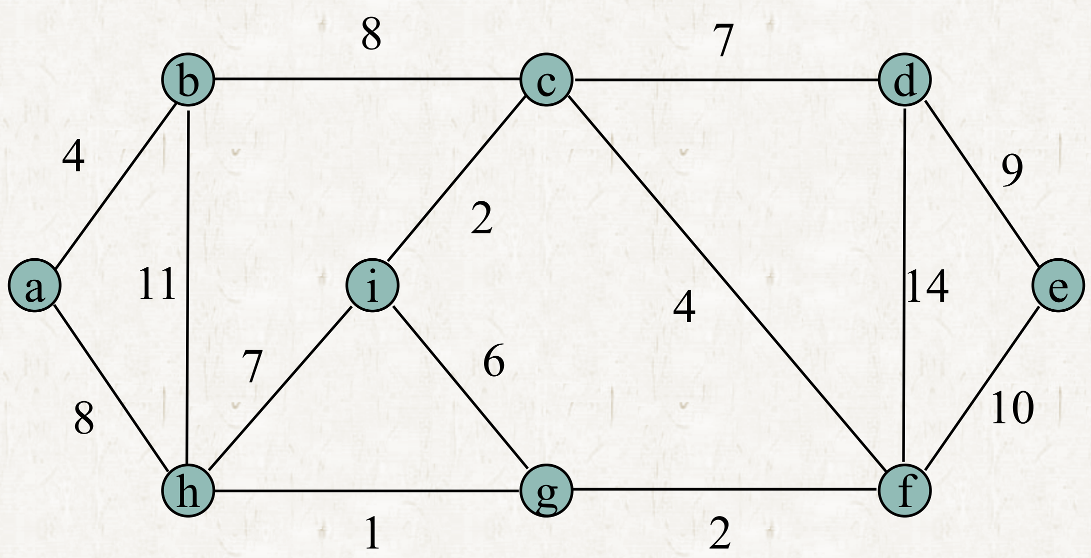
Minimum Spanning Trees

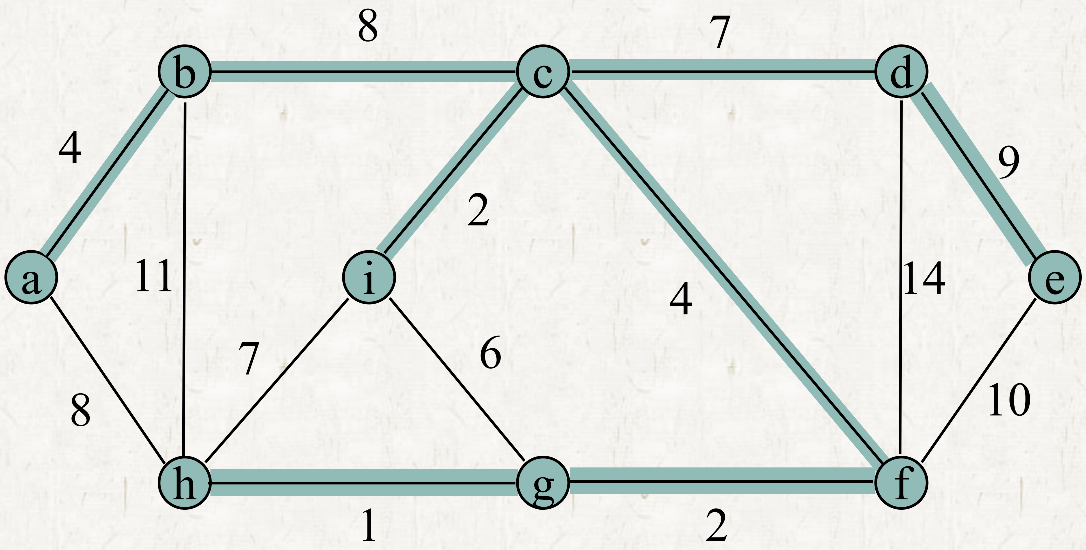
Weighted Undirected Graphs

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



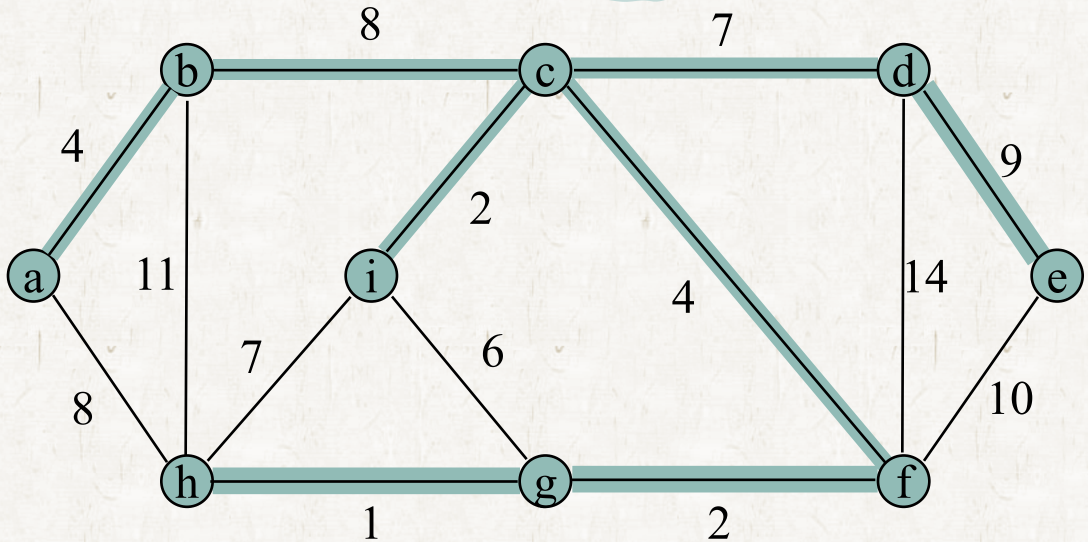
Spanning Trees

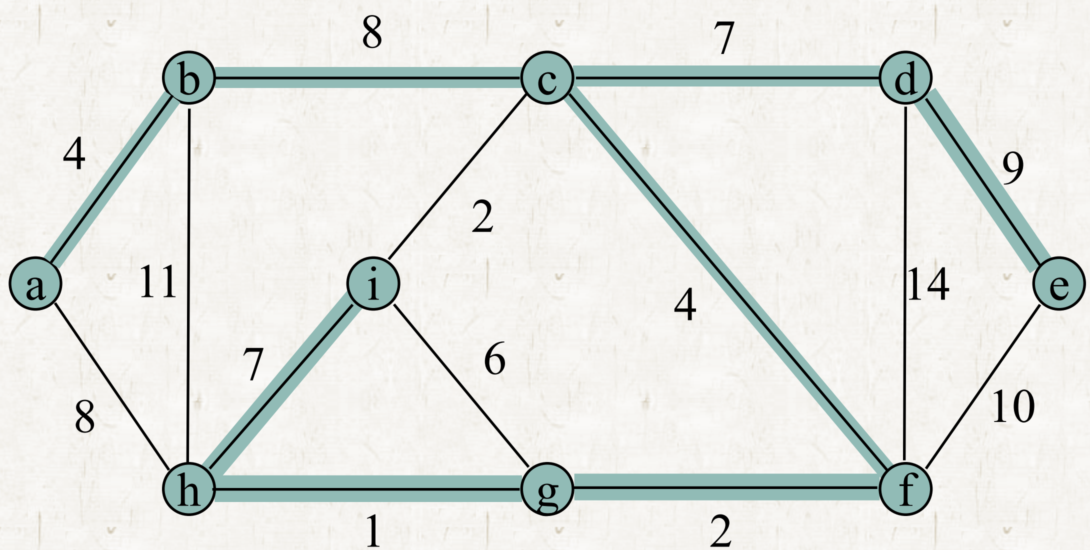
- **A *spanning tree*** for *G*.
  - A tree containing all of the vertices in *G* and edges of the tree are selected from the edges in *G*.



  - There are many spanning trees.

Spanning Trees





Minimum Spanning Trees

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

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

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

Minimum Spanning Trees

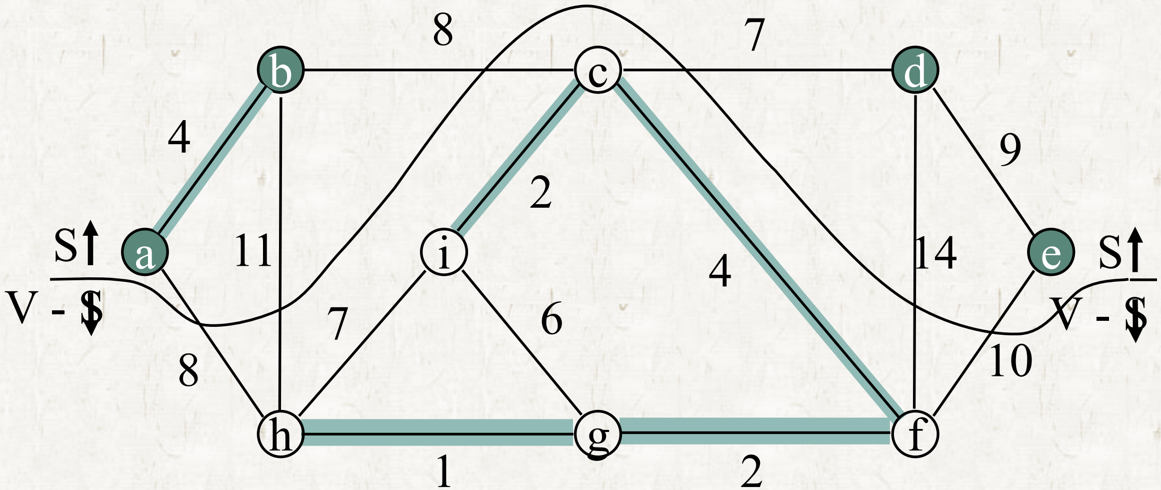
- GENERIC-MST

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

- It grows the minimum spanning tree one edge at a time.
- A is a subset of some minimum spanning tree.
- 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.

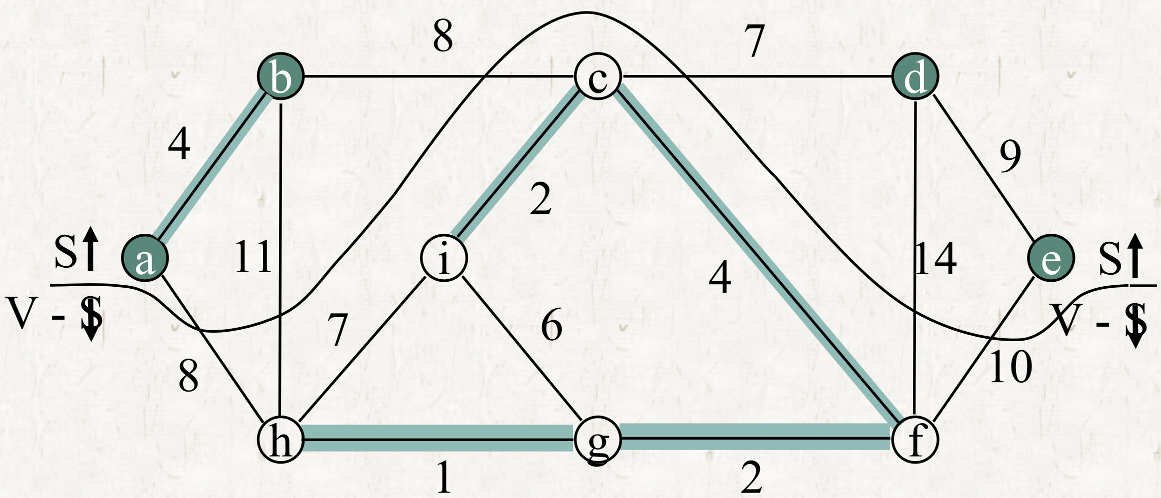
Minimum Spanning Trees

- 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 $V - S$.



Minimum Spanning Trees

- 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.



