最小生成樹：①kruskal 演算法
②Prim 演算法
→用最少的成本讓城鎮可以相連接

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** < 通用型演算法 利用"安全边"找MST
- The algorithms of Kruskal and Prim

如何找安全边

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.

$w(u,v)$：修復道路所需成本

目標：用最小成本使城鎮可以相連接

生活問題 $\xrightarrow{\text{graph}}$ 校學問題

- **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.

house $\xrightarrow{\text{graph}}$ vertex

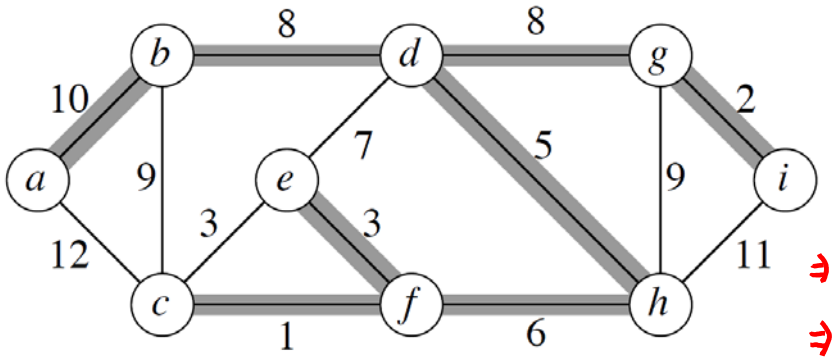
road $\xrightarrow{\text{graph}}$ edge

Overview$_{2/2}$

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

  最小生成樹：在所有生成樹中具有最少成本的生成樹

- Example of such a graph:



MST 非唯一

→將 (e, f) 換成 (c, e) 可得另一 MST

→兩者的 weight 相同

  - 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.

3個性質：
① 恰有 n-1 個邊
② 沒有 cycle
③ 不唯一

基本想法：
$A \leftarrow \phi$，一次加一個"安全"的邊，使得邊集合 A 為某一個 MST 的子集合

(u,v) 對 A 而言是"安全"的 ⇔ $A \cup \{(u,v)\}$ 是某一個 MST 的子集合

Generic MST algorithm

GENERIC-MST(G, w)

1. $A \leftarrow \emptyset$
2. **while** A does not form a spanning tree
3. find an edge (u, v) that is **safe** for A
4. $A \leftarrow A \cup \{(u, v)\}$
5. **return** A

每次加入一個安全邊
直到 A 是一個 spanning tree

維持的性質：A 集合為某一個 MST 的子集合

- Use the loop invariant to show that this generic algorithm works.

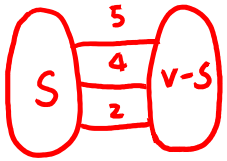
一開始 A 為空集合，自然成立（$\emptyset$ 為任何集合的子集合）

  - **Initialization:** The empty set trivially satisfies the loop invariant.
  - **Maintenance:** Since we add only safe edges, A remains a subset of some MST. 因為加的為安全邊，所以成立
  - **Termination:** All edges added to A are in an MST, so when we stop, A is a spanning tree that is also an MST.

loop invariant + A 是 spanning tree ⇒ A 是 MST

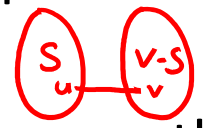
Finding a safe edge$_{1/2}$ 找安全邊

- Edge (c, f) has the lowest weight of any edge in the graph.
  - Is it safe for $A = \emptyset$? A = ∅ 時，weight 最小的邊是安全邊嗎？
- Intuitively: 基本想法：
  - 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? 選 weight 最小的邊？



MST 中一定有 edge 將 S 和 V-S 相連

- A **cut** $(S, V-S)$ is a partition of vertices into disjoint sets S and $V - S$. cut：將 V 分成 S 與 V-S 兩個集合
- Edge $(u, v) \in E$ **crosses** cut $(S, V-S)$ if one endpoint is in S and the other is in $V - S$. (u,v) crosses (S, V-S)

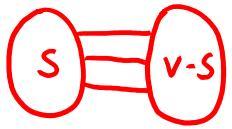


- A cut **respects** A if and only if no edge in A crosses the cut.

cut respects A：A 的 edge 都出現在 S 中或 V-S 中

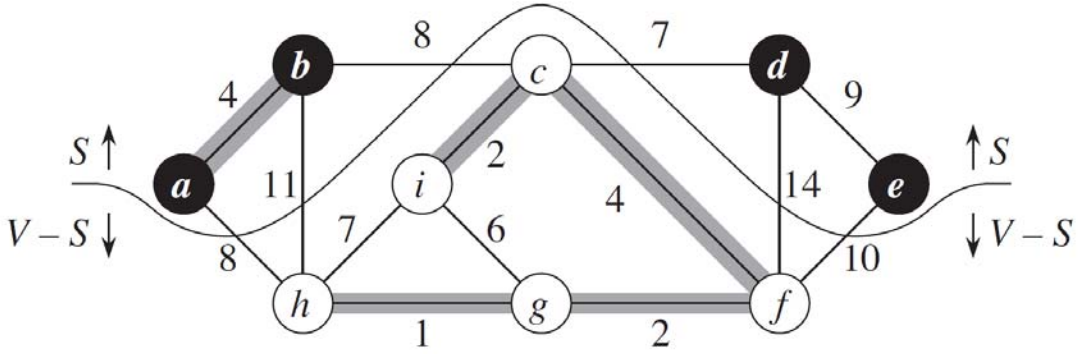
敬重

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



light edge : weight 最小的 edge (不唯一)

- 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.

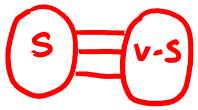


- An example:

light edge : (d, c)

  - The edge (d, c) is the unique light edge crossing the cut. A: 淺灰的 edge 形成的集合
  - 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. (S, V-S) respects A

Theorem 23.1 1/2



A中所有edge都在S中或V-S中

- **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. light edge (u, v) 对 A 而言是安全的

- **Proof**:
  - Let T be an MST that includes A. T: 一個包含 A 的 MST
  - If T contains (u, v), done. 情形1: (u,v) ∈ T ⇒ 完成
  - Otherwise, suppose that T does not contain (u, v). 情形2: (u,v) ∉ T ⇒ 找出另一個 MST T', 使得 A∪{(u,v)} ⊆ T'
  - 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.

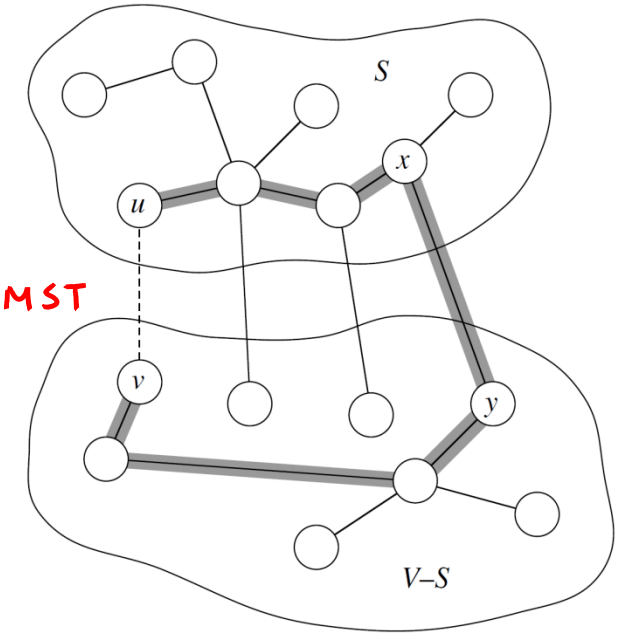
因為T是MST, 所以有一條 path p 會連接 u 和 v
p 會 cross (S, V-S) 至少一次

Theorem 23.1 2/2

讓 $T' = T - \{(x,y)\} \cup \{(u,v)\}$

證明 T' 是 MST

- Let (x, y) be an edge of p that crosses the cut. ⇒ T' 也是一個 spanning tree
- Clearly, we have $w(u, v) \le w(x, y)$.
- Let $T' = T - \{(x, y)\} \cup \{(u, v)\}$.
- Clearly, T' is also a spanning tree.
- $w(T') = w(T) - w(x, y) + w(u, v) \le w(T)$. ⇒ T' 是 MST
- T' is also an MST. 因為 (u,v) weight 較小



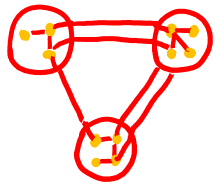
證明 (u,v) 是安全的

- 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 $(x, y) \notin A \Rightarrow A \subseteq T - \{(x, y)\} \subseteq T'$. ① $A \subseteq T'$
- $A \cup \{(u, v)\} \subseteq T'$. ② $(u,v) \subseteq T'$
- Since T' is an MST, (u, v) is safe for A. ① + ② → 目標

↳ 目標 $A \cup \{(u,v)\} \subseteq T'$

Properties of GENERIC-MST 通用型MST演算法的性質

- 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. 安全边会将2個connected component合成一個
  - 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.



可以將A看成由許多connected component所形成的森林

程式開始時A=中沒有边，所以有"n"個 connected components
已經过 n-1 次加入安全边，只剩一個 connected component

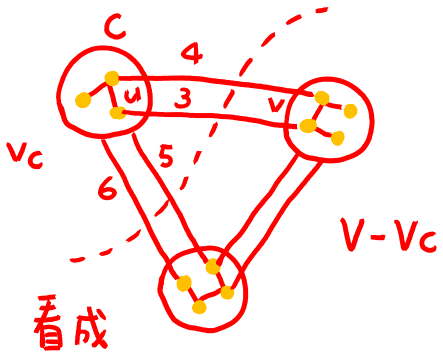
Corollary 23.2

- **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.

(u,v) 是 C 往外連的 edge 中 weight 最小的

⇒ cut (Vc, V-Vc) respects A + (u,v) 是 light edge

⇒ (u,v) 是安全邊



Outline

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

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

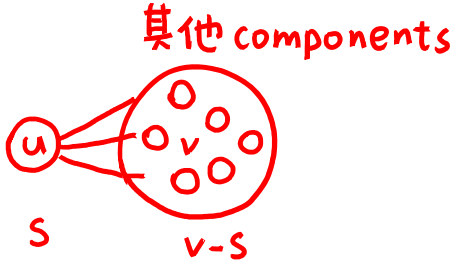
- $G = (V, E)$ is a connected, undirected, weighted graph.
  $w : E \rightarrow \mathbf{R}$.
  - Starts with each vertex being its own component.
  - Repeatedly merges two components into one by choosing the light edge that connects them.
  - Scans the set of edges in monotonically increasing order by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

使用 disjoint - set 去测试会不会形成 cycle

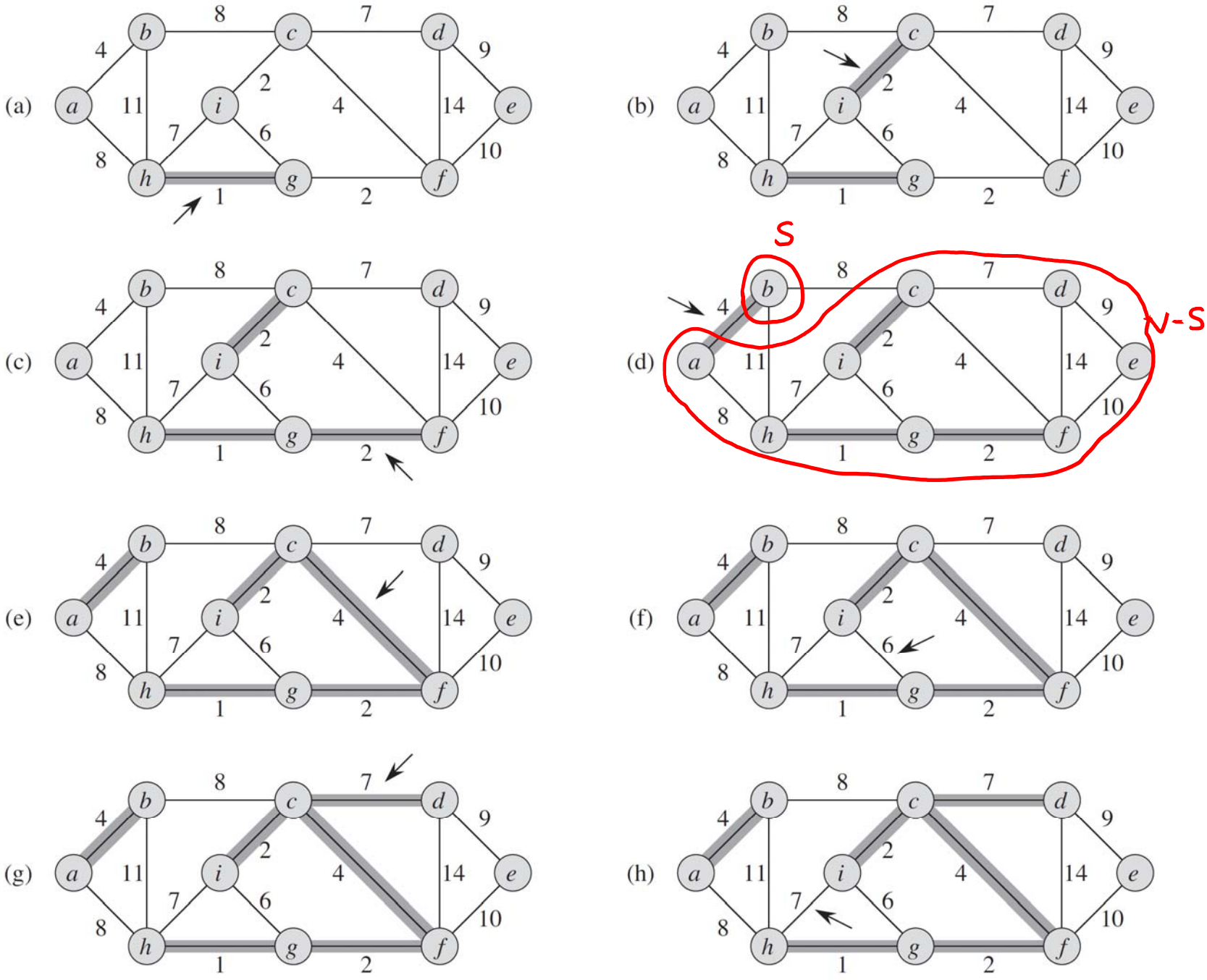
Kruskal's algorithm 2/2

```
MST-KRUSKAL(G, w)
1.  A ← ∅
2.  for each vertex v ∈ V[G]
3.      MAKE-SET(v)
4.  sort the edges of E into nondecreasing order by weight w
5.  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6.      if FIND-SET(u) ≠ FIND-SET(v)
7.          A ← A ∪ {(u, v)}
8.          UNION(u, v)
9.  return A
```

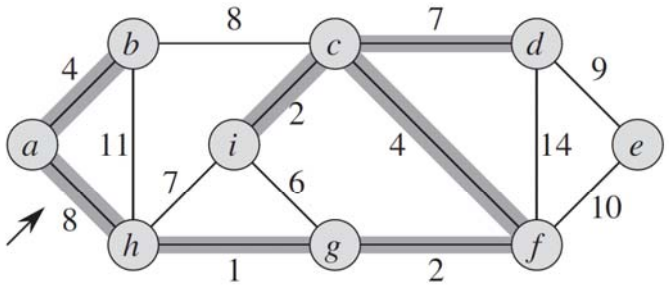
(u,v) 是選連接 2 個不同 component 的 edge 中 weight 最小的
⇒ (u,v) 是 S 往外連的 edge 中 weight 最小的
⇒ (u,v) 是安全邊
⇒ kruskal's 演算法是正確的



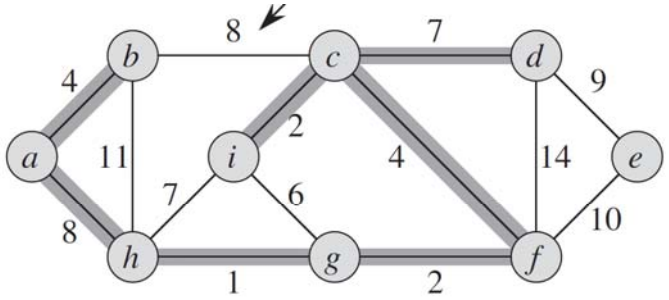
- In Kruskal's algorithm, the **set *A* is a forest** whose vertices are all those of the given graph.
- The safe edge added to *A* is always a least-weight edge in the graph that connects two distinct components.


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

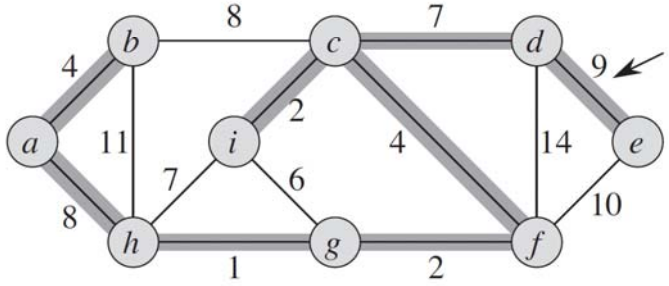
(i)


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

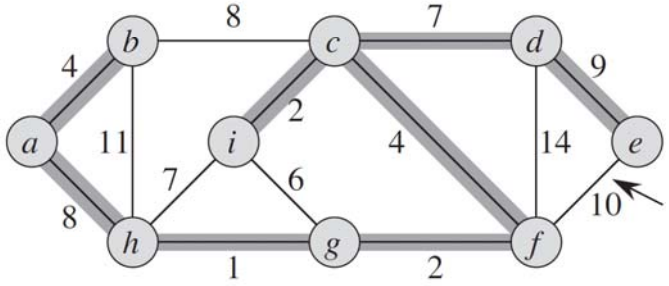
(j)


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

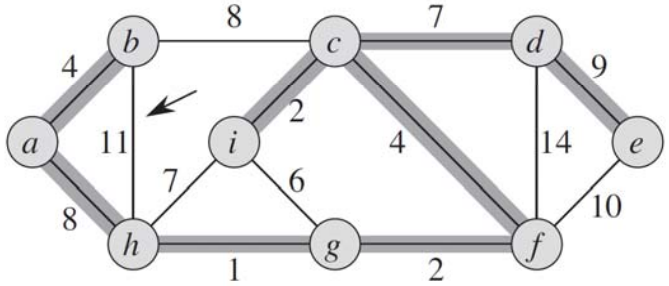
(k)


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

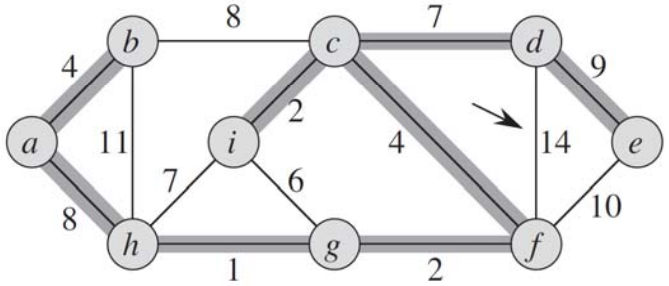
(l)


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

(m)


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

(n)


