طراحی الگوریتم ها

جلسه ۱۲ و ۱۳

ملکی مجد

مباحث

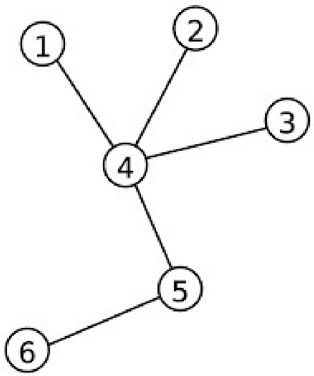
Minimum Spanning Tree

درخت پوشای کمینه

- Kruskal
- Prim

مبحث MST از فصل ۲۳ کتاب CLRS تدریس می شود.

- حداقل چند تا خط لازم داریم تا n تا نقطه را به هم وصل کنیم؟



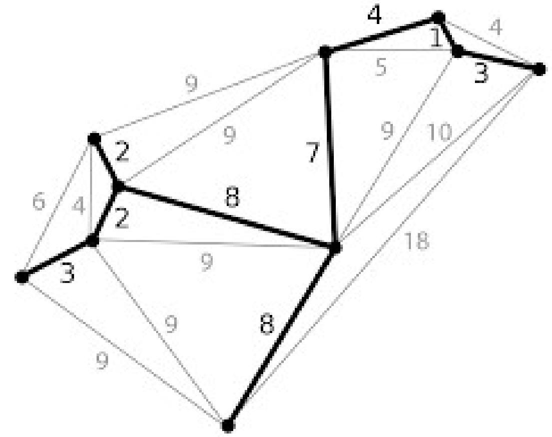
- حداقل چند تا خط لازم داریم تا n تا نقطه را به هم وصل کنیم؟ n-1
- فرض کنید قرار است که n تا پین را با سیم به هم وصل کنیم. کمترین طول مورد نیاز سیم چقدر است؟

- حداقل چند تا خط لازم داریم تا n تا نقطه را به هم وصل کنیم؟ n-1

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

  موضوع درس است

Model the wiring problem



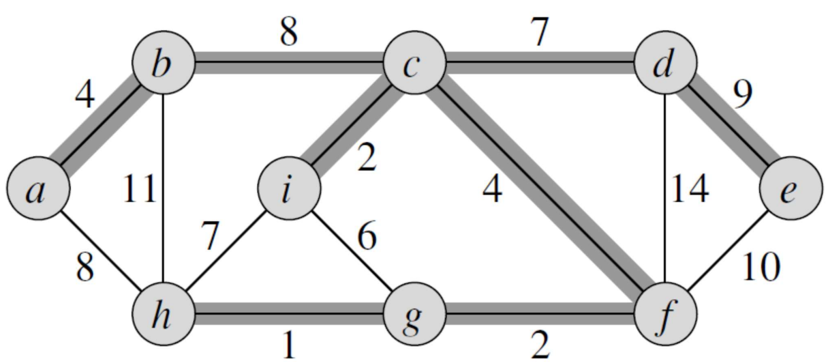
- Model with:
  - a **connected, undirected graph $G = (V, E)$**,
  - where V is the set of pins
  - E is the set of **possible interconnections** between pairs of pins
  - for each edge $(u, v) \in E$, we have a **weight $w(u, v)$ specifying the cost** (amount of wire needed) to connect u and v

- 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 (MST)

- 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 *with minimum weight* the ***minimum-spanning-tree problem***.



