طراحی الگوریتم

۱۹ آبان ۹۸

ملکی مجد

Topic	Reference
Recursion and Backtracking	Ch.1 and Ch.2 JeffE
Dynamic Programming	Ch.3 JeffE and Ch.15 CLRS
Greedy Algorithms	Ch.4 JeffE and Ch.16 CLRS
Amortized Analysis	Ch.17 CLRS
Elementary Graph algorithms	Ch.6 JeffE and Ch.22 CLRS
Minimum Spanning Trees	Ch.7 JeffE and Ch.23 CLRS
Single-Source Shortest Paths	Ch.8 JeffE and Ch.24 CLRS
All-Pairs Shortest Paths	Ch.9 JeffE and Ch.25 CLRS
Maximum Flow	Ch.10 JeffE and Ch.26 CLRS
String Matching	Ch.32 CLRS
NP-Completeness	Ch.12 JeffE and Ch.34 CLRS

- چند تا خط لازم داریم تا n تا نقطه را به هم وصل کنیم؟
- فرض کنید قرار است که n تا پین را با سیم به هم وصل کنیم. کمترین طول مورد نیاز سیم چقدر است؟

- model the wiring problem with
  - a connected, undirected graph $G = (V, E)$,
  - where V is the set of pins
  - E is the set of possible interconnections between pairs of pins
  - for each edge $(u, v) \in E$, we have a weight $w(u, v)$ specifying the cost (amount of wire needed) to connect u and v

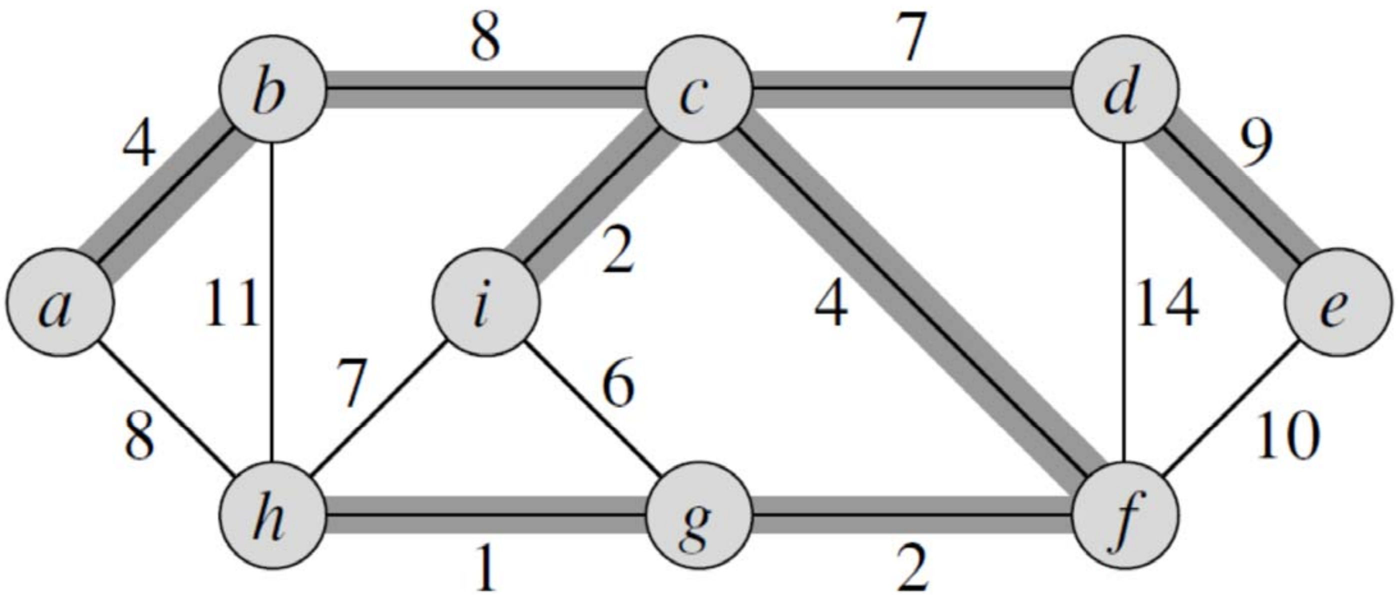
- wish to find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight $w(T)$ is minimized.
  - $w(t) = \sum_{(u,v)\in T} w(u, v)$

minimum-spanning-tree

- Since *T* is acyclic and connects all of the vertices, it must form a tree, which we call a ***spanning tree*** since it "spans" the graph *G*.

- We call the problem of determining the tree *T* the ***minimum-spanning-tree problem***.

example



آیا درخت فراگیر کمینه دیگری وجود دارد؟

two algorithms for solving the MST problem

- Kruskal's algorithm
- Prim's algorithm

- These two algorithms are **greedy** algorithms

- At each step of an algorithm, one of several possible choices must be made (the choice that is the **best at the moment**)
- we can prove that certain greedy strategies do yield a spanning tree with minimum weight.

Time complexity for solving MST

- Kruskal's algorithm
- Prim's algorithm

- using ordinary binary heaps
  - run in time $O(E \lg V)$
- using Fibonacci heaps
  - Prim's algorithm can be sped up to run in time $O(E + V \lg V)$
  - is an improvement if $|V|$ is much smaller than $|E|$

In the following …

- First
  - Learn a **generic** minimum-spanning-tree algorithm that grows a spanning tree by adding one edge at a time

- Then
  - Learn two ways to implement the generic algorithm
  1. Kruskal
  2. Prim

Growing a minimum spanning tree (assumption)

- **Assume** that we have a connected, undirected graph $G = (V, E)$ with a weight function $w : E \rightarrow \mathbf{R}$, and we wish to find a minimum spanning tree for G.

Growing a minimum spanning tree (loop invariant)

- greedy strategy grows the minimum spanning tree **one edge at a time.**

- The algorithm manages a set of edges A, maintaining the following loop invariant:

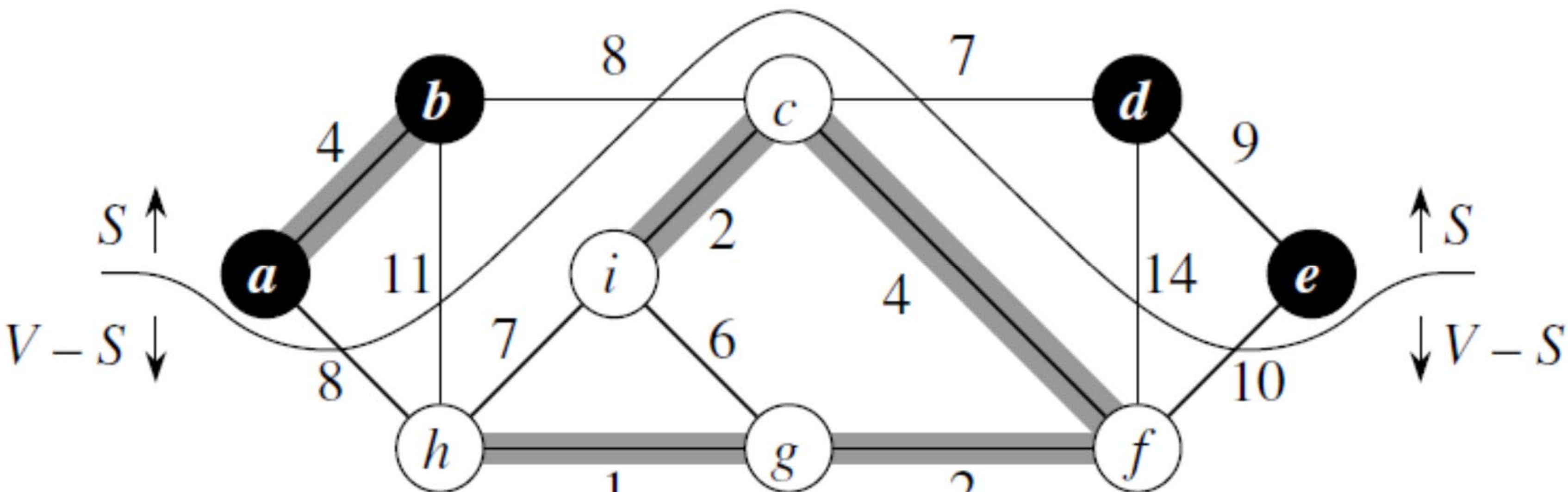
Prior to each iteration, A is a subset of some minimum spanning tree.

