Algorithms
Chapter 23
Minimum Spanning Trees

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **Growing a minimum spanning tree**
- The algorithms of Kruskal and Prim

Overview$_{1/2}$

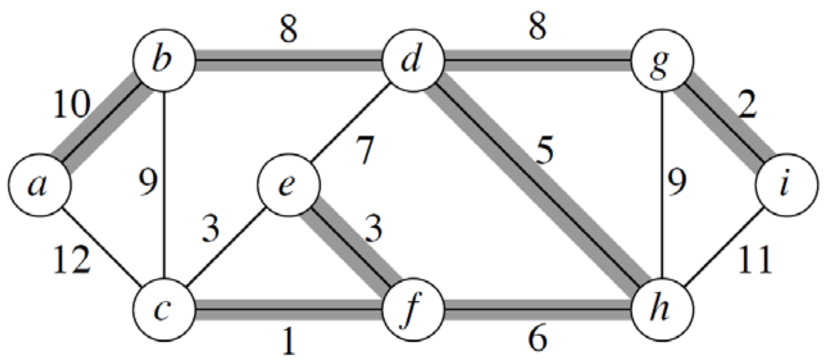
- **Problem**:
  - A town has a set of houses and a set of roads.
  - A road connects 2 and only 2 houses.
  - A road connecting houses *u* and *v* has a repair cost *w*(*u*, *v*).
  - **Goal**: Repair enough roads such that
    - everyone stays connected, and
    - total repair cost is minimum.

- **Model as a graph**:
  - Undirected graph $G = (V, E)$. **Weight** $w(u, v)$ on each edge $(u, v) \in E$.
  - Find $T \subseteq E$ such that
    - T connects all vertices (T is a spanning tree), and
    - $w(T) = \sum_{(u,v)\in T} w(u,v)$ is minimized.

Overview$_{2/2}$

- A spanning tree whose weight is minimum over all spanning trees is called a **minimum-spanning-tree**, or **MST**.

- Example of such a graph:



  - In this example, there is more than one MST.
  - Replace edge (e, f) by (c, e).
  - Get a different spanning tree with the same weight.

Growing a minimum spanning tree

- Some properties of an MST:
  - It has $|V| - 1$ edges.
  - It has no cycles.
  - It might not be unique.
- **Building up the solution**
  - We will build a set *A* of edges.
  - Initially, *A* has no edges.
  - As we add edges to *A*, maintain a loop invariant:

    Loop invariant: *A* is a subset of some MST.
  - Add only edges that maintain the invariant.
  - If *A* is a subset of some MST, an edge (u, v) is **safe** for *A* if and only if $A \cup \{(u, v)\}$ is also a subset of some MST.

Generic MST algorithm

GENERIC-MST(G, w)

$A \leftarrow \emptyset$

while A does not form a spanning tree

  find an edge (u, v) that is **safe** for A

  $A \leftarrow A \cup \{(u, v)\}$

return A

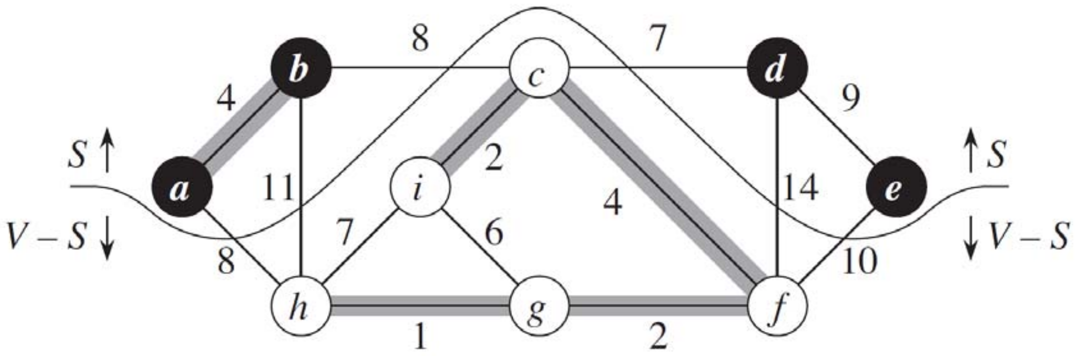
- Use the loop invariant to show that this generic algorithm works.
  - **Initialization:** The empty set trivially satisfies the loop invariant.
  - **Maintenance:** Since we add only safe edges, A remains a subset of some MST.
  - **Termination:** All edges added to A are in an MST, so when we stop, A is a spanning tree that is also an MST.