Here we focus on

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
  - using ordinary binary heaps : run in time $\boldsymbol{O(E \lg V)}$
  - using Fibonacci heaps: Prim's can be run in time $\boldsymbol{O(E + V \lg V)}$(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

- Second
  - Learn two ways to implement this generic algorithm
  1. Kruskal
  2. Prim

Growing a minimum spanning tree

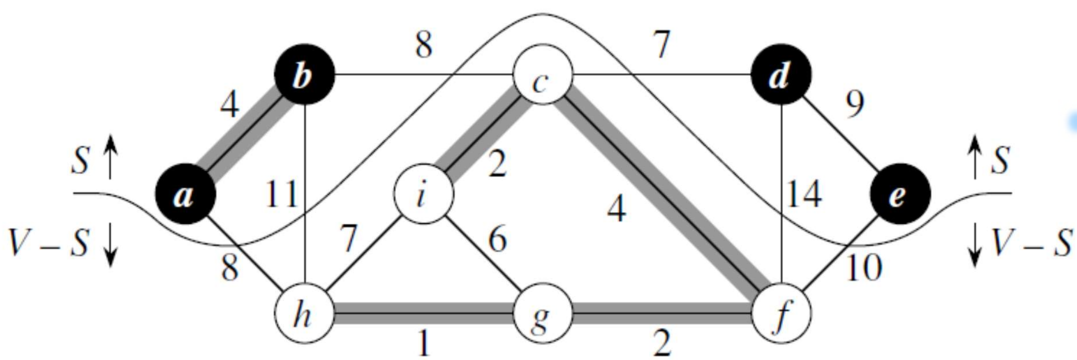
- Assumption
  - **Assume** that we have a connected, undirected graph *G = (V, E)* with a weight function *w* : *E* → **R**, and we wish to find a minimum spanning tree for *G*.

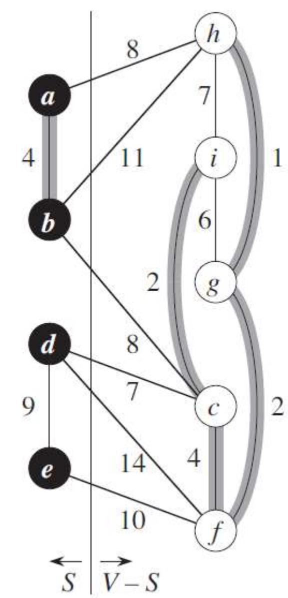
Greedy strategy grows the minimum spanning tree **one edge at a time.** Algorithm **manages a set of edges** A

- loop invariant
  - Prior to each iteration, A is a **subset of some minimum spanning tree**.
- safe edge
  - At each step, we determine an edge (u, v) (***safe edge***) 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.

CUT

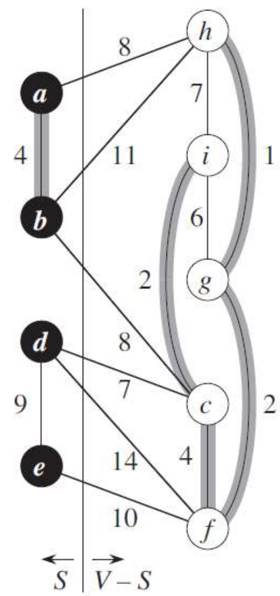
- A ***cut*** $(\boldsymbol{S}, \boldsymbol{V} - \boldsymbol{S})$ of an undirected graph $G = (V, E)$ is a **partition of** $\boldsymbol{V}$
  - We say that an edge $(u, v) \in E$ ***crosses*** the cut $(S, V - S)$ if one of its endpoints is in S and the other is in $V - S$.
  - We say that a cut ***respects***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





Light edge

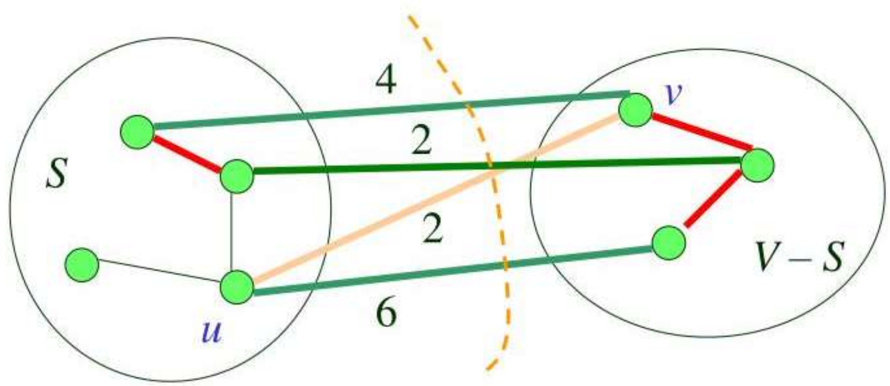
- An edge is a ***light edge*** crossing a cut if its weight is the minimum of any edge crossing the cut



