Design and Analysis of Algorithms

CAIRO UNIVERSITY

Lecture 07: Minimum Spanning Trees

Ahmed Hamdy

Agenda

- Disjoint sets

- Minimum Spanning Trees
  - Definition
  - Kruskal’s algorithm
  - Prim’s algorithm

Disjoint sets

- Let $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$ be a collection of disjoint dynamic sets.
- Each set is identified by a ***representative*** (set member).
- In some applications, it doesn't matter how to select the representative.
- Main application is to identify whether two members belong to the same set, or to two different sets.

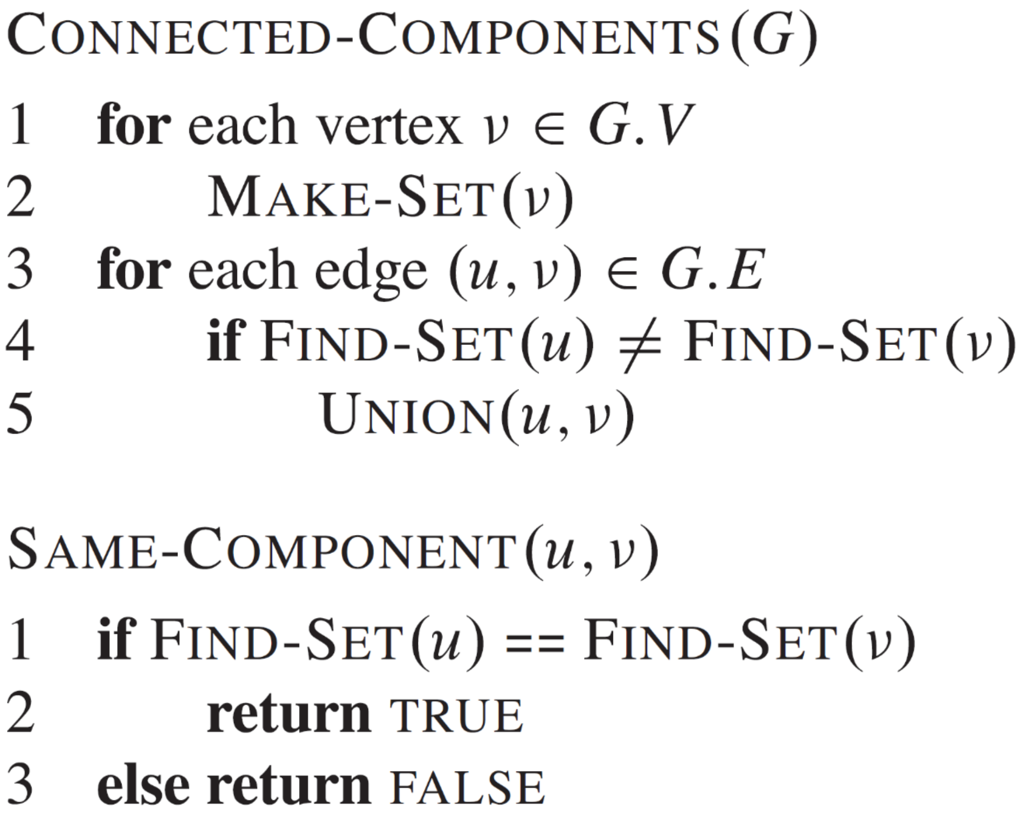
Disjoint sets

Operations:

- **MAKE-SET($\boldsymbol{x}$):** creates a new set whose only member (and thus representative) is x.
- **UNION($\boldsymbol{x}, \boldsymbol{y}$):** unites the dynamic sets that contain x and y. The representative can be any member in $S_x \cup S_y$.
- **FIND-SET($\boldsymbol{x}$):** returns a pointer to the representative of the set containing x.

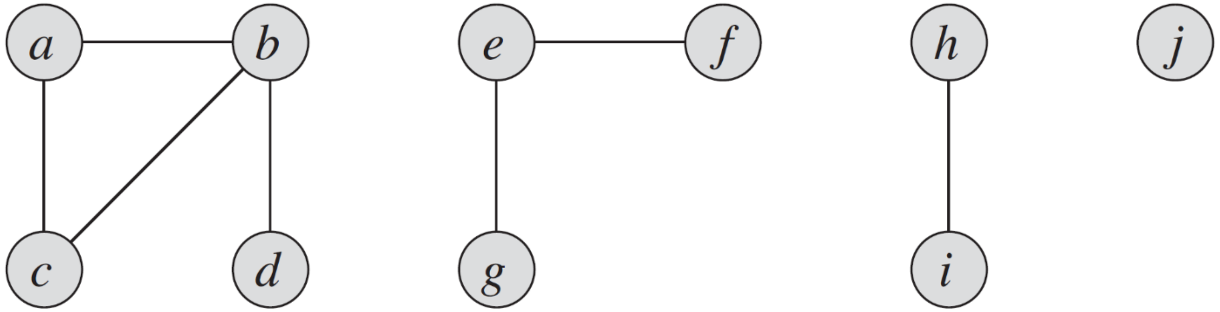
Disjoint sets: