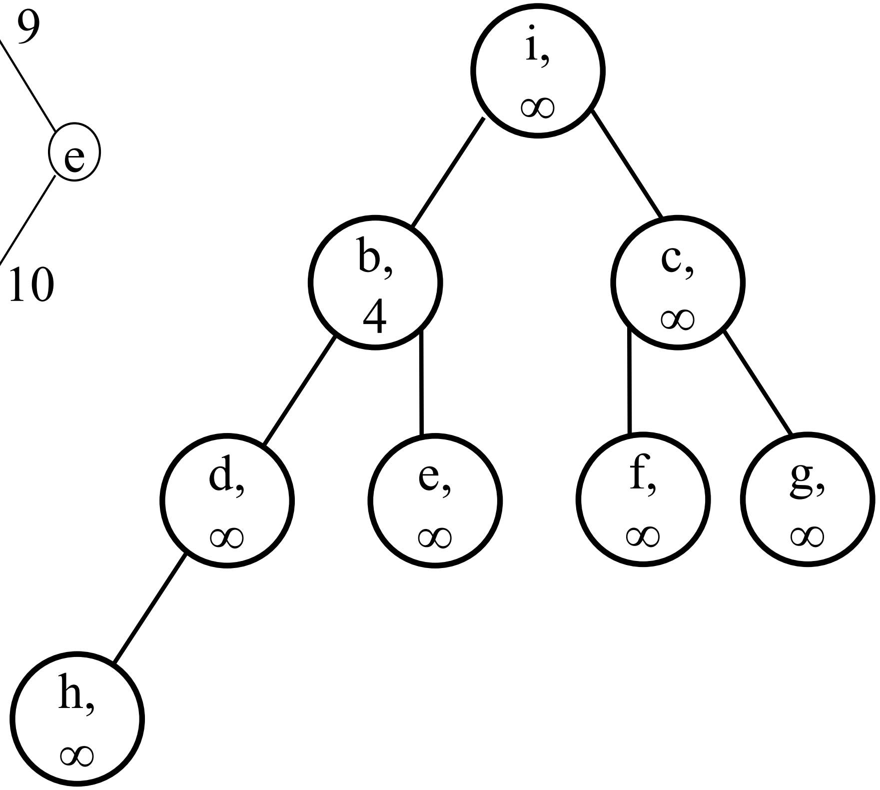
Prim's Algorithm





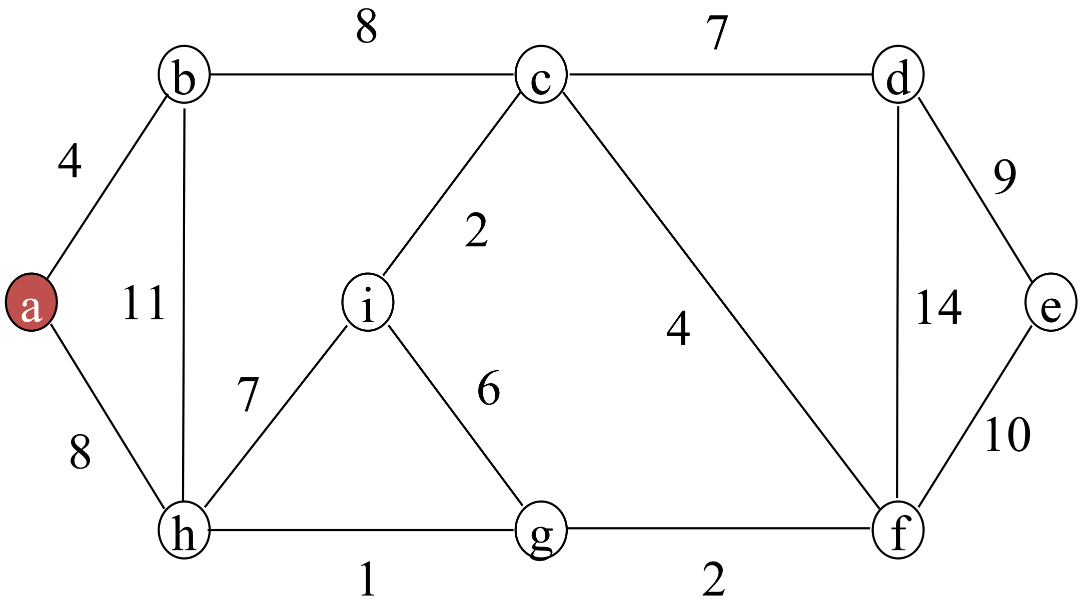
a	b	c	d	e	f	g	h	i
	a							

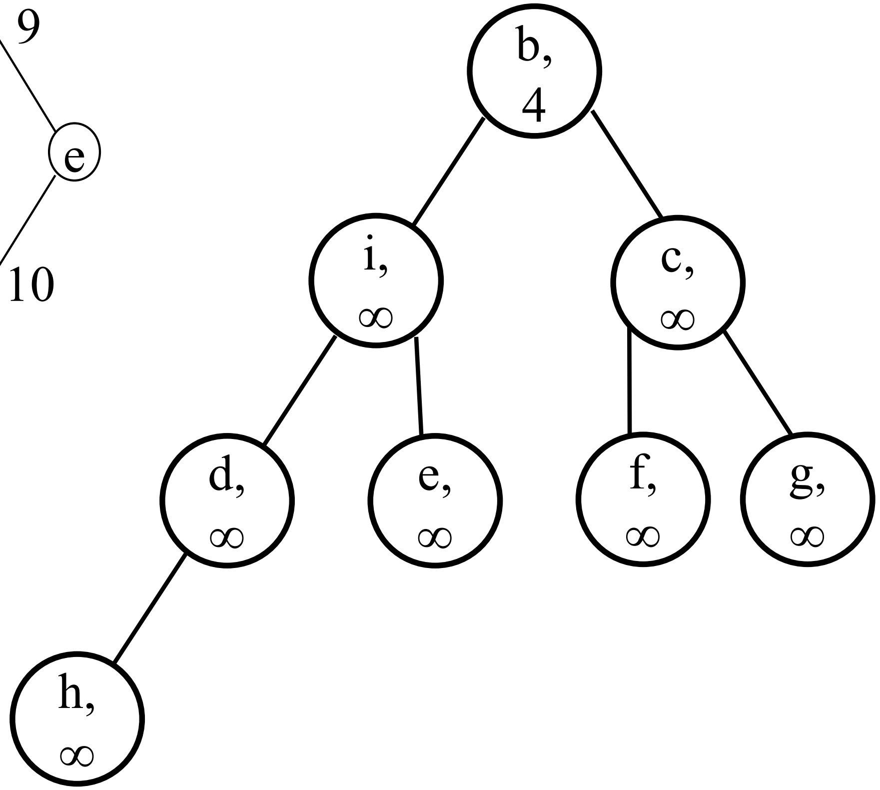




a	b	c	d	e	f	g	h	i
	a							

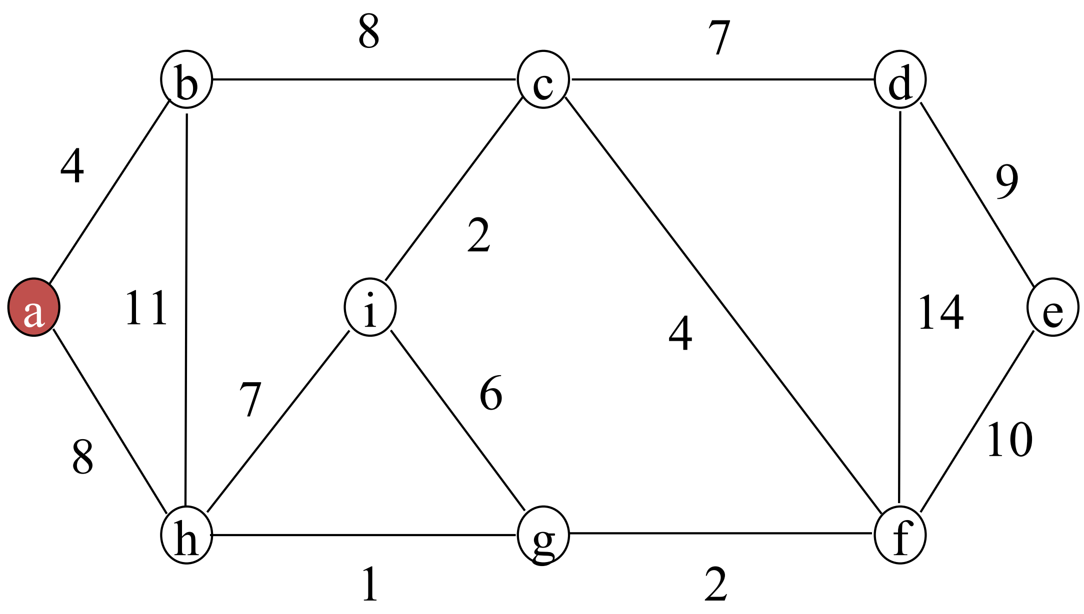
Prim’s Algorithm

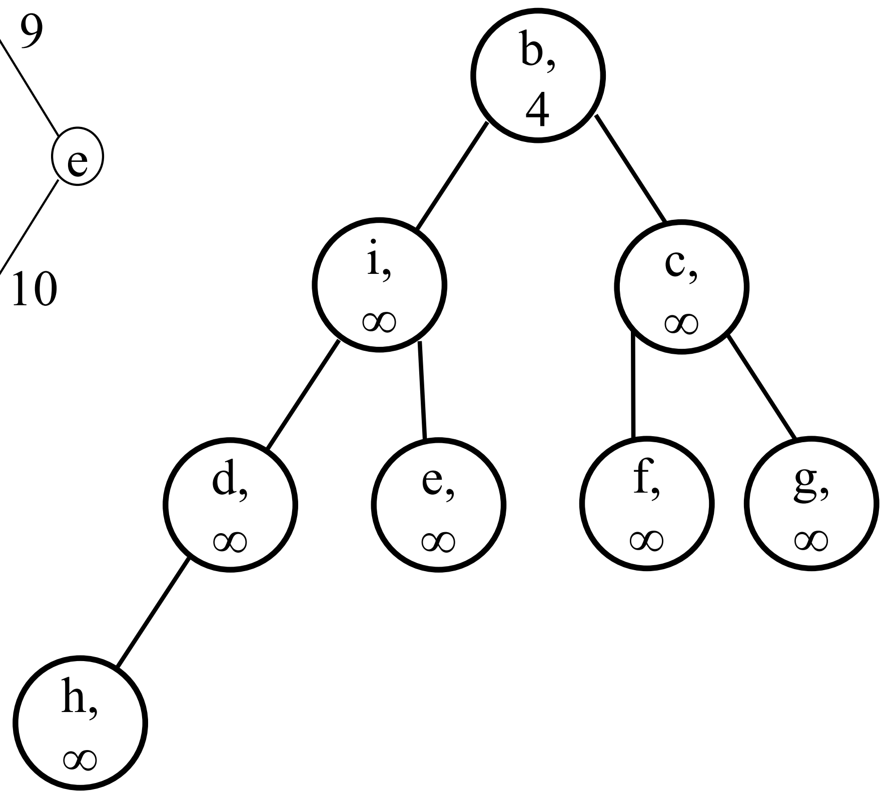




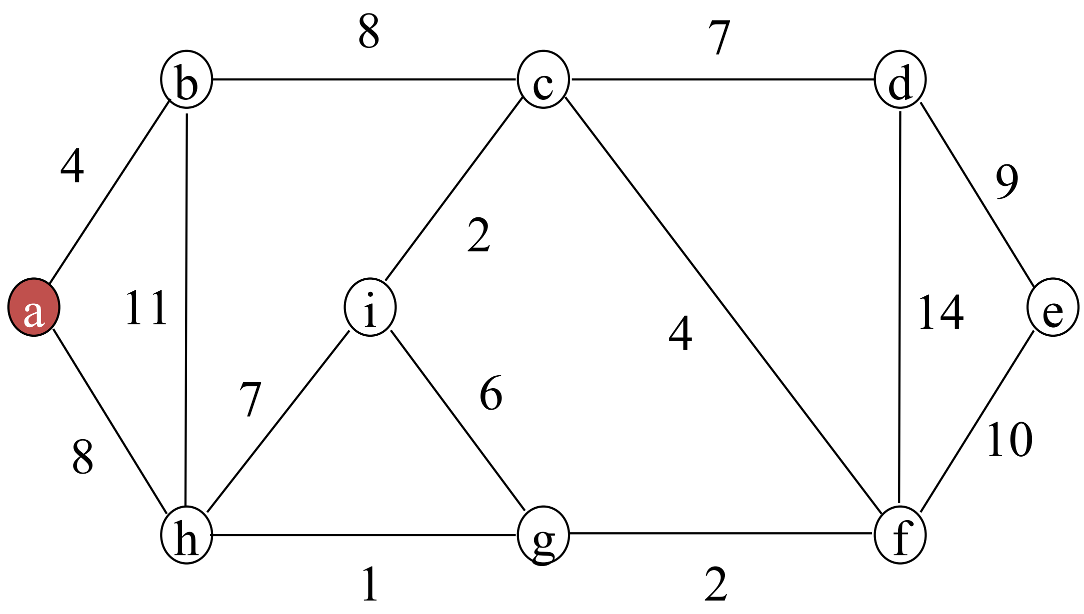
a	b	c	d	e	f	g	h	i
	a							

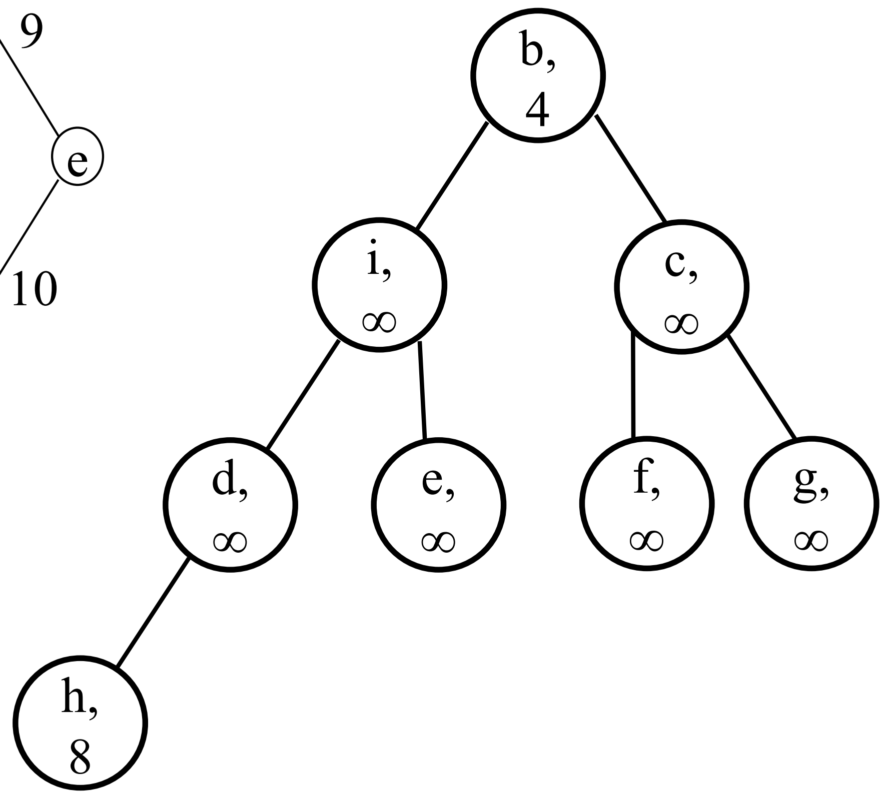
Prim's Algorithm





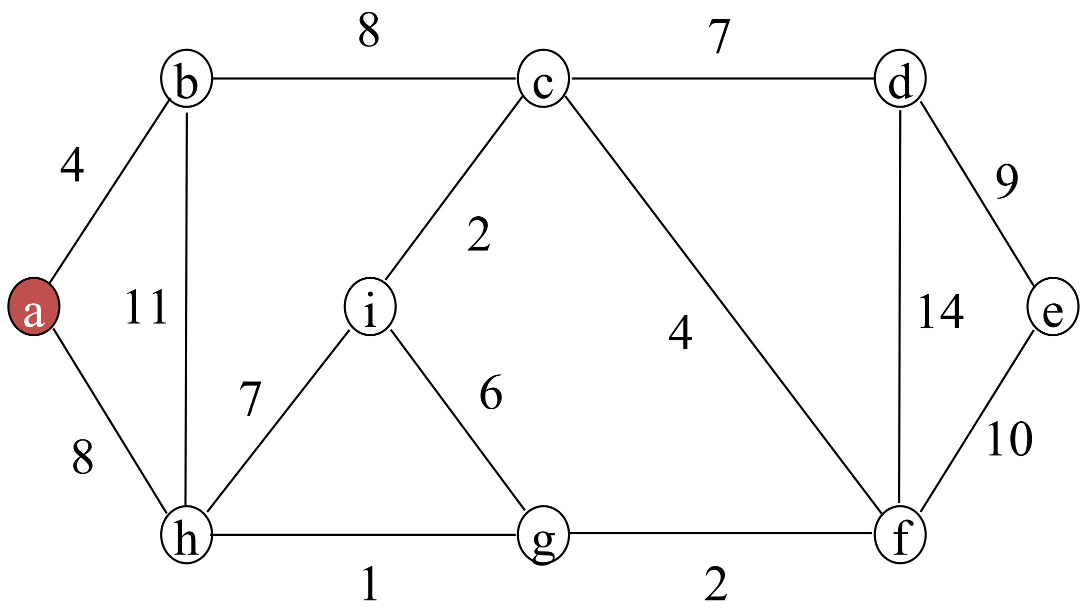
a	b	c	d	e	f	g	h	i
	a						a	

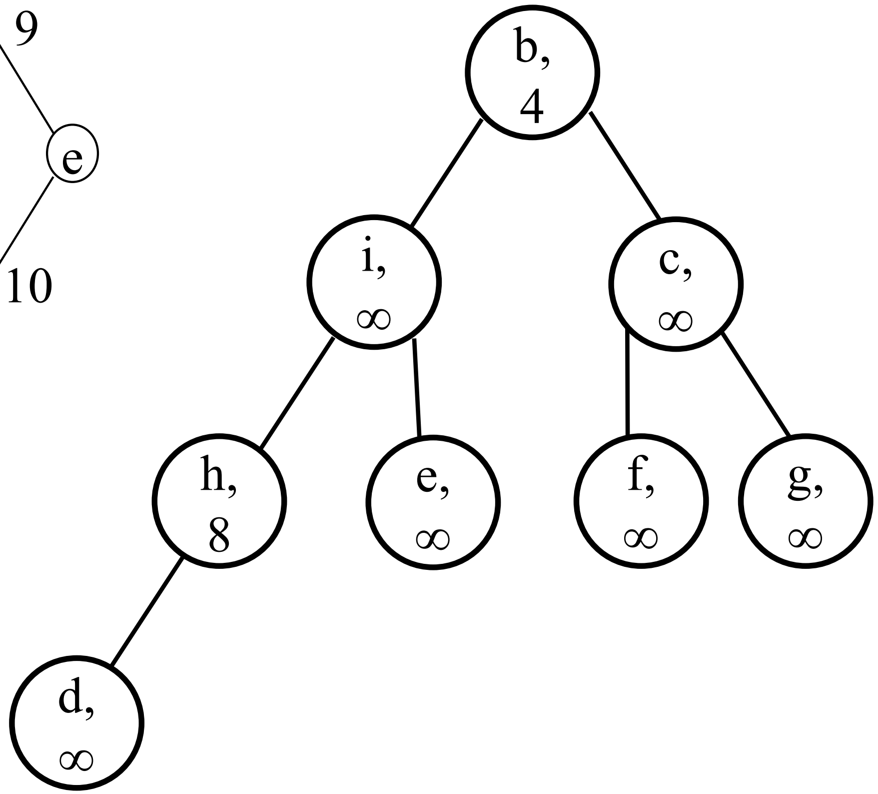




a	b	c	d	e	f	g	h	i
	a						a	

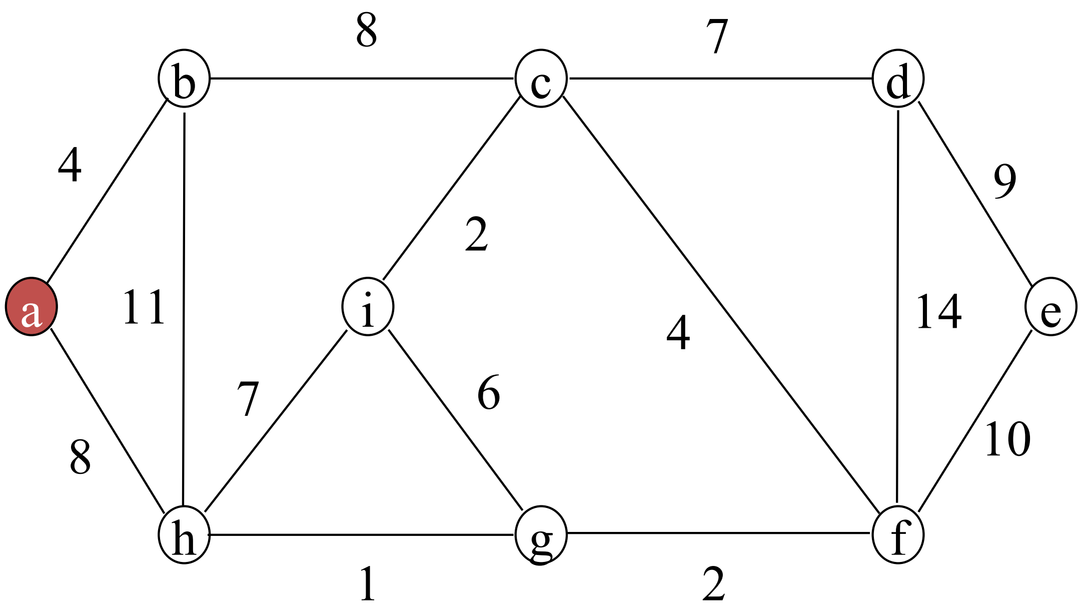
Prim's Algorithm

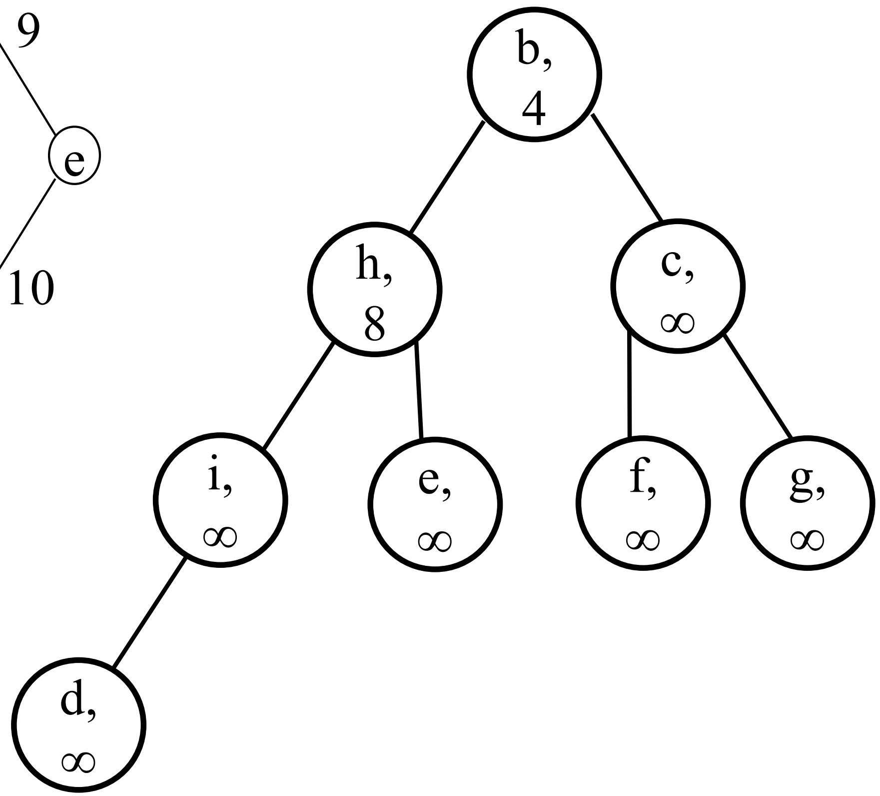




a	b	c	d	e	f	g	h	i
	a						a	

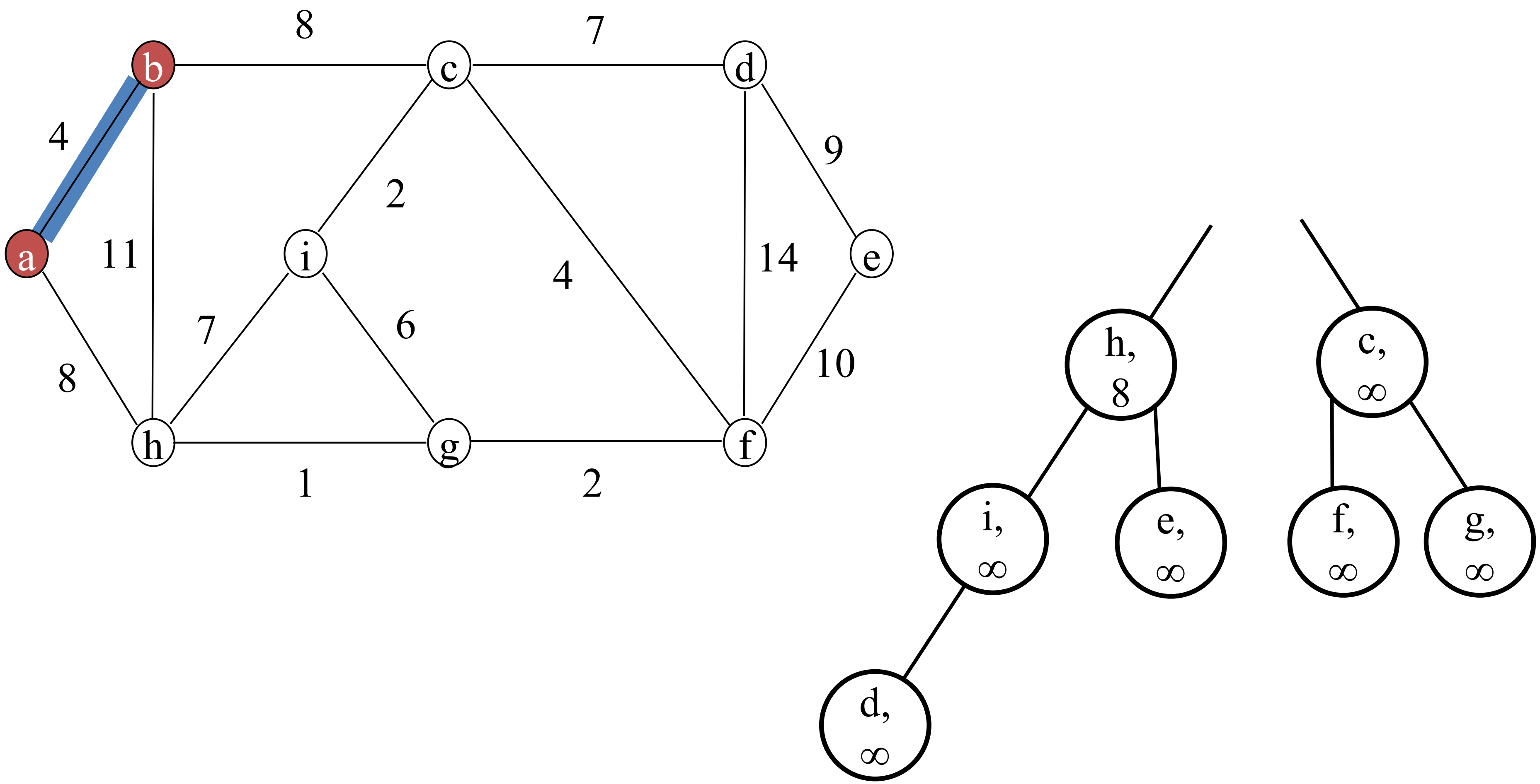
Prim's Algorithm





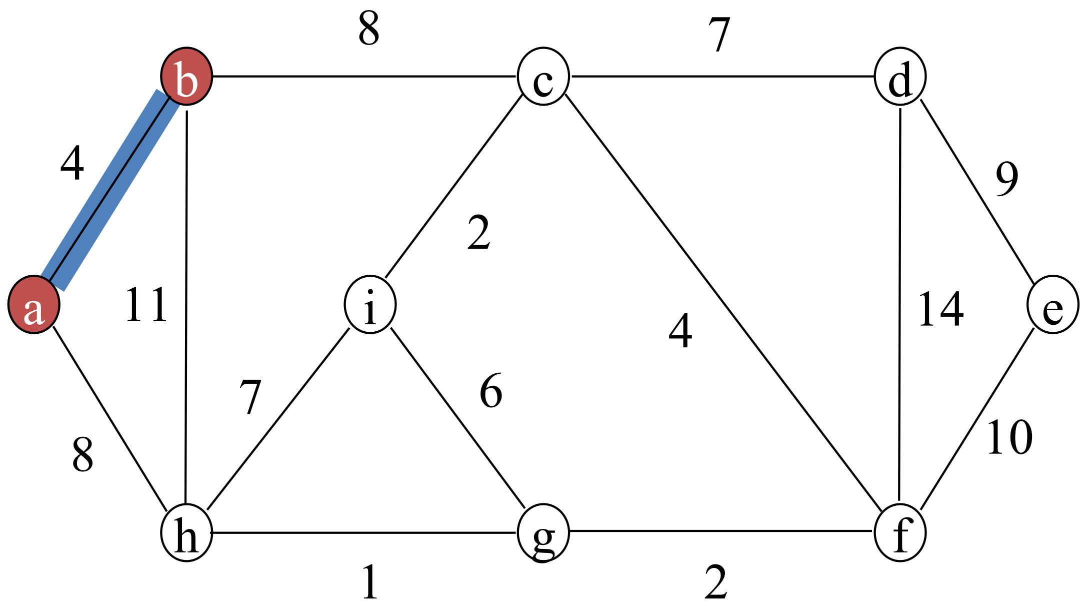
a	b	c	d	e	f	g	h	i
	a						a	

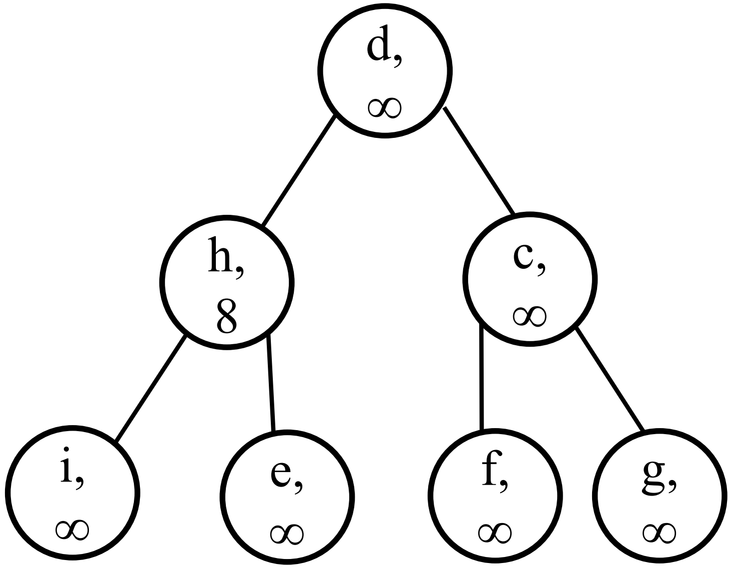
Prim's Algorithm



a	b	c	d	e	f	g	h	i
	a						a	

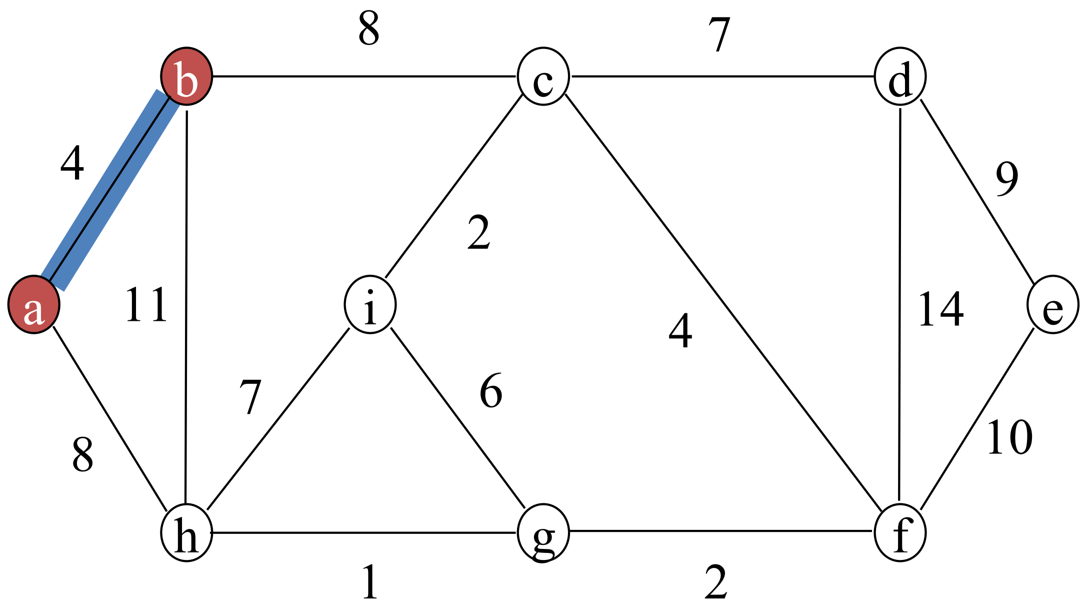
Prim's Algorithm

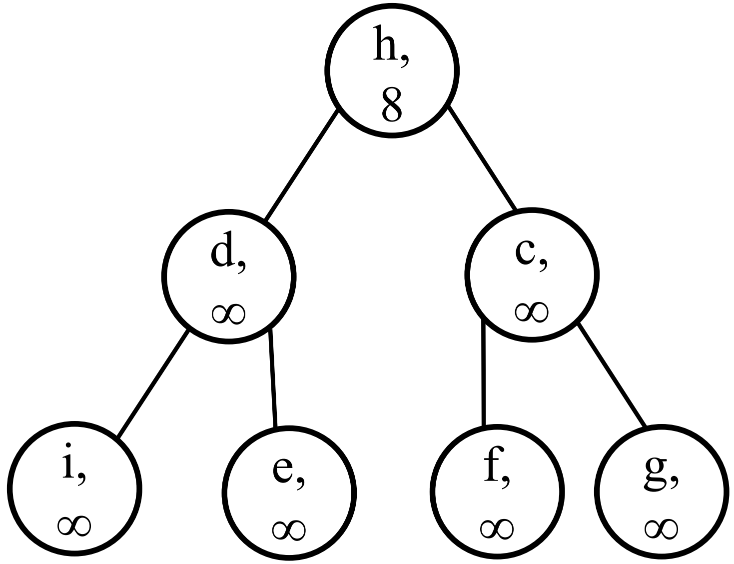




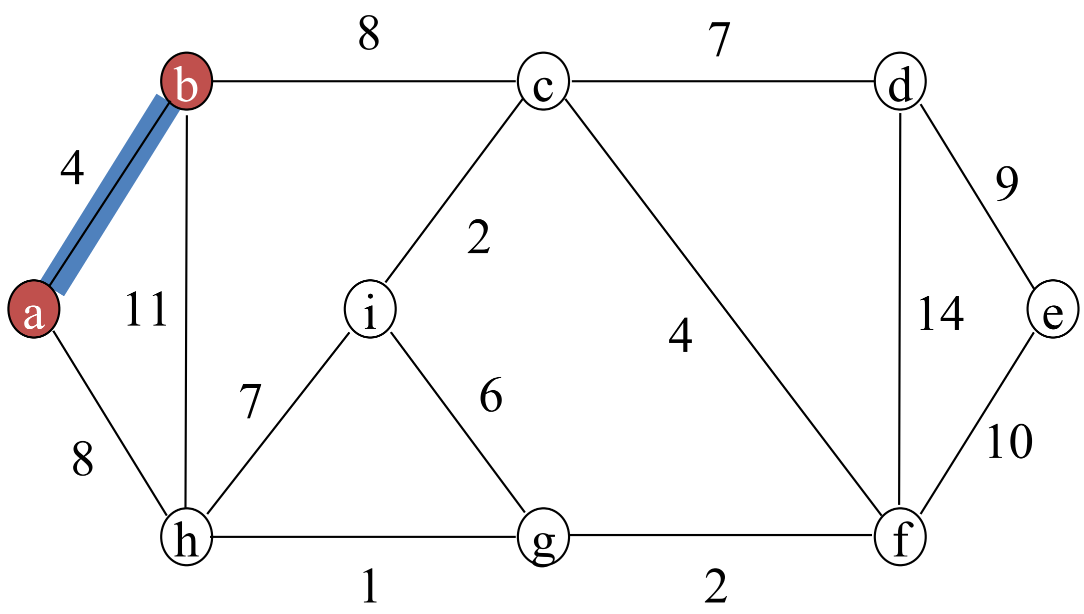
a	b	c	d	e	f	g	h	i
	a						a	

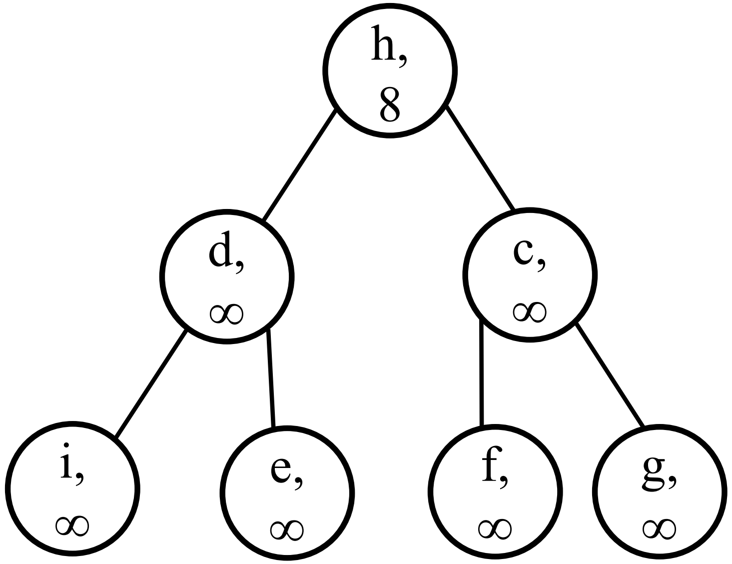
Prim's Algorithm





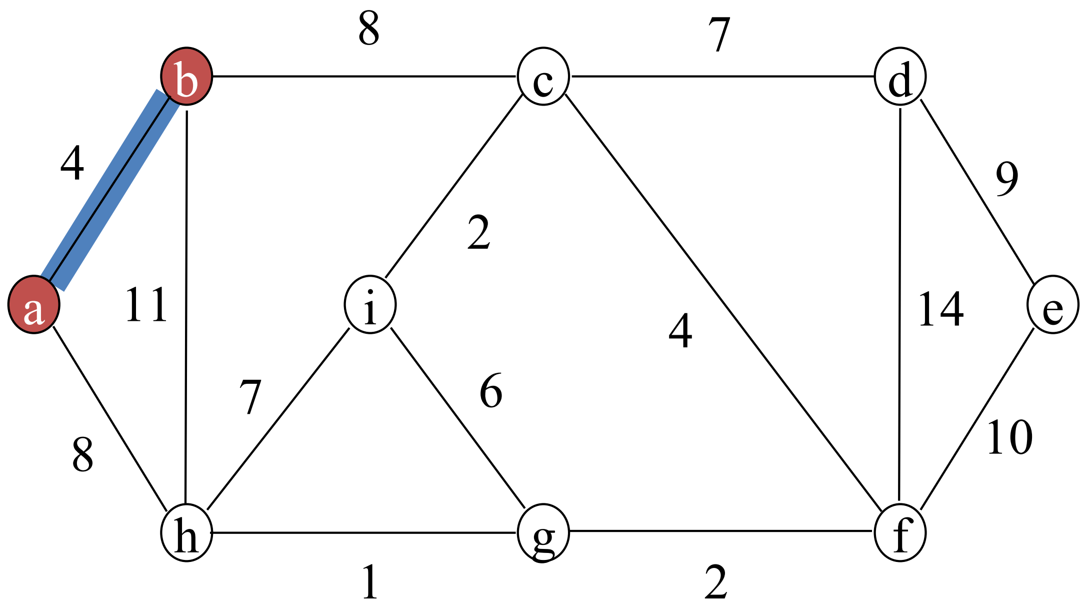
a	b	c	d	e	f	g	h	i
	a						a	

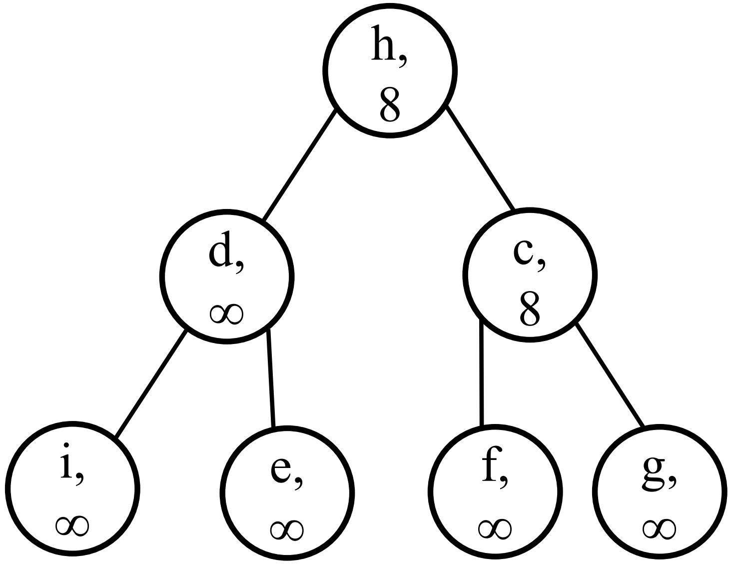




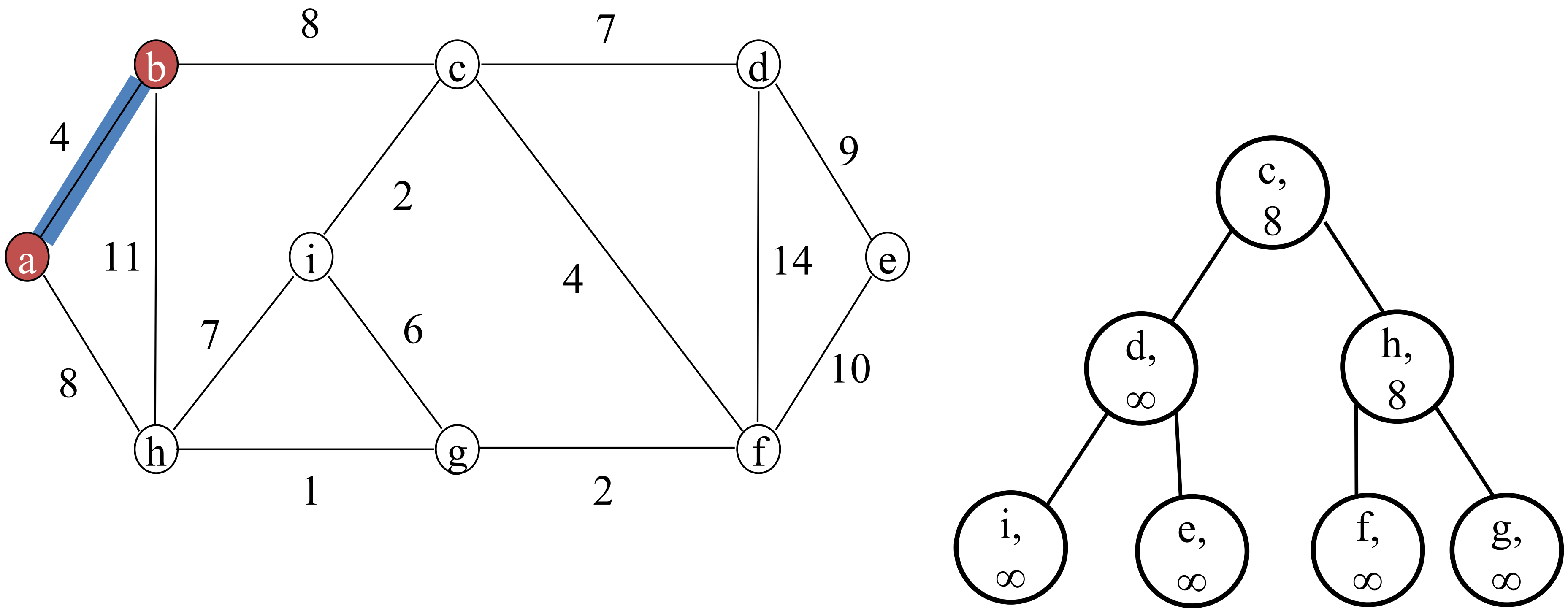
a	b	c	d	e	f	g	h	i
	a	b					a	

Prim’s Algorithm

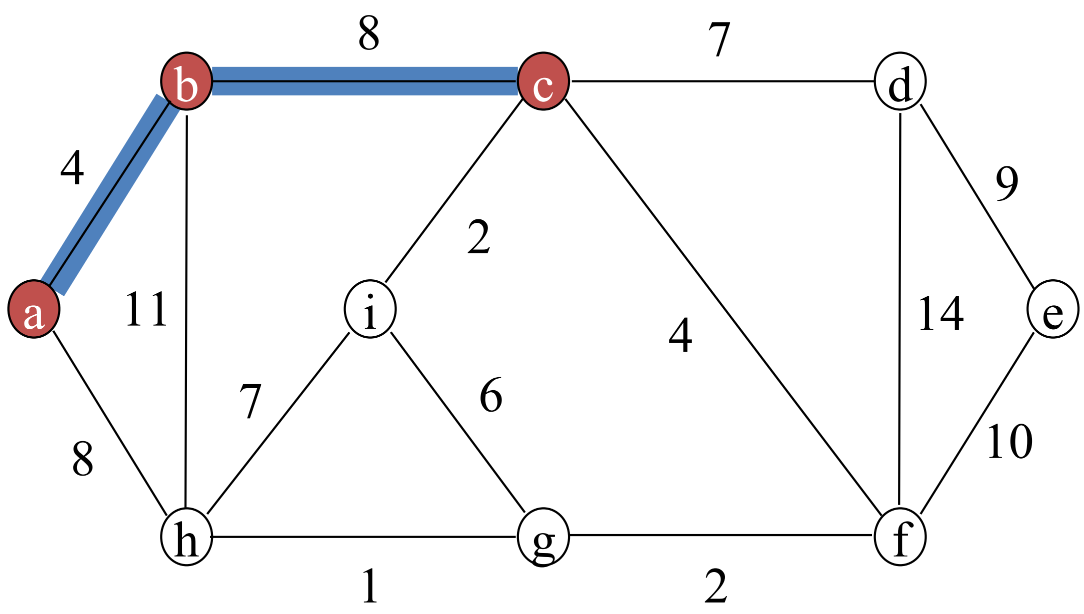


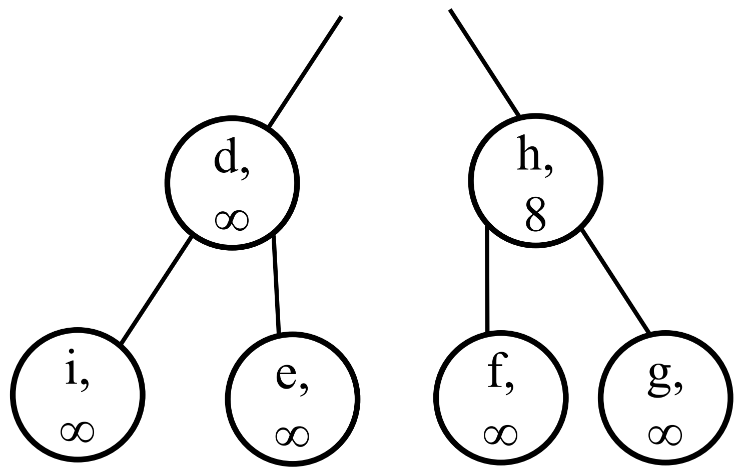


a	b	c	d	e	f	g	h	i
	a	b					a	

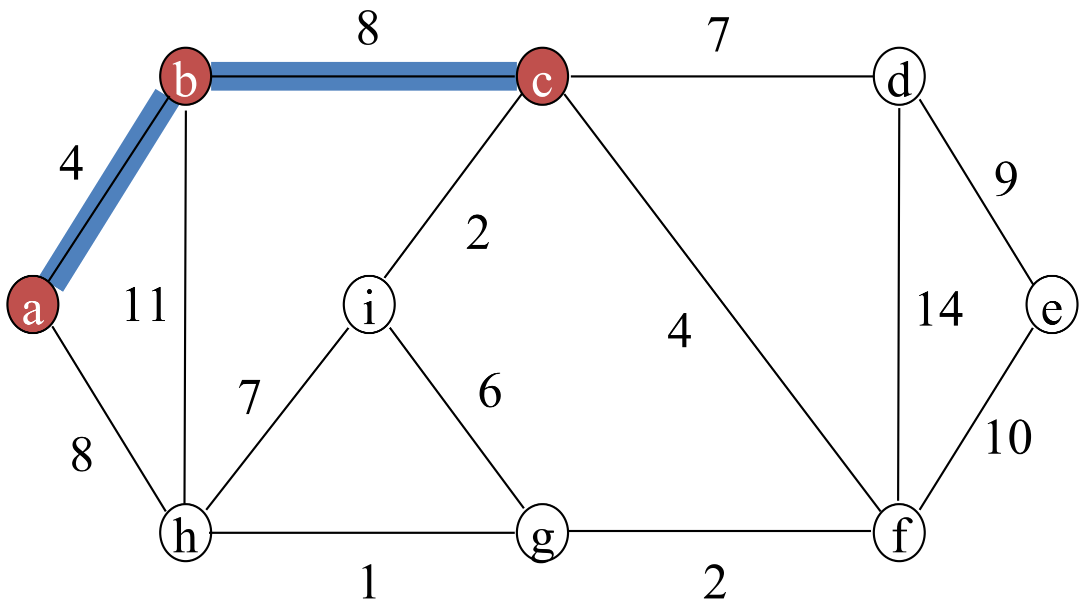


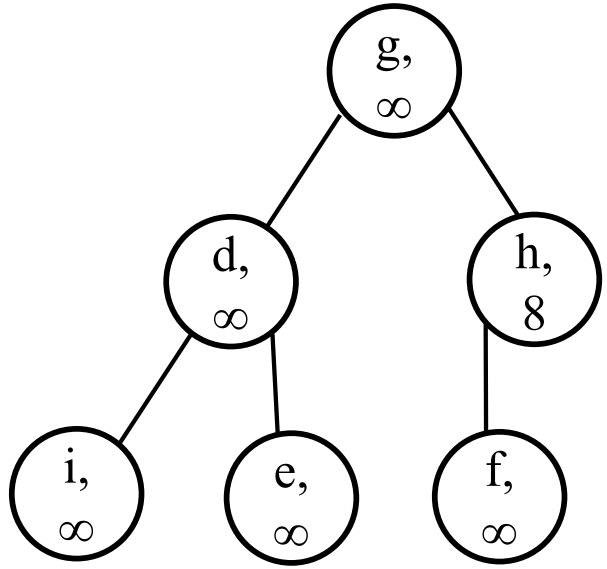
a	b	c	d	e	f	g	h	i
	a	b					a	



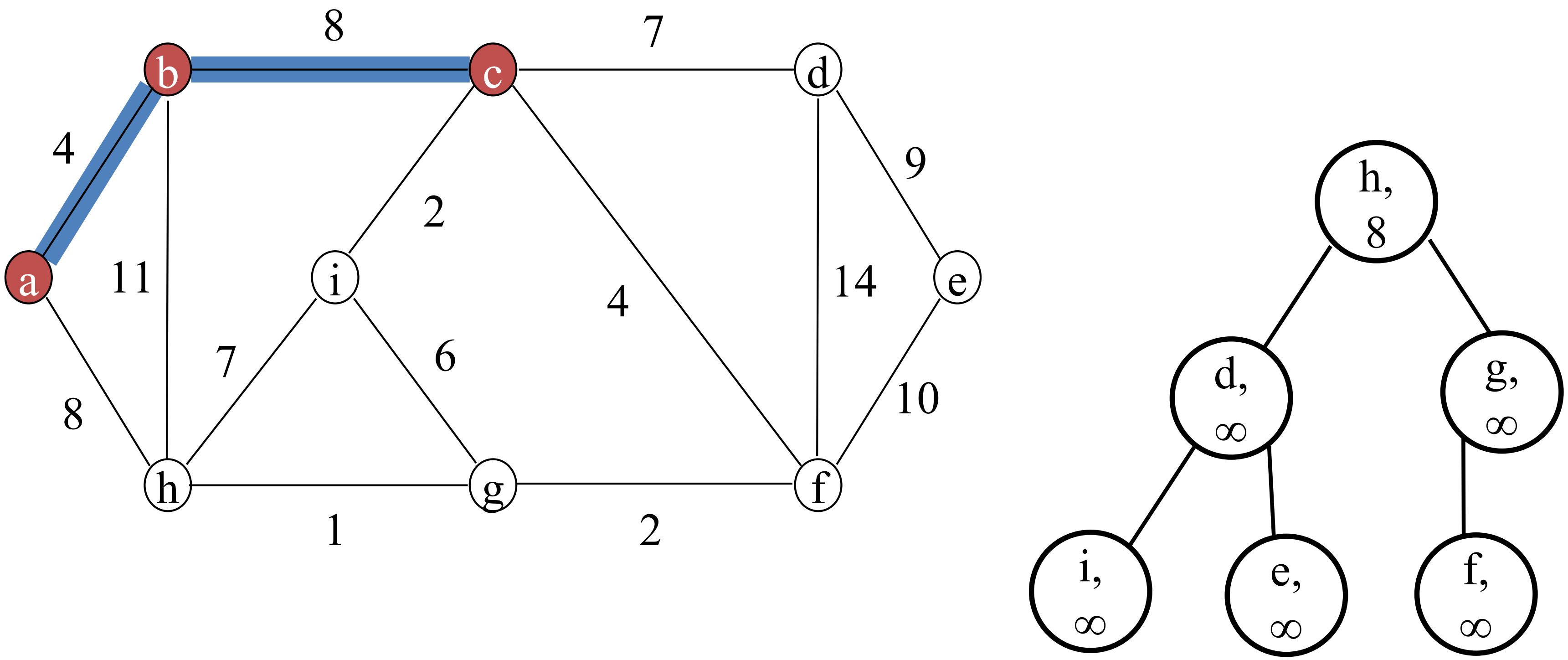


a	b	c	d	e	f	g	h	i
	a	b					a	

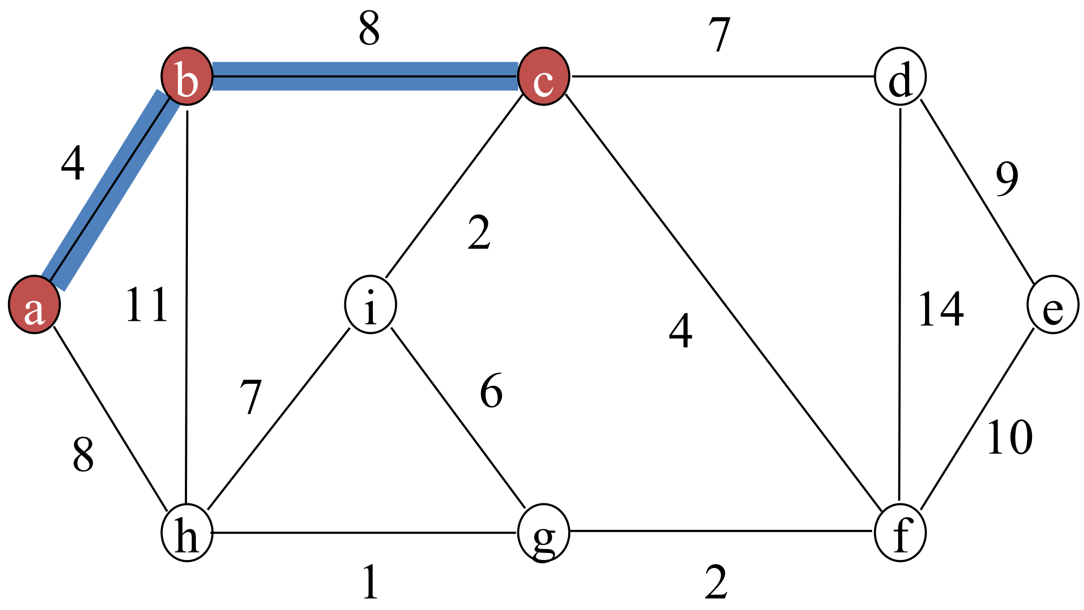


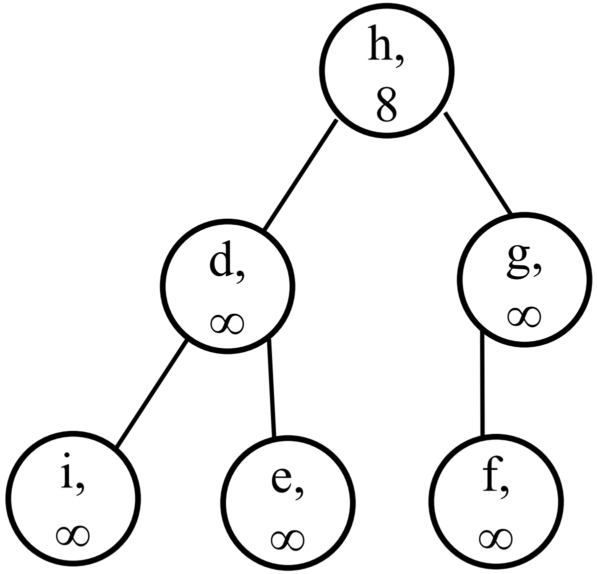


a	b	c	d	e	f	g	h	i
	a	b					a	

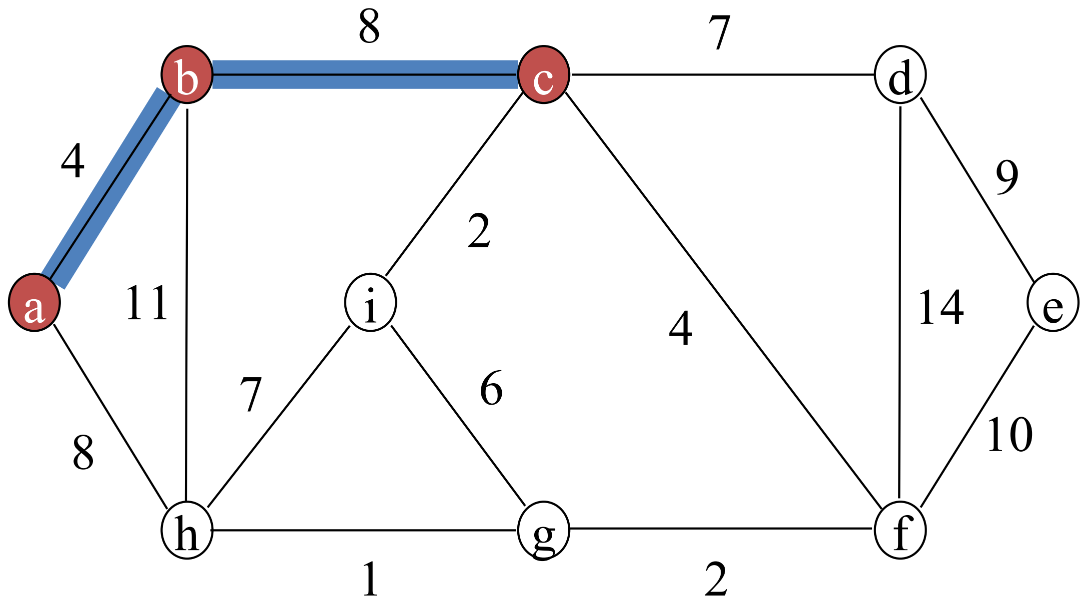


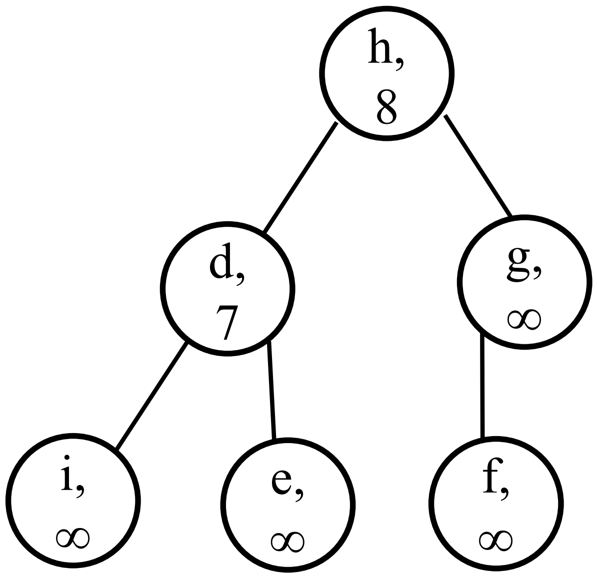
a	b	c	d	e	f	g	h	i
	a	b					a	





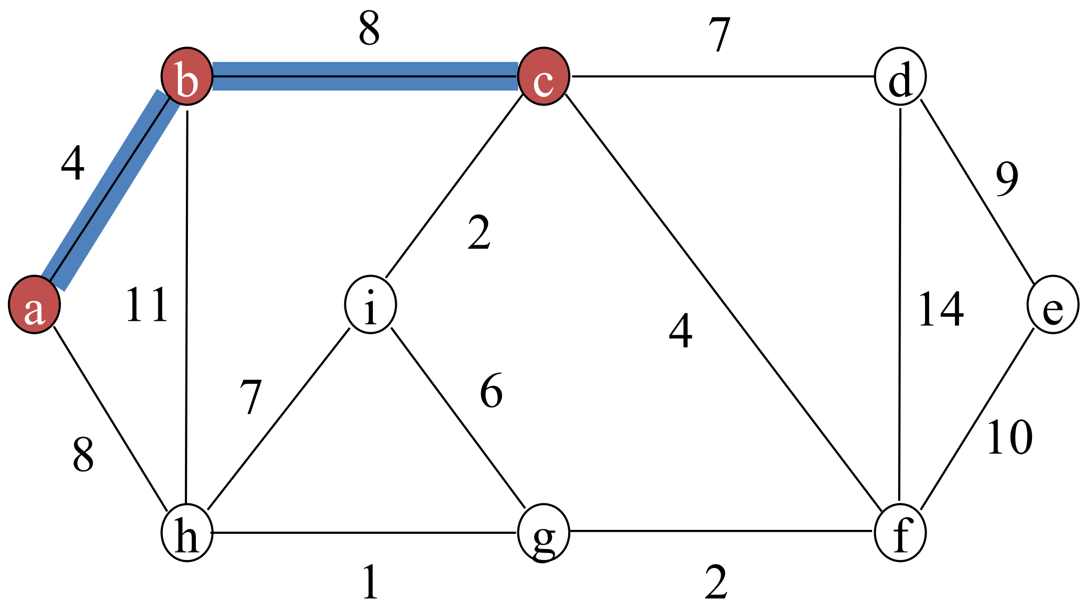
a	b	c	d	e	f	g	h	i
	a	b	c				a	

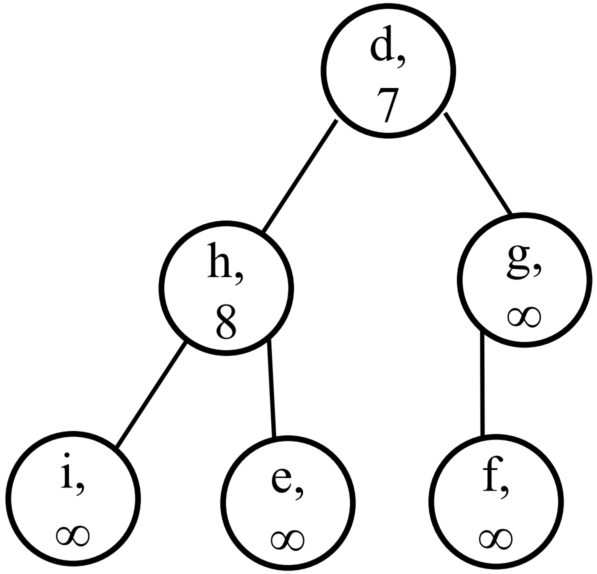




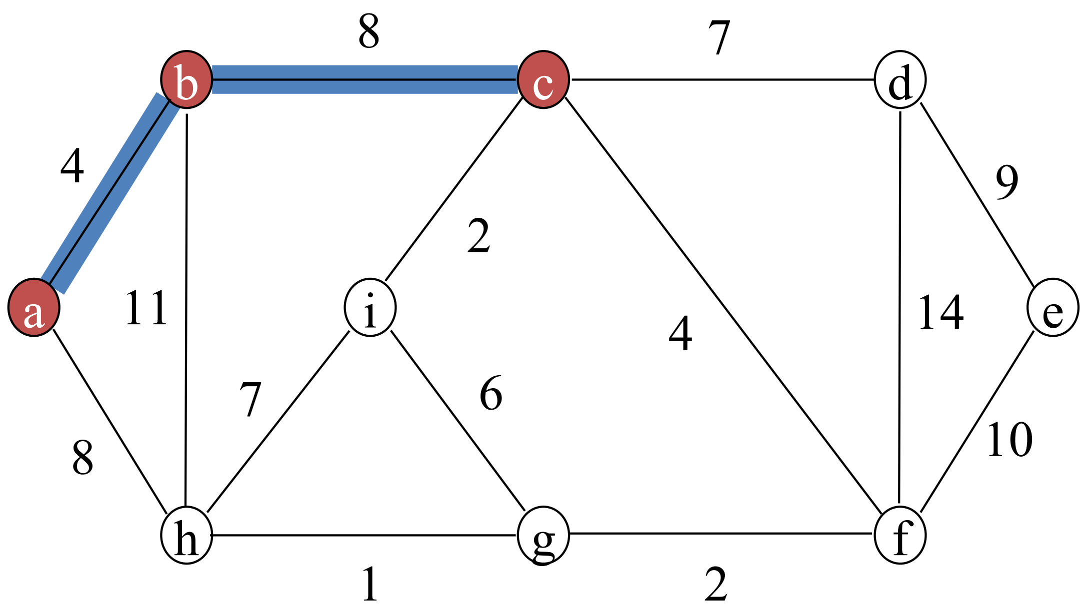
a	b	c	d	e	f	g	h	i
	a	b	c				a	

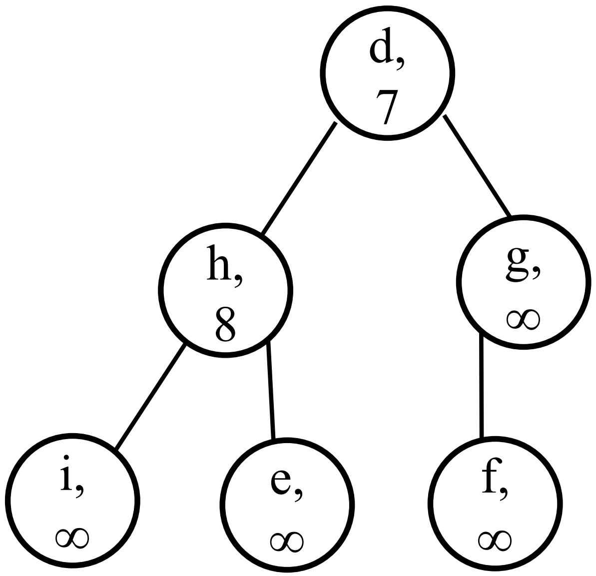
Prim's Algorithm



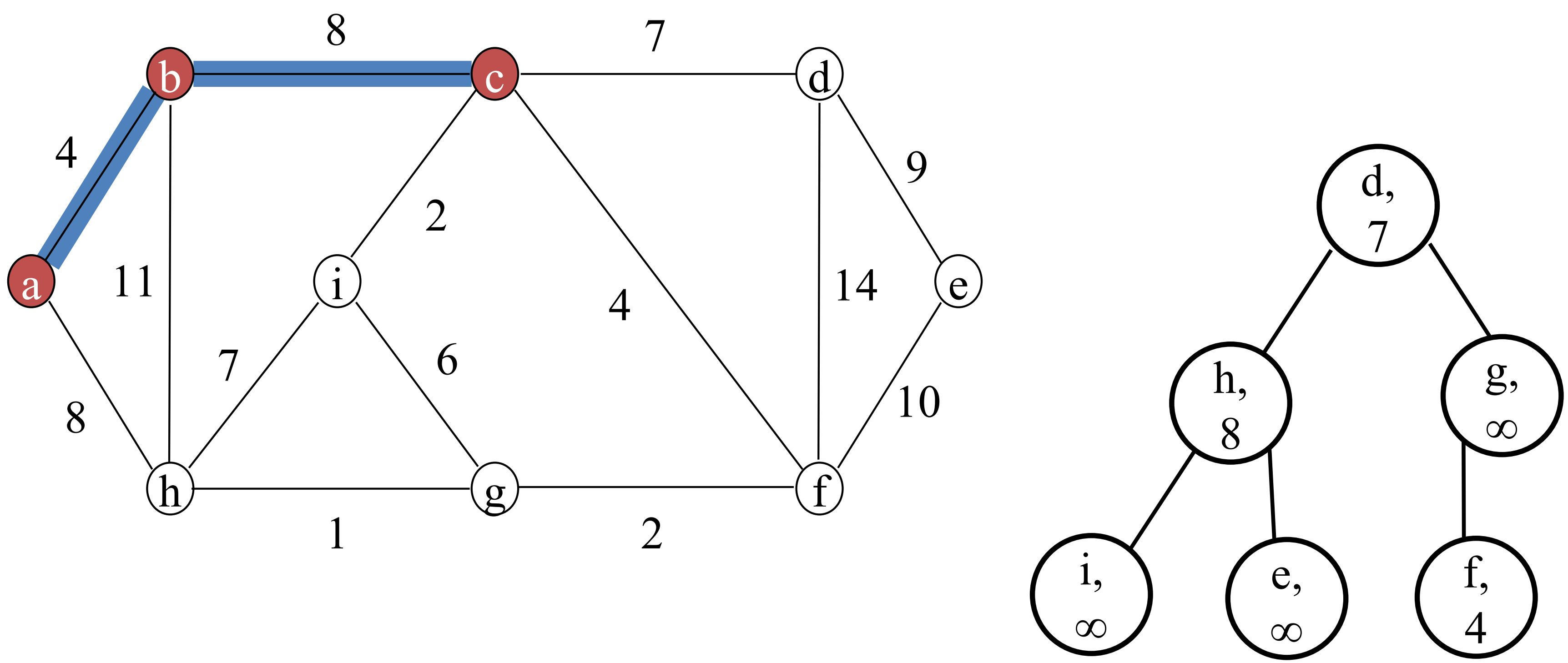


a	b	c	d	e	f	g	h	i
	a	b	c				a	



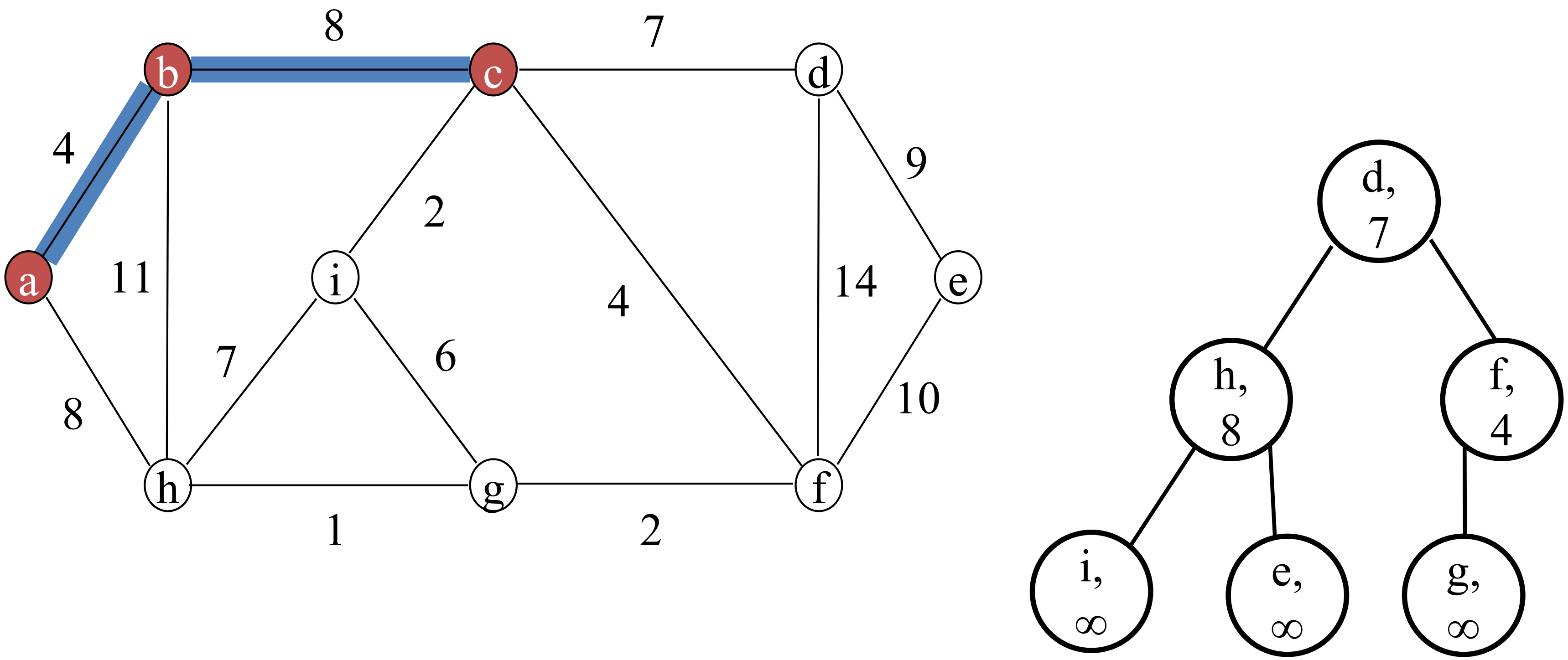


a	b	c	d	e	f	g	h	i
	a	b	c		c		a	

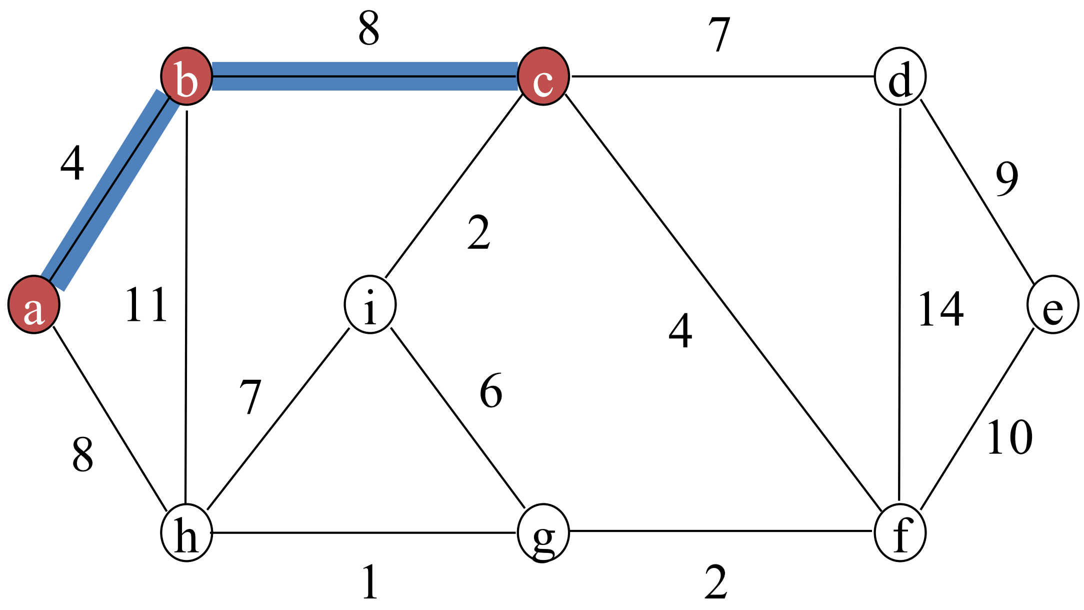


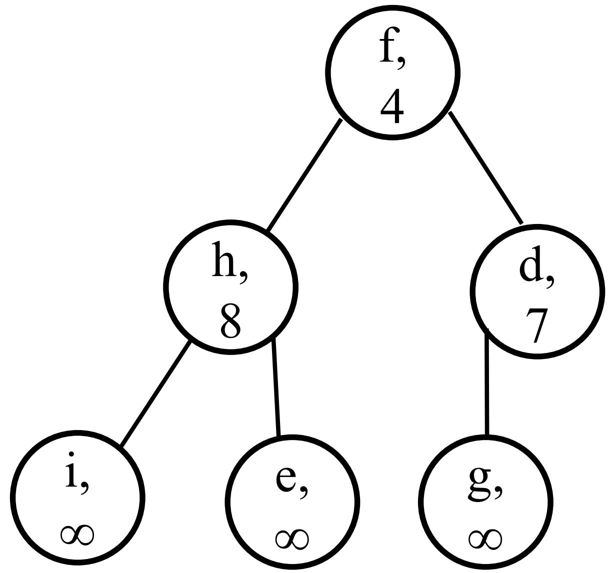
a	b	c	d	e	f	g	h	i
	a	b	c		c		a	

Prim's Algorithm

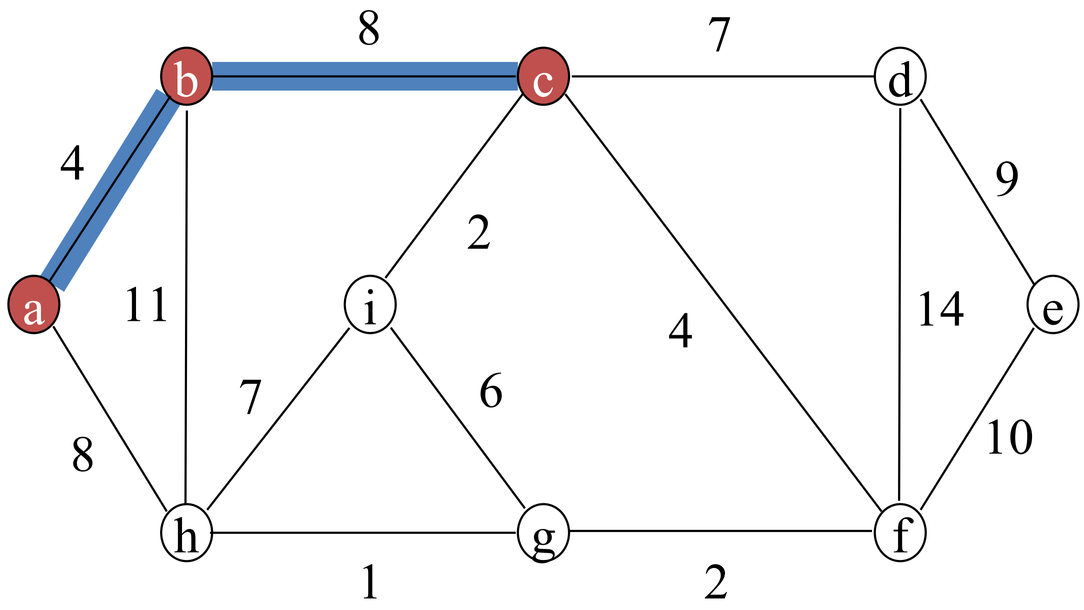


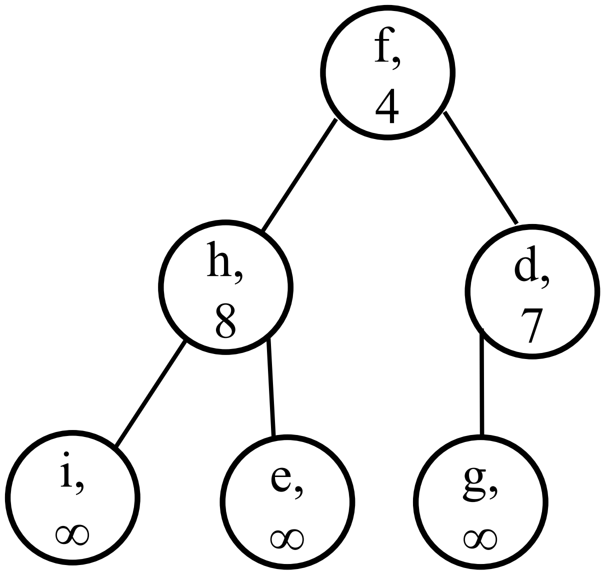
a	b	c	d	e	f	g	h	i
	a	b	c		c		a	





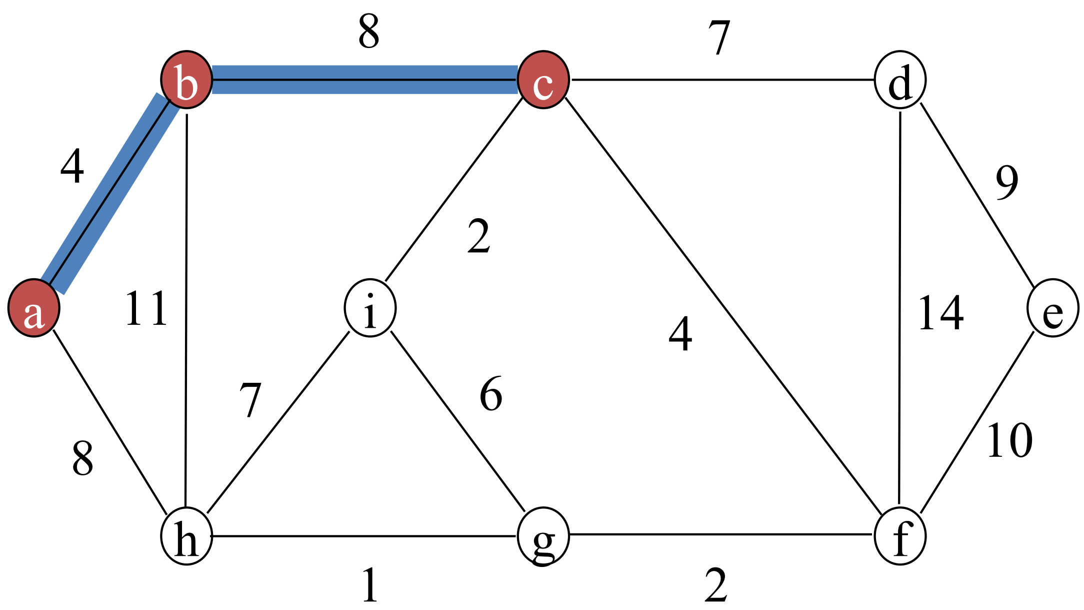
a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**
	a	b	c		c		a	

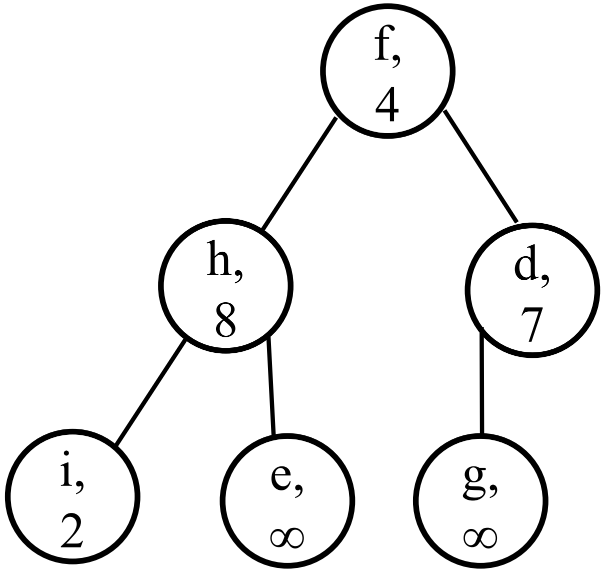




a	b	c	d	e	f	g	h	i
	a	b	c		c		a	c

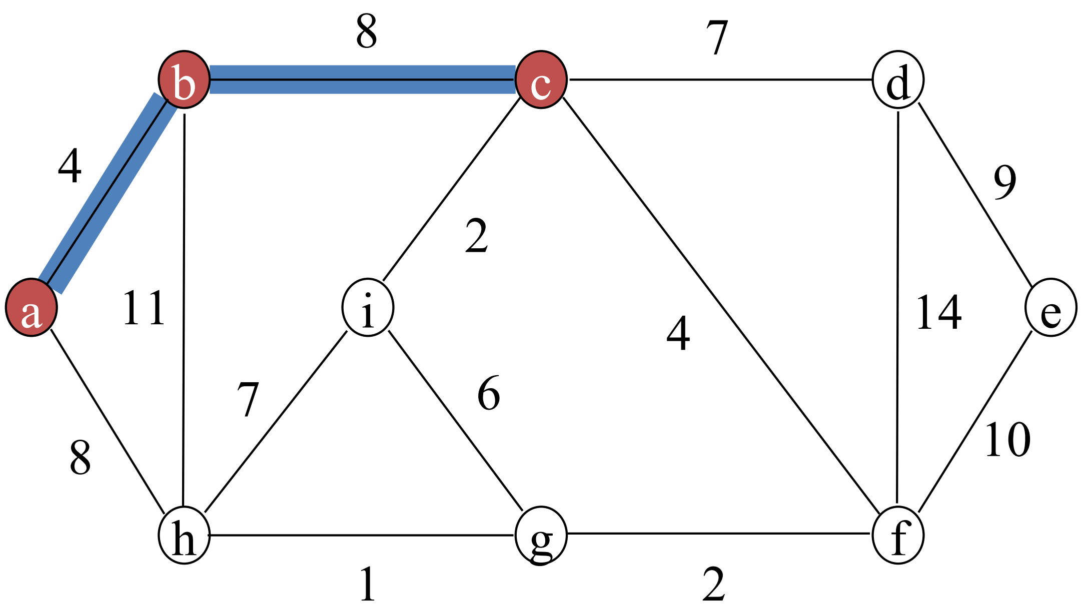
Prim's Algorithm

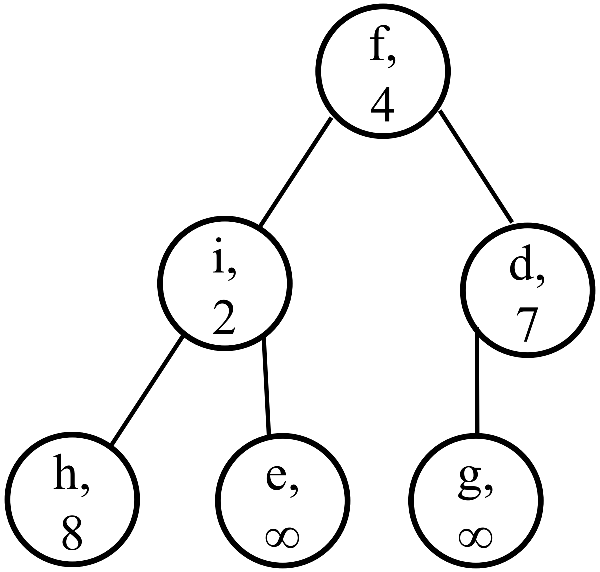




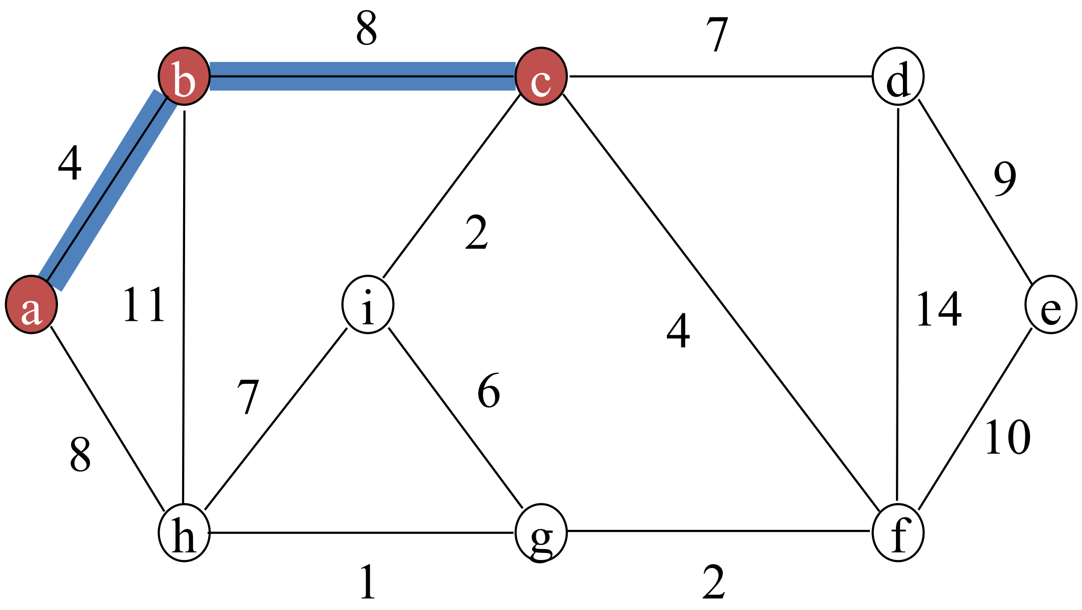
a	b	c	d	e	f	g	h	i
	a	b	c		c		a	c

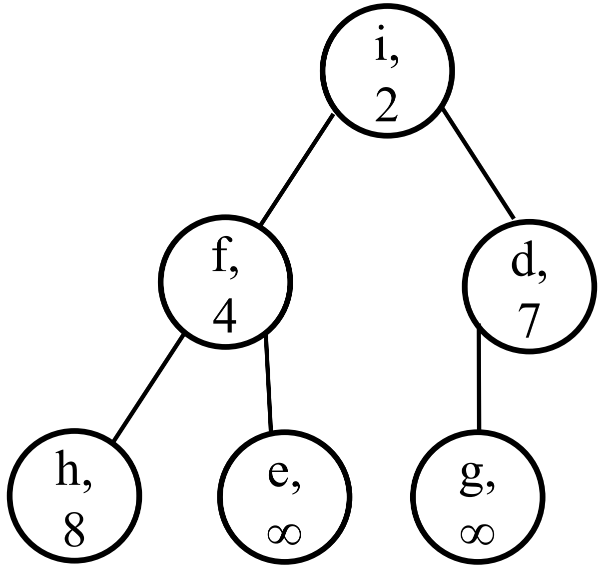
Prim's Algorithm



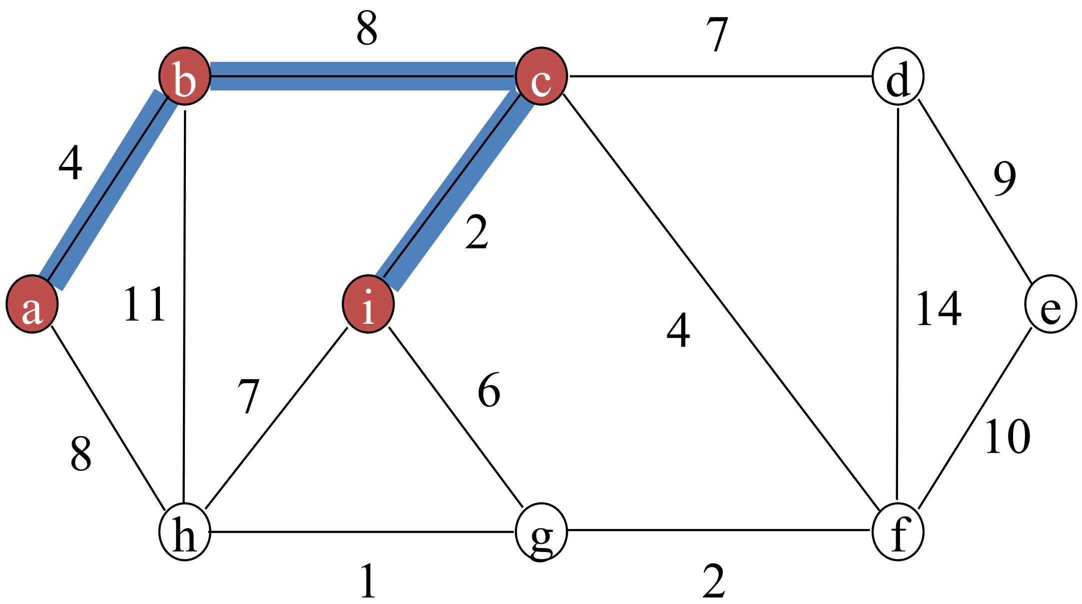


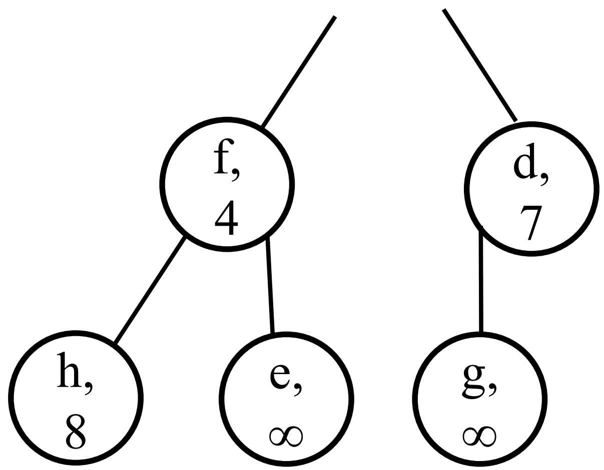
a	b	c	d	e	f	g	h	i
	a	b	c		c		a	c





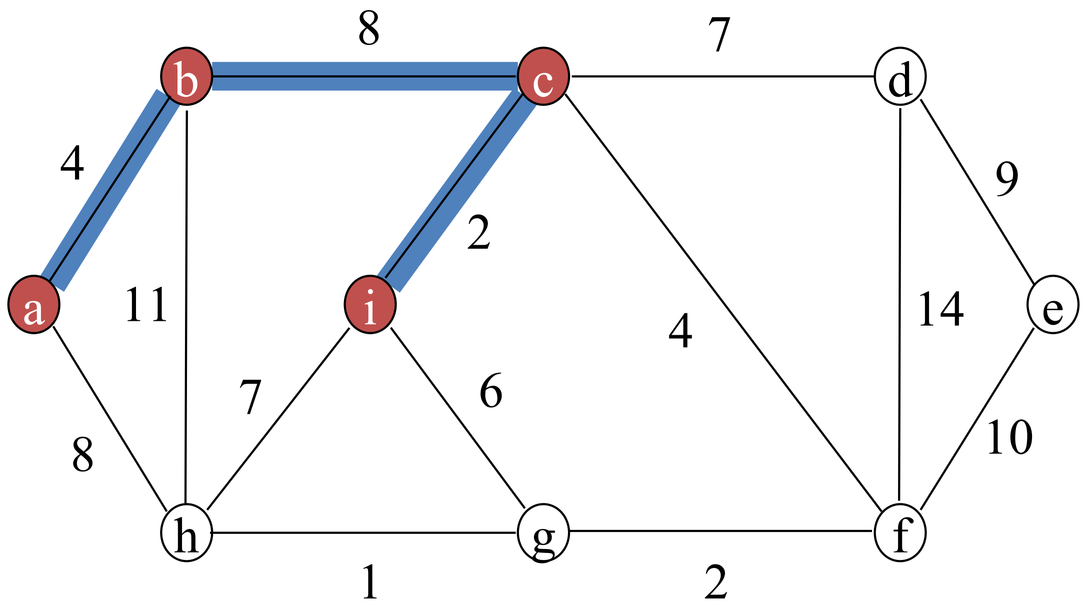
a	b	c	d	e	f	g	h	i
	a	b	c		c		a	c

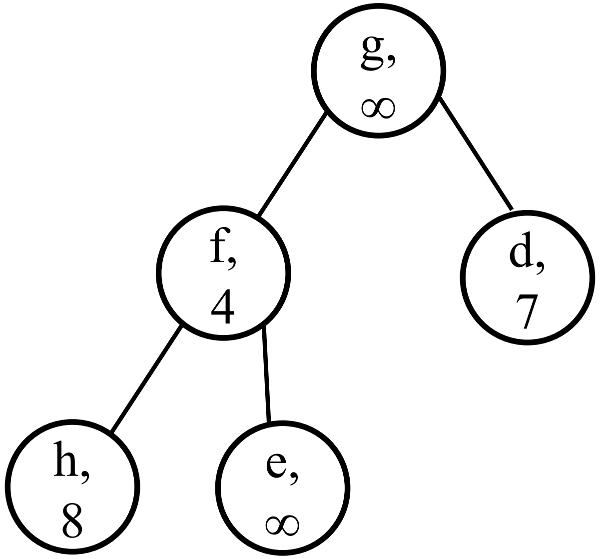




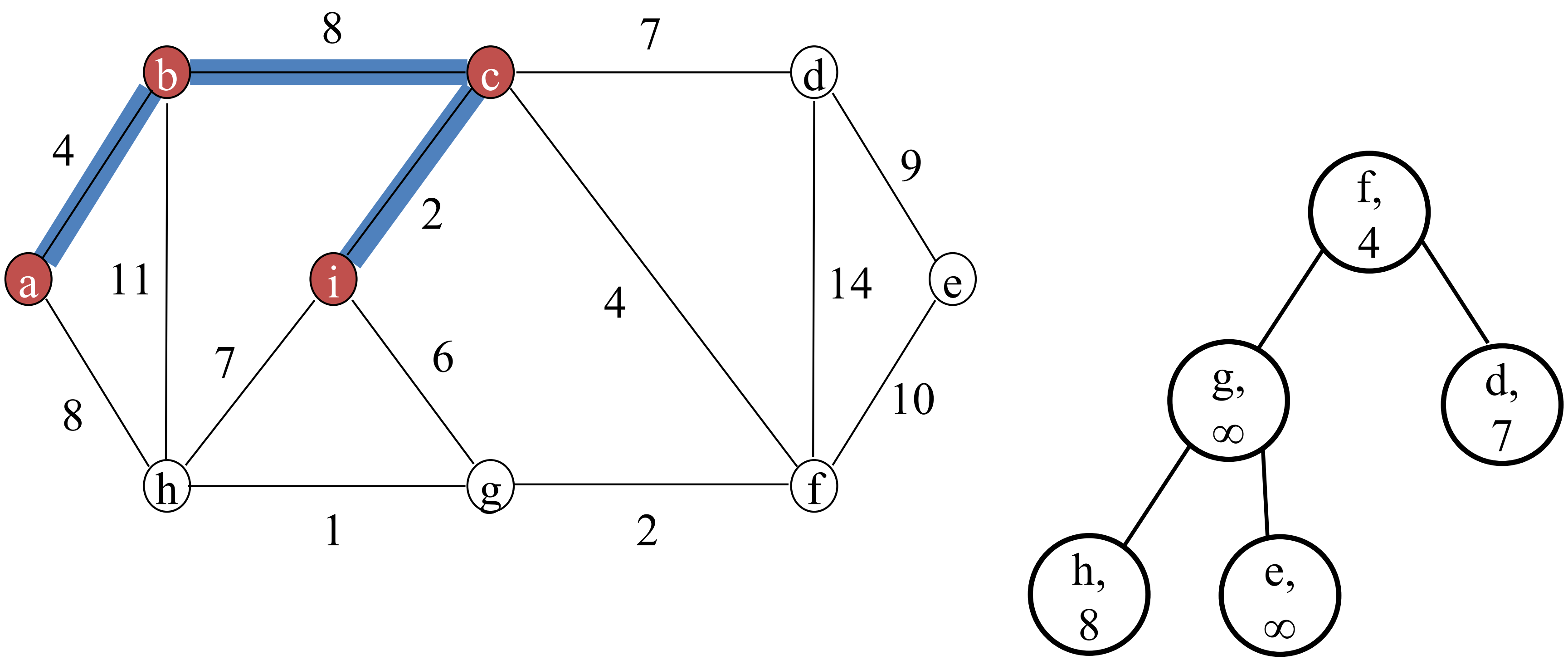
a	b	c	d	e	f	g	h	i
	a	b	c		c		a	c

Prim's Algorithm

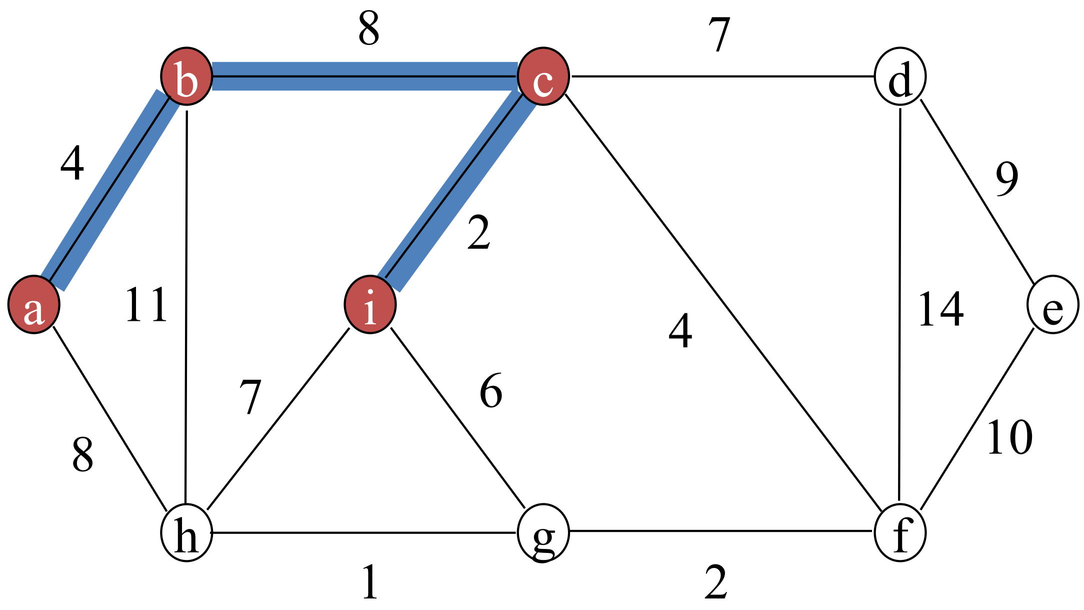


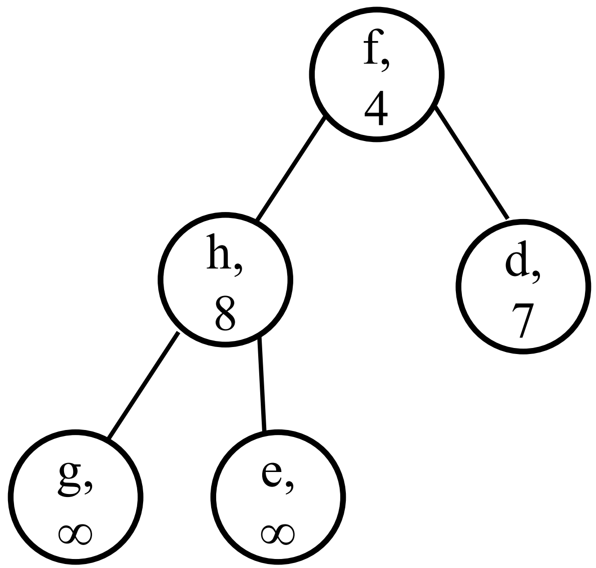


a	b	c	d	e	f	g	h	i
	a	b	c		c		a	c

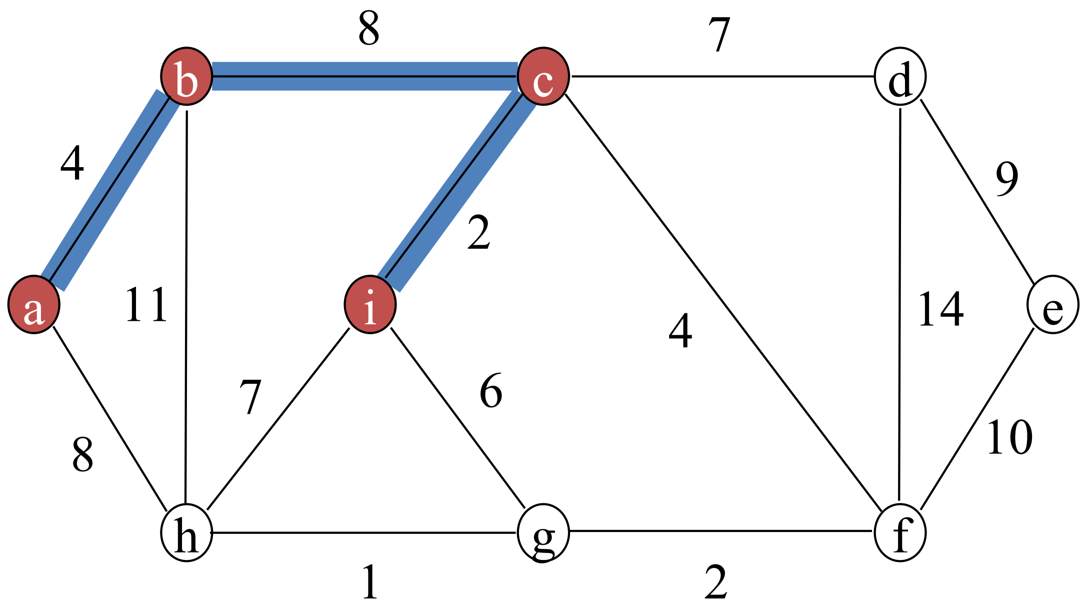


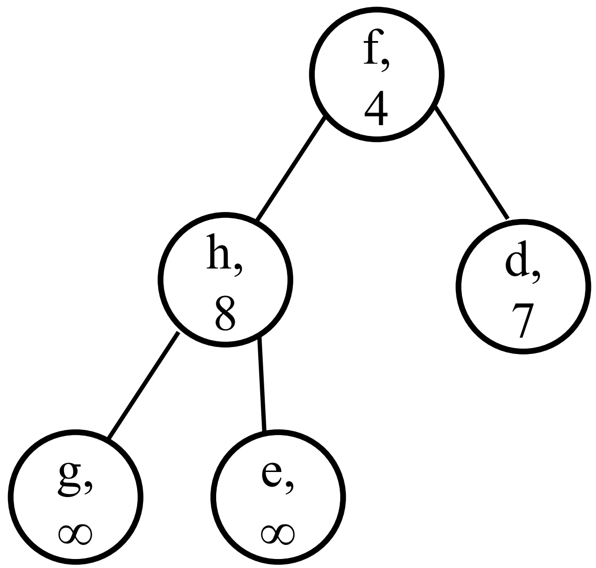
a	b	c	d	e	f	g	h	i
	a	b	c		c		a	c





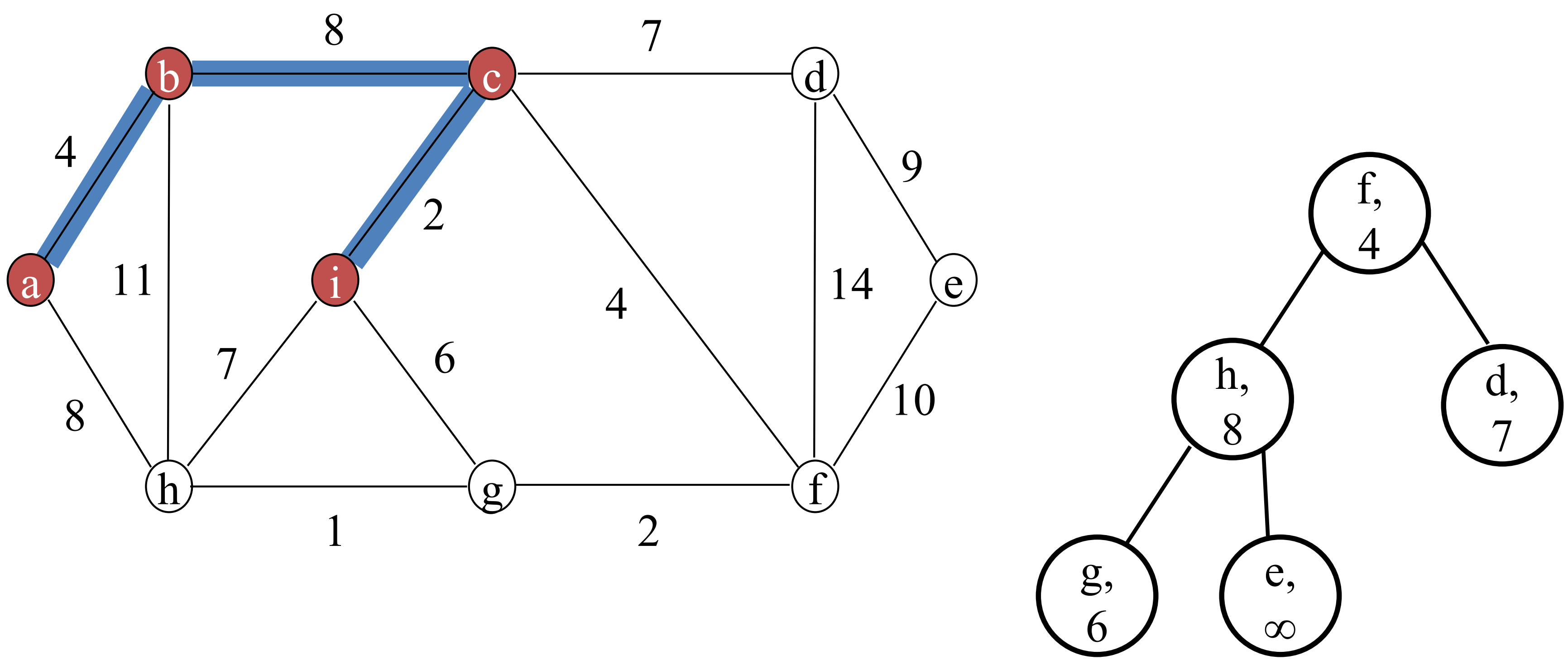
a	b	c	d	e	f	g	h	i
	a	b	c		c		a	c





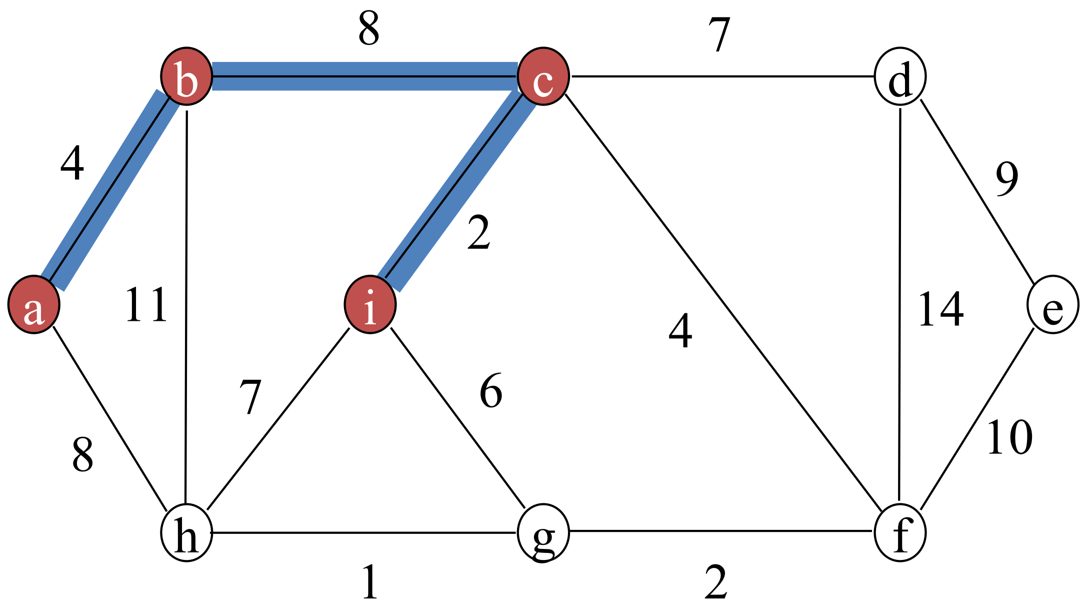
a	b	c	d	e	f	g	h	i
	a	b	c		c	i	a	c

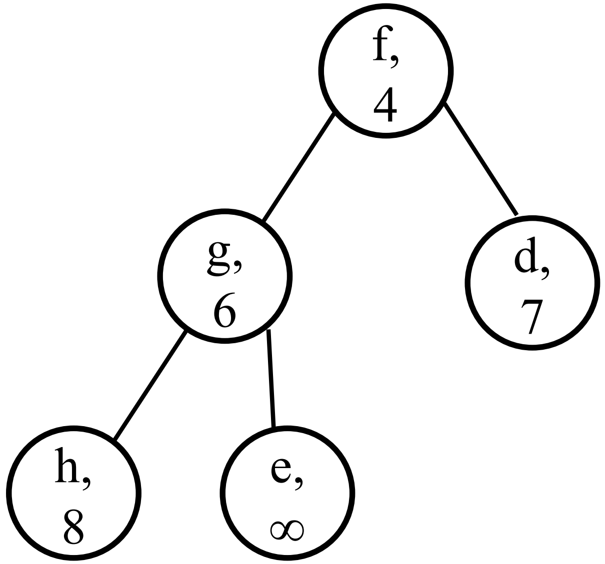
Prim's Algorithm



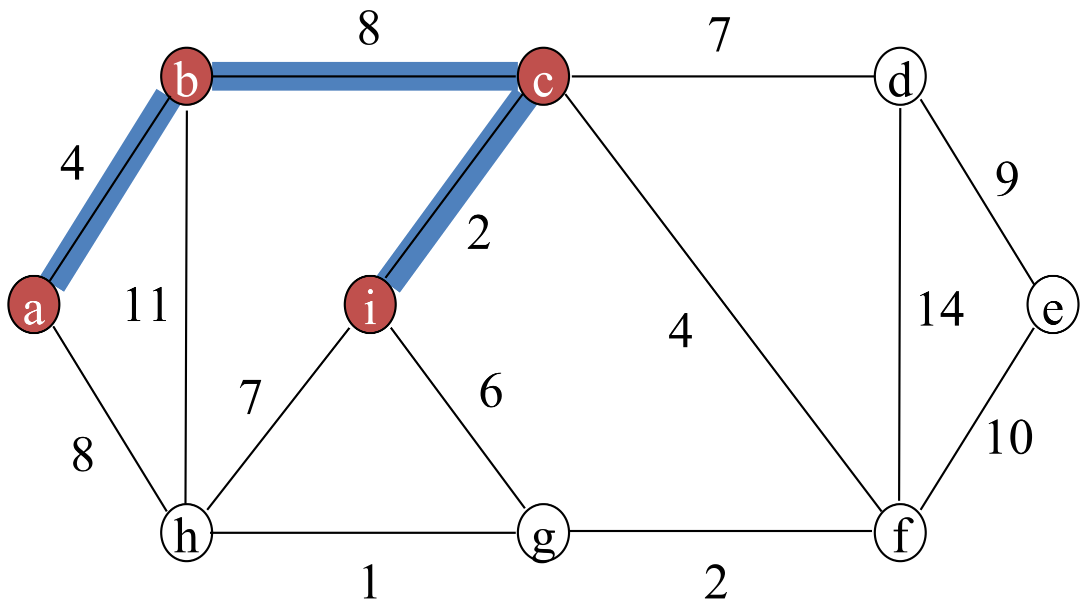
a	b	c	d	e	f	g	h	i
	a	b	c		c	i	a	c

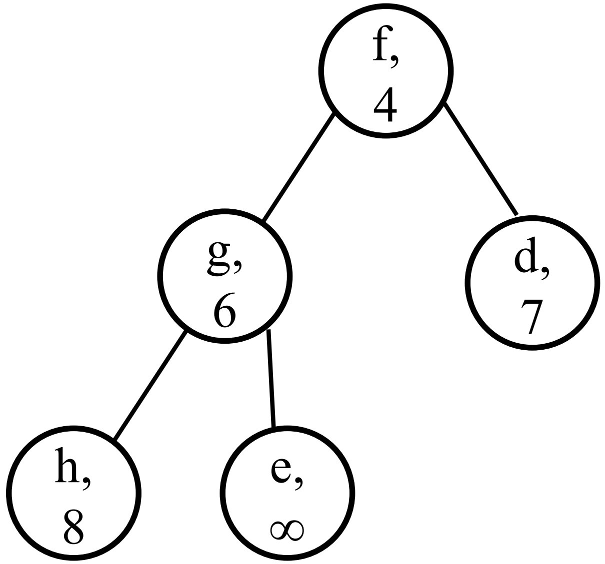
Prim's Algorithm



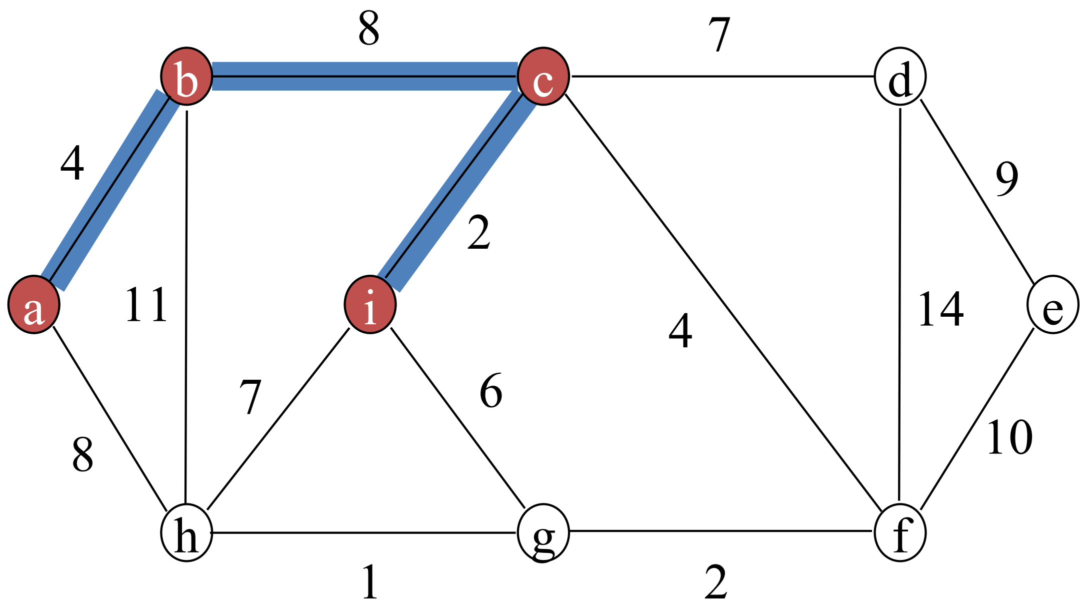


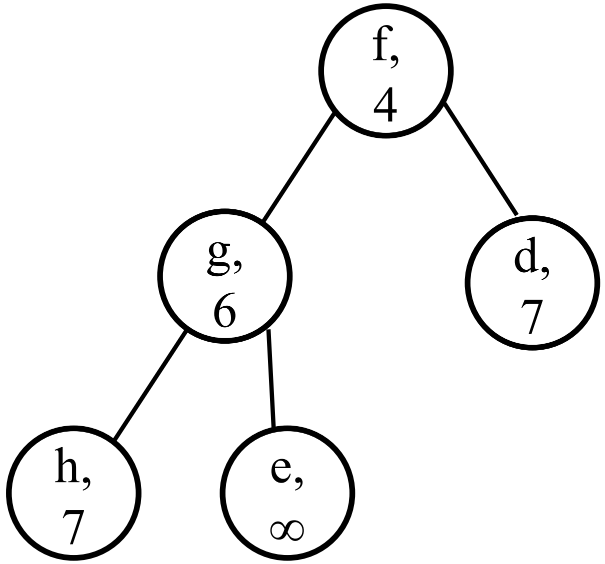
a	b	c	d	e	f	g	h	i
	a	b	c		c	i	a	c





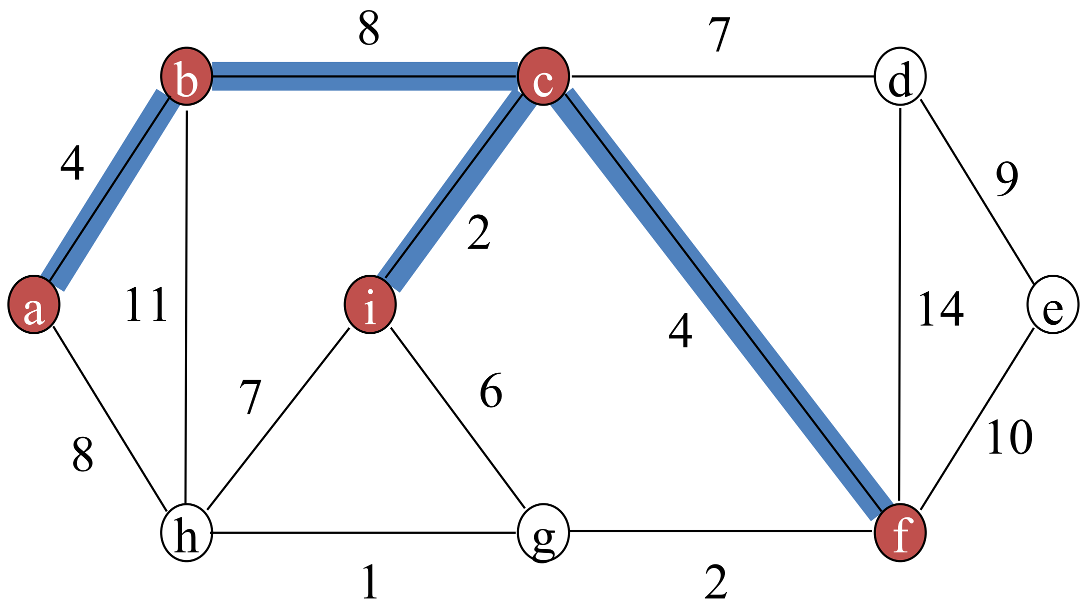
a	b	c	d	e	f	g	h	i
	a	b	c		c	i	i	c

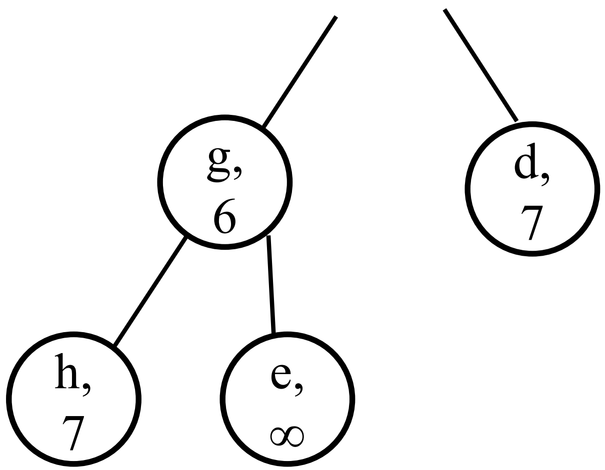




a	b	c	d	e	f	g	h	i