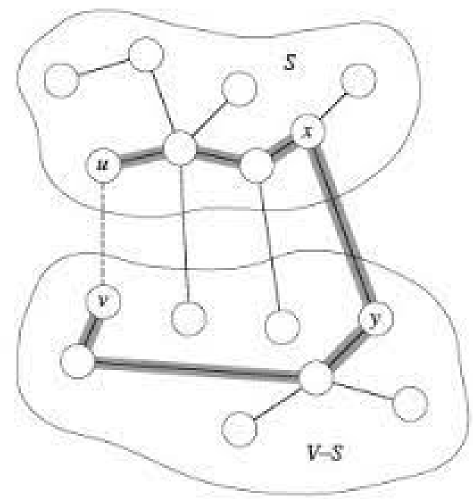
Recognizing safe edges (Theorem 23.1)

- 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 edge** for A



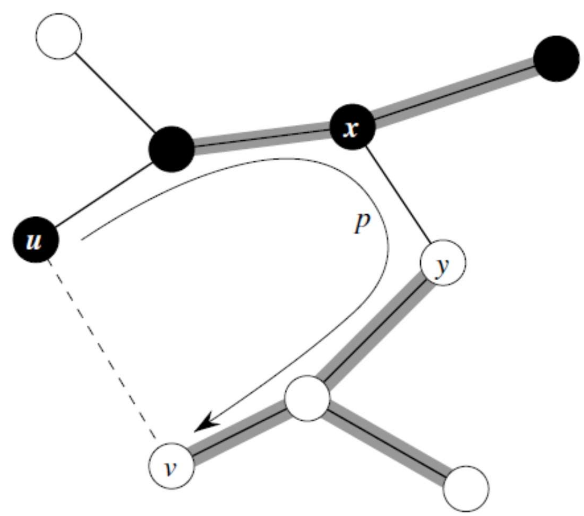
Recognizing safe edges (Theorem 23



- 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 $(S, V - S)$.

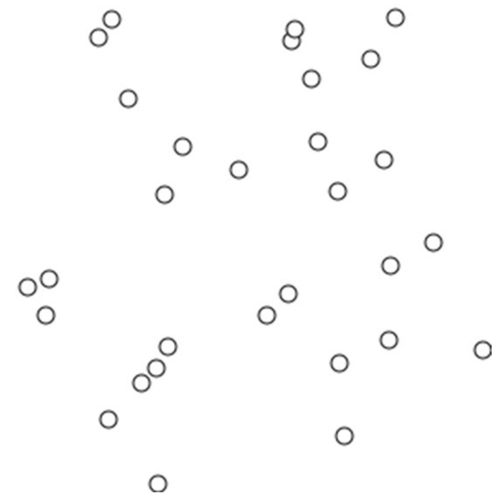
Then, edge (u, v) is **safe edge** for A

Proof idea
Edges of MST T are shown!
Highlighted edges are in A
(x,y) is an edge that contradicts (u,v) not a light edge!