Growing a minimum spanning tree (safe edge)

- At each step, we determine an edge (u, v) that can be added to A without violating this invariant, in the sense that $A \cup \{(u, v)\}$ is also a subset of a minimum spanning tree. We call such an edge a ***safe edge*** for A, since it can be safely added to *A* while maintaining the invariant.

Cut and light edge

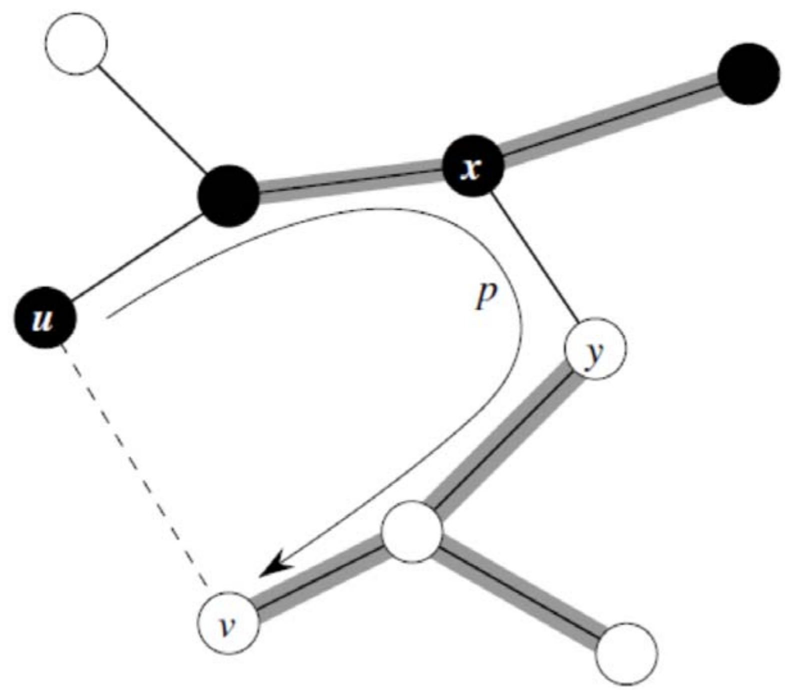
- A ***cut*** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V
- We say that an edge $(u, v) \in E$ ***crosses*** the cut $(S, V - S)$ if one of its endpoints is in S and the other is in $V - S$.
- We say that a cut ***respects*** a set A of edges if no edge in A crosses the cut.
- An edge is a ***light edge*** crossing a cut if its weight is the minimum of any edge crossing the cut



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

Recognizing safe edges (Theorem 23.1)

- Let $G = (V, E)$ be a connected, undirected 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 $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a **light edge** crossing $(S, V - S)$. Then, edge (u, v) is **safe** for …



x
u
p
y
v

GENERIC-MST

GENERIC-MST(G, w)

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

- The loop in lines 2–4 of GENERIC-MST is executed |V| - 1 times as each of the |V|-1 edges of a minimum spanning tree is successively determined.
- Initially, when $A = \emptyset$, there are $|V|$ trees in G_A, and each iteration reduces that number by 1.
- When the forest contains **only a single tree**, the algorithm terminates.

Corollary 23.2

- Let $G = (V, E)$ be a connected, undirected 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, and 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.

Kruskal

- Consider GENERIC-MST

- 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.

- It uses a disjoint-set data structure to maintain several disjoint sets of elements (contains the vertices in a tree).

MST-KRUSKAL

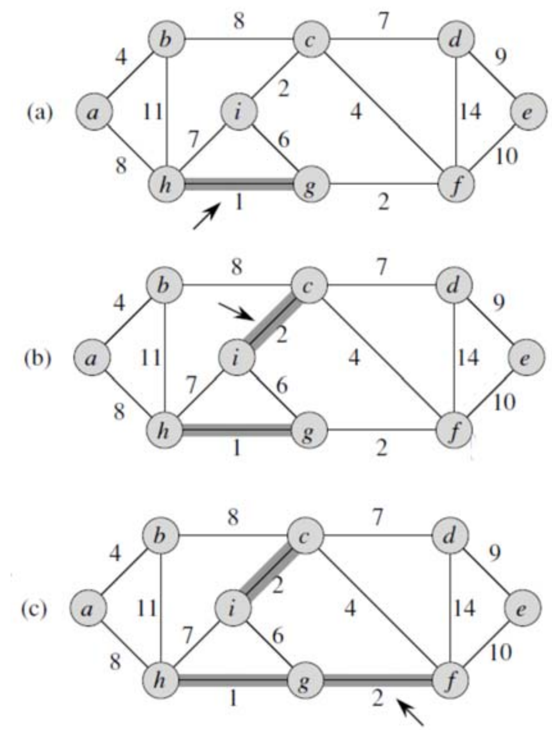
MST-KRUSKAL(G, w)