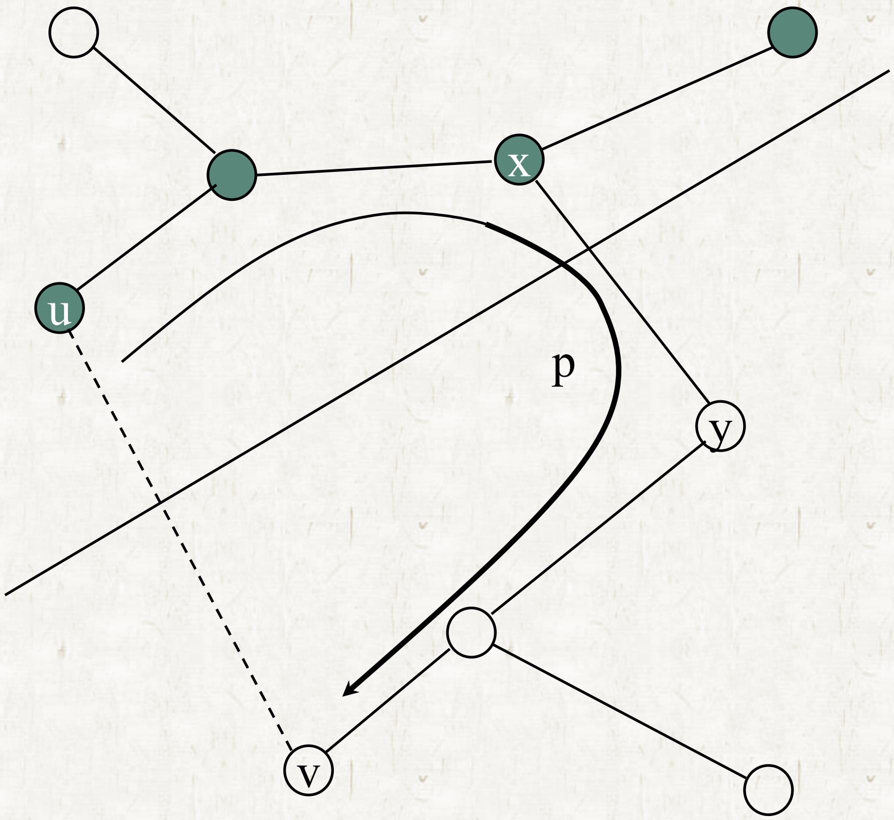
Minimum Spanning Trees

- 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 $A \cup \{(u, v)\}$.

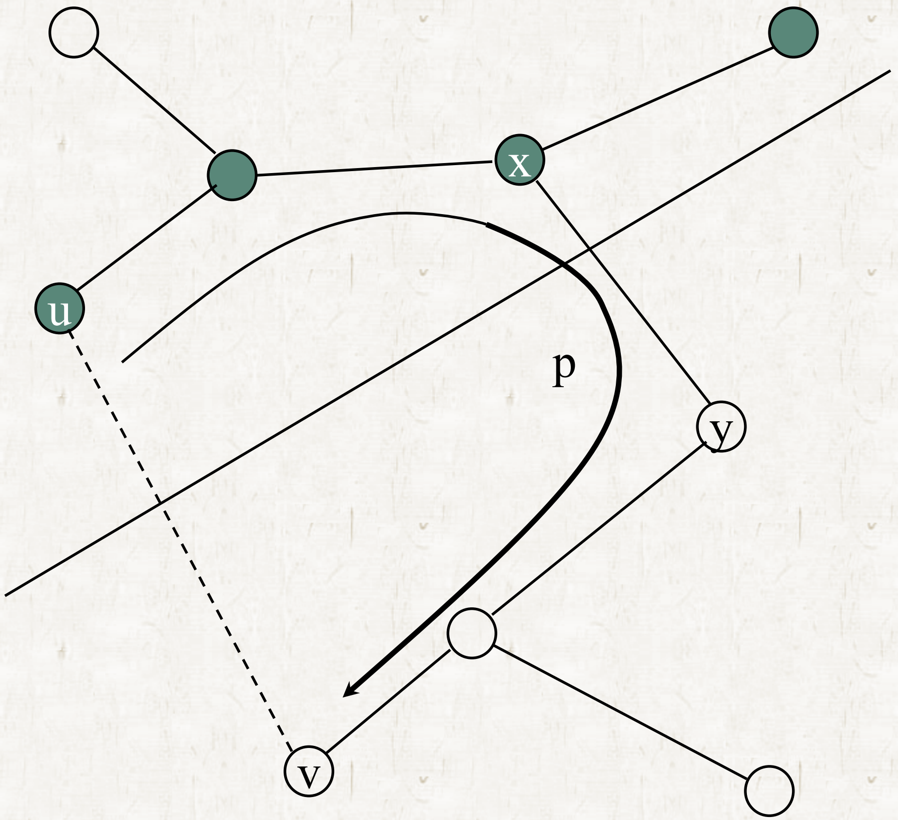
Minimum Spanning Trees

- 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.



Minimum Spanning Trees

- 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
  - $T' = T - \{(x, y)\} \cup \{(u, v)\}$.



Minimum Spanning Trees

- 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, $w(u, v) \leq w(x, y)$.

$$w(T') = w(T) - w(x, y) + w(u, v)$$
$$\leq w(T)$$

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

Minimum Spanning Trees

- 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.

Minimum Spanning Trees

- Corollary 23.2
  - Let $G = (V, E)$ be a graph with a real-valued weight function w defined on E.
  - 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.

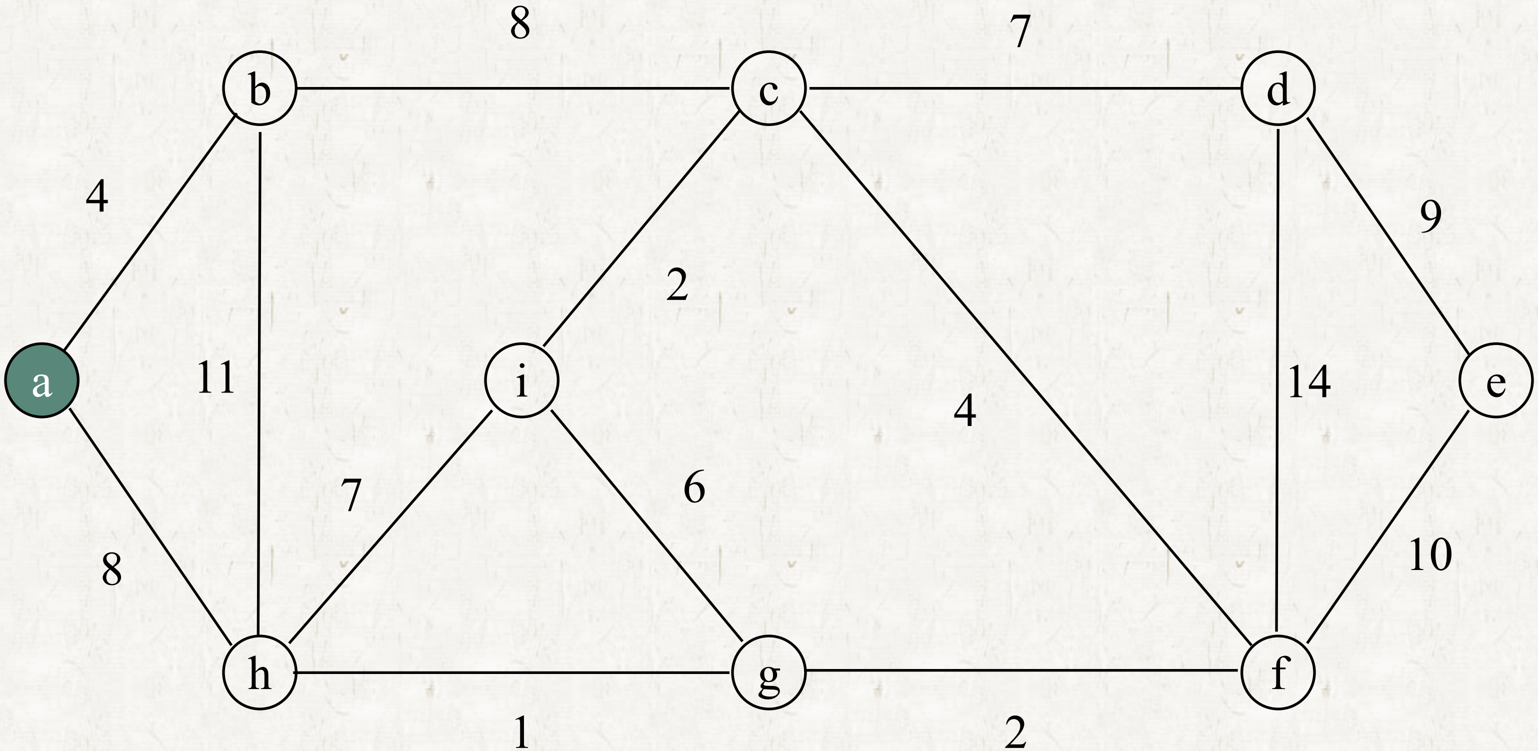
Minimum Spanning Trees

- ***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.

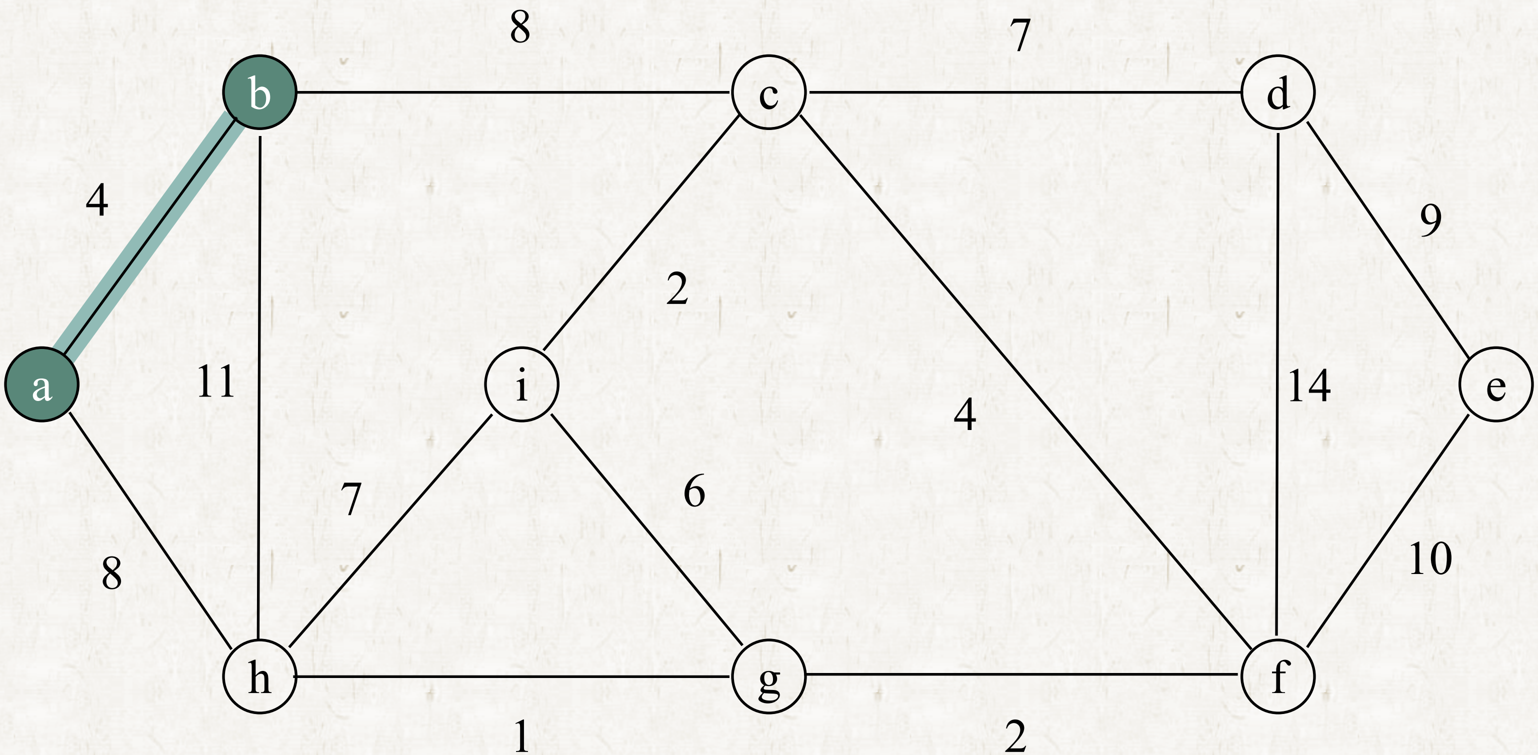
Prim's Algorithm

- 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.