Finding a safe edge$_{1/2}$

- Edge (c, f) has the lowest weight of any edge in the graph.
  - Is it safe for $A = \emptyset$?
- Intuitively:
  - Let $S \subset V$.
  - In any MST, there has to be one edge that connects S with $V - S$.
  - Why not choose the edge with minimum weight?
- A **cut** $(S, V-S)$ is a partition of vertices into disjoint sets S and $V - S$.
- Edge $(u, v) \in E$ **crosses** cut $(S, V-S)$ if one endpoint is in S and the other is in $V - S$.
- A cut **respects** A if and only if no edge in A crosses the cut.

Finding a safe edge$_{2/2}$

- An edge is a **light edge** crossing a cut if and only if its weight is minimum over all edges crossing the cut.
  - For a given cut, there can be more than 1 light edge crossing it.



- An example:
  - The edge (d, c) is the unique light edge crossing the cut.
  - A subset A of the edges is shaded; note that the cut $(S, V-S)$ respects A, since no edge of A crosses the cut.

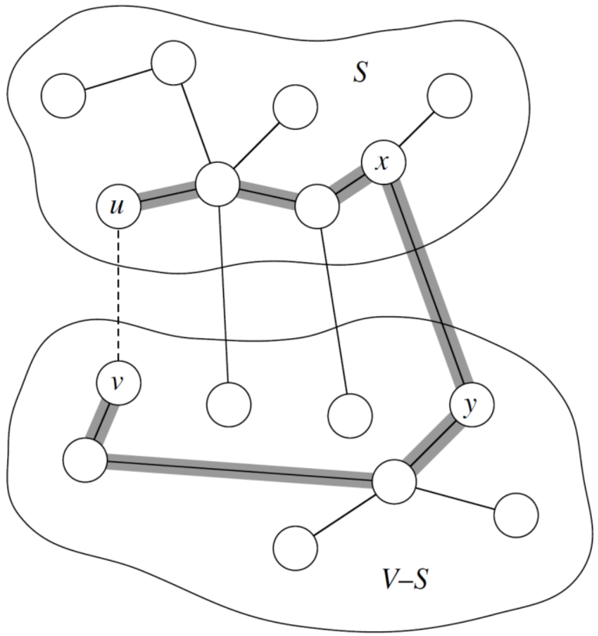
Theorem $23.1_{1/2}$

- **Theorem 23.1**
  Let A be a subset of some MST, $(S, V-S)$ be a cut that respects A, and (u,v) be a light edge crossing $(S, V-S)$.
  Then (u,v) is safe for A.

- **Proof**:
  - Let T be an MST that includes A.
  - If T contains (u, v), done.
  - Otherwise, suppose that T does not contain (u, v).
  - We'll construct a different MST T' that includes $A \cup \{(u,v)\}$.
  - Since T is an MST, it contains a **unique path *p*** between u and v.
  - Path p must cross the cut $(S, V-S)$ at least once.

Theorem $23.1_{2/2}$

- Let (x, y) be an edge of p that crosses the cut.
- Clearly, we have $w(u, v) \leq w(x, y)$.
- Let $T' = T - \{(x, y)\} \cup \{(u, v)\}$.
- Clearly, T' is also a spanning tree.
- $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$.
- T' is also an MST.
- It remains to show that (u, v) is safe for A.
- Since the cut respects A,
  edge (x, y) is not in A.
- $A \subseteq T$ and $(x, y) \notin A \Rightarrow A \subseteq T - \{(x, y)\} \subseteq T'$.
- $A \cup \{(u, v)\} \subseteq T'$.
- Since T' is an MST, (u, v) is safe for A.