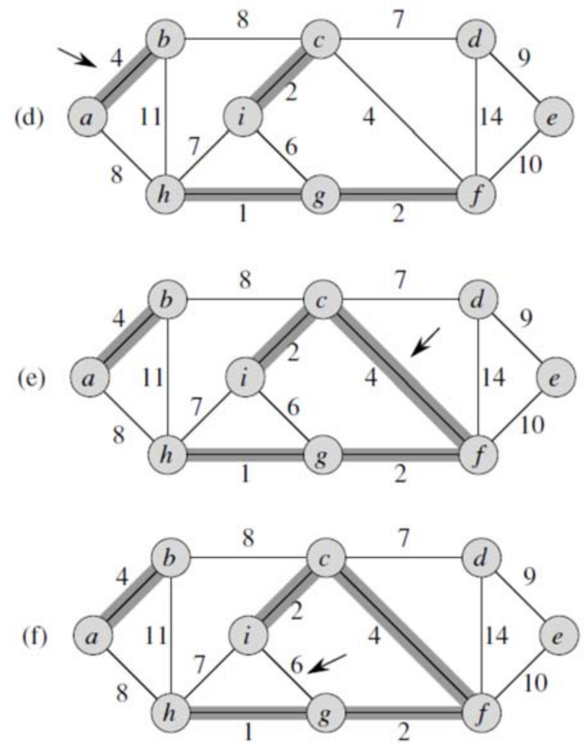
$A \leftarrow \emptyset$
for each vertex $v \in V[G]$
    do $MAKE - SET(v)$
sort the edges of *E* into nondecreasing order by weight w
for each edge $(u, v) \in E$, taken in nondecreasing order by weight
    do if $FIND - SET(u) \neq FIND - SET(v)$
        then $A \leftarrow A \cup \{(u, v)\}$
            $UNION(u, v)$
return A