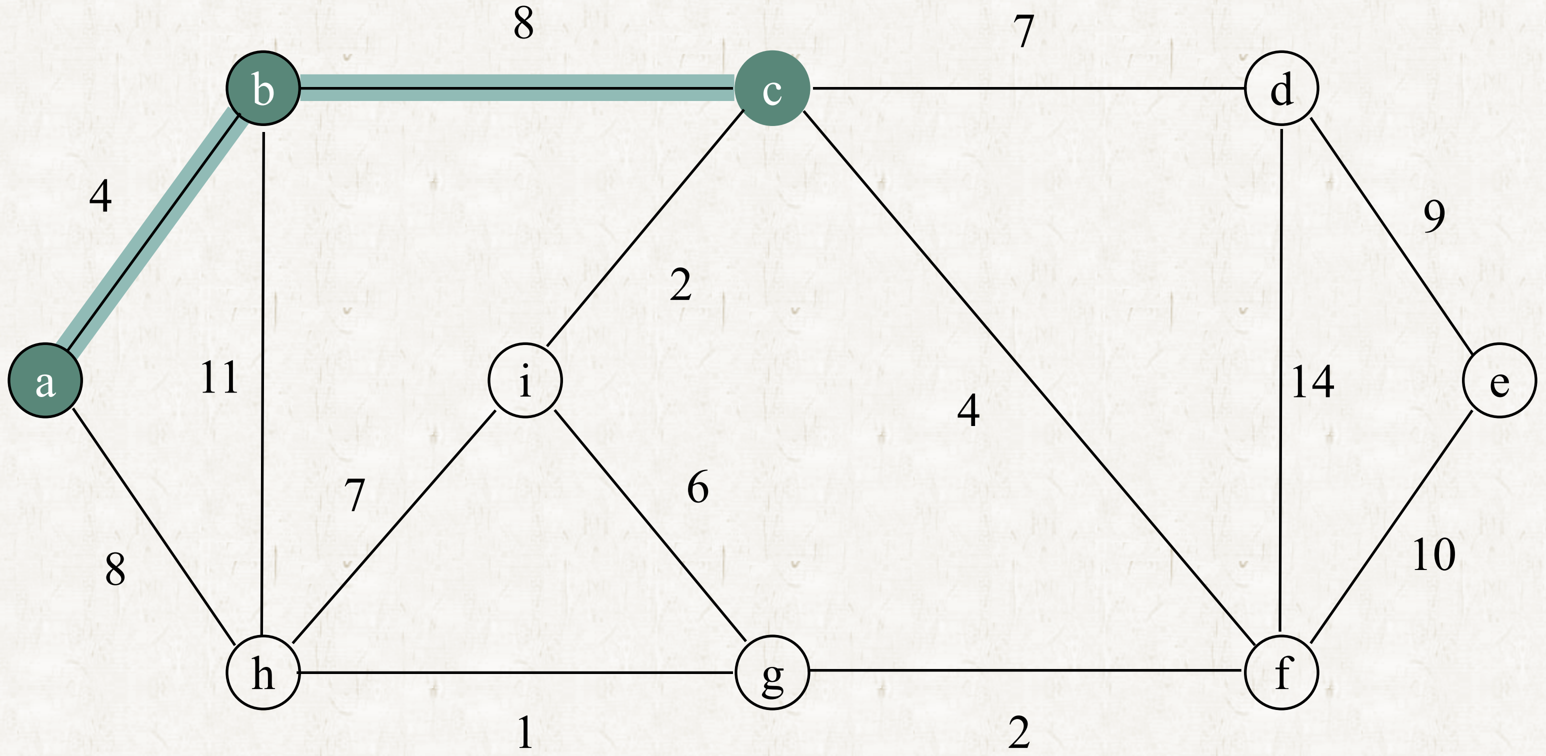
Prim's Algorithm



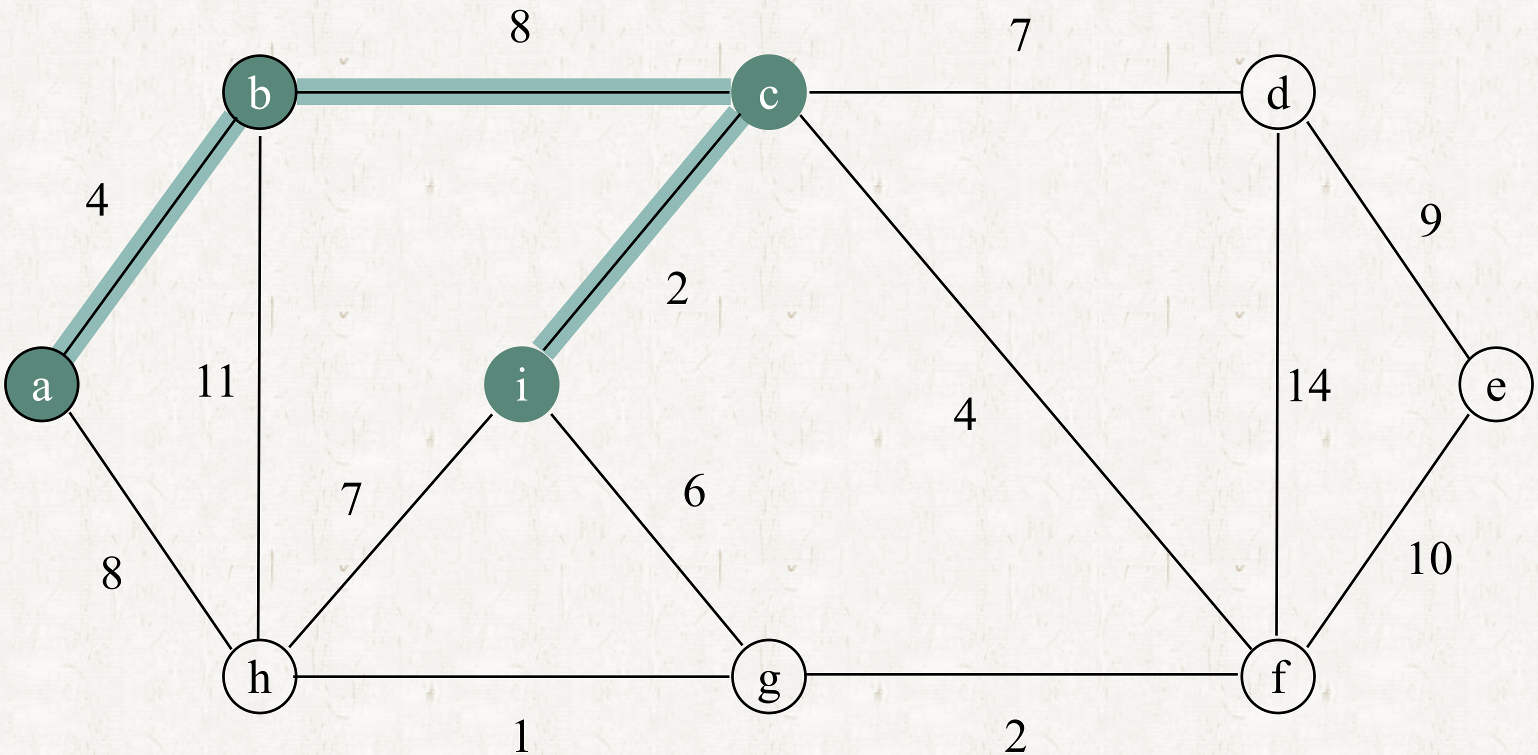
Prim's Algorithm



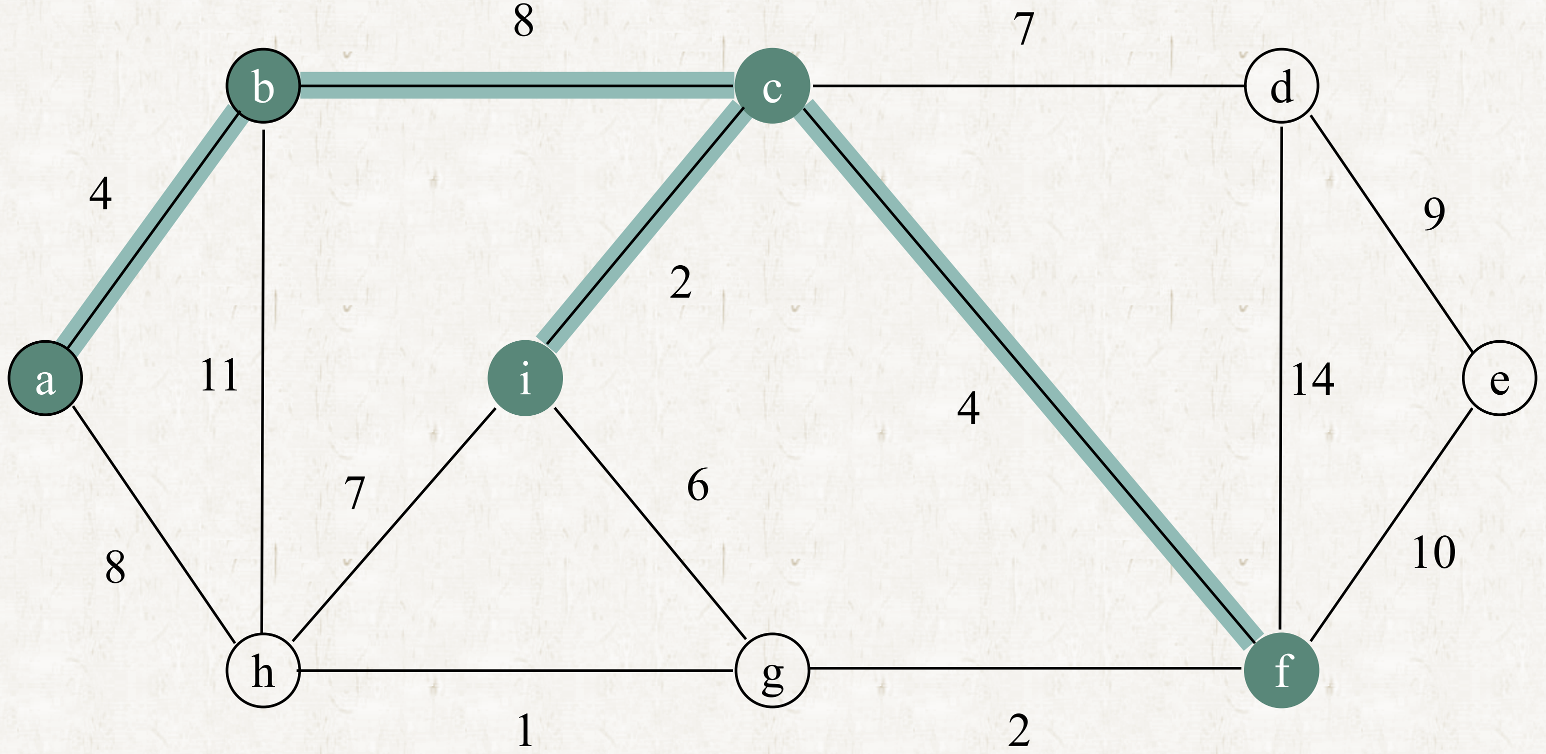
Prim's Algorithm



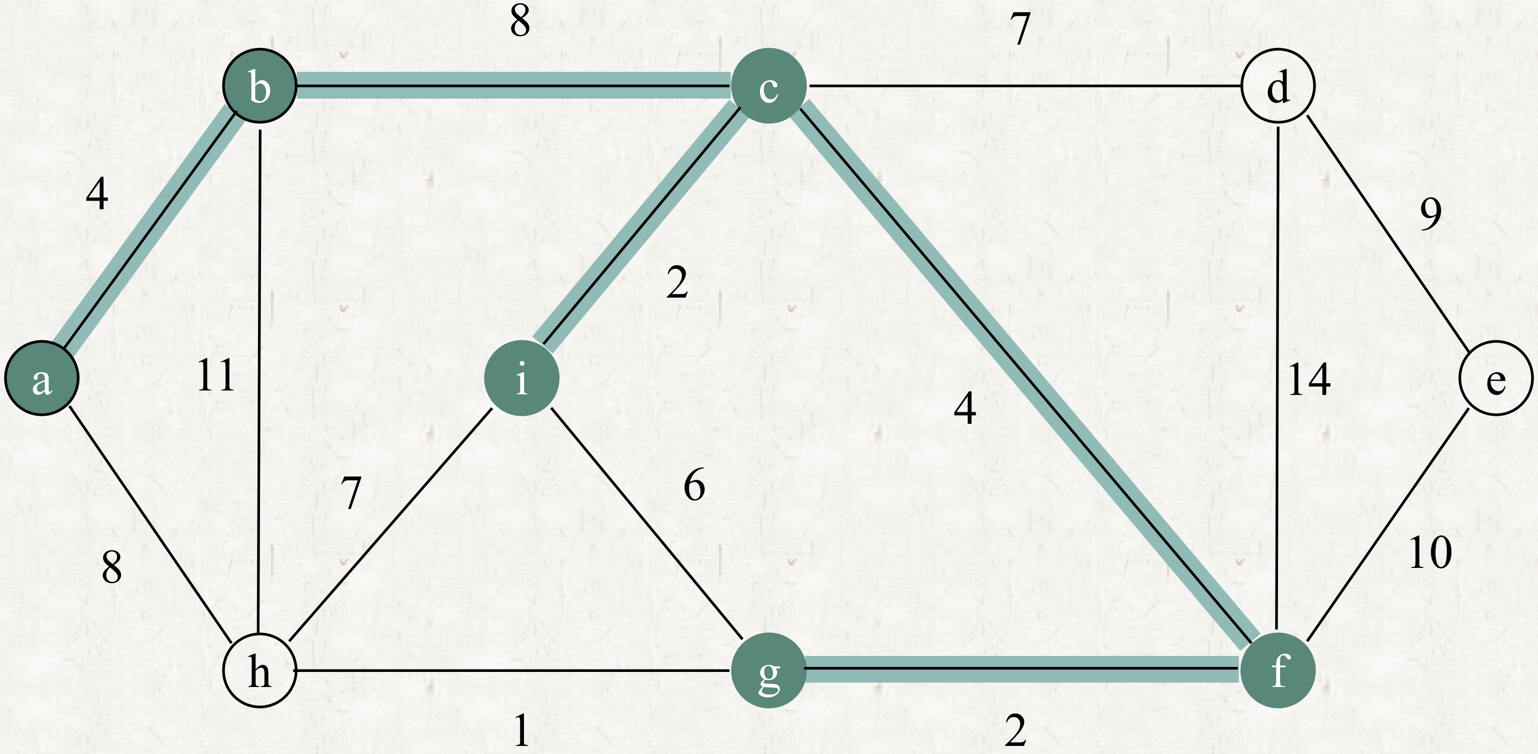
Prim's Algorithm



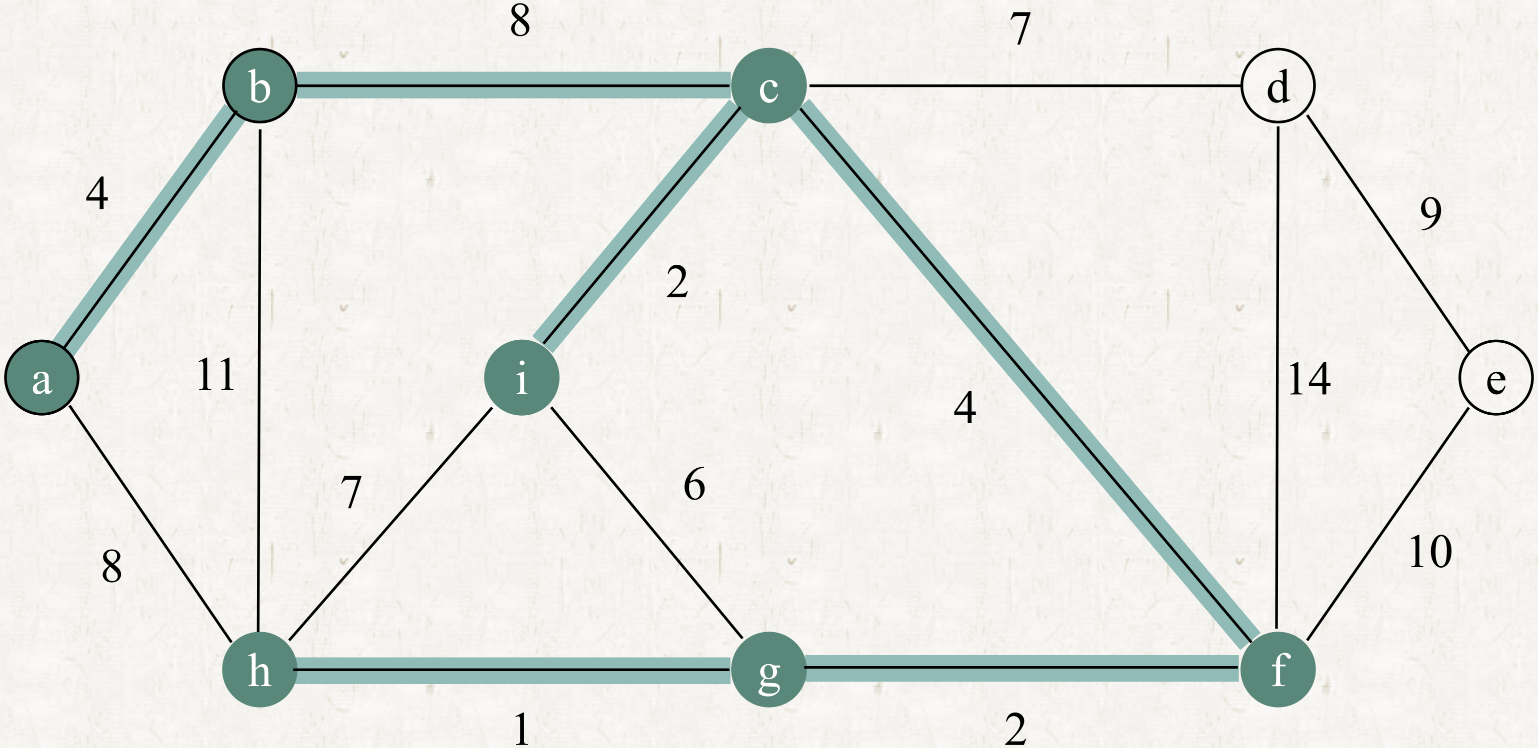
Prim's Algorithm



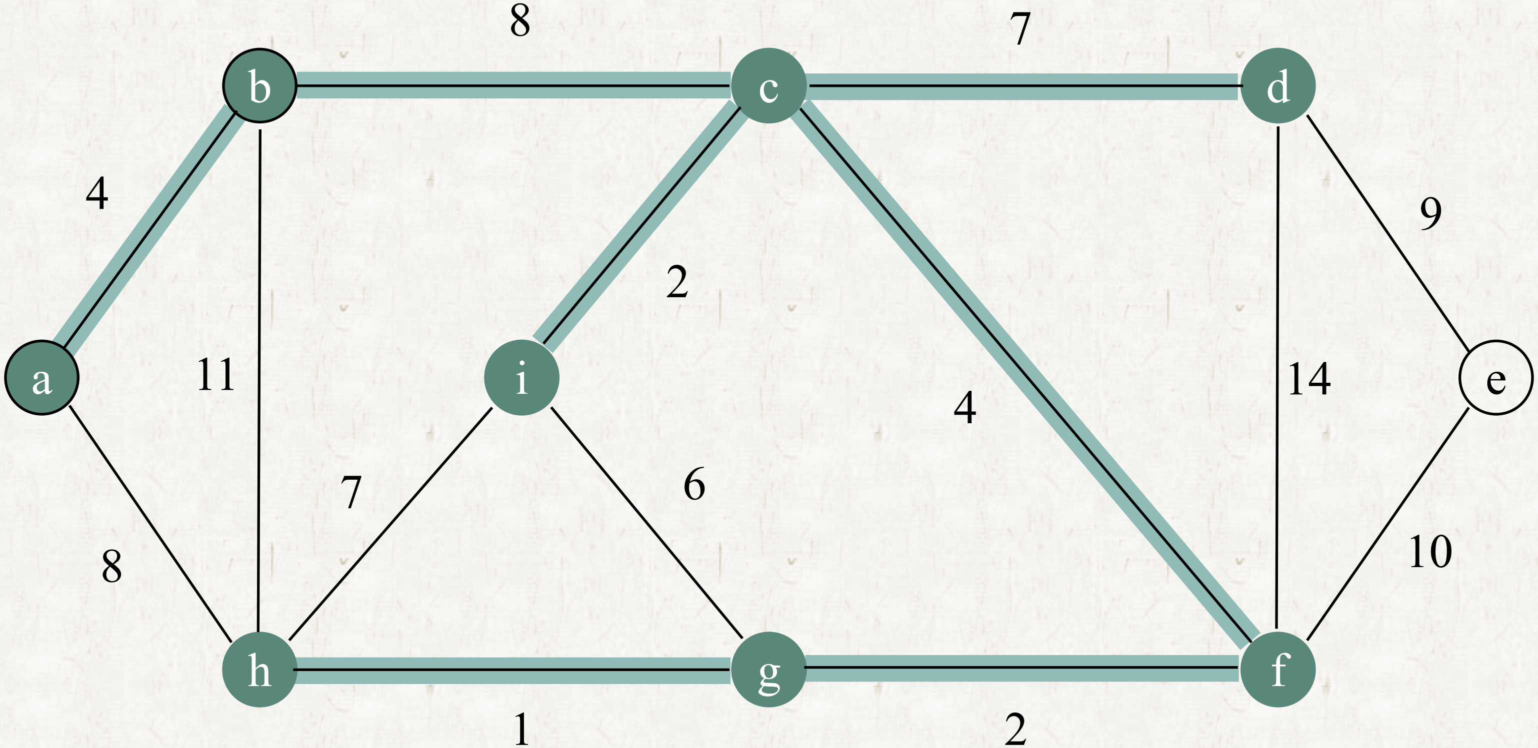
Prim's Algorithm



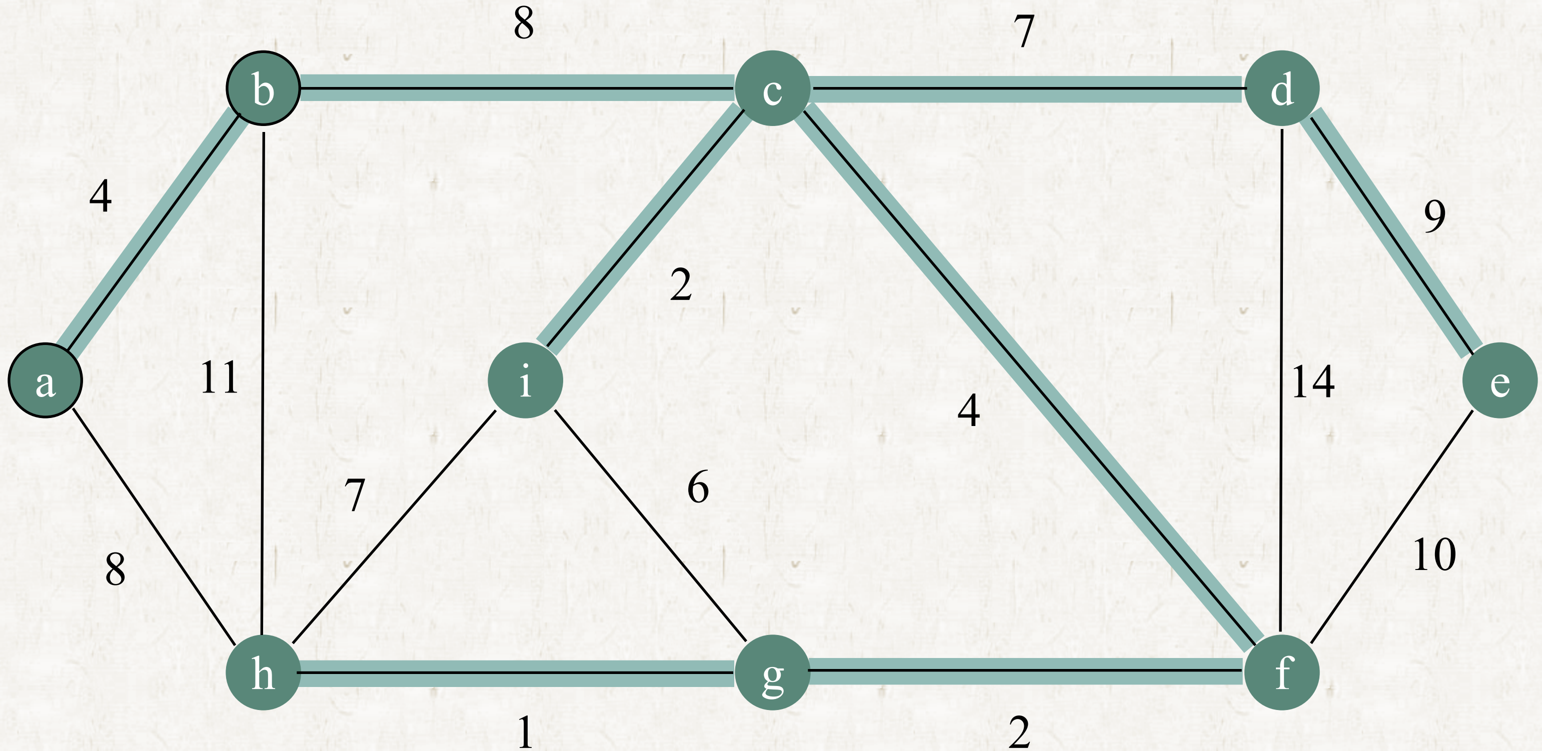
Prim's Algorithm