Properties of GENERIC-MST

- So, in GENERIC-MST, we have:
  - A is a forest containing connected components.
  - Initially, each component is a single vertex.
  - Any safe edge merges two of these components into one.
  - Each component is a tree.
  - Since an MST has exactly $|V|-1$ edges, the **for** loop iterates $|V|-1$ times.
  - Equivalently, after adding $|V|-1$ safe edges, we're down to just one component.

Corollary 23.2

- **Corollary 23.2**
  If $C = (V_C, E_C)$ is a connected component in the forest $G_A = (V, A)$ and (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.

Outline

- Growing a minimum spanning tree
- **The algorithms of Kruskal and Prim**

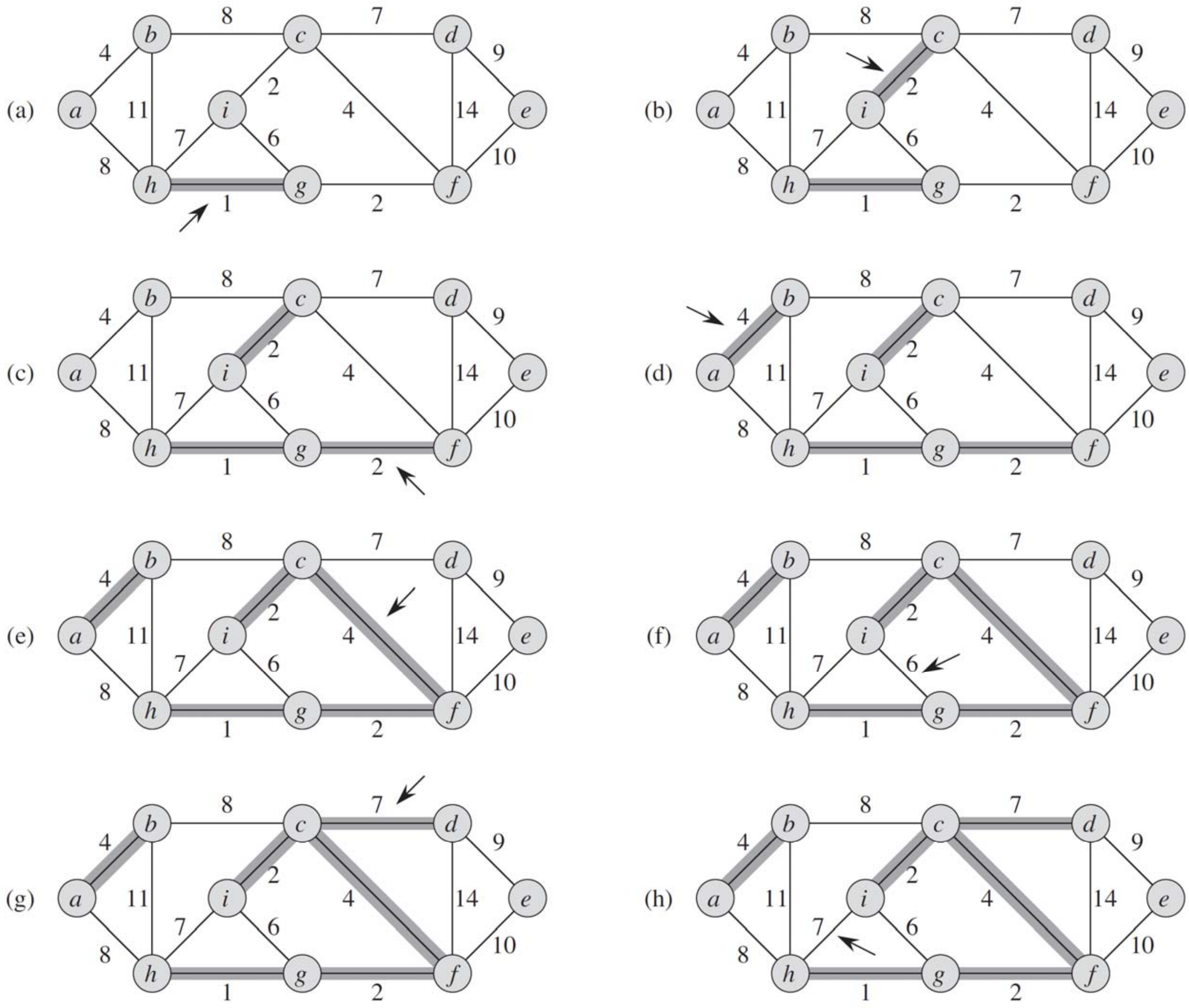
Kruskal's algorithm$_{1/2}$