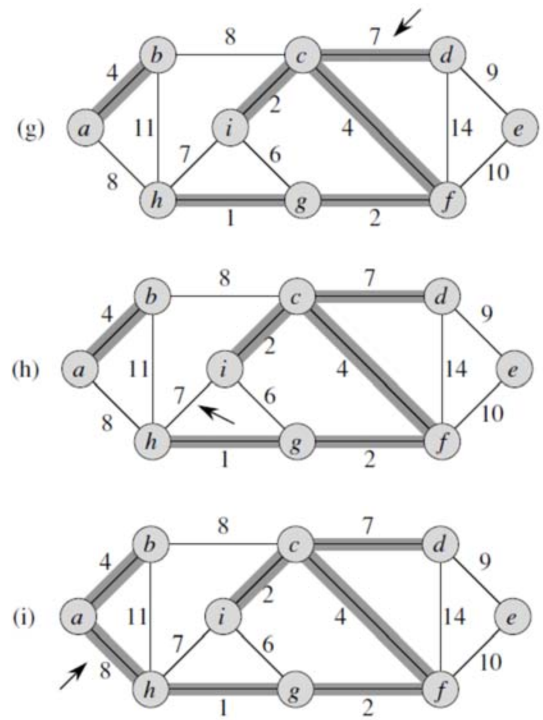
running time of Kruskal

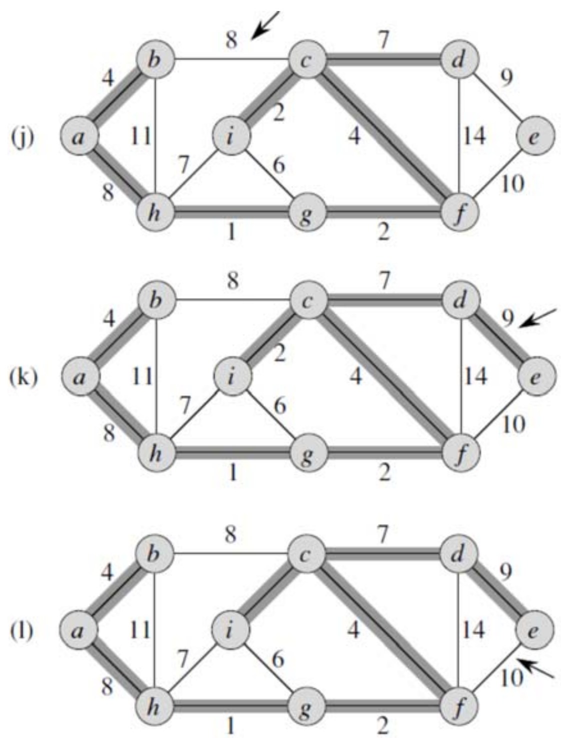
- The running time of Kruskal's algorithm for a graph $G = (V, E)$ depends on the implementation of the disjoint-set data structure.

- We shall assume the disjoint-set-forest implementation of Section 21.3 with the **union-by-rank** and **path-compression heuristics**, since it is the asymptotically fastest implementation known.
  - disjoint-set operations take $O(E\ \alpha(V))$ time
  - since $\alpha(|V|) = O(\lg V) = O(\lg E)$,
  - the running time of Kruskal's algorithm : $O(E \lg V)$.

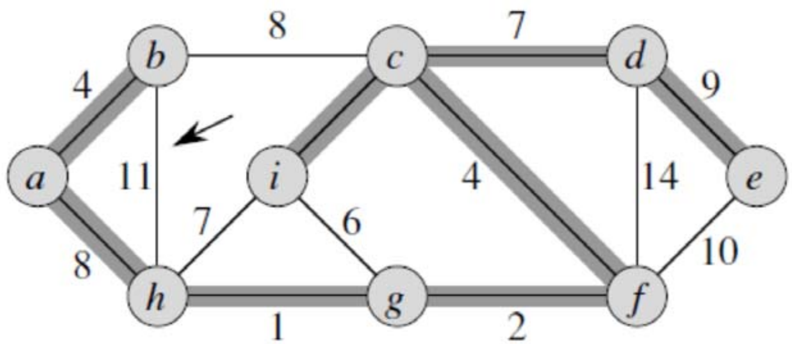

(a)
(b)
(c)


(d)
(e)
(f)


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

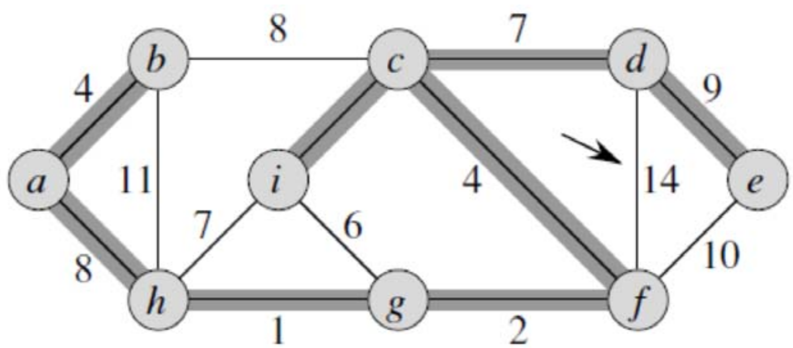

(j)
a b c d e f g h i
4 8 11 7 2 6 1 2 4 7 14 9 10 8
(k)
a b c d e f g h i
4 8 11 7 2 6 1 2 4 7 14 9 10 8
(l)
a b c d e f g h i
4 8 11 7 2 6 1 2 4 7 14 9 10 8

(m)



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

(n)



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

Prim

- Consider GENERIC-MST

- The set A forms a **single tree**
- The tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V
- The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

- The key to implementing Prim's algorithm efficiently is to make it easy to select a new edge to be added to the tree formed by the edges in A.
  - min-priority queue($key[v]$ is the minimum weight of any edge connecting v to a vertex in the tree)