Prim's Algorithm



Prim's Algorithm



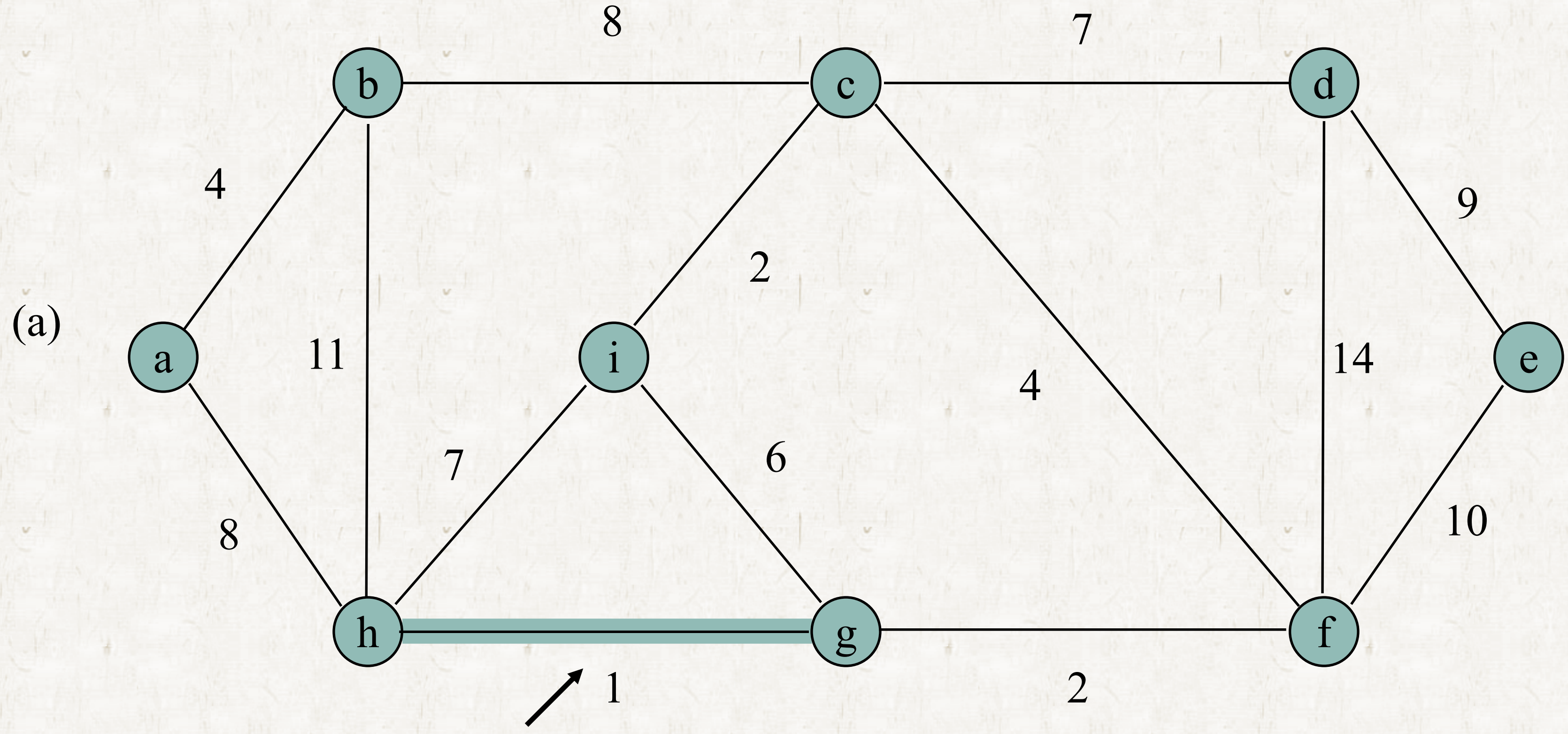
Prim's Algorithm

```
MST-PRIM(G, w, r)
    for each u ∈ V[G]
        do key[u] ← ∞
           π[u] ← NIL
    key[r] ← 0
    Q ← V[G]
    while Q ≠ Ø
        do u ← EXTRACT-MIN(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)
```

Kruskal's Algorithm

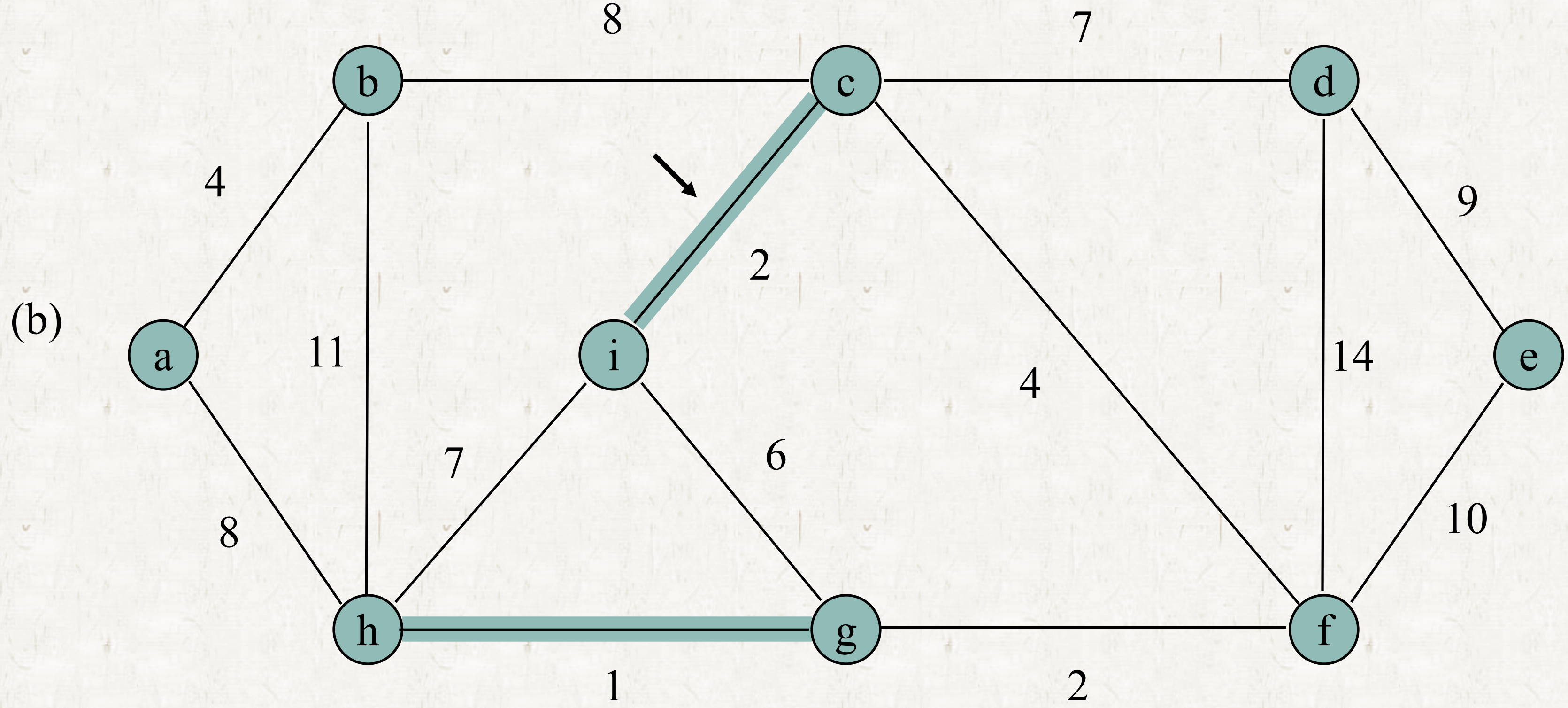
- It finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u, v) of least weight.