- $G = (V, E)$ is a connected, undirected, weighted graph.
  $w : E \to \mathbf{R}$.
  - Starts with each vertex being its own component.
  - Repeatedly merges two components into one by choosing the light edge that connects them.
  - Scans the set of edges in monotonically increasing order by weight.
  - Uses a **disjoint-set** data structure to determine whether an edge connects vertices in different components.

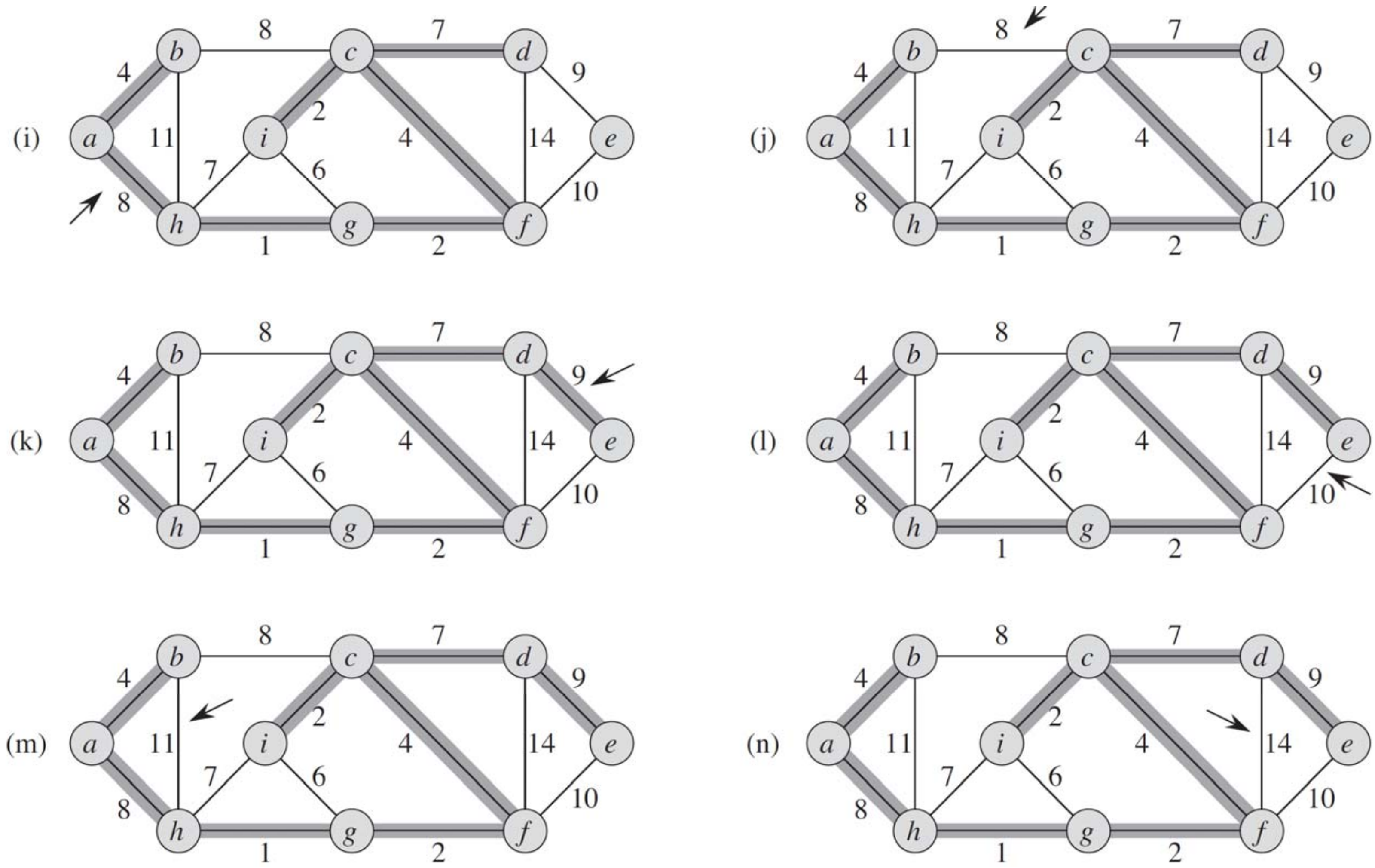
Kruskal's algorithm$_{2/2}$

```
MST-KRUSKAL(G, w)
    A ← Ø
    for each vertex v ∈ V[G]
        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
        if FIND-SET(u) ≠ FIND-SET(v)
            A ← A ∪ {(u, v)}
            UNION(u, v)
    return A
```