GENERIC-MST

(Note it is generic)



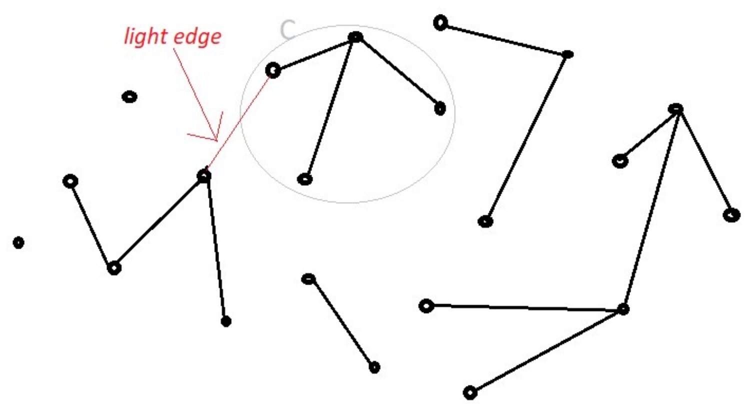
GENERIC-MST(G, w)

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
- 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.



Think together ☺

23.1-1
Let (u, v) be a minimum-weight edge in a connected graph G. Show that (u, v) belongs to some minimum spanning tree of G.

Kruskal

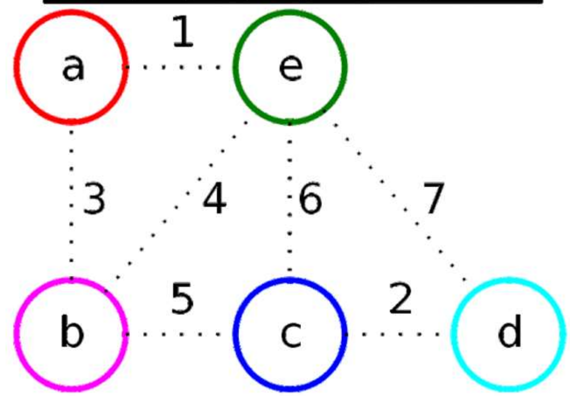
- Consider GENERIC-MST

- 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

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



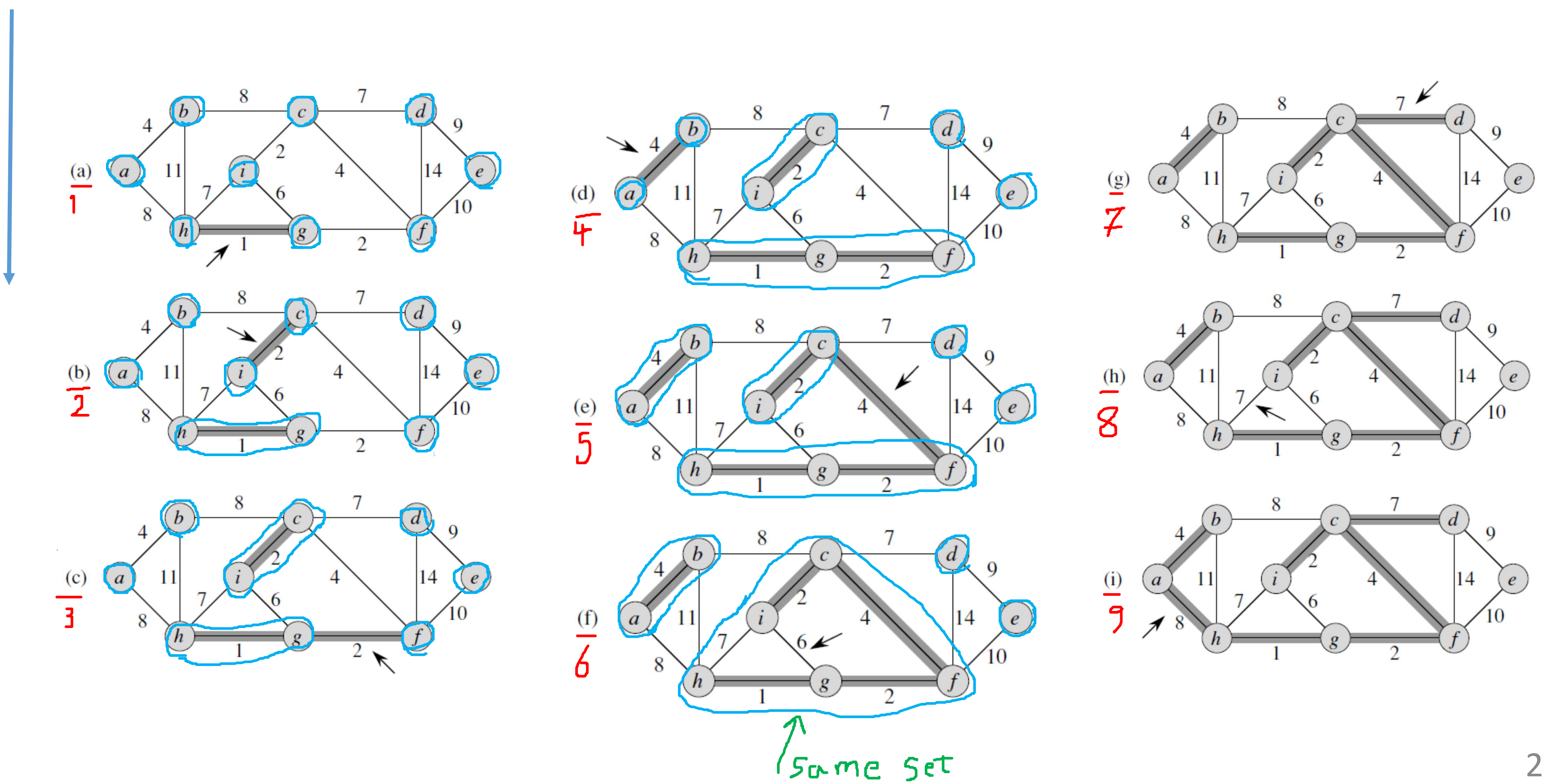
MST-KRUSKAL(G, w)