- Let C_1 and C_2 denote the two trees that are connected by (u, v).
- Since (u, v) must be a light edge connecting C_1 to some other tree, Corollary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

Kruskal's Algorithm



Kruskal’s algorithm

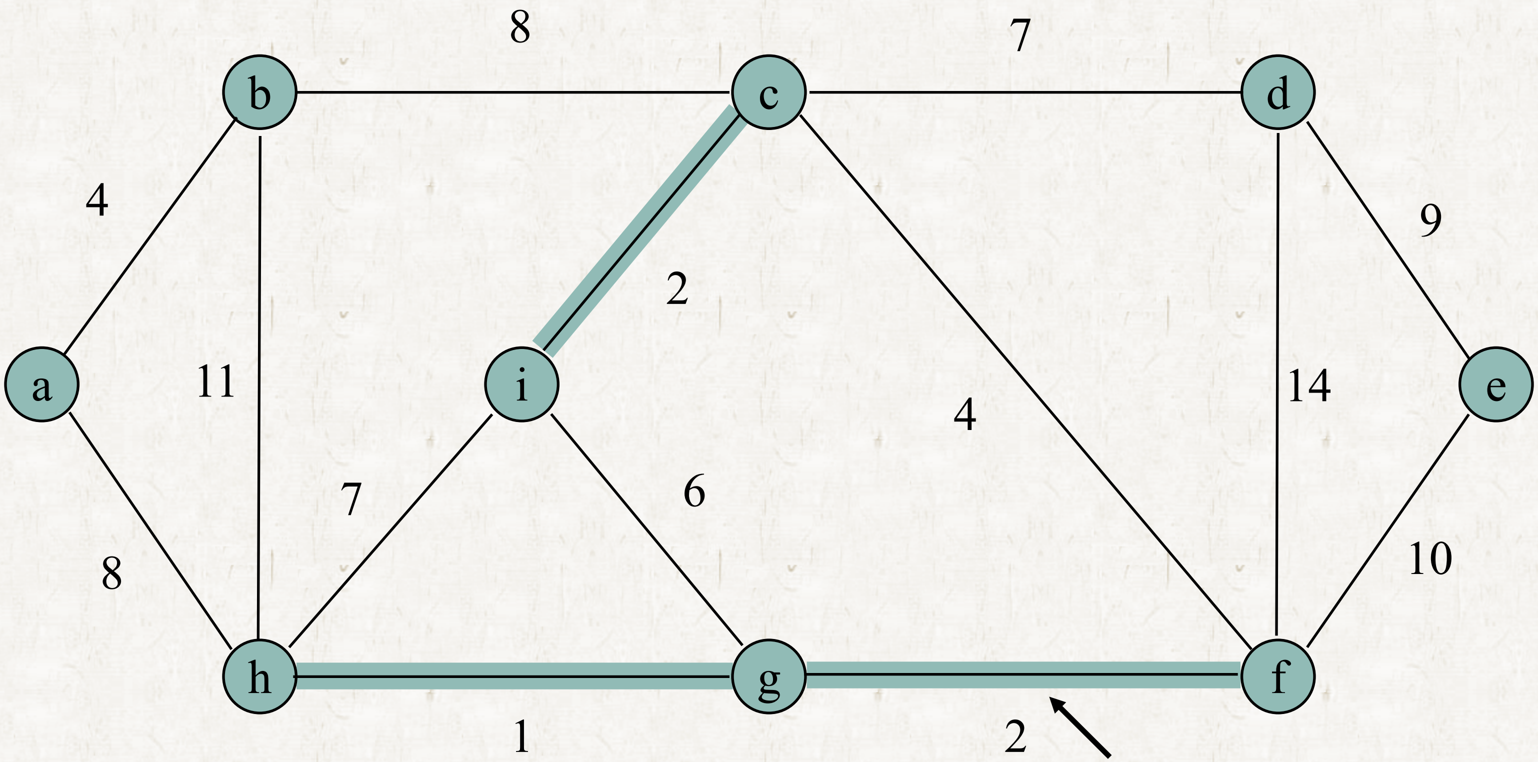
Kruskal's Algorithm



Kruskal’s algorithm

Kruskal's Algorithm

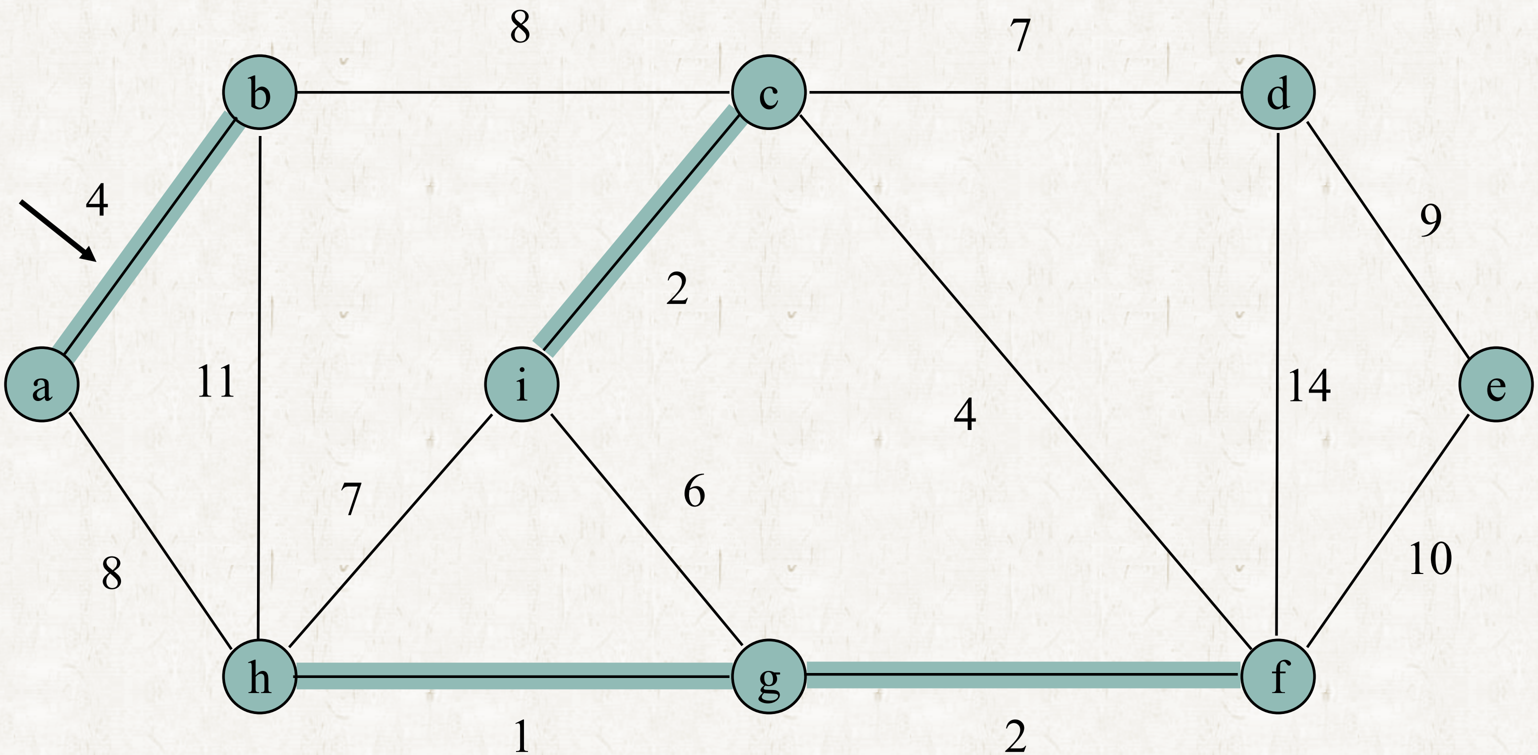
(c)