	a	b	c		c	i	i	c

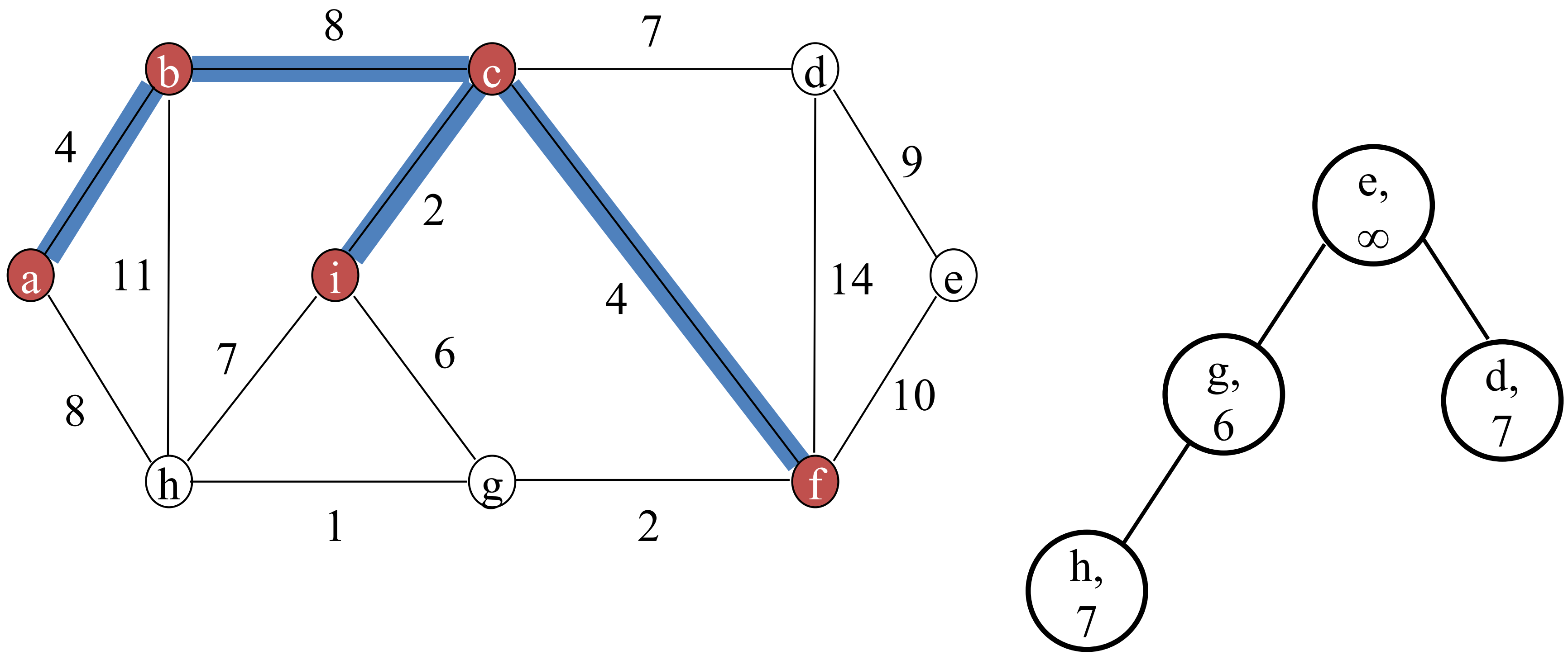
Prim's Algorithm





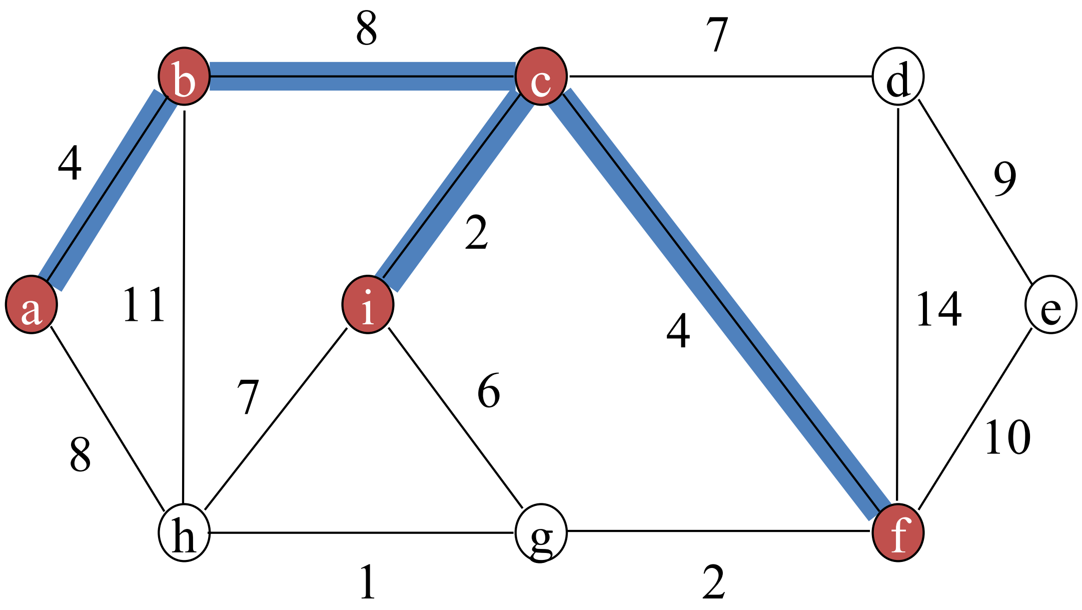
a	b	c	d	e	f	g	h	i
	a	b	c		c	i	i	c

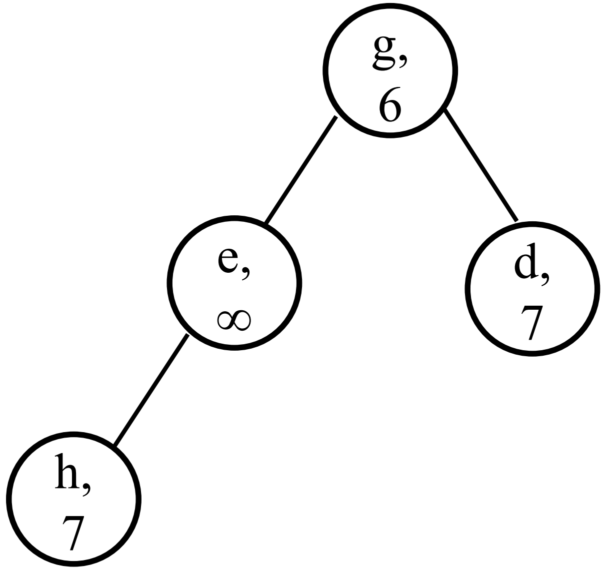
Prim's Algorithm



a	b	c	d	e	f	g	h	i
	a	b	c		c	i	i	c

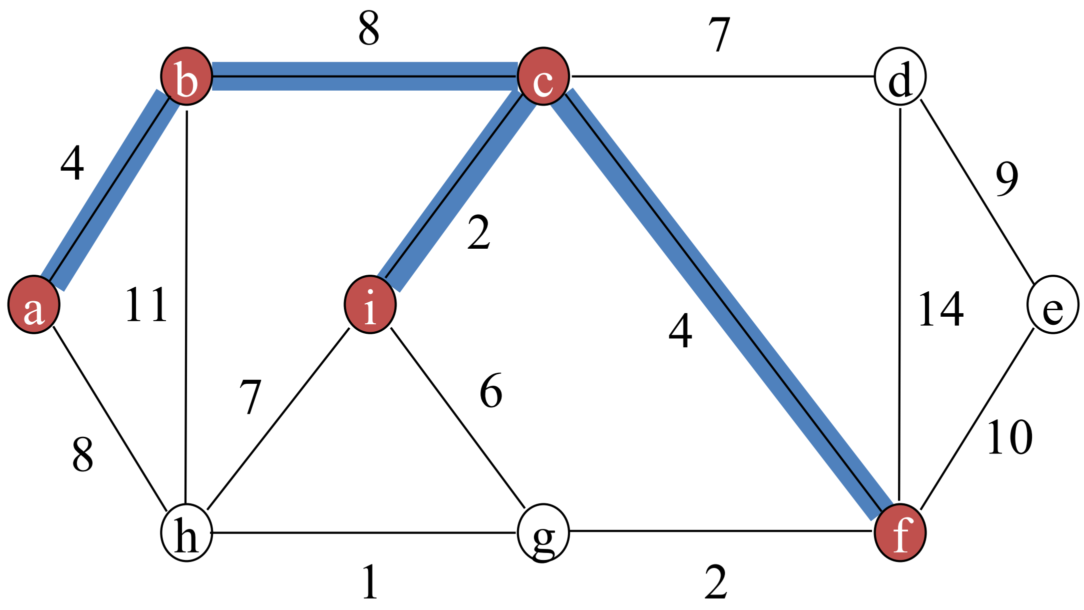
Prim's Algorithm

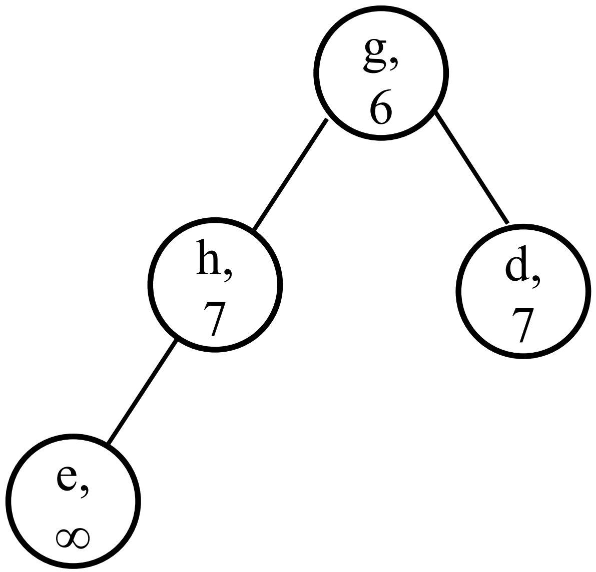




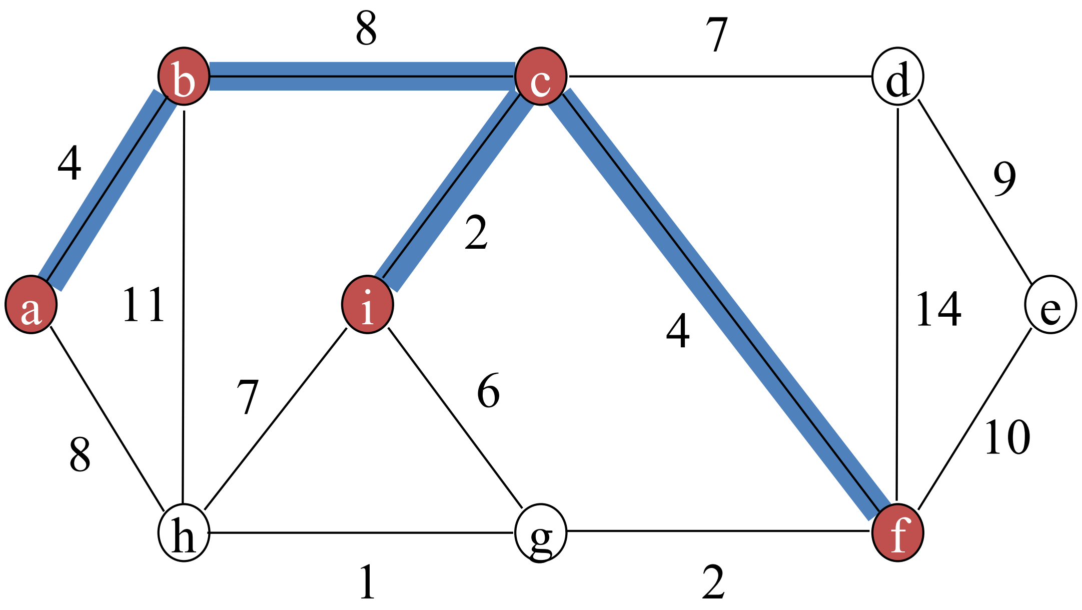
a	b	c	d	e	f	g	h	i
	a	b	c		c	i	i	c

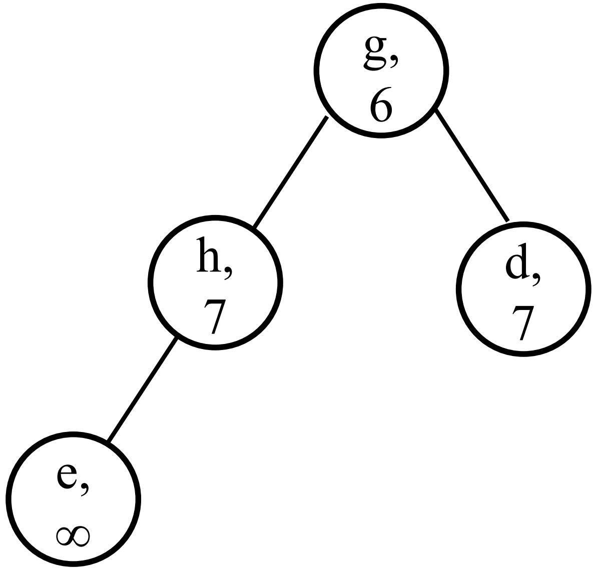
Prim's Algorithm



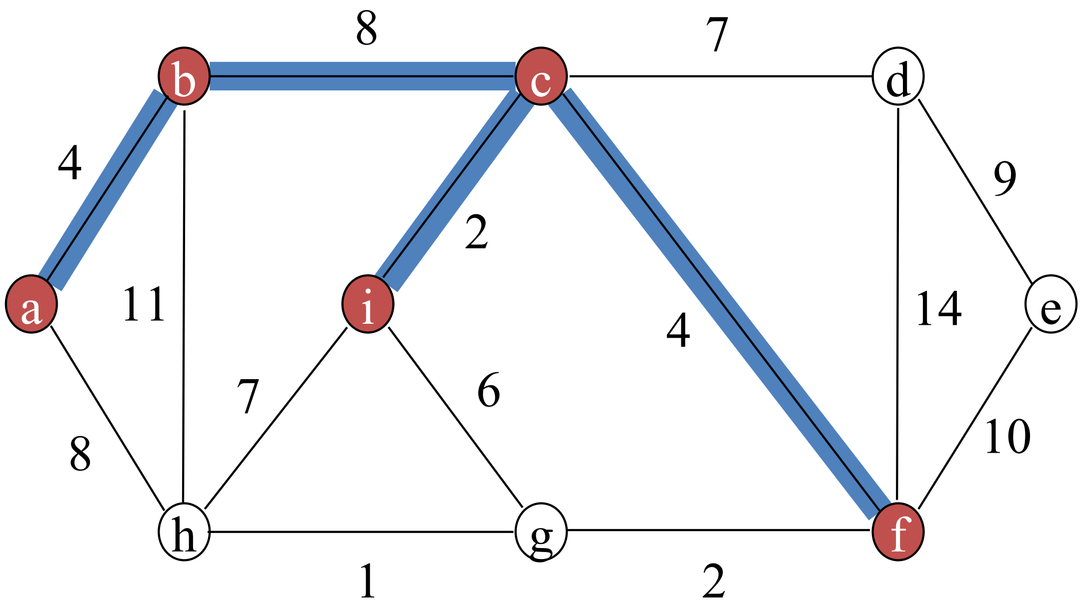


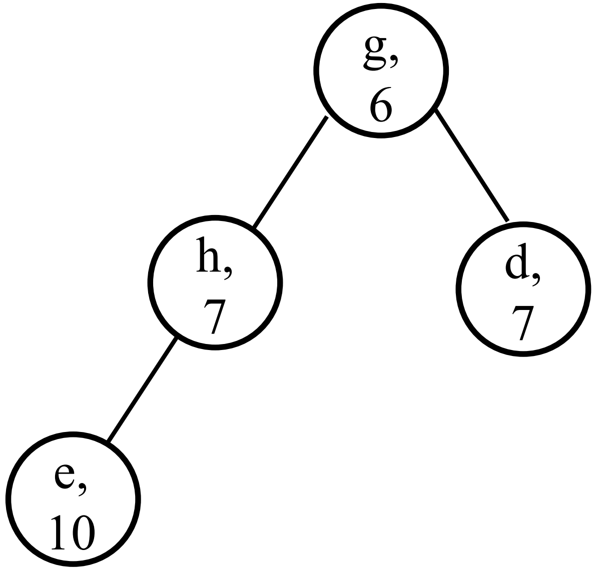
a	b	c	d	e	f	g	h	i
	a	b	c		c	i	i	c





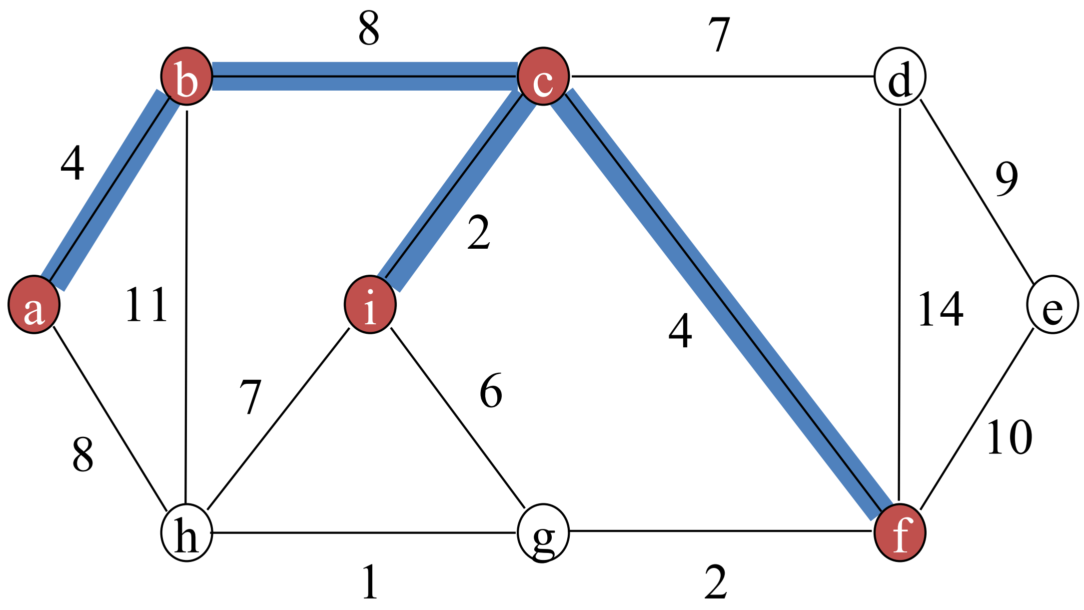
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	i	i	c

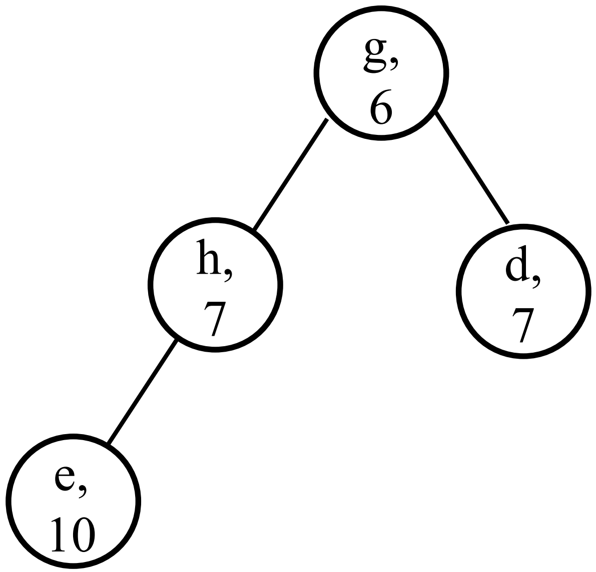




a	b	c	d	e	f	g	h	i
	a	b	c	f	c	i	i	c

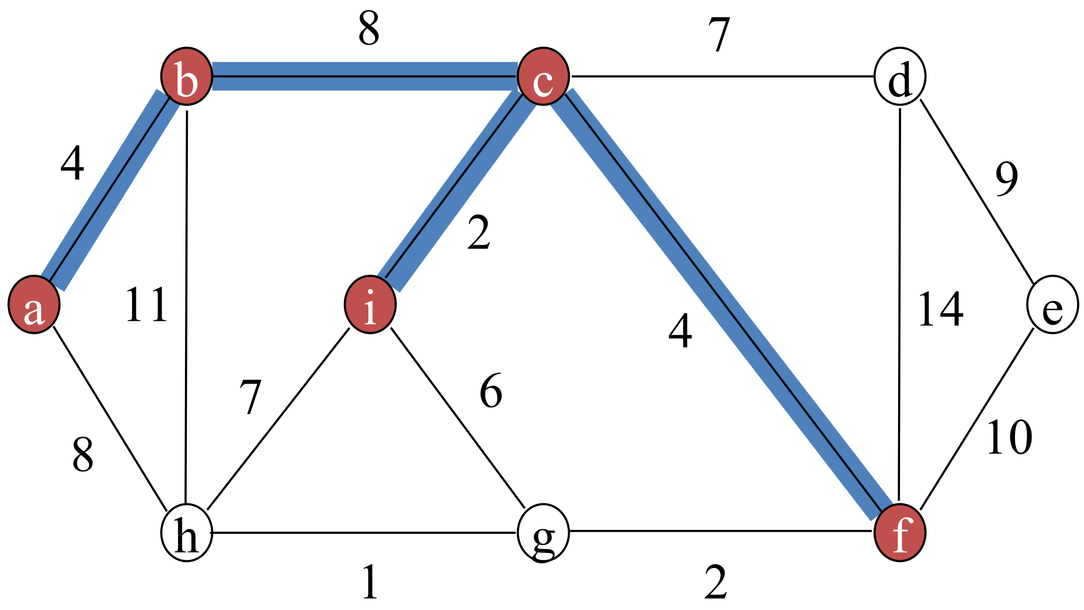
Prim's Algorithm

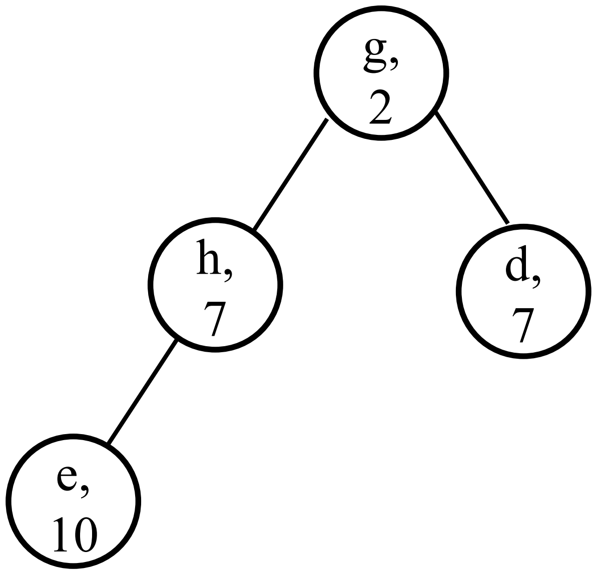




a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	i	c

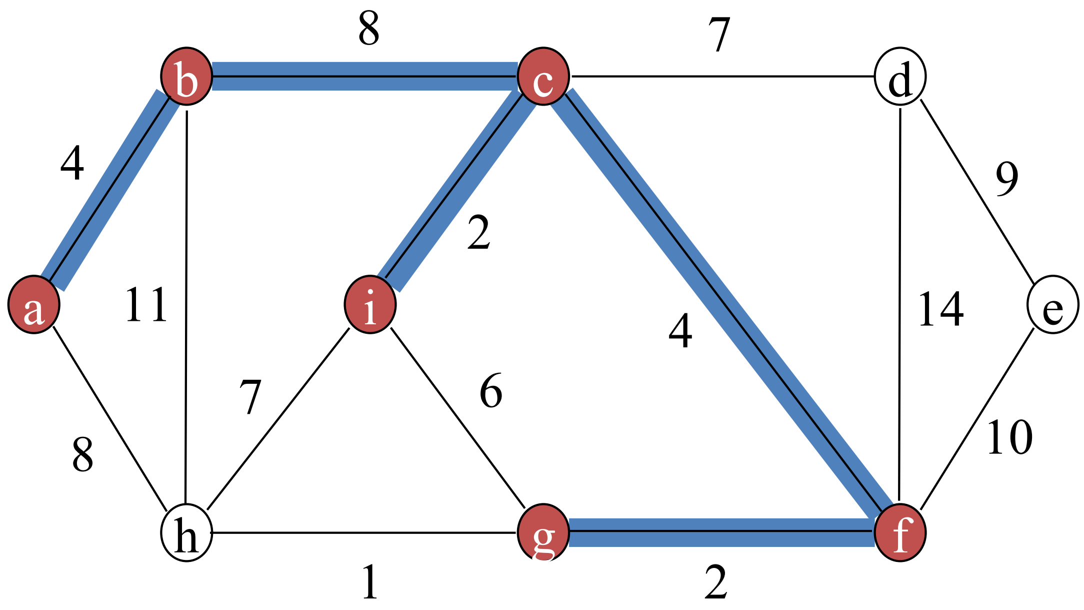
Prim’s Algorithm

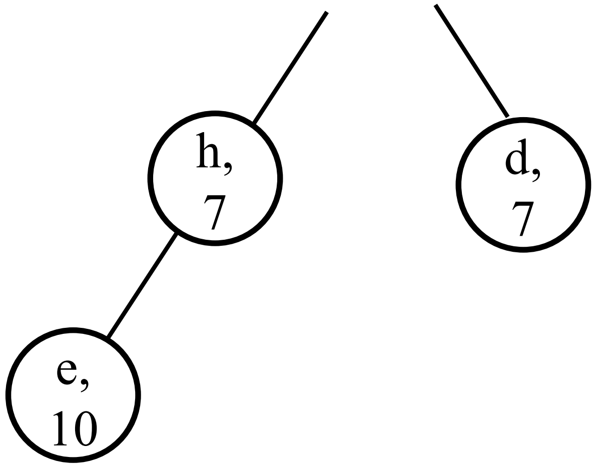




a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	i	c

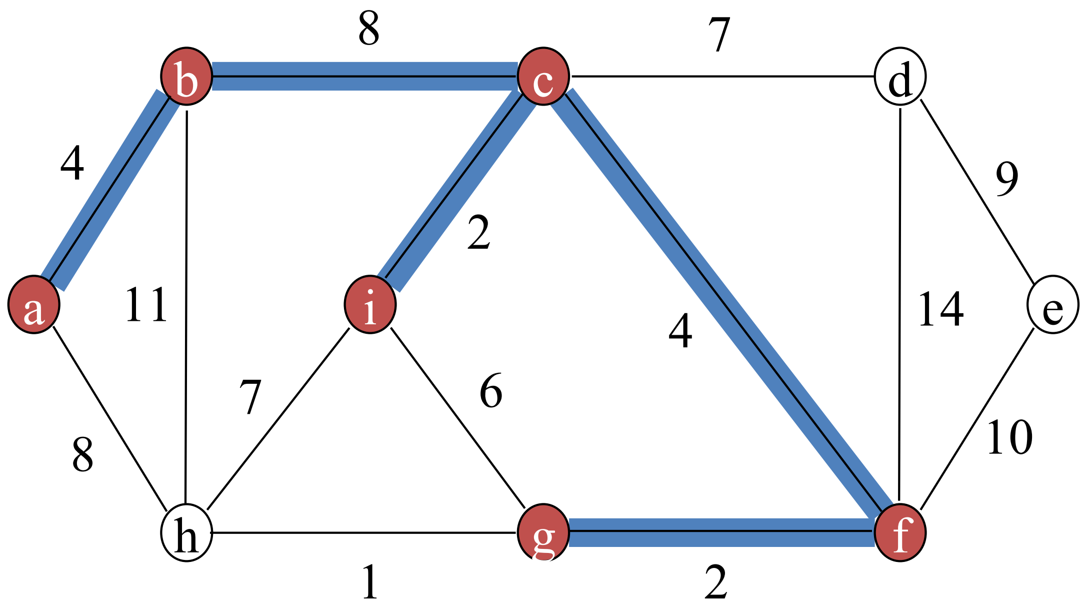
Prim's Algorithm

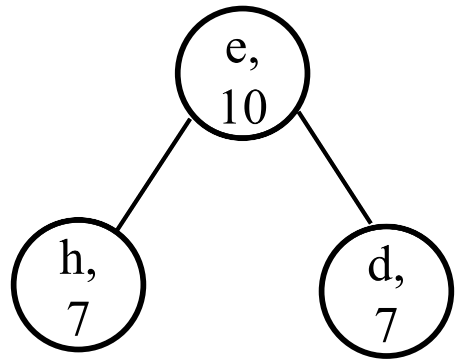




a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	i	c

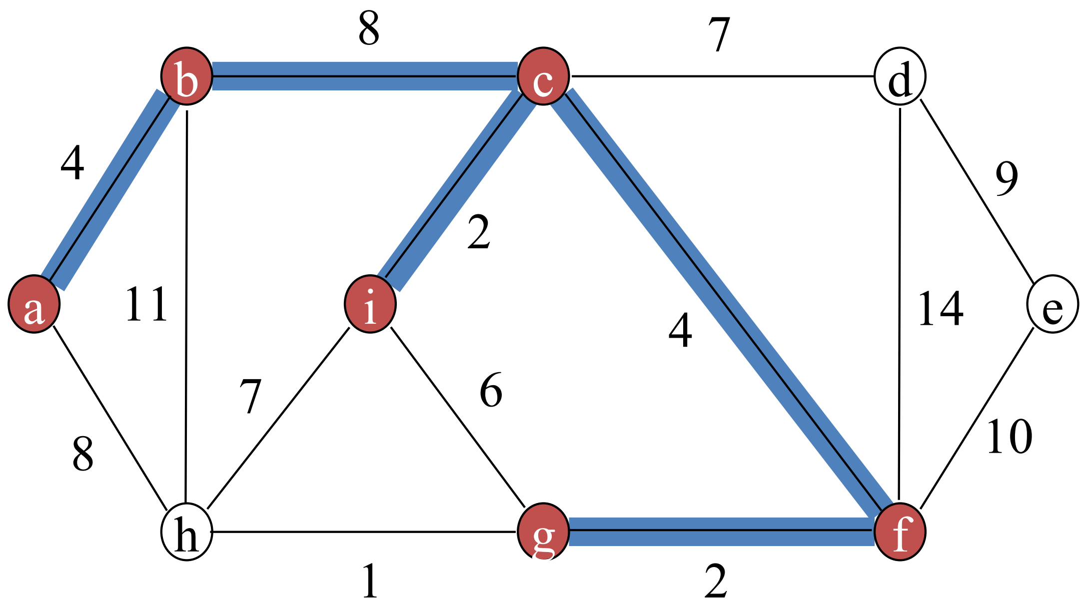
Prim's Algorithm

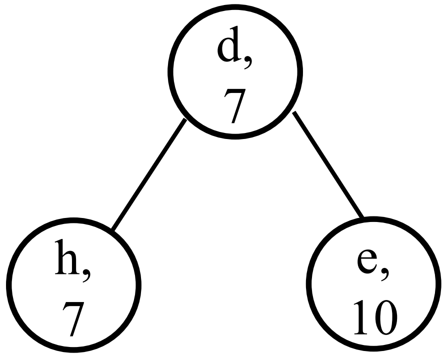




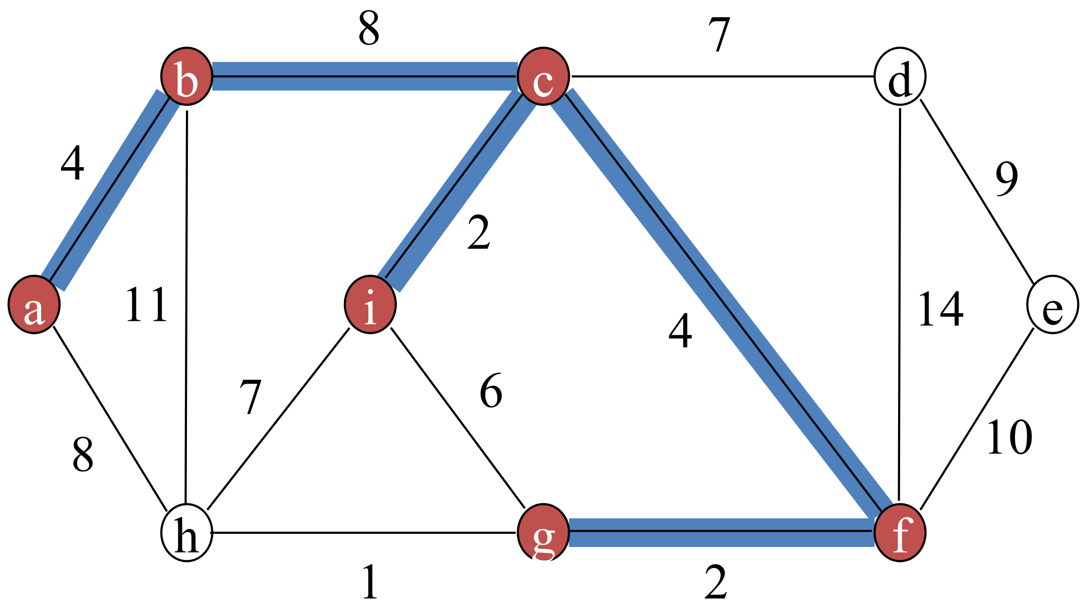
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	i	c