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

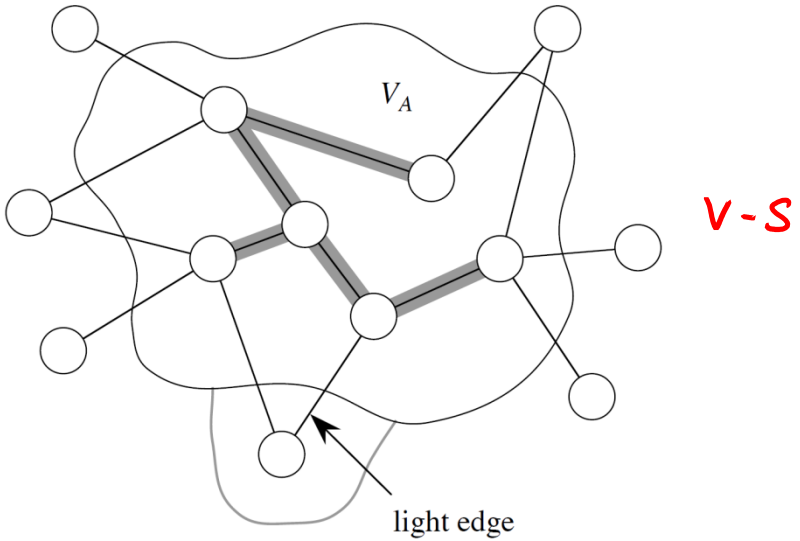
Analysis

- Time complexity
  - Initialize A: $O(1)$
  - First for loop: n MAKE-SETS
  - Sort E: $O(m \lg m)$ 排序
  - Second for loop: $O(m)$ FIND-SETS and UNIONS union-and-find 的時間