Kruskal’s algorithm

Kruskal's Algorithm

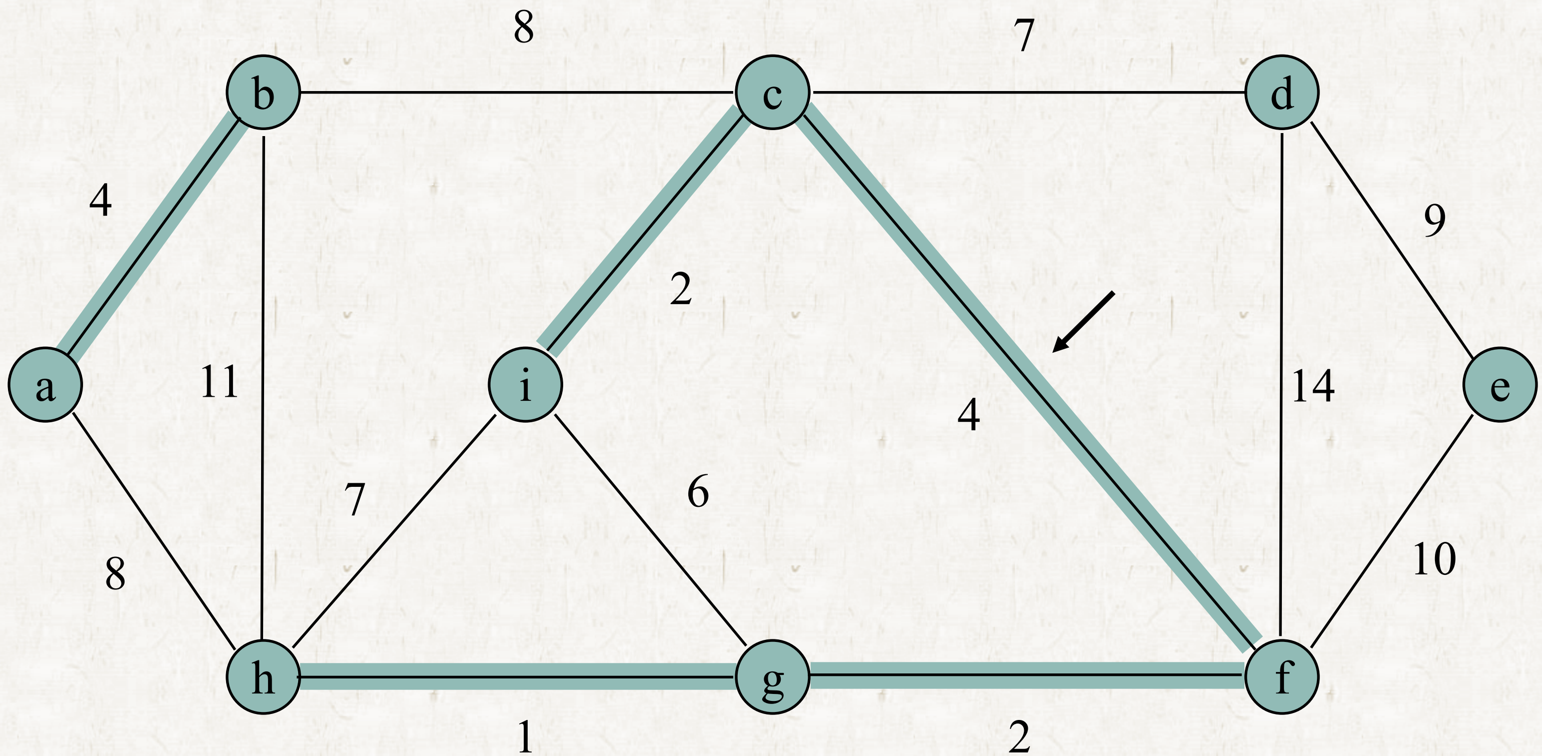
(d)



Kruskal’s algorithm

Kruskal's Algorithm

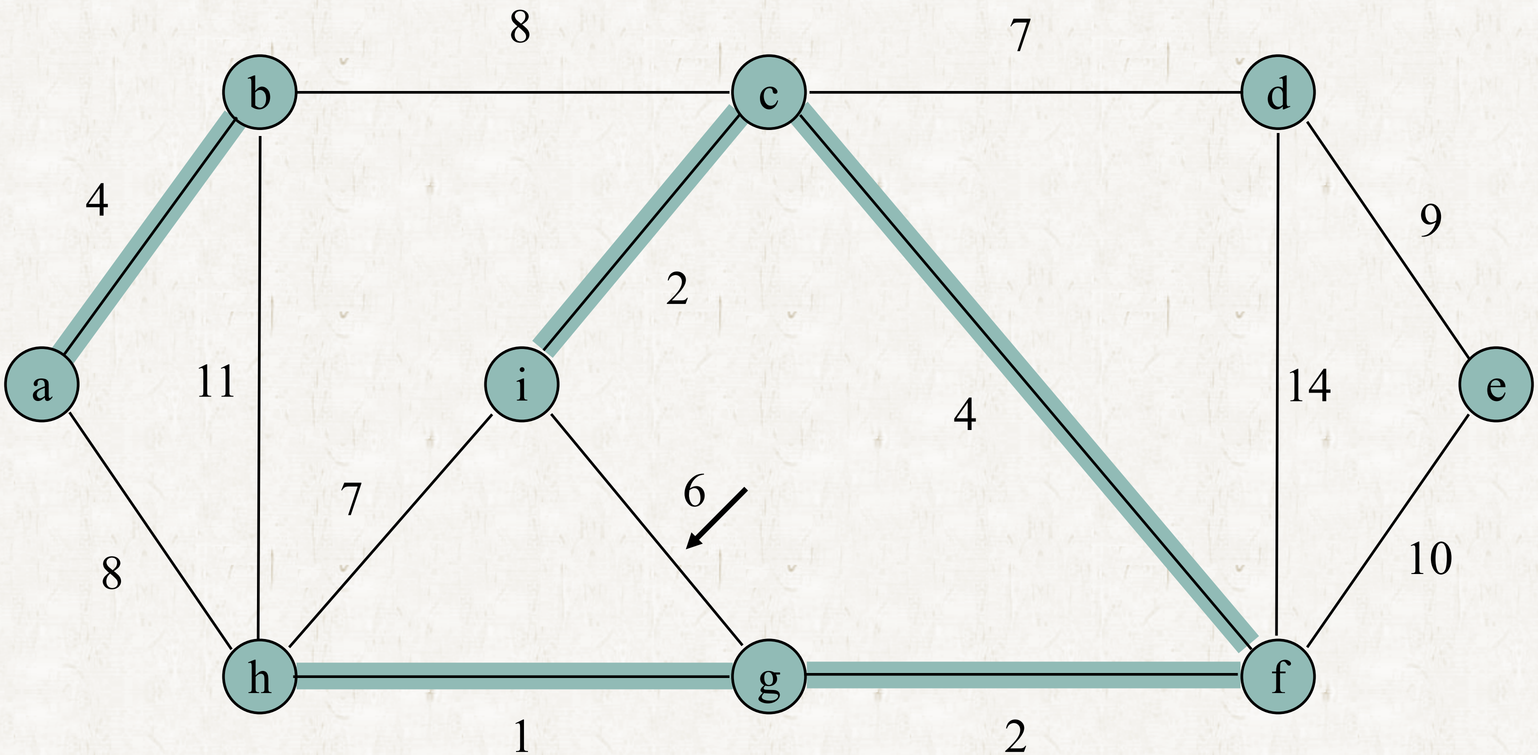
(e)



Kruskal's algorithm

Kruskal's Algorithm

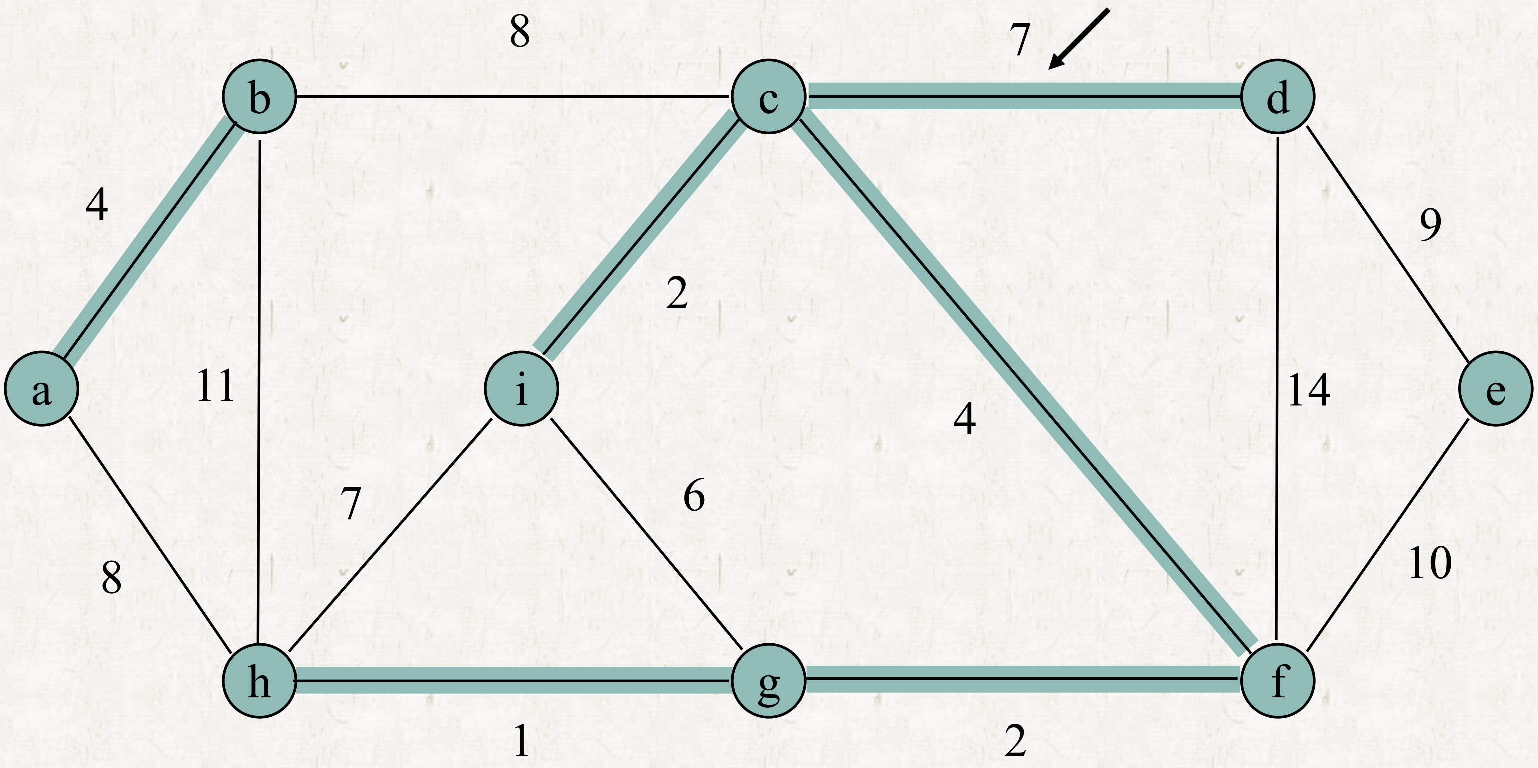
(f)