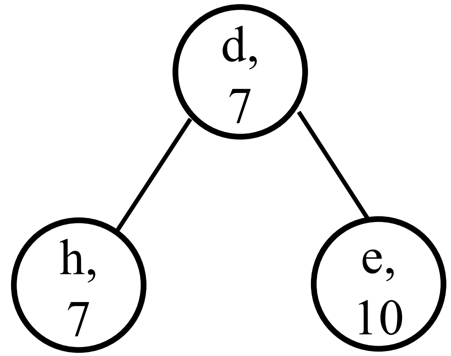
Prim’s Algorithm





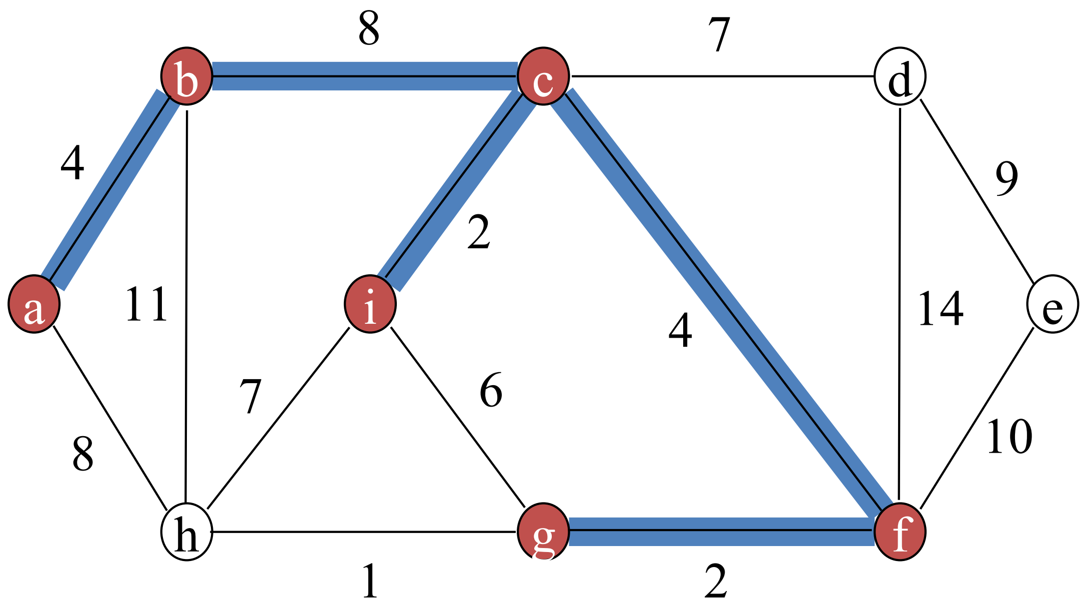
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	i	c

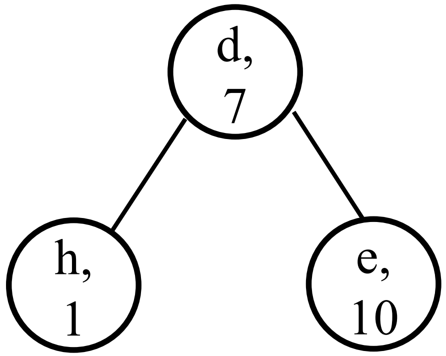




a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	g	c

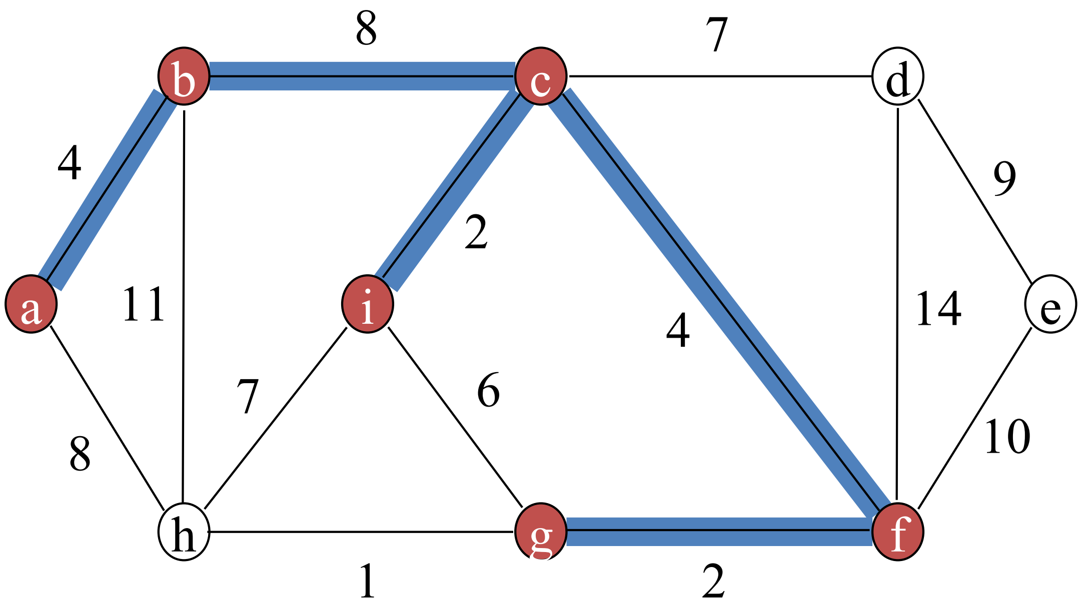
Prim's Algorithm

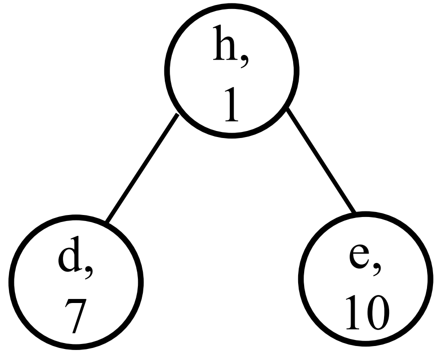




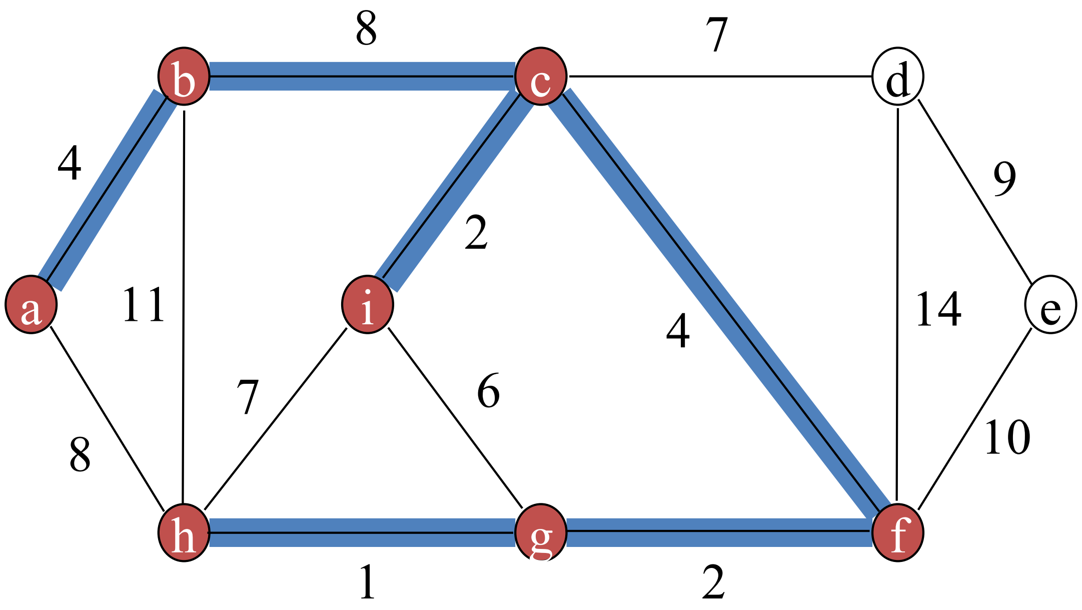
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	g	c

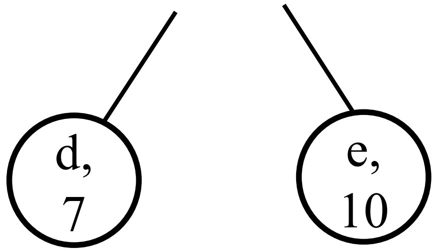
Prim’s Algorithm





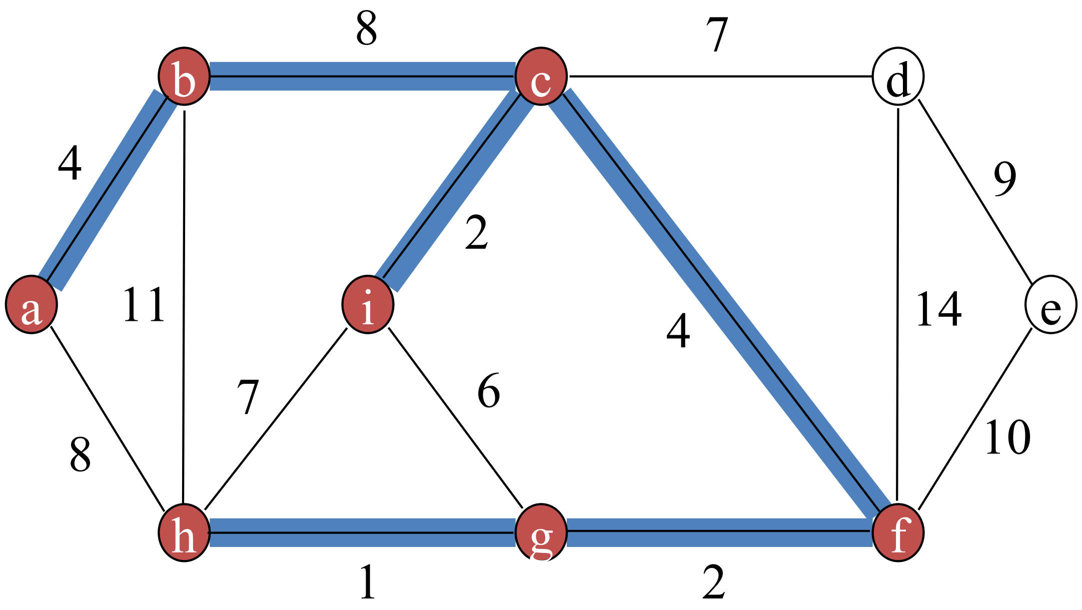
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	g	c

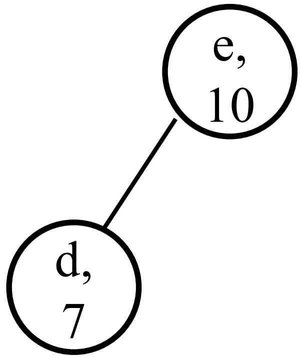




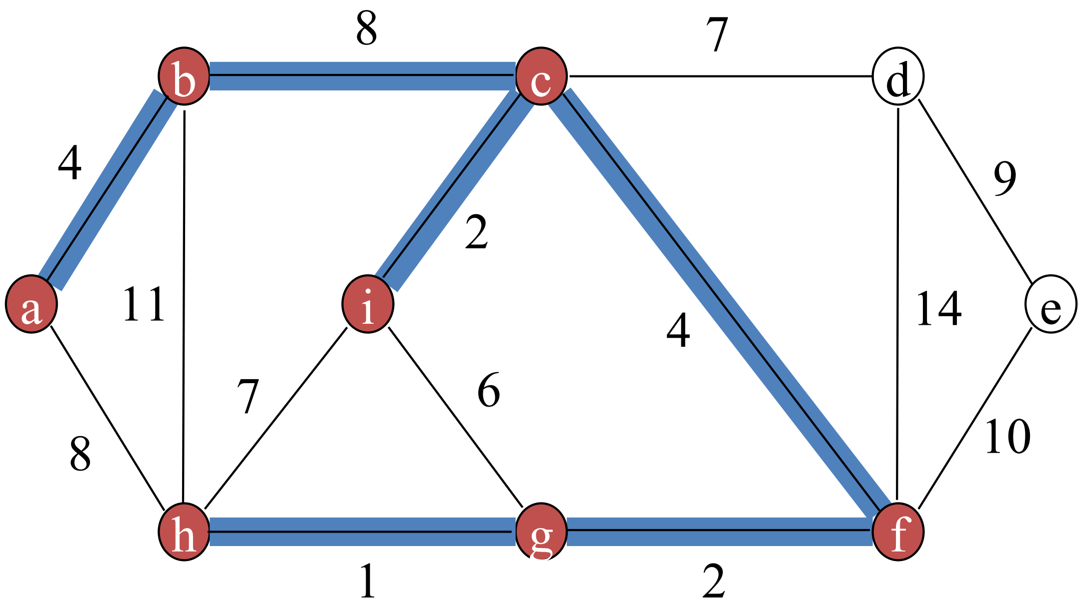
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	g	c

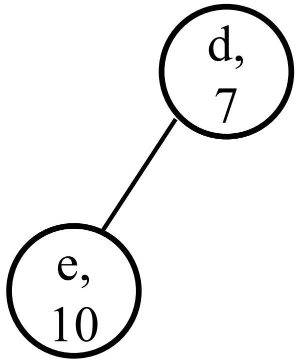
Prim's Algorithm



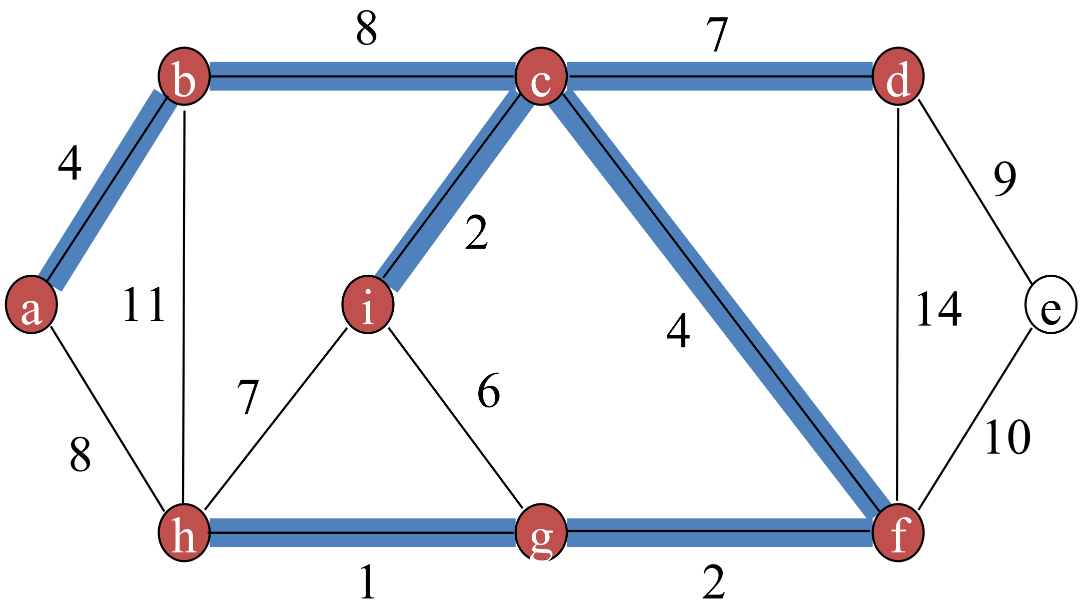


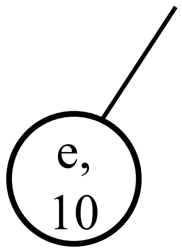
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	g	c





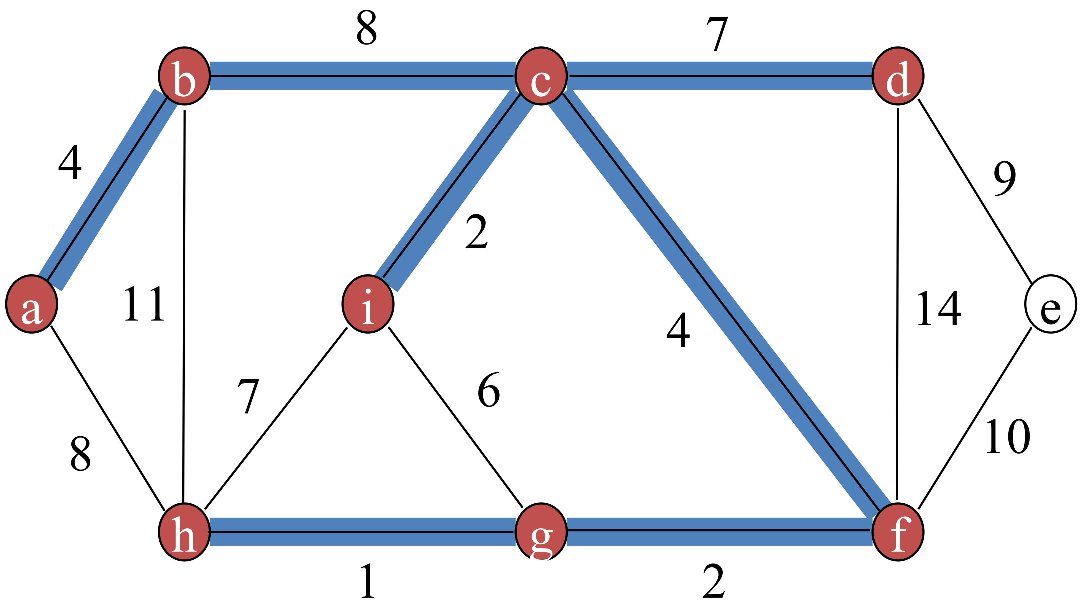
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	g	c

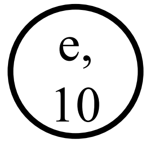




a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	g	c

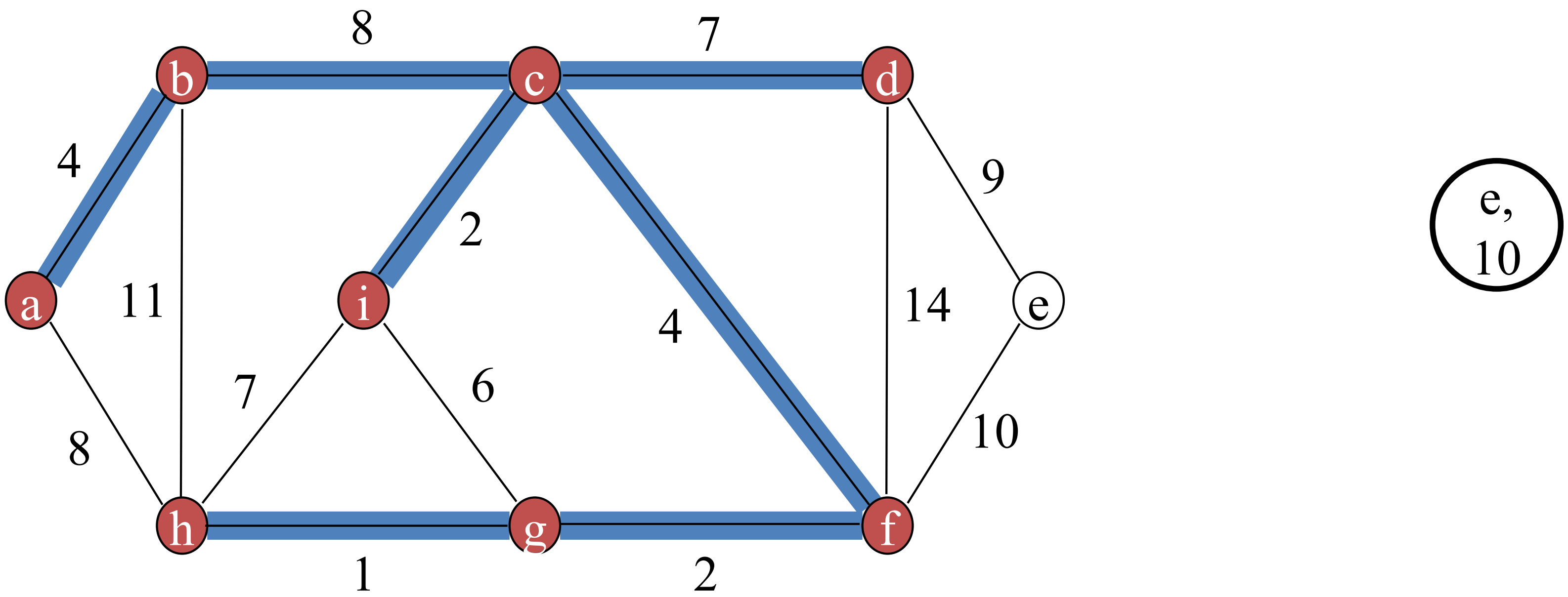
Prim's Algorithm





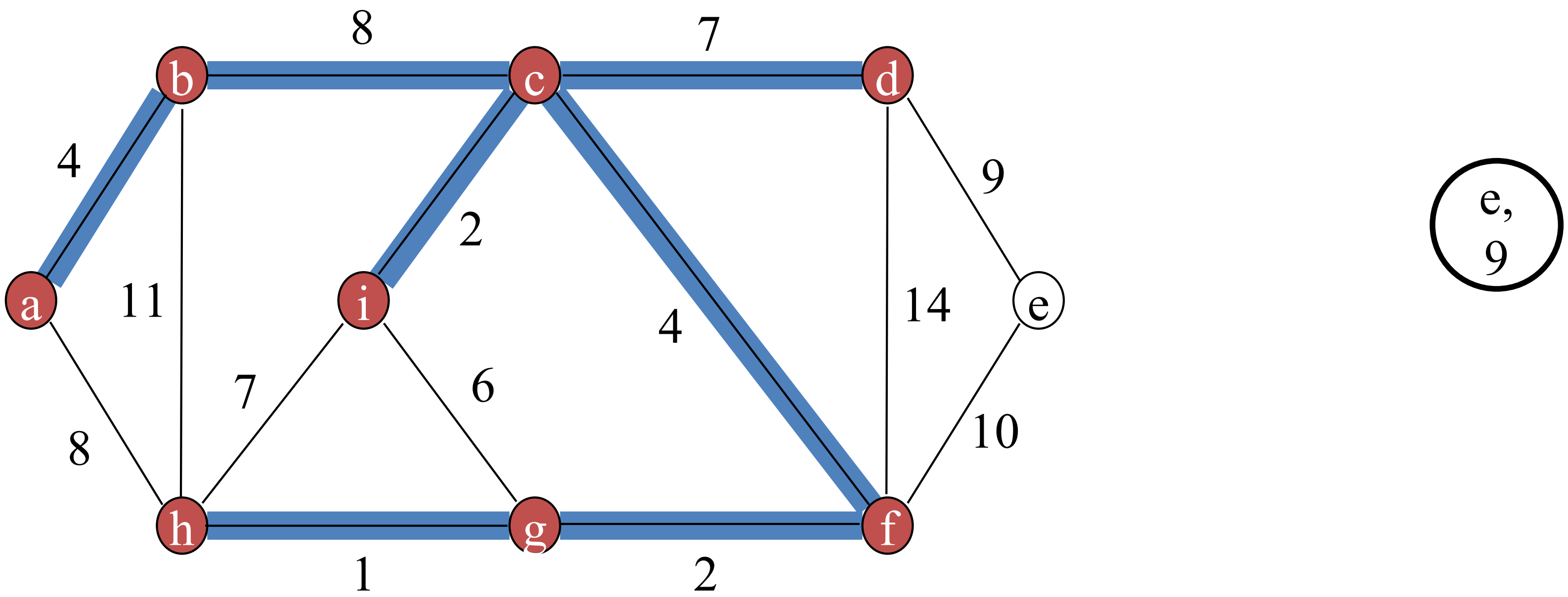
a	b	c	d	e	f	g	h	i
	a	b	c	f	c	f	g	c

Prim's Algorithm

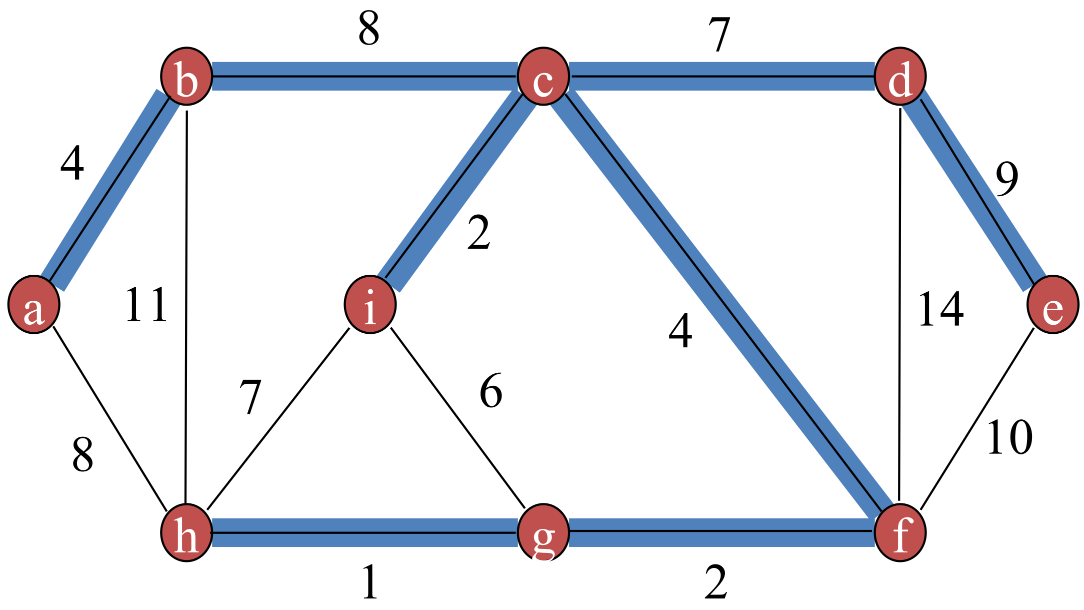


a	b	c	d	e	f	g	h	i
	a	b	c	d	c	f	g	c

Prim's Algorithm



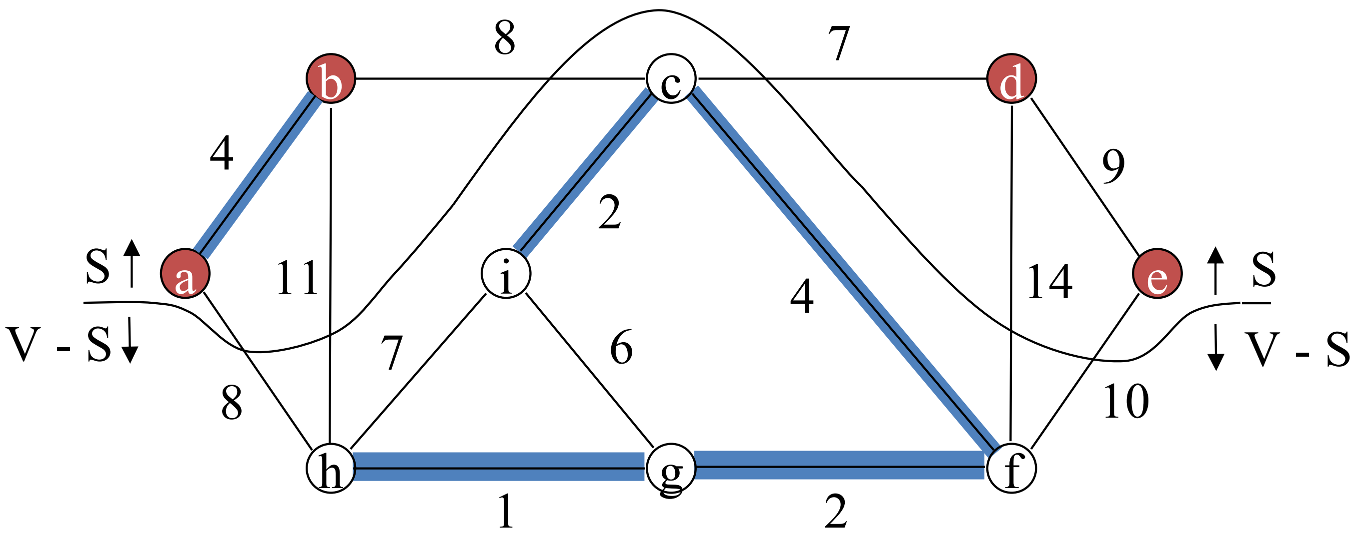
a	b	c	d	e	f	g	h	i
	a	b	c	d	c	f	g	c



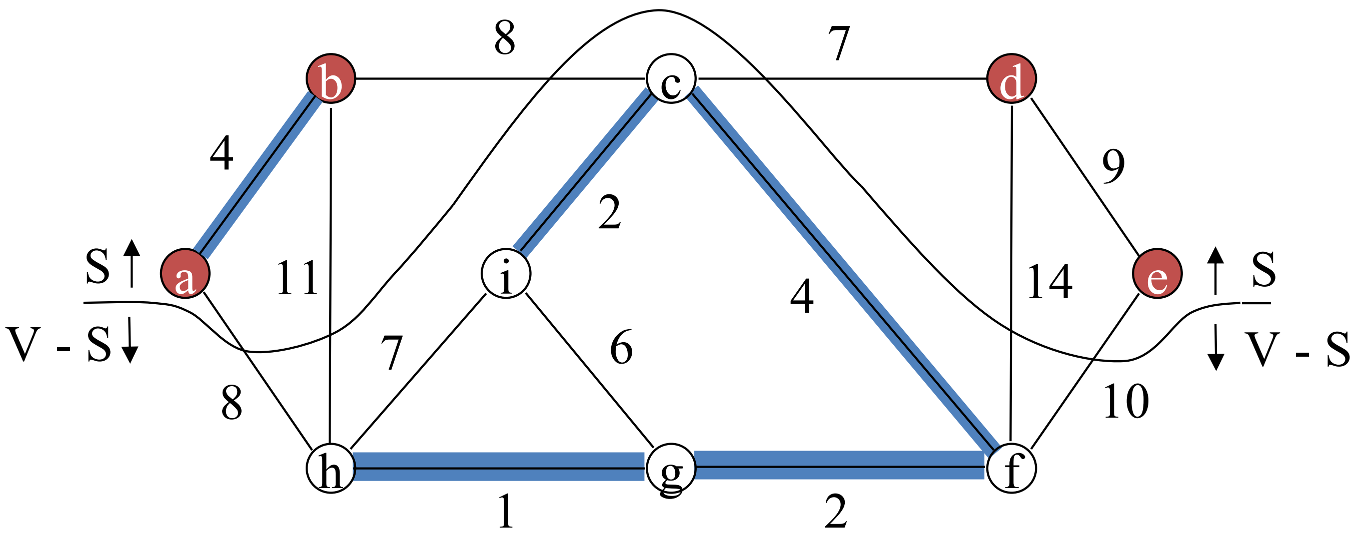
a	b	c	d	e	f	g	h	i
	a	b	c	d	c	f	g	c

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ Ø
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```

- A ***cut*** $(S, V - S)$ of an undirected graph $G = (V, E)$
  - A partition of V
- An edge $(u, v) \in E$ ***crosses*** the cut $(S, V - S)$
  - if one of edge $(u, v) \in E$ endpoints is in S and the other is in $V - S$.