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

Running time of Kruskal

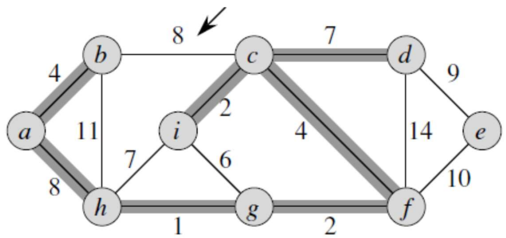
- The running time of Kruskal's algorithm for a graph $G = (V, E)$ depends on the implementation of the disjoint-set data structure.

- 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 : $\boldsymbol{O(E\ lg\ V)}$**.**

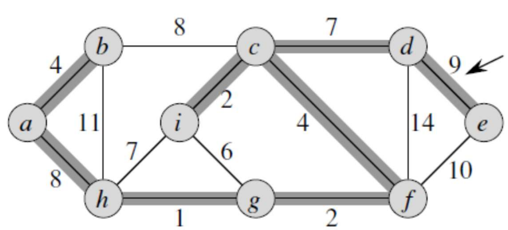


same set

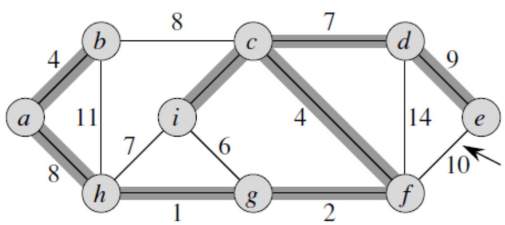
(j)
10



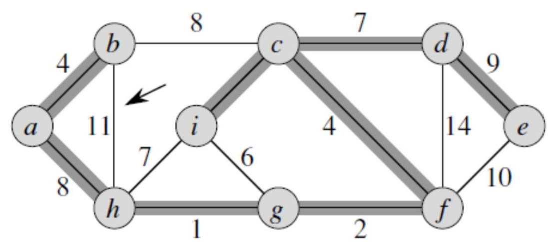
(k)
11



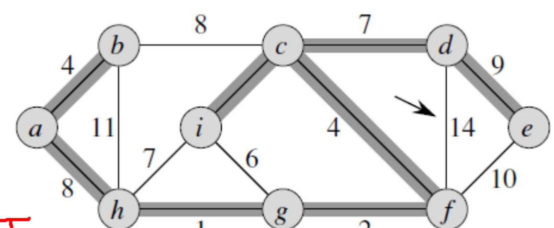
(l)
12



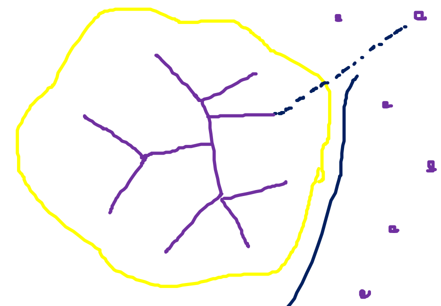
(m)
13



(n)
14
MST



Prim



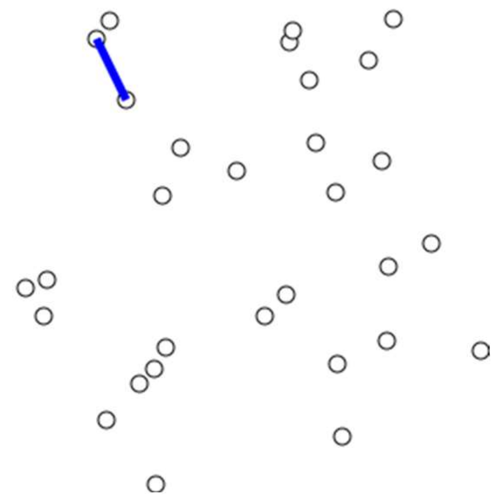
- Consider GENERIC-MST

- 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 l**east-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 $u \in V[G]$
    do $key[u] \leftarrow \infty$
       $\pi[u] \leftarrow NIL$
$key[r] \leftarrow 0$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
    do $u \leftarrow EXTRACT-MIN(Q)$
        for each $v \in Adj[u]$
            do if $v \in Q$ and $w(u, v) < key[v]$
                then $\pi[v] \leftarrow u$
                    $key[v] \leftarrow w(u, v)$