Kruskal’s algorithm

Kruskal's Algorithm

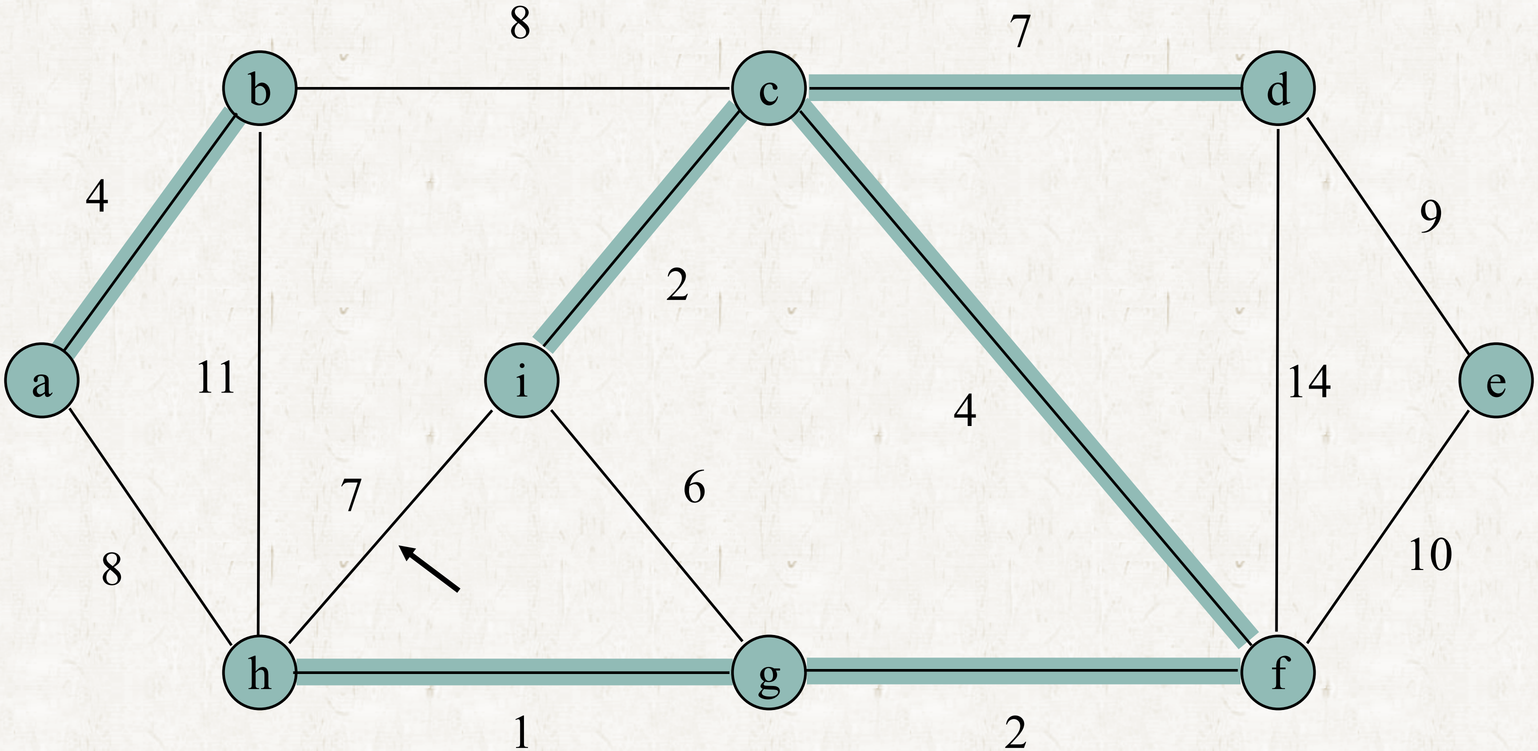
(g)



Kruskal’s algorithm

Kruskal's Algorithm

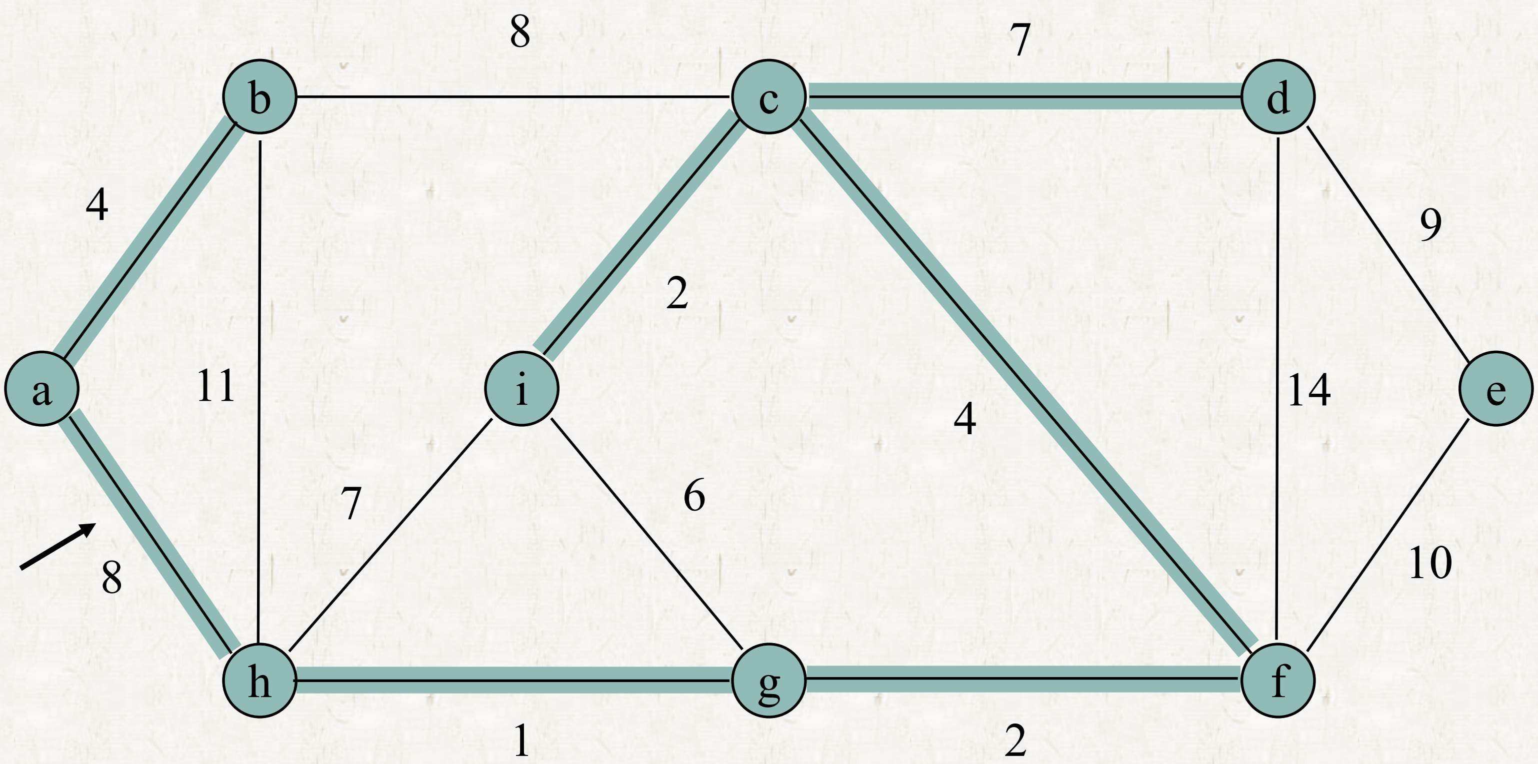
(h)



Kruskal’s algorithm

Kruskal's Algorithm

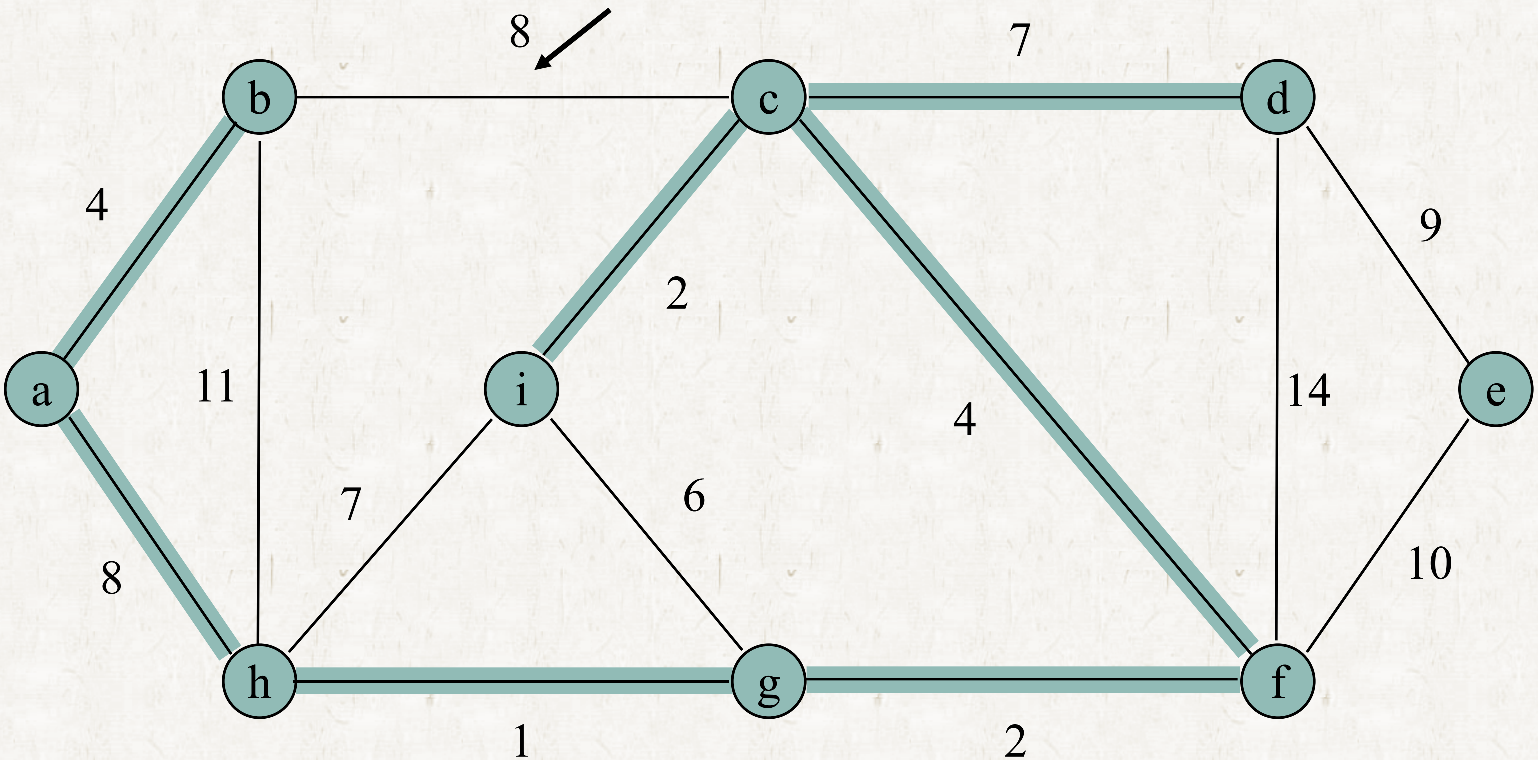
(i)



Kruskal’s algorithm

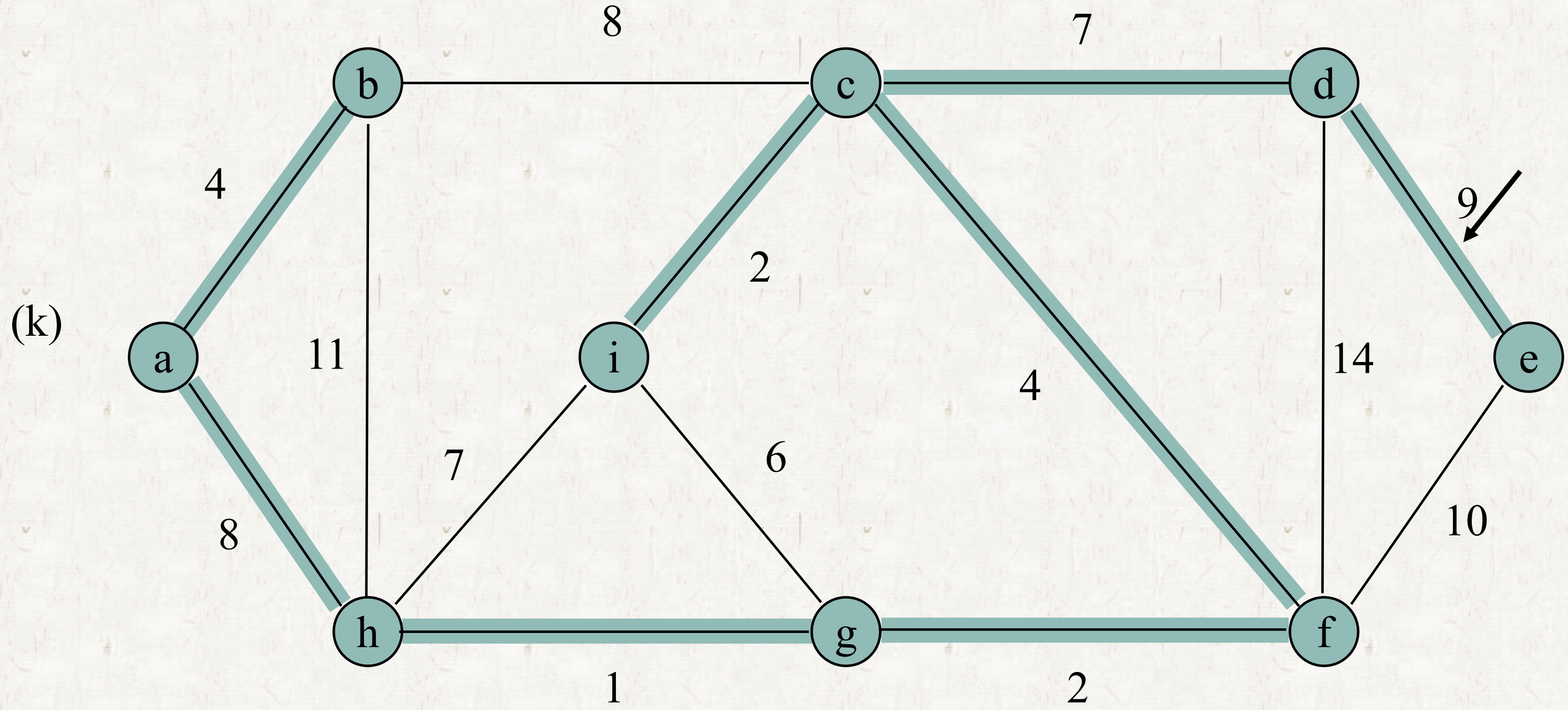
Kruskal's Algorithm

(j)