make-sets : n 次
union : n-1 次
find : 2m 次

- 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)
1.   for each u ∈ V[G]
2.       key[u] ← ∞
3.       π[u] ← NIL
4.   key[r] ← 0
5.   Q ← V [G]
6.   while Q ≠ Ø
7.       u ← EXTRACT-MIN(Q)
8.       for each v ∈ Adj[u]
9.           if v ∈ Q and w(u, v) < key[v]
10.              π[v] ← u
11.              key[v] ← w(u, v)
```

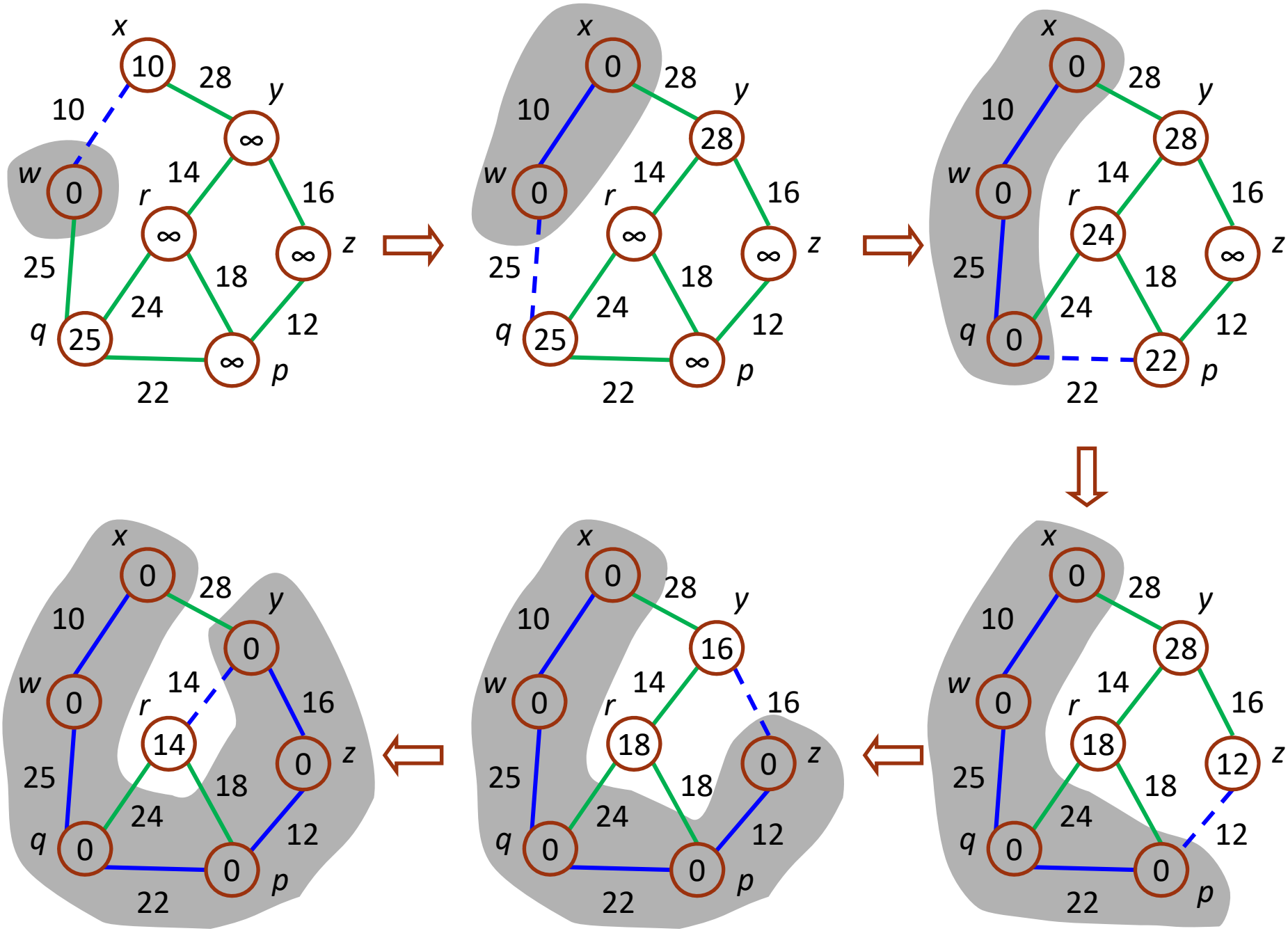
key[u]: 和"S"的最短距離

π(u): u 的父親

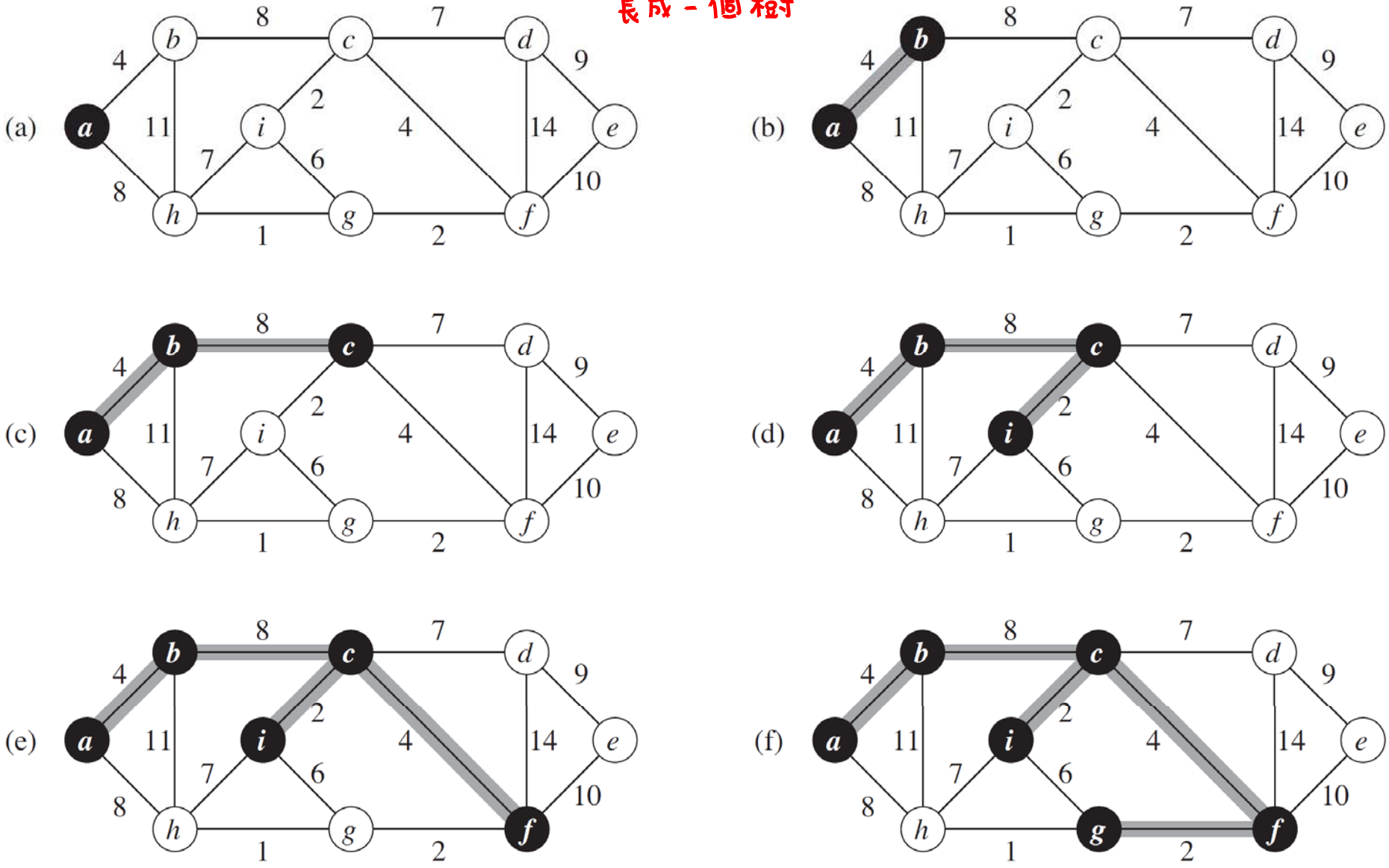
Q: heap，利用Q來找出 light edge

如果u的鄰居，因u加入而距離變短
→更新鄰居v的資訊 ① 距離
② 父親

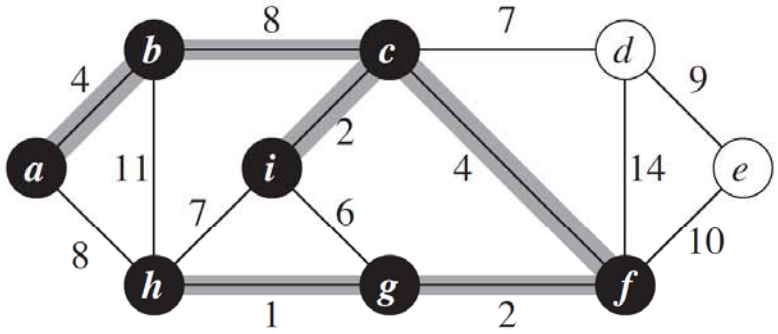
- In Prim's algorithm, the set A forms a single tree.