- A cut ***respects*** a set A of edges
  - if no edge in A crosses the cut.
- An edge is a ***light edge***
  - if its weight is the minimum of any edge crossing the cut.

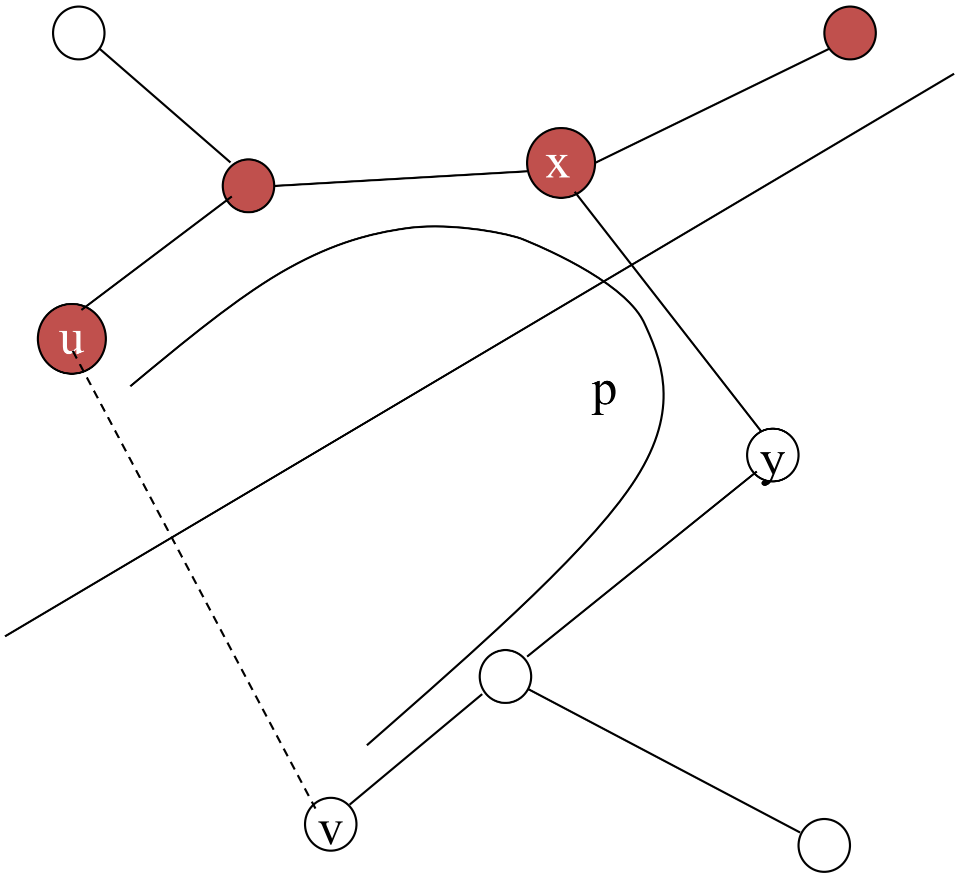


- Theorem 23.1
  - Consider an edge subset A contained in some MST.
  - Consider a cut respecting A.
  - Then, a light edge crossing the cut is safe for A.

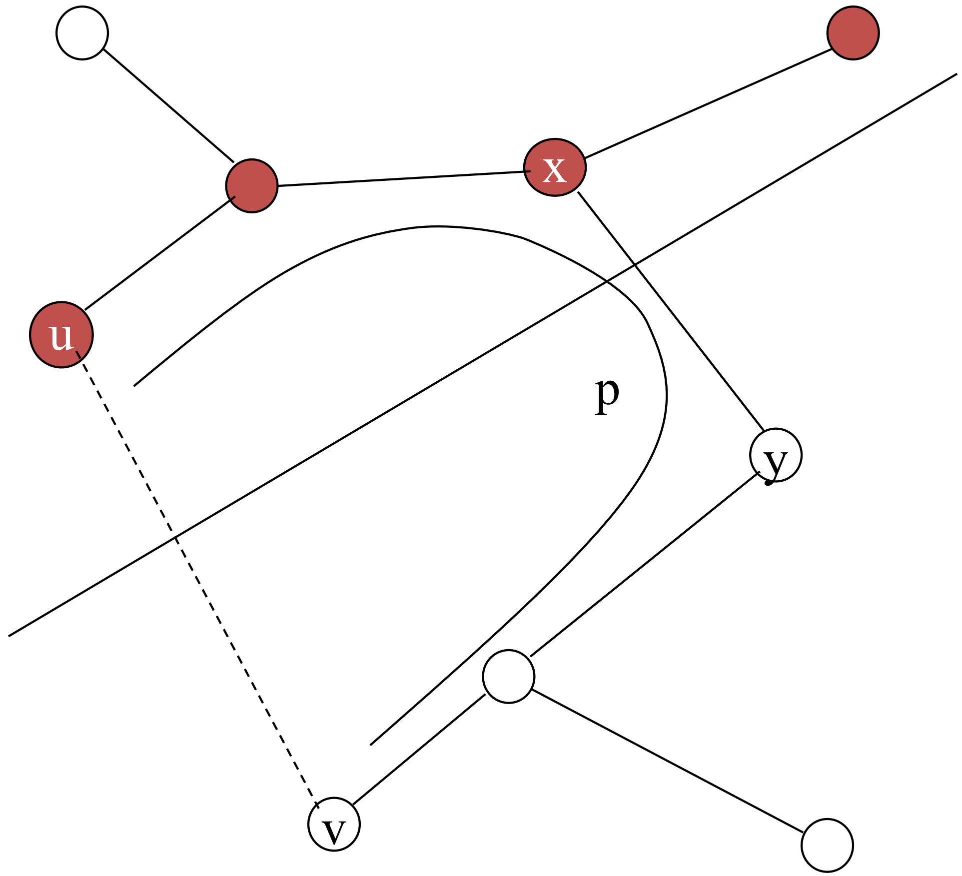
- Outline of the proof
  - Let T be a minimum spanning tree that includes A.
    - Assume that T does not contain the light edge (u, v).
  - It constructs another minimum spanning tree T' that includes $A \cup \{(u, v)\}$.

- The edge (u, v) forms a cycle with the edges on the path p from u to v in T.

- Since u and v are on opposite sides of the cut $(S, V - S)$,
  - there is at least one edge in T on the path p that also crosses the cut.
  - Let (x, y) be any such edge.



- The edge (x, y) is not in A.
  - Because the cut respects A.
- Removing (x, y) breaks T into two components.
  - Because (x, y) is on the unique path from u to v in T.
- Adding (u, v) reconnects them to form a new spanning tree
  - $T' = T - \{(x, y)\} \cup \{(u, v)\}$.



- We next show that T' is a minimum spanning tree.
  - Since (u, v) is a light edge crossing $(S, V - S)$ and (x, y) also crosses this cut, $w(u, v) \leq w(x, y)$.

$$
\begin{aligned}
w(T') &= w(T) - w(x, y) + w(u, v) \\
&\leq w(T)
\end{aligned}
$$

  - But T is a minimum spanning tree, so that $w(T) \leq w(T')$; thus, T' must be a minimum spanning tree, too.

- We show that (u, v) is actually a safe edge for A.
  - $A \subseteq T$ and $(x, y) \notin A \Rightarrow A \subseteq T'$
    - Thus $A \cup \{(u, v)\} \subseteq T'$.
  - Since T' is a minimum spanning tree, (u, v) is safe for A.

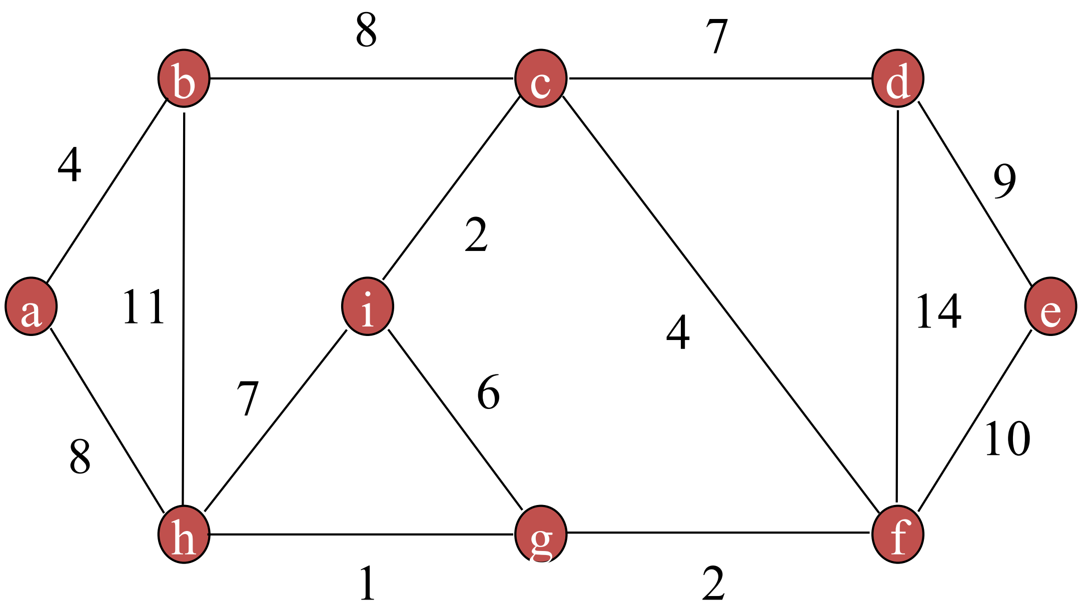
- **Corollary 23.2**
  - Let $G = (V, E)$ be a graph.
  - Let A be a subset of E that is included in some minimum spanning tree for G.
  - Let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_A = (V, A)$.
  - If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.

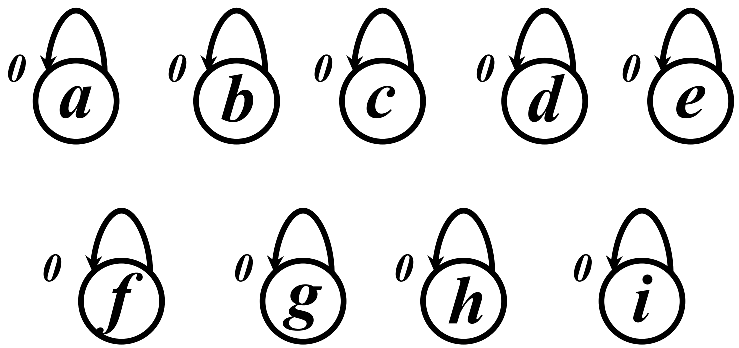
- ***Proof***
  - The cut $(V_C, V - V_C)$ respects A, and (u, v) is a light edge for this cut.
  - Therefore, (u, v) is safe for A.

- The edges in the set A always form a single tree.
- The tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V.
- At each step, a light edge is added to the tree A that connects A to an isolated vertex of $G_A = (V, A)$.
- By Corollary 23.2, this rule adds only edges that are safe for A.
- Therefore, when the algorithm terminates, the edges in A form a minimum spanning tree.

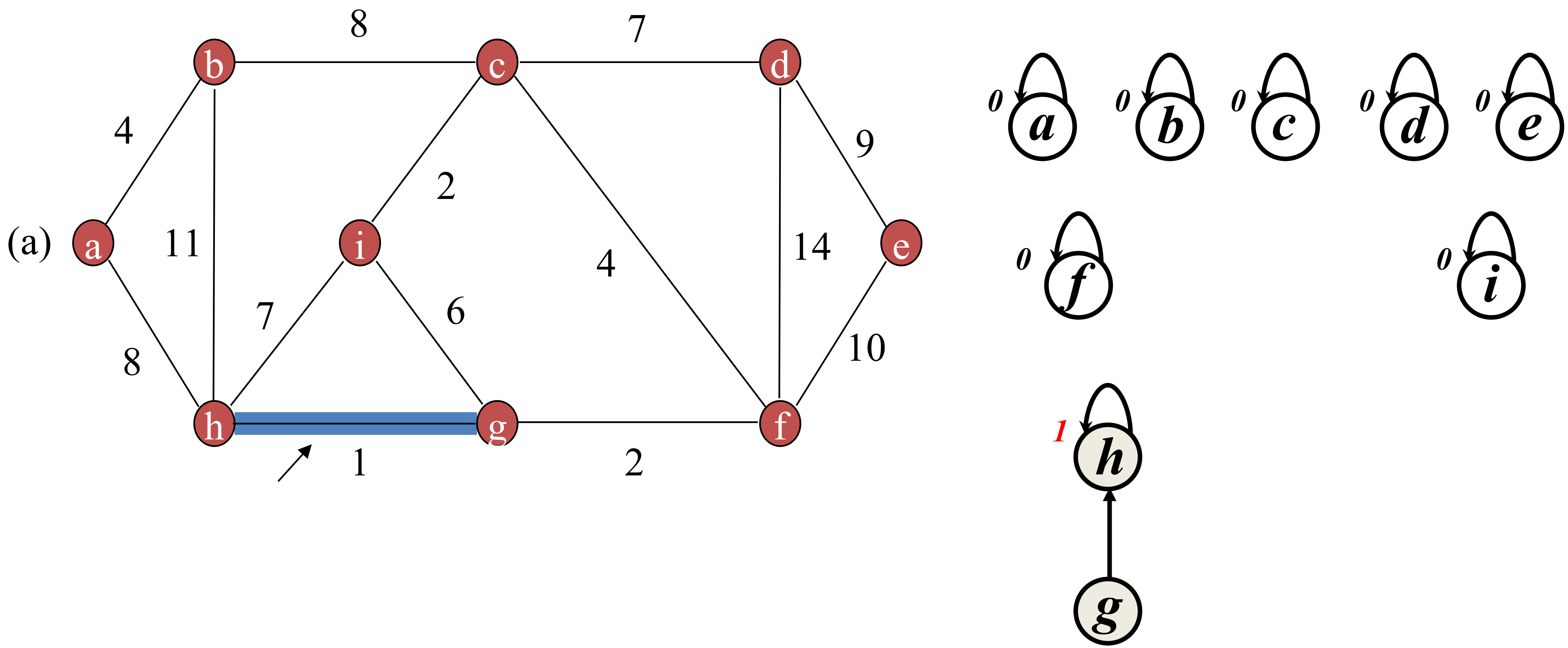
- It finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u, v) of least weight.
- Let C_1 and C_2 denote the two trees that are connected by (u, v) .
- Since (u, v) must be a light edge connecting C_1 to some other tree, Corollary 23.2 implies that (u, v) is a safe edge for C_1.

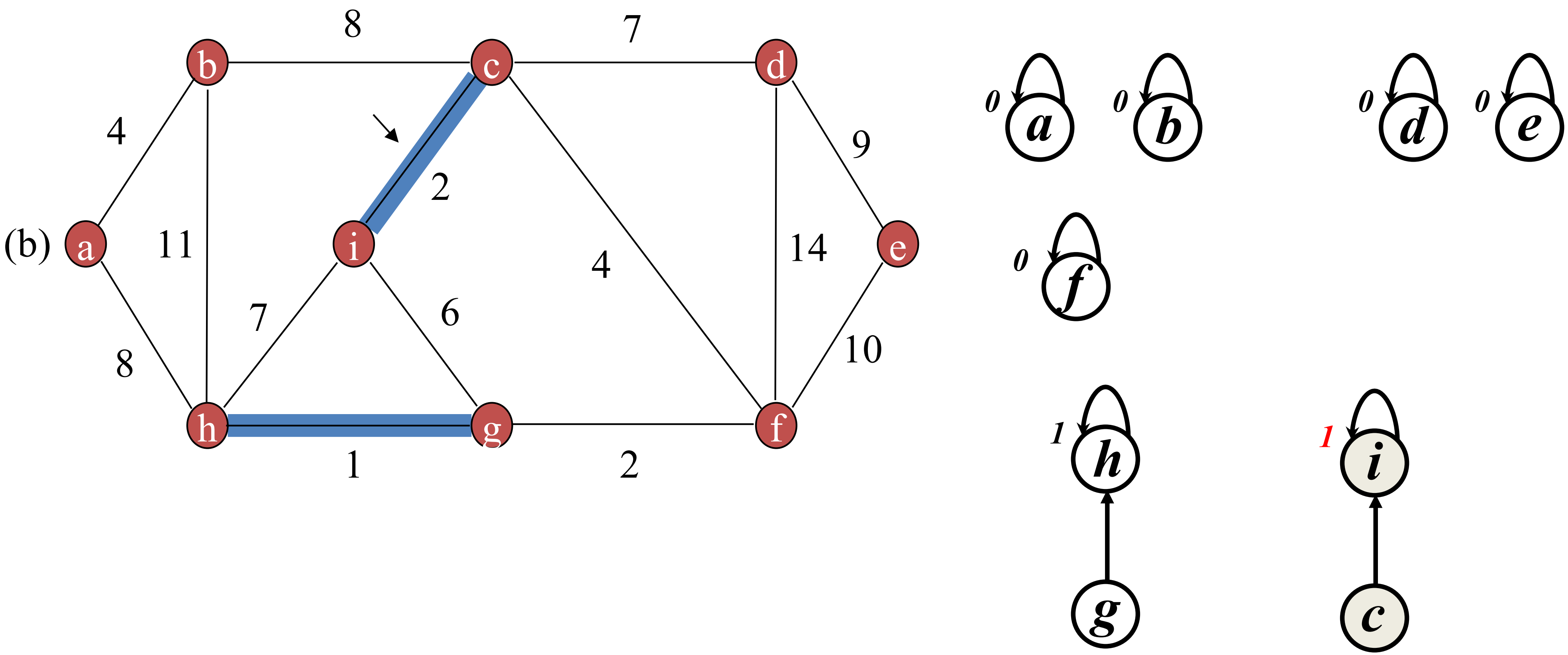
Kruskal's Algorithm

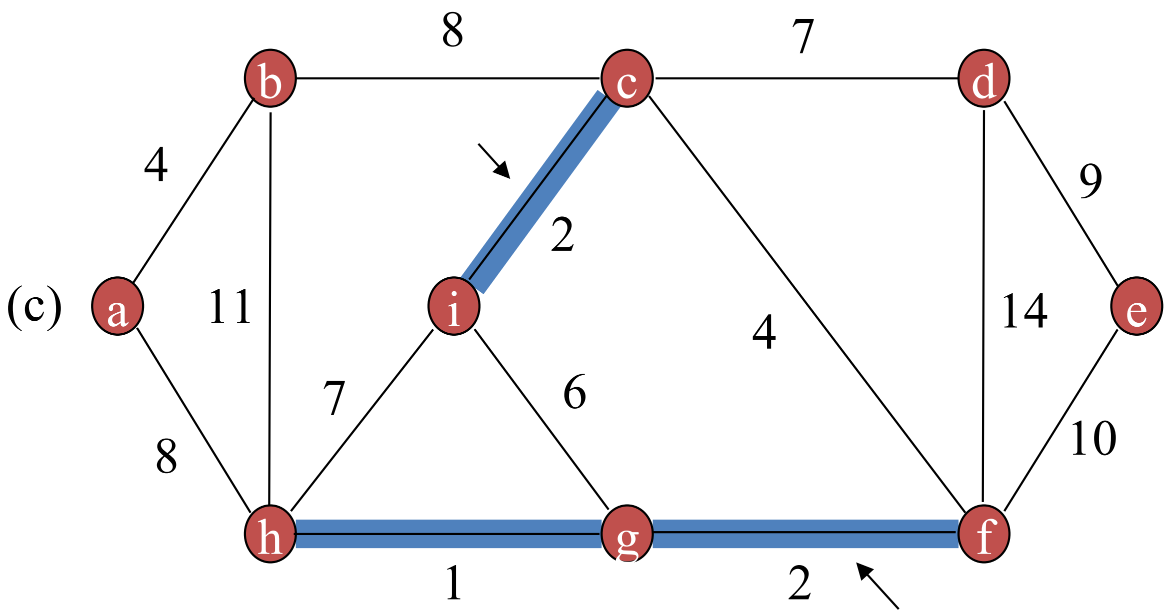


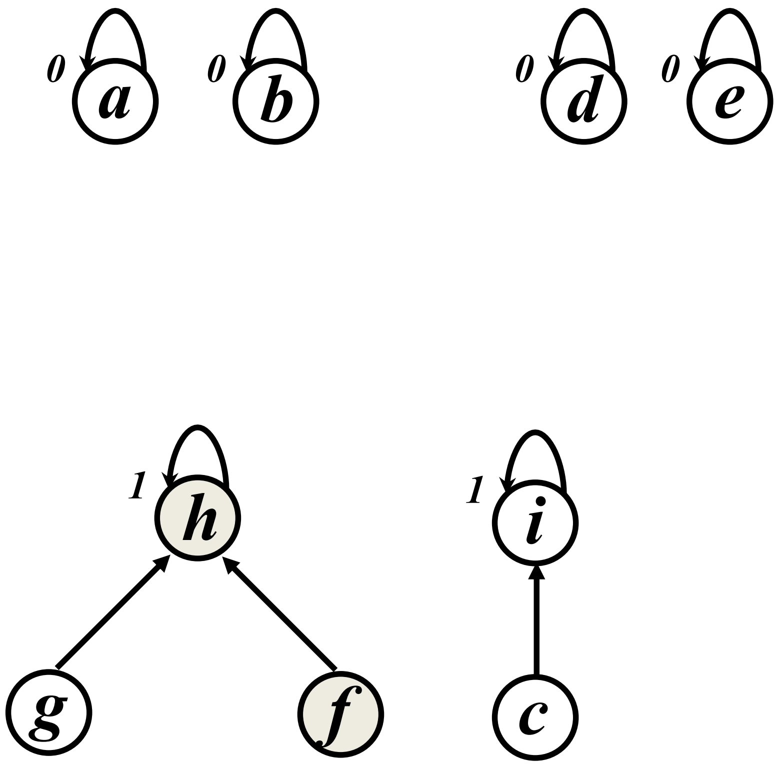


Kruskal's Algorithm

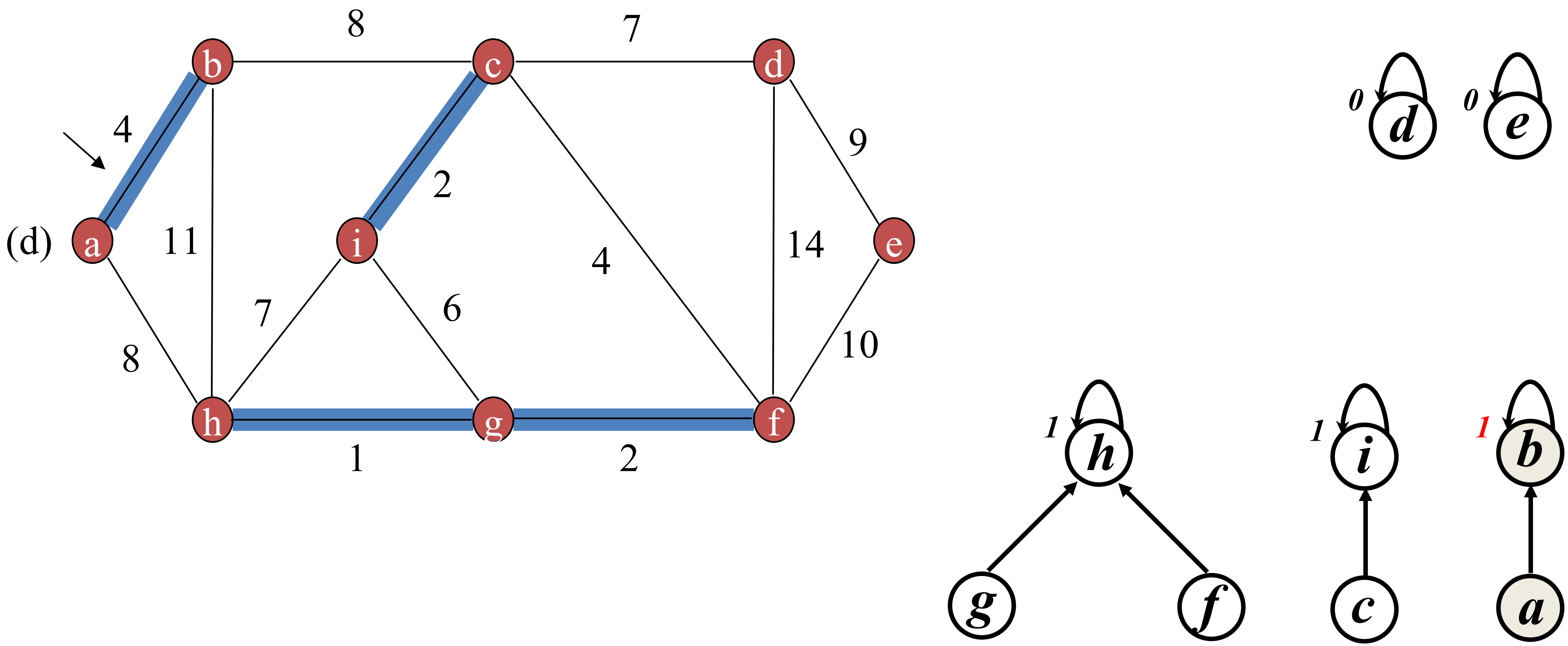



(b)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
14
4
7
6
10
8
1
2
0
a
0
b
0
d
0
e
0
f
1
h
g
1
i
c