Kruskal’s algorithm

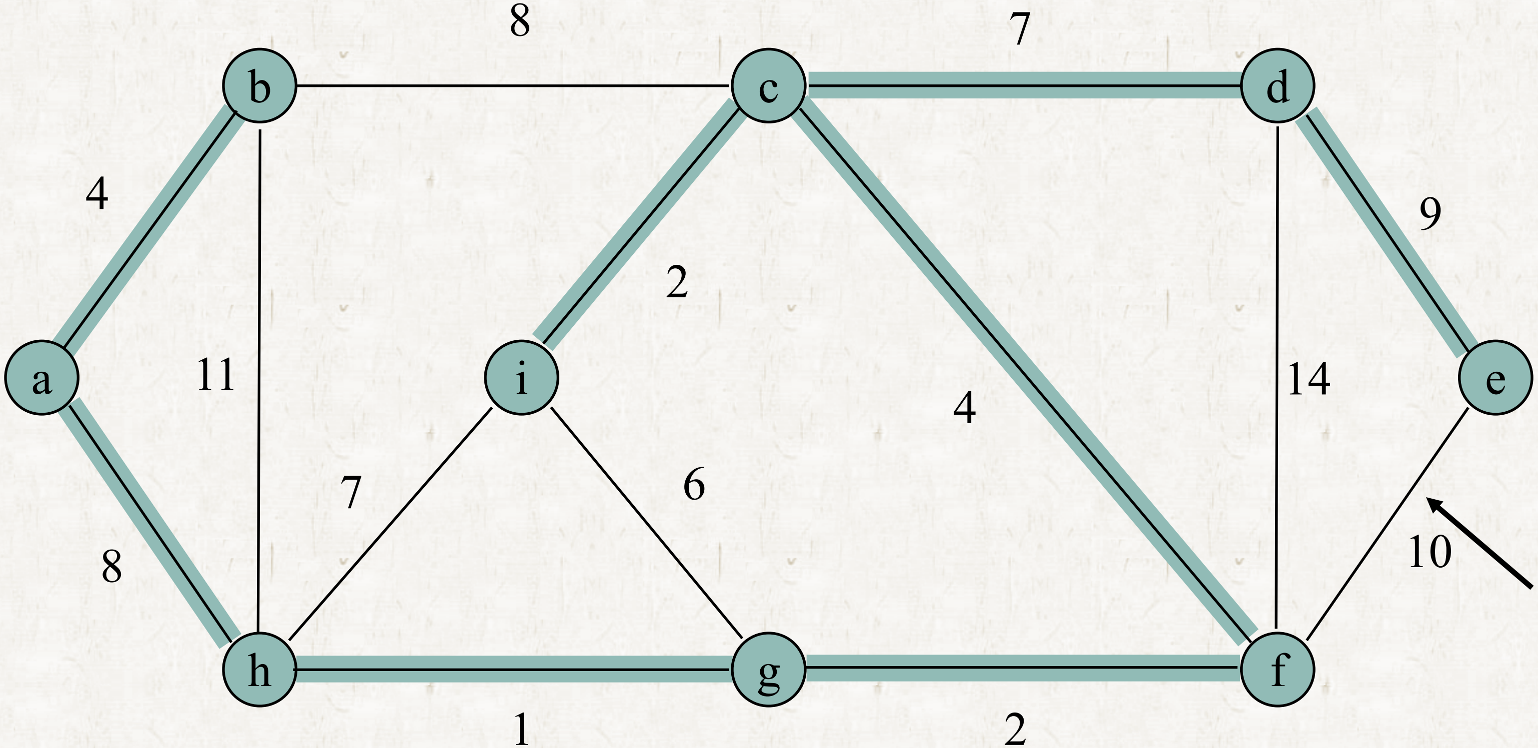
Kruskal's Algorithm



Kruskal’s algorithm

Kruskal's Algorithm

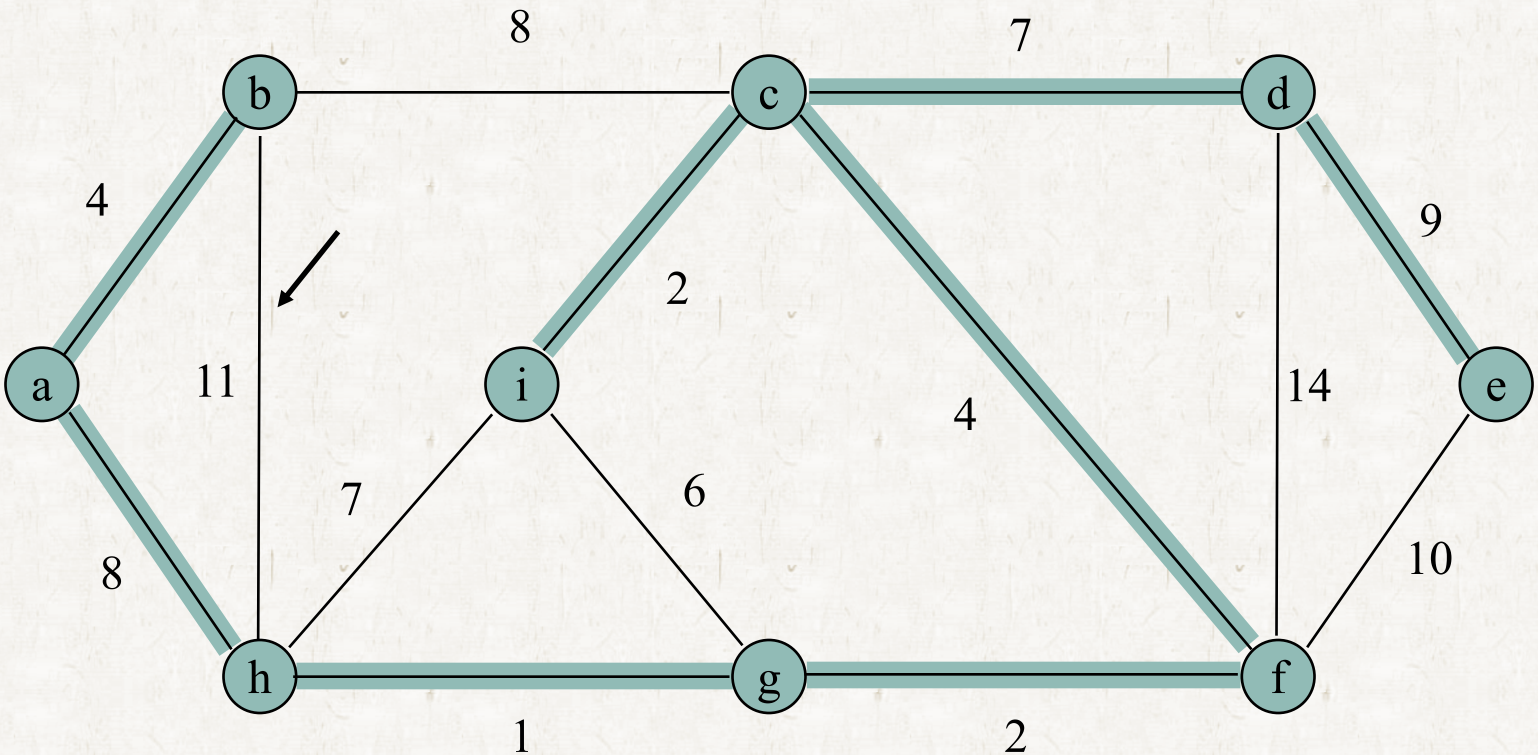
(l)



Kruskal’s algorithm

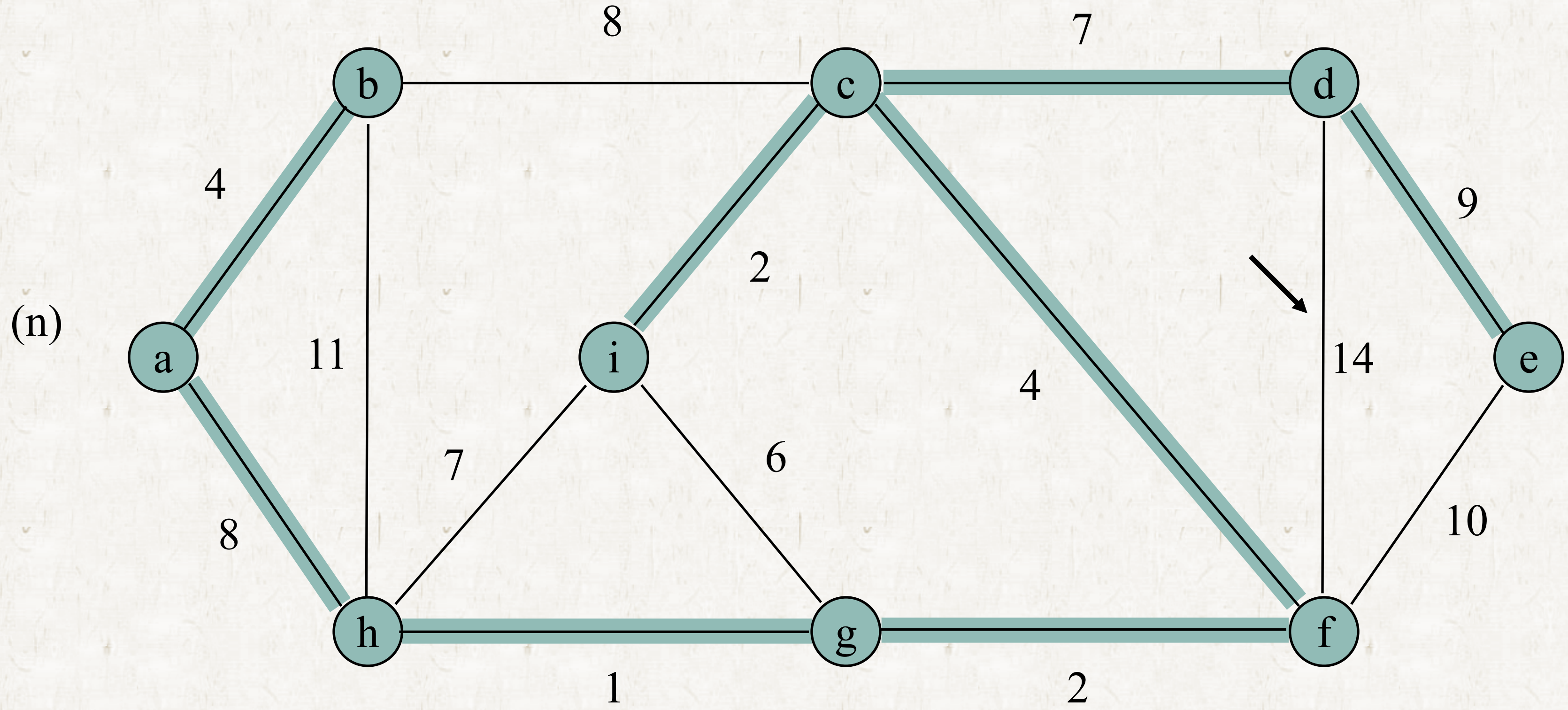
Kruskal's Algorithm

(m)



Kruskal’s algorithm

Kruskal's Algorithm



Kruskal’s algorithm

Kruskal's Algorithm

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

Time Complexity

- Prim's algorithm
  - BUILD-HEAP takes $O(|V|)$
  - **while** loop is repeated $|V|$ times, EXTRACT-MIN takes $O(\lg |V|)$ time.
  - **for** loop within the **while** loop is repeated $|E|$ times, assignment in the **for** loop requires DECREASE-KEY
- Kruskal's algorithm
  - Sorting edges can take $O(|E| \bullet \log |E|)$
  - **for** loop is repeated $|E|$ times and within the loop FIND-SET and UNION operations are called.
    => $O(|E| \bullet \alpha(|V|)) = O(|E| \bullet \log |V|)$