- The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.


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

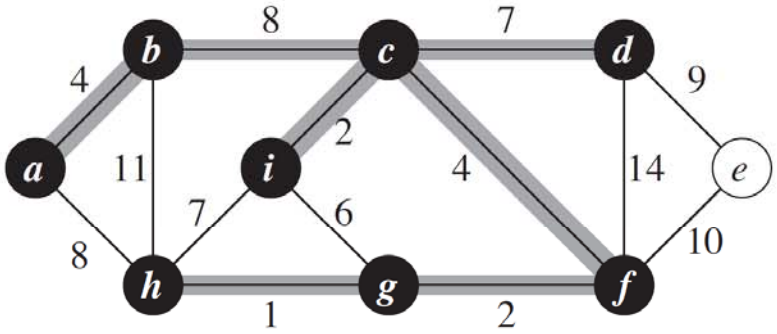
a: root，從 a 開始往外長
長成一個樹



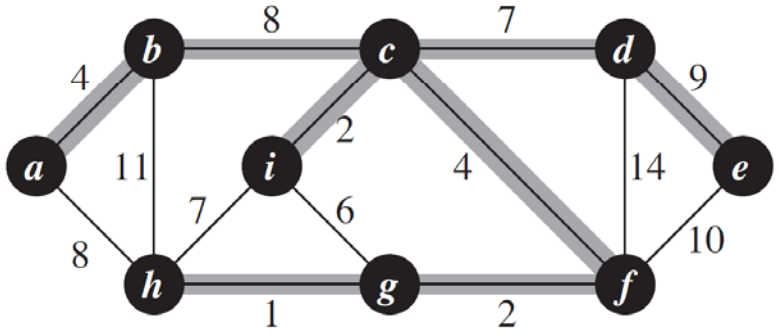
(g)


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

(h)


a
b
c
d
e
f
g
h
i
4
8
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
8
7
9
11
2
4
14
7
6
10
8
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: $O(m + n \lg n)$

找最小，最多作 n 次

更新資訊，最多作 m 次

binary heap: Extract-min, $O(\lg n)$

Decrease-key, $O(\lg n)$

Fibonacci heap: Extract-min, $O(\lg n)$

Decrease-key, $O(1)$