- In Kruskal's algorithm, the **set *A* is a forest** whose vertices are all those of the given graph.
- The safe edge added to *A* is always a least-weight edge in the graph that connects two distinct components.



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

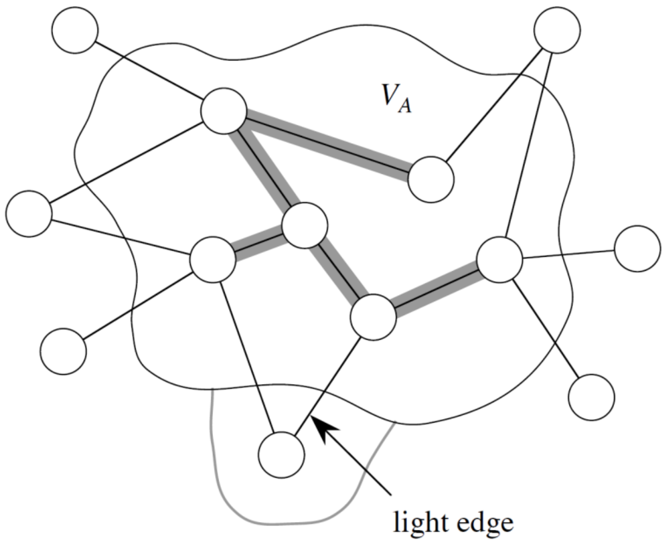


(i)
a b c d e f g h i
4 8 7 9 11 2 4 14 7 6 10 8 1 2
(j)
a b c d e f g h i
8 4 7 9 2 11 4 14 7 6 10 8 1 2
(k)
a b c d e f g h i
8 4 7 9 11 2 4 14 7 6 10 8 1 2
(l)
a b c d e f g h i
8 4 7 9 2 11 4 14 7 6 10 8 1 2
(m)
a b c d e f g h i
8 4 7 9 11 2 4 14 7 6 10 8 1 2
(n)
a b c d e f g h i
8 4 7 9 2 11 4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

Analysis

- Time complexity
  - Initialize A: $O(1)$
  - First for loop: n MAKE-SETS
  - Sort E: $O(m \lg m)$
  - Second for loop: $O(m)$ FIND-SETS and UNIONS
- Using the disjoint-set data structure in Chapter 21.
  - Time complexity: $O((n+m)\ \alpha(n)) + O(m \lg m)$
  - Since G is connected, $m \geq n - 1 \Rightarrow O(m\alpha(n)) + O(m \lg m)$
  - $\alpha(n) = O(\lg n) = O(\lg m)$.
  - Therefore, total time is $O(m \lg m)$.
  - $m \leq n^2 \Rightarrow \lg m = O(2 \lg n) = O(\lg n)$.
  - Therefore, $O(m \lg n)$ time.