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)
```

When the algorithm terminates,

the min-priority queue Q is empty;

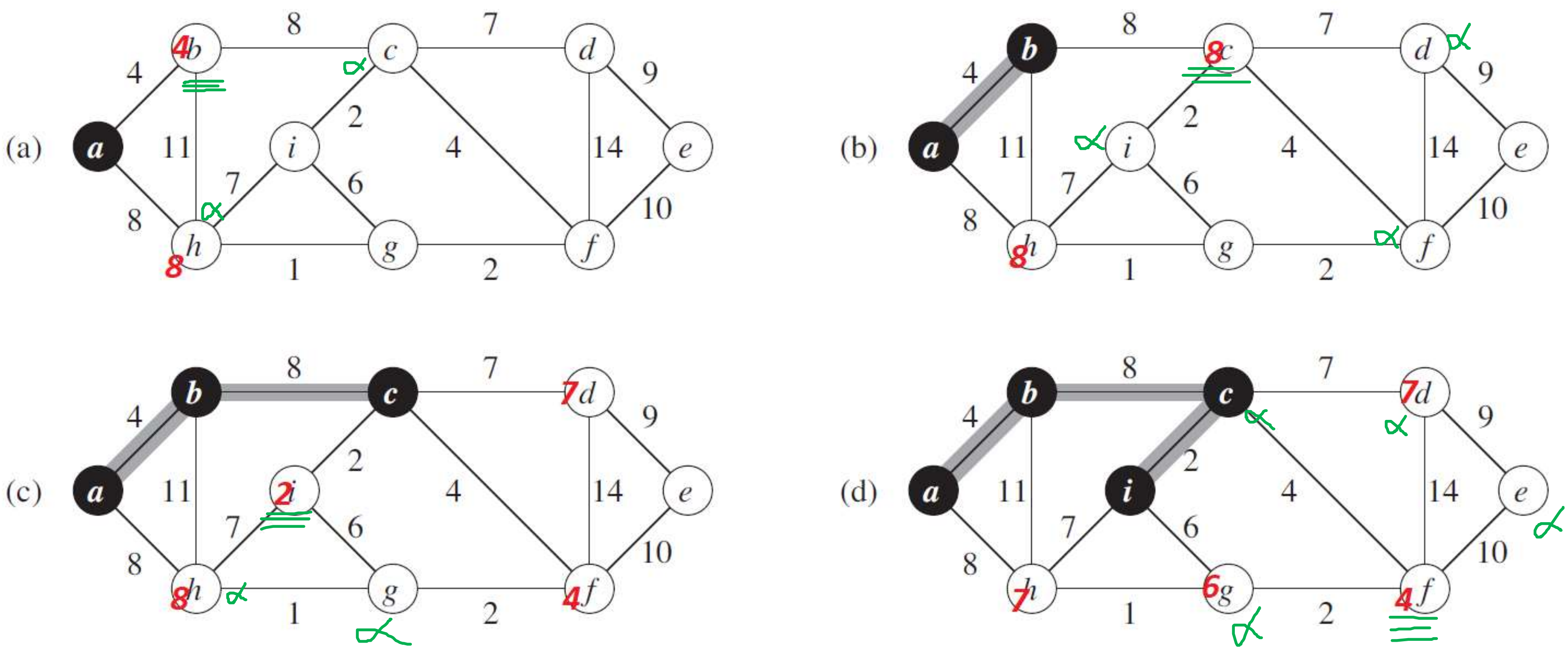
the minimum spanning tree A for G is thus

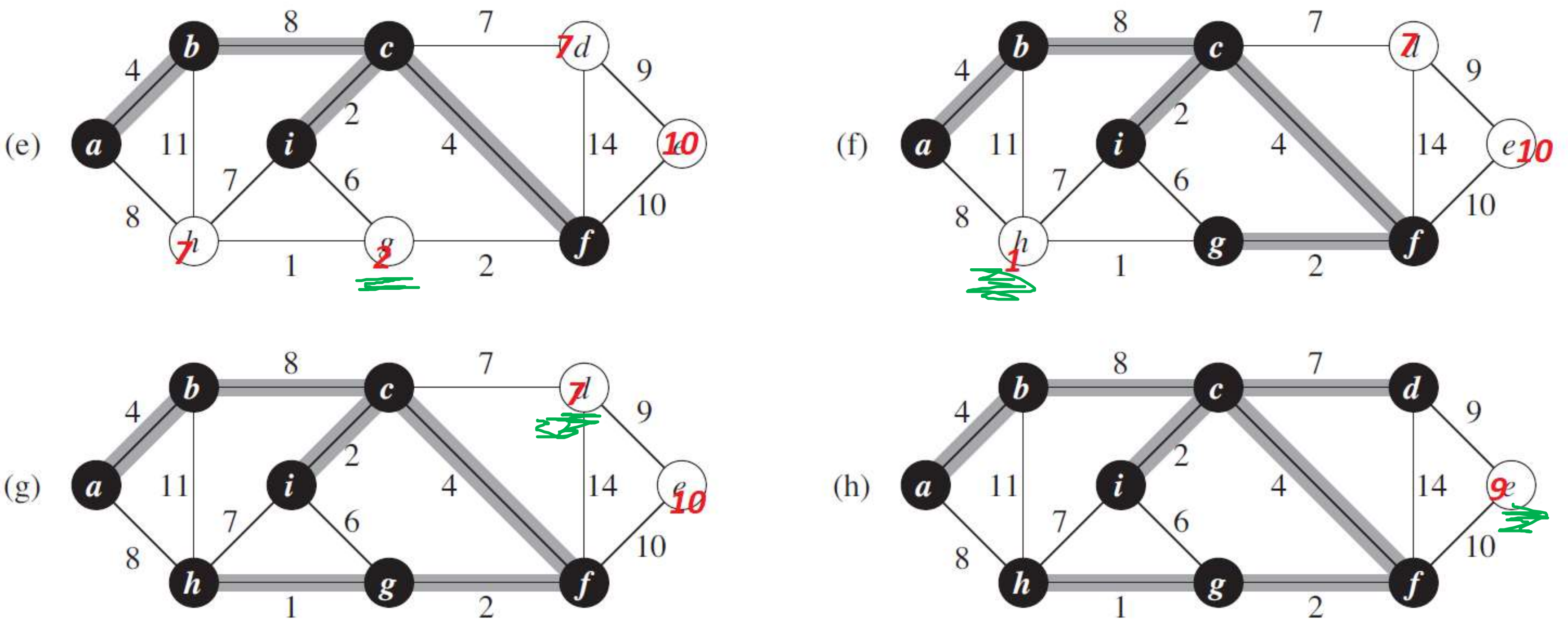
$$A = \{(v, \pi(v)) : v \in V - \{r\}\}$$

- Examine the time complexity of Prim when the min-priority-queue is implemented by an ordinary array?

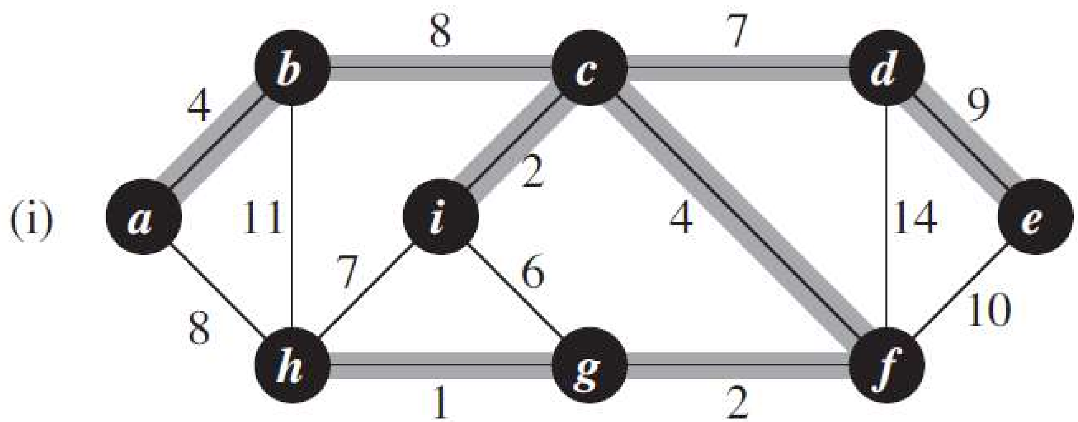
- Examine the time complexity of Prim when the min-priority-queue is implemented by an ordinary array?
    - $O(V * V + E)$

Example: MST by Prim




(e)
(f)
(g)
(h)

Resulted MST:
(Not unique!)



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

- Binary min-heap (at first min heap can be build in $O(V)$)
  - Each extract min and decrease key take $O(\lg V)$ time
  - $O(V \lg V + E \lg V) = O(E \lg V)$
- Fibonacci heaps
  - Each extract min takes O(lg V) amortized time, and each decrease key takes $O(1)$ amortized time
  - $O(V \lg V + E)$
  - Fibonacci heaps use amortized analysis

23.2-8

Professor Borden proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

Think together

23.2-4

Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Kruskal's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

23.2-5

Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Prim's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?