(c)
a
b
c
d
e
f
g
h
i
8
7
4
9
2
11
4
14
7
6
8
10
1
2

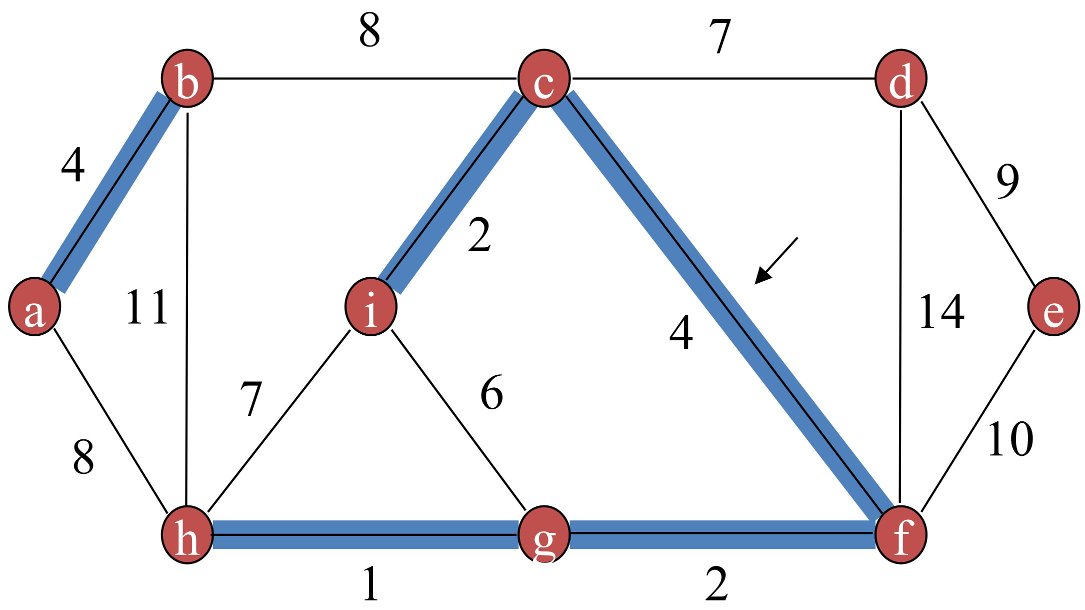

0
a
0
b
0
d
0
e
1
h
g
f
1
i
c

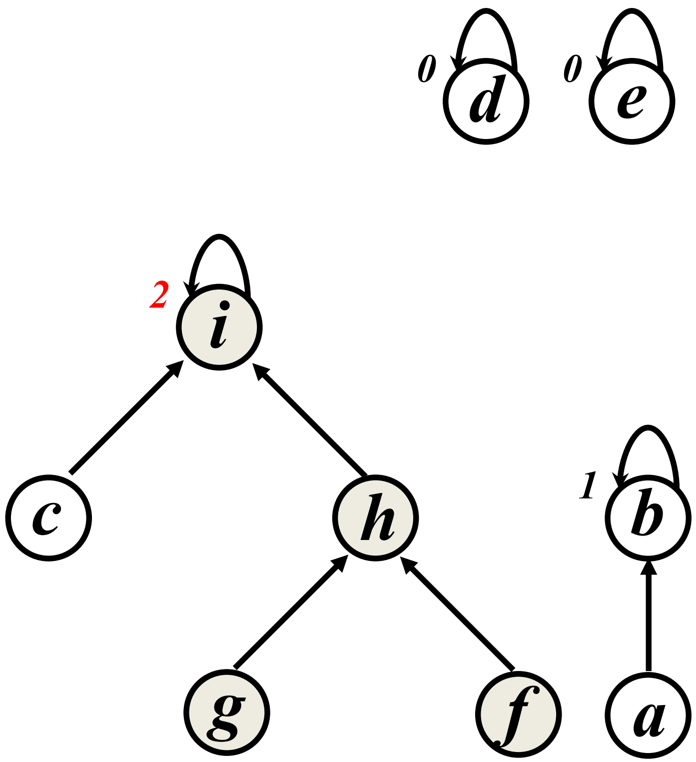
Kruskal's Algorithm



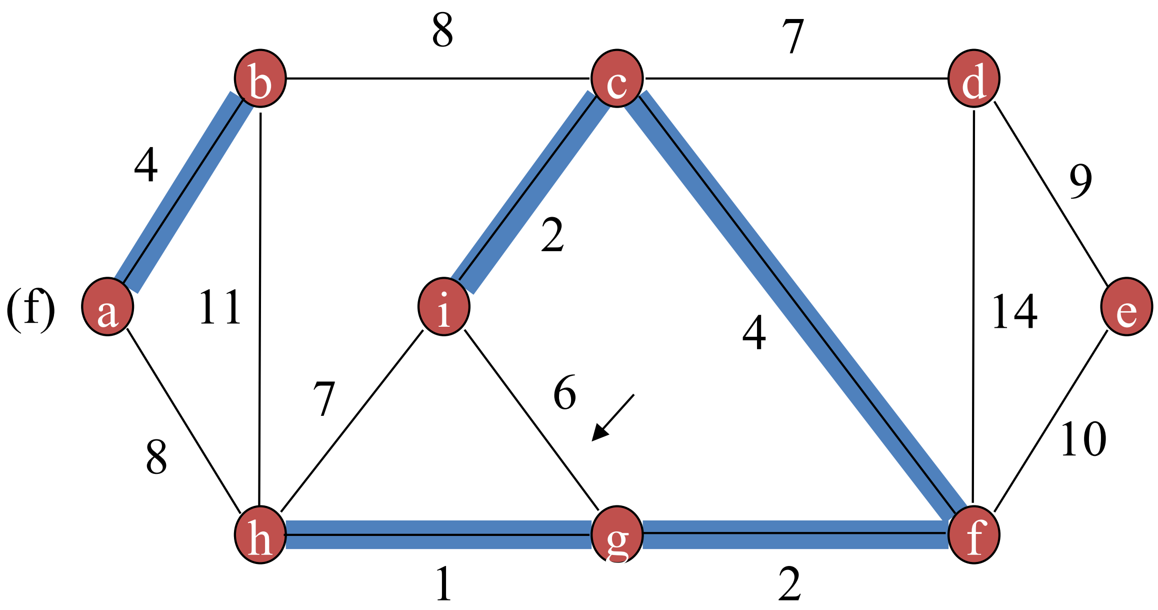
Kruskal's Algorithm

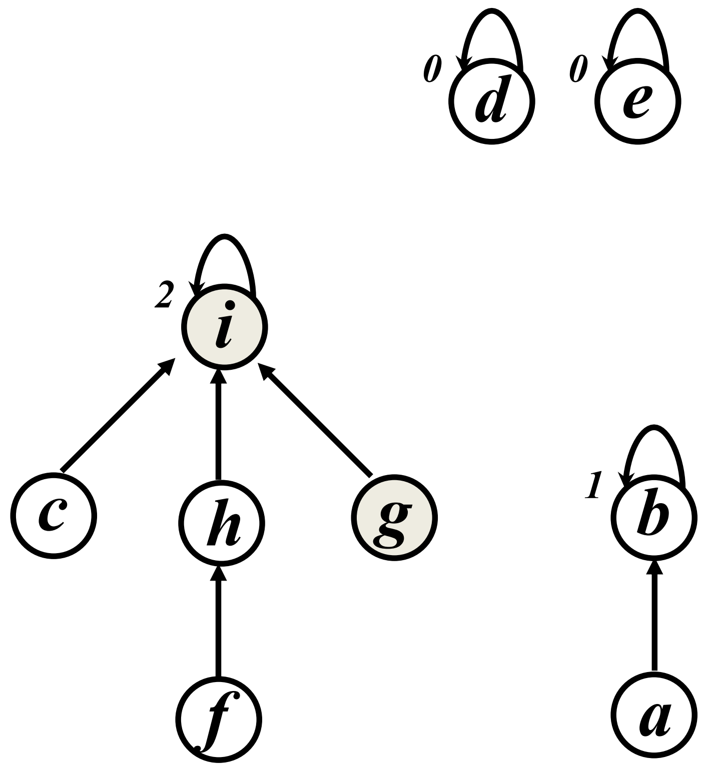
(e)



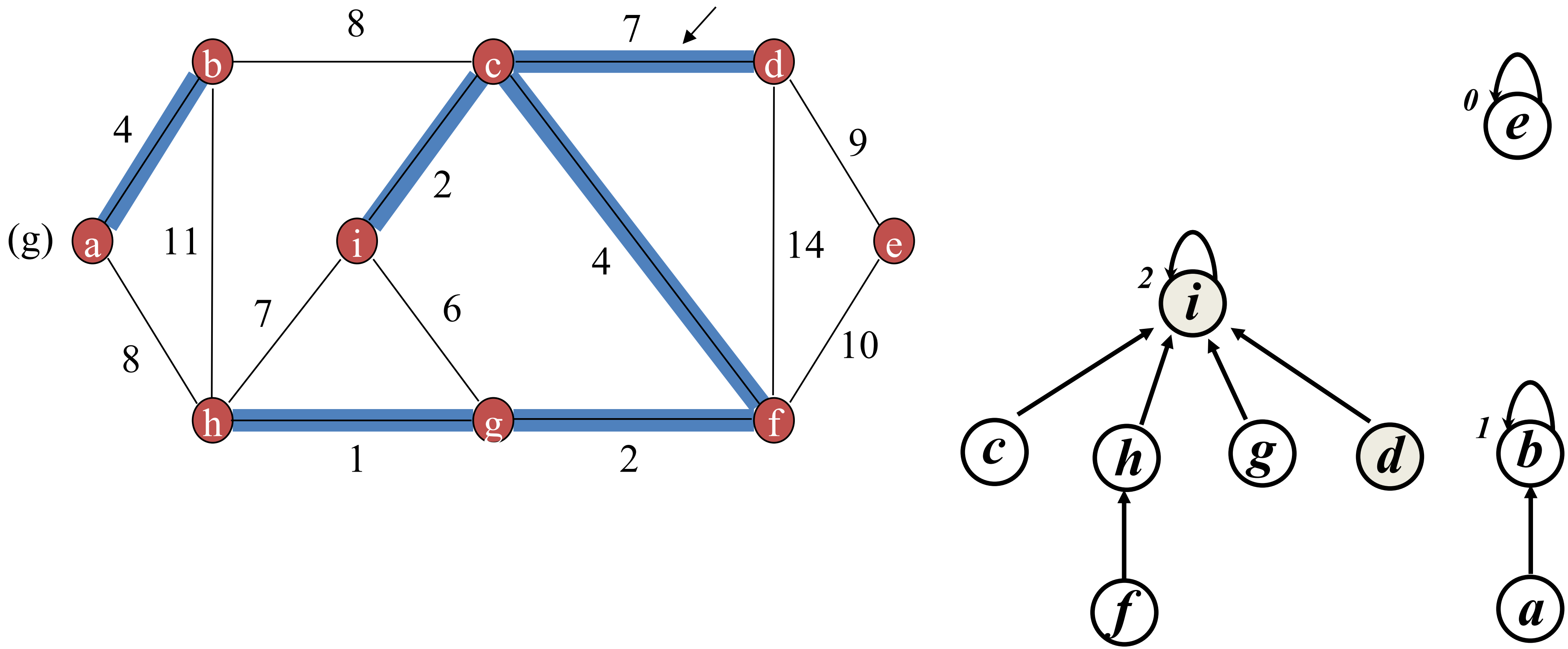


Kruskal's Algorithm

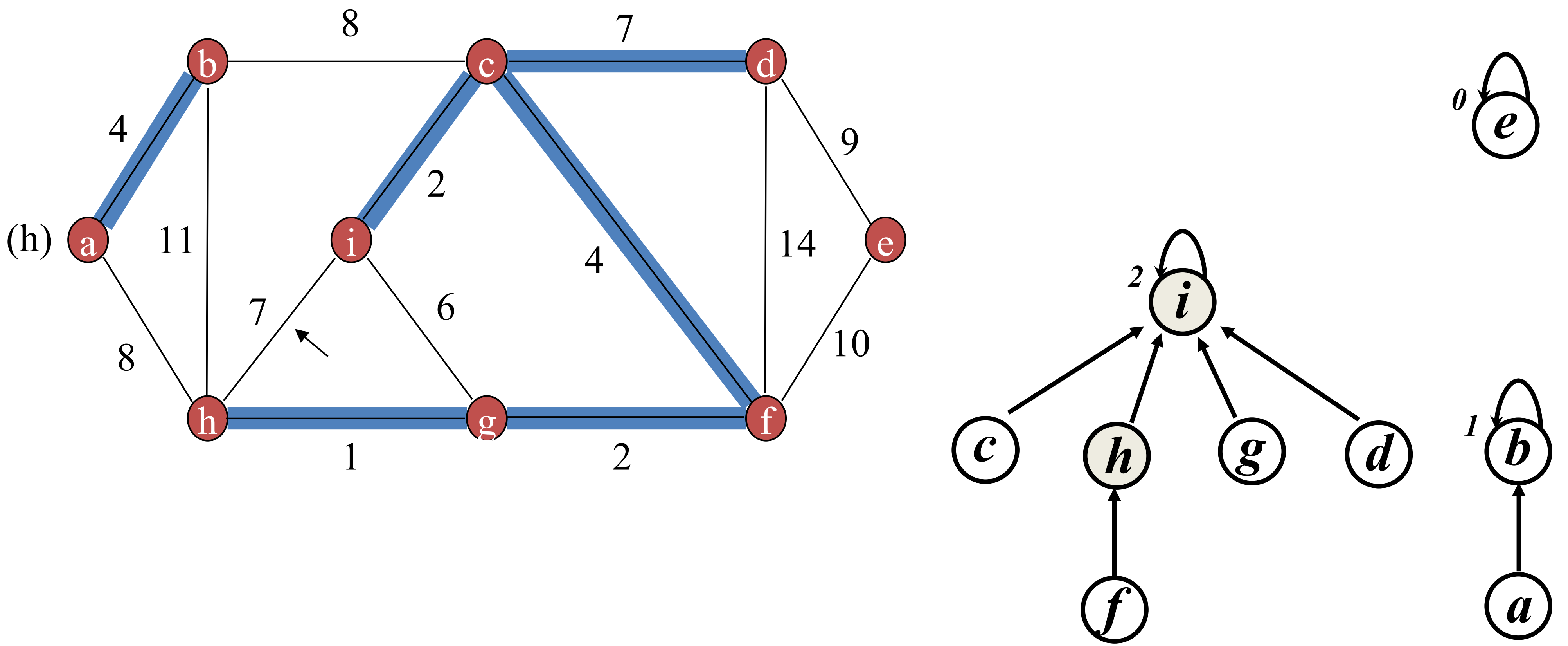


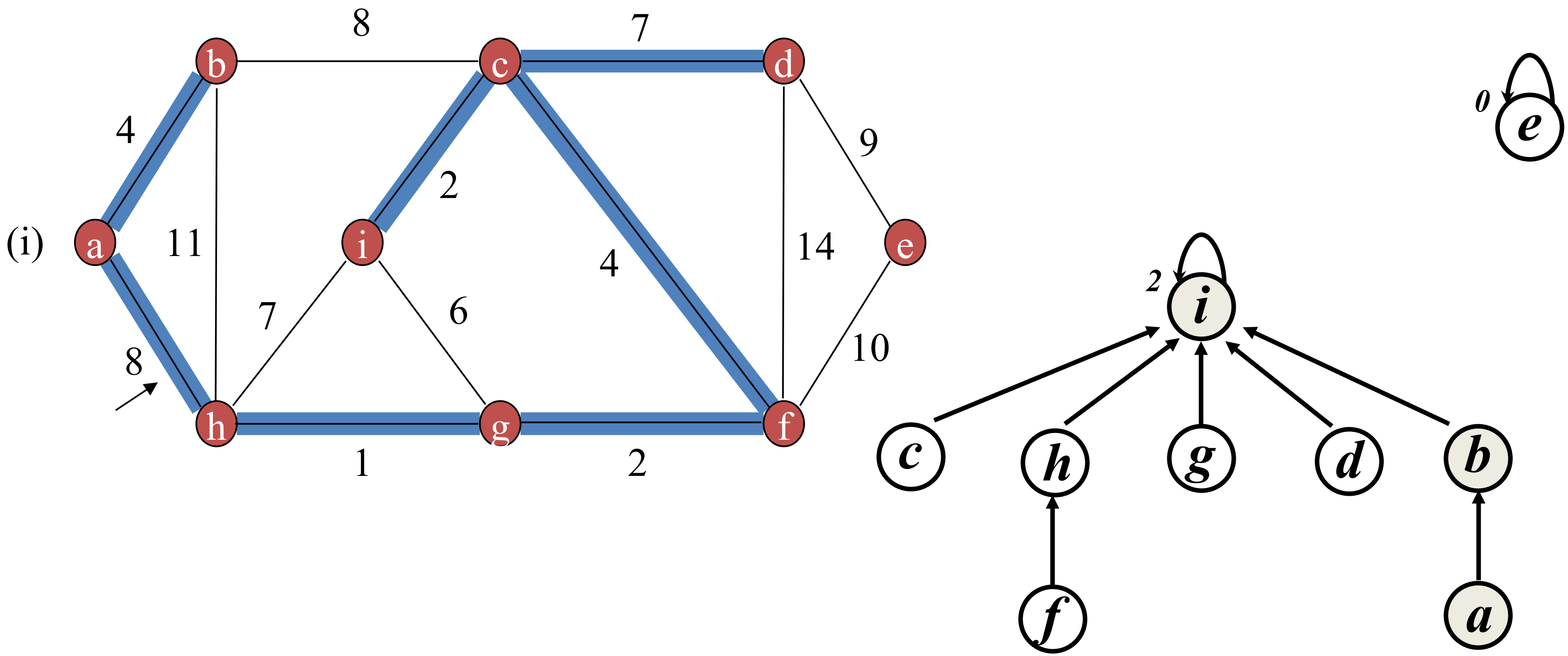


Kruskal's Algorithm

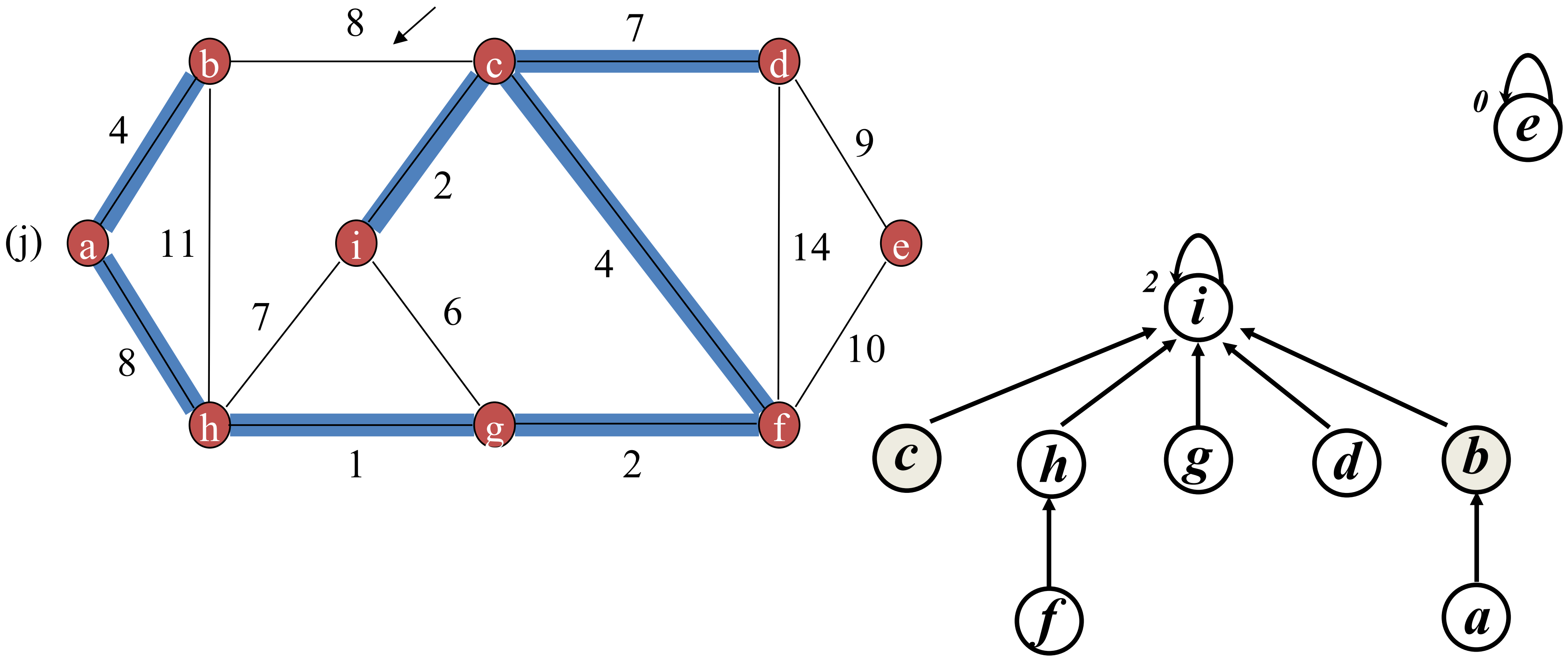


Kruskal's Algorithm



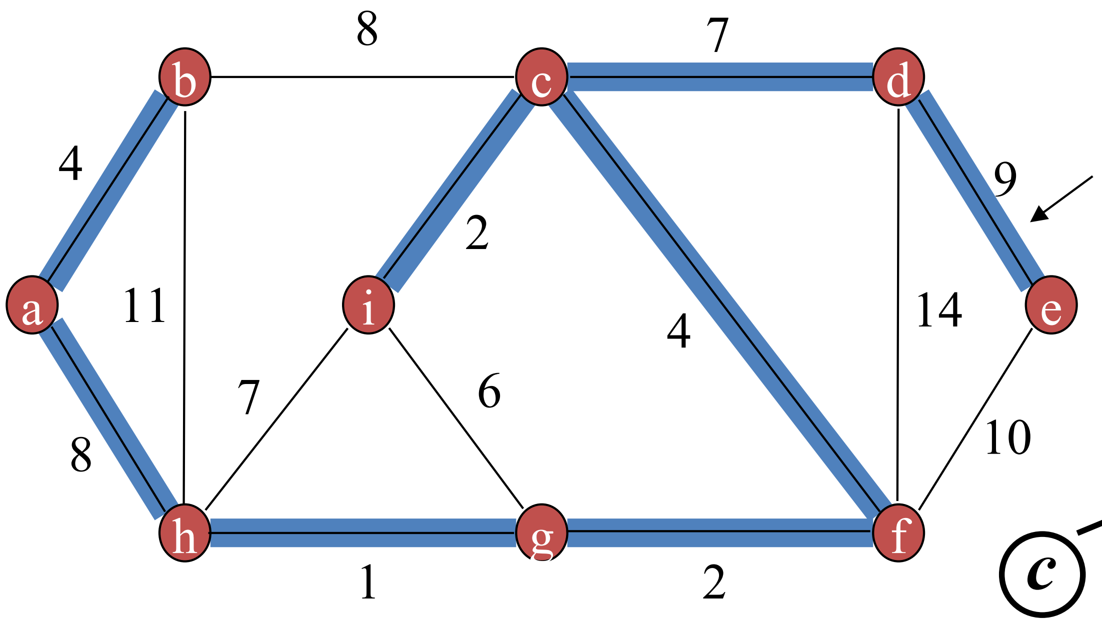

(i)
a
b
c
d
e
f
g
h
i
4
8
7
11
2
4
14
9
7
6
10
8
1
2
0
e
2
i
c
h
g
d
b
f
a

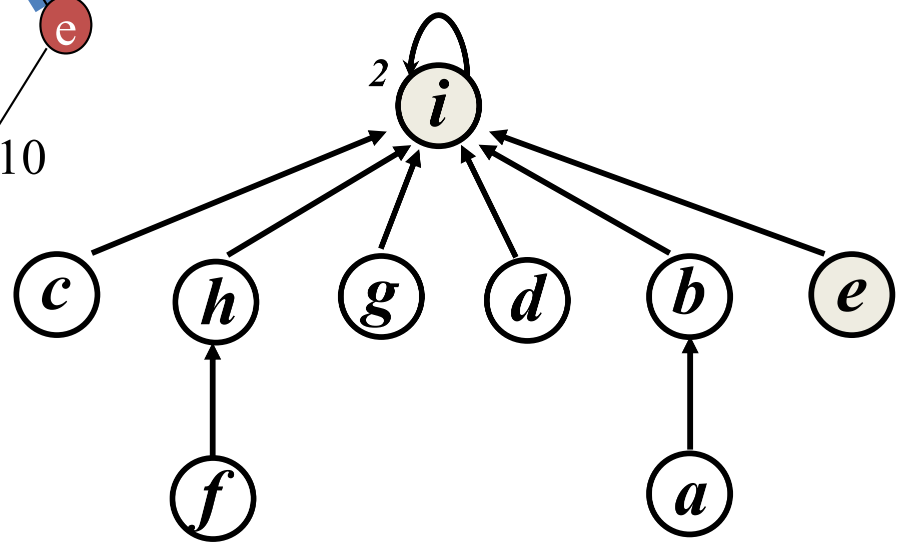
Kruskal's Algorithm



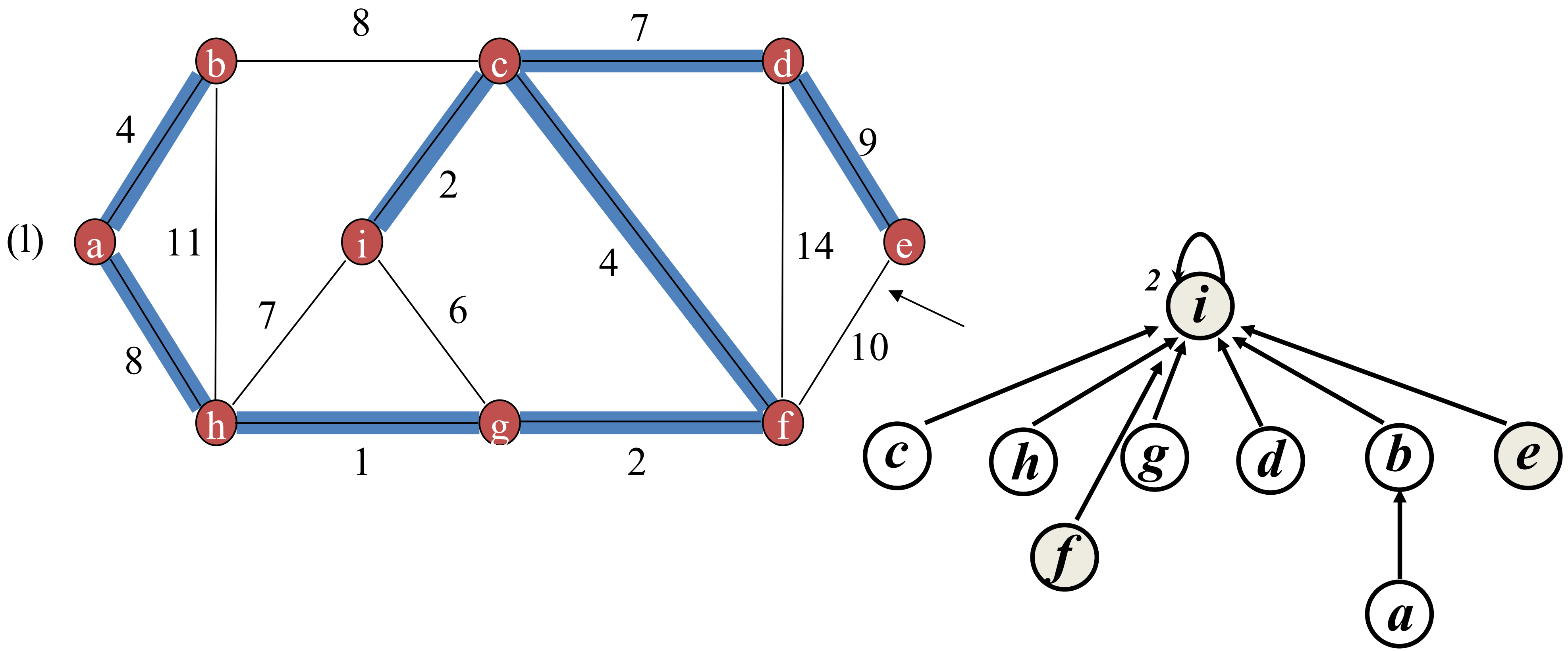
Kruskal's Algorithm

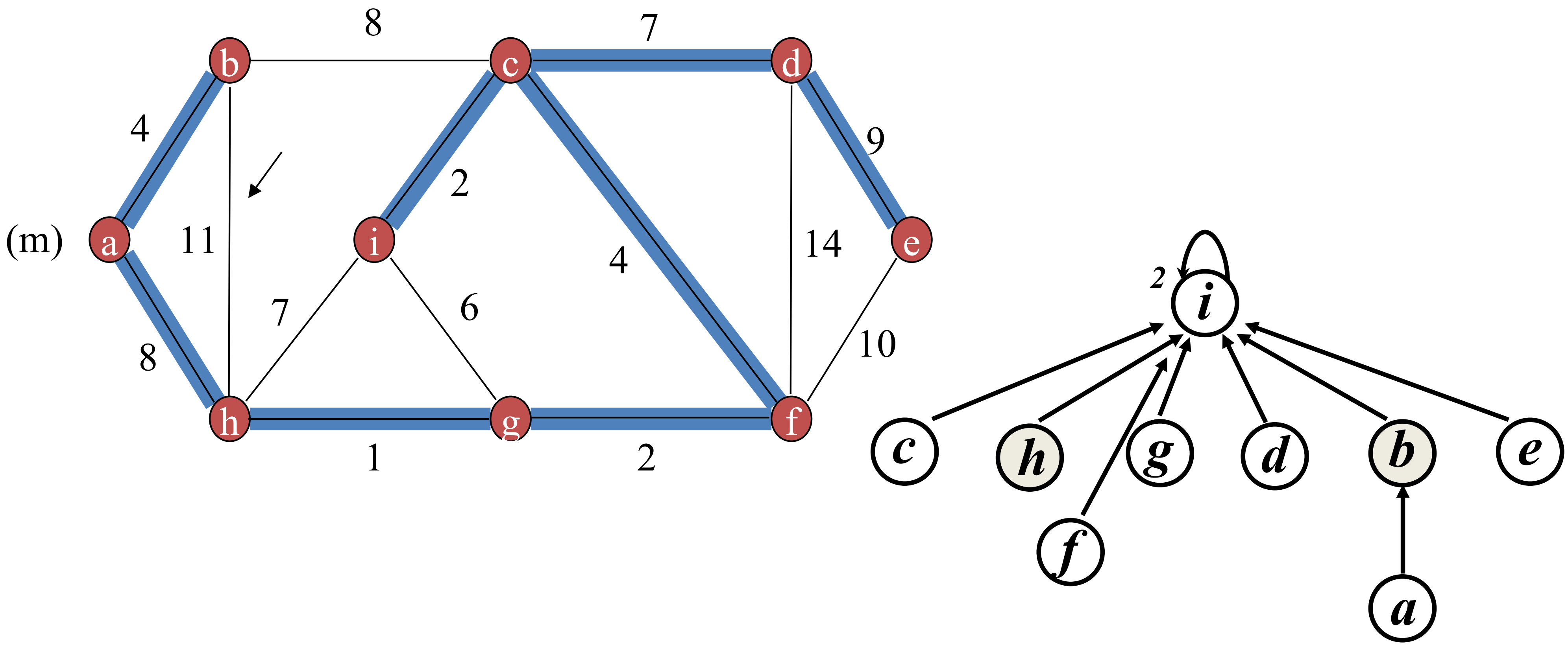
(k)



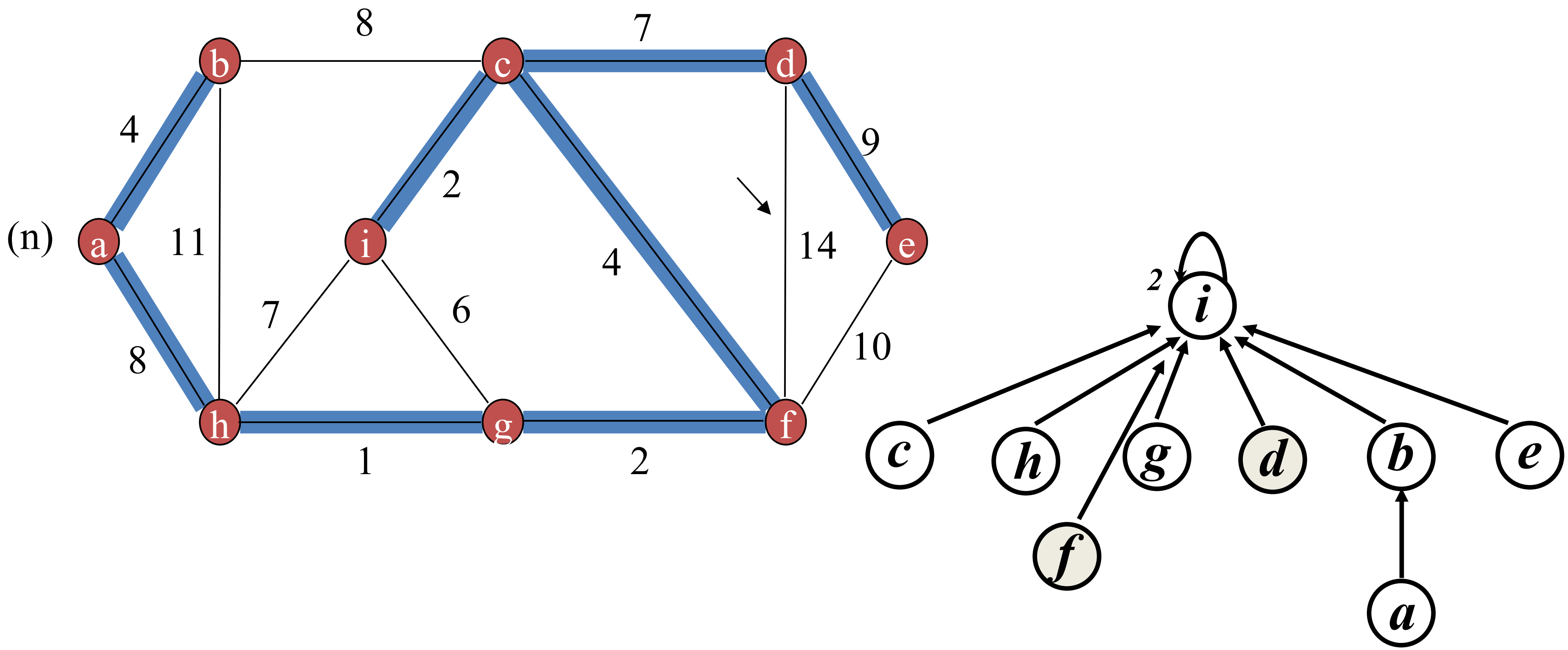


Kruskal's Algorithm




(m)
a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2
2
i
c
h
g
d
b
e
f
a

Kruskal's Algorithm



Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```