Prim's algorithm$_{1/2}$

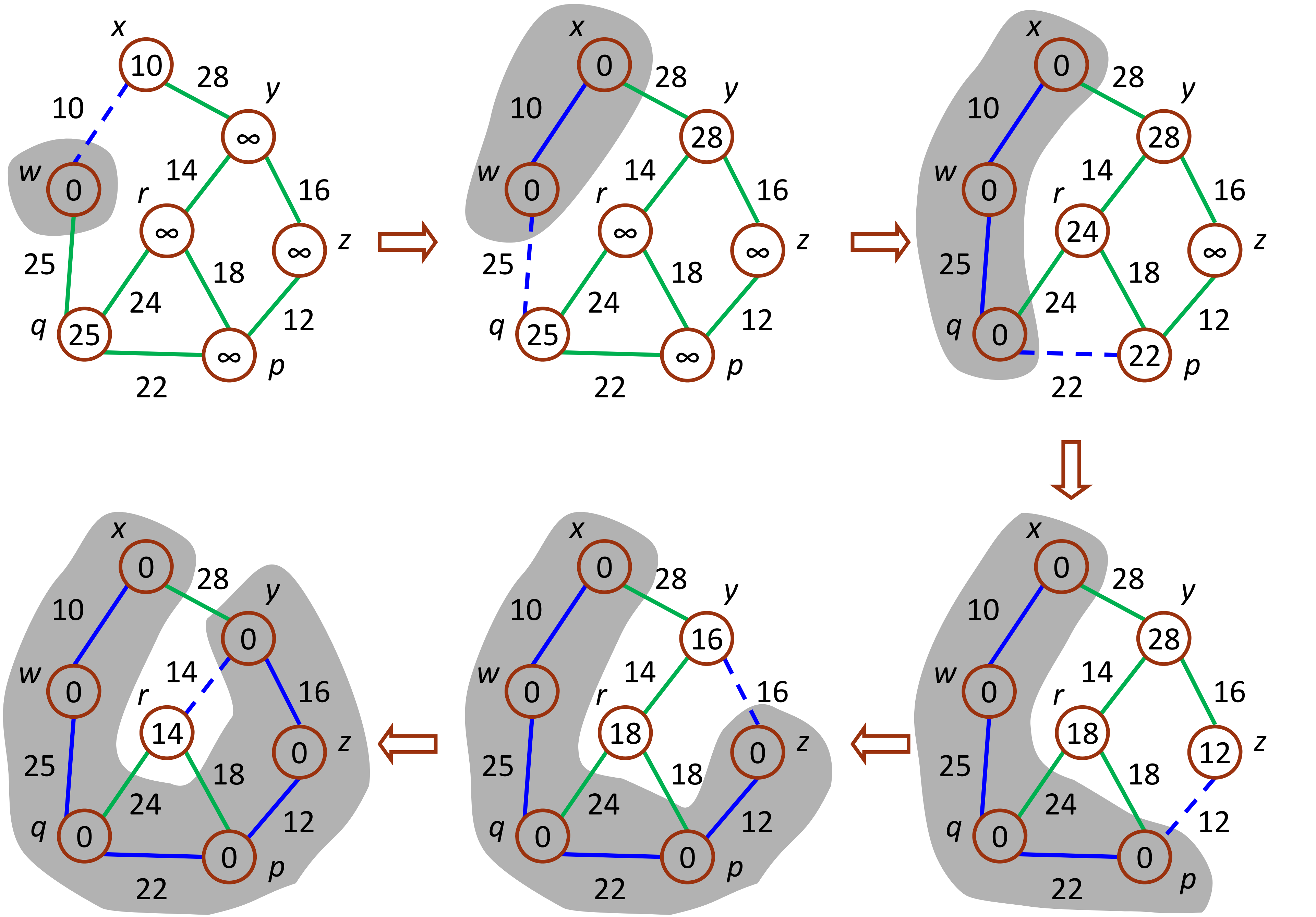


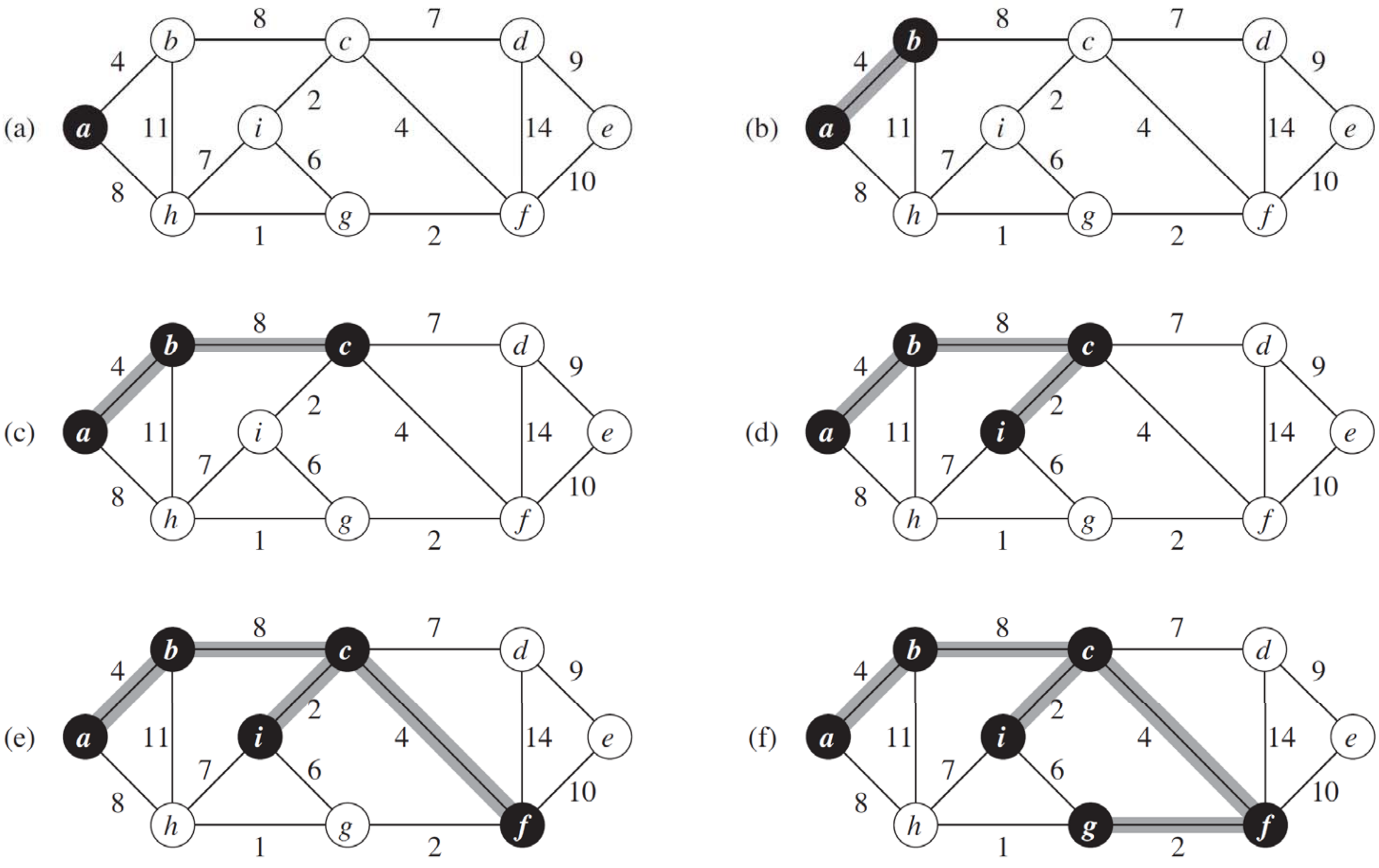
- $G = (V, E)$ is a connected, undirected, weighted graph.
  - Builds one tree, so A is always a tree.
  - Starts from an arbitrary "root".
  - At each step, find a light edge crossing cut $(V_A, V - V_A)$, where V_A = vertices that A is incident on.
  - Add this edge to A.