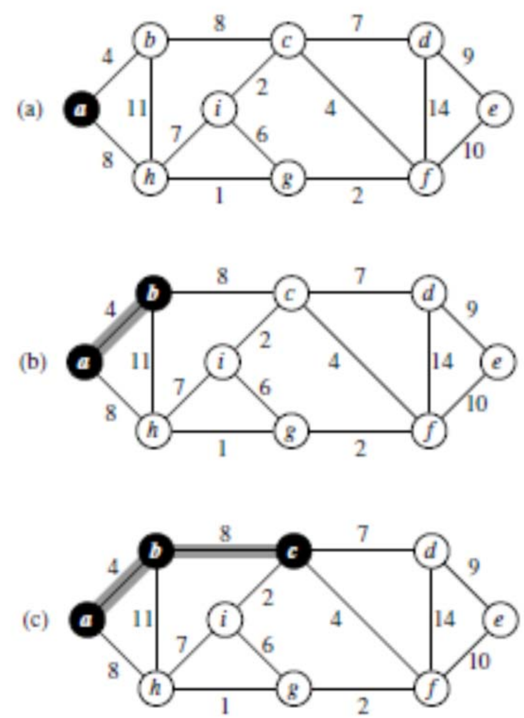
MST-PRIM(G, w, r)

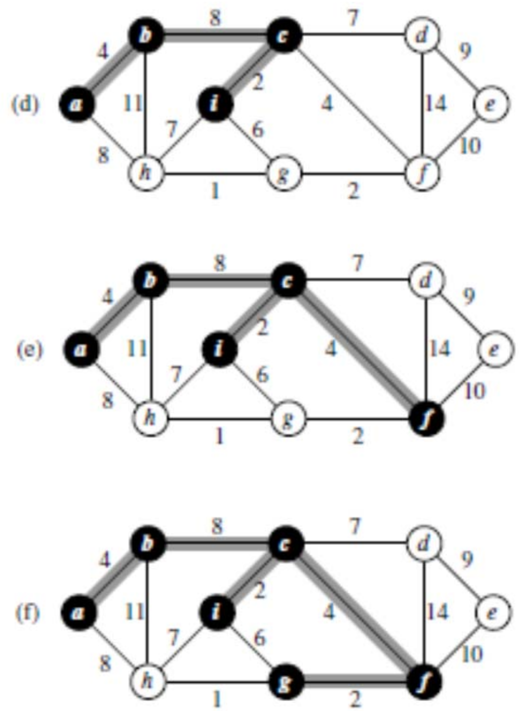
```
for each u ∈ V[G]
   do key[u] ← ∞
      π[u] ← NIL
key[r] ← 0
Q ← V[G]
while Q ≠ ∅
   do u ← EXTRACT − MIN(Q)
      for each v ∈ Adj[u]
         do if v ∈ Q and w(u, v) < key[v]
            then π[v] ← u
                 key[v] ← w(u, v)
```

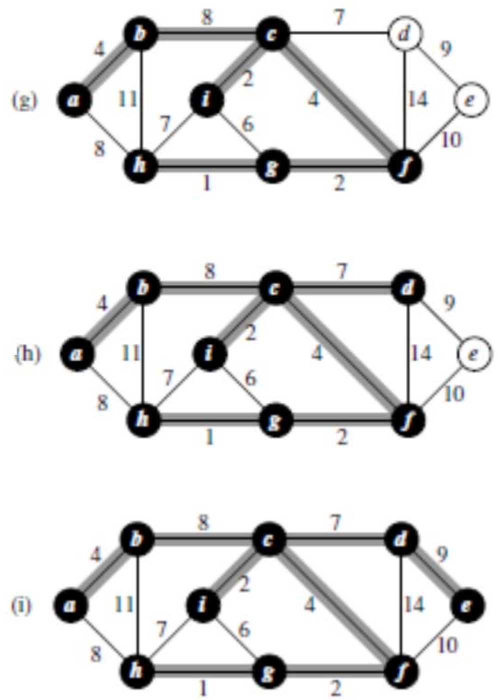
- The performance of Prim's algorithm depends on how we implement the min priority queue Q.

- Binary min-heap
  - $O(V \lg V + E \lg V) = O(E \lg V)$
- Fibonacci heaps
  - $O(E + V \lg V)$
  - Fibonacci heaps use amortized analysis

Example: MSP by Prim




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


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