Prim’s algorithm$_{2/2}$

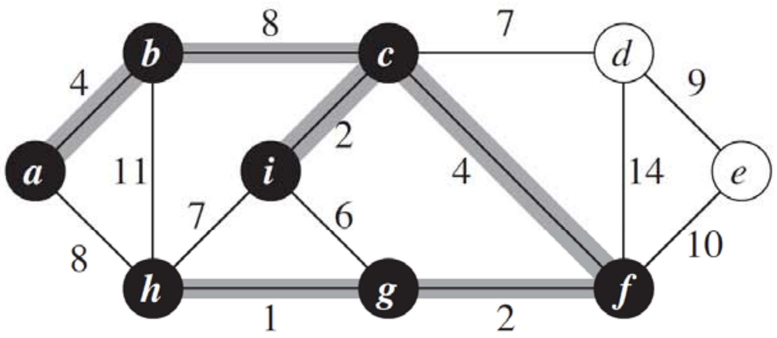
```
MST-PRIM(G, w, r)
    for each u ∈ V[G]
        key[u] ← ∞
        π[u] ← NIL
    key[r] ← 0
    Q ← V [G]
    while Q ≠ Ø
        u ← EXTRACT-MIN(Q)
        for each v ∈ Adj[u]
            if v ∈ Q and w(u, v) < key[v]
                π[v] ← u
                key[v] ← w(u, v)
```

- 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.

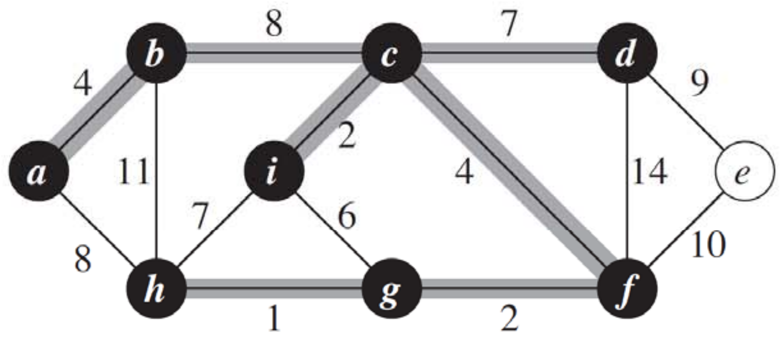

x
10
28
y
10
w
0
14
r
16
25
z
18
24
q
25
12
22
p
28
0
∞
16
12
14
24
18


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

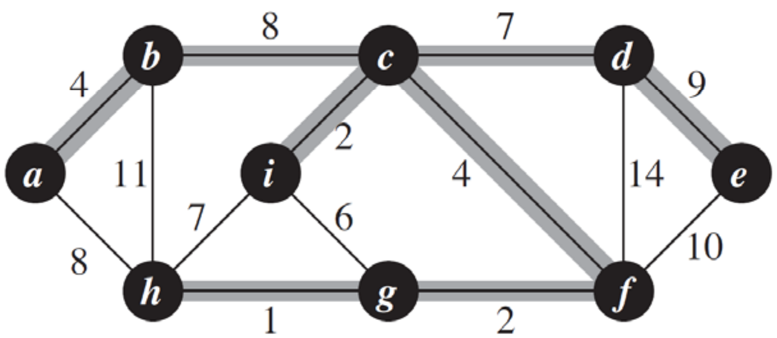
(g)


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

(h)


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

(i)


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

Analysis

- Time complexity depends on how the priority queue is implemented.
- Suppose Q is a **binary heap**. (**worst case**)
  - Initialize Q and first **for** loop: $O(n)$
  - **while** loop: n EXTRACT-MIN calls $\Rightarrow O(n \lg n)$
  - $\leq m$ DECREASE-KEY calls $\Rightarrow O(m \lg n)$
  - Total: $O(m \lg n)$
- Suppose Q is a **Fibonacci heap**. (**amortized**)
  - Initialize Q and first **for** loop: $O(n)$
  - **while** loop: n EXTRACT-MIN calls $\Rightarrow O(n \lg n)$
  - $\leq m$ DECREASE-KEY calls $\Rightarrow O(m)$
  - Total: $O(m + n \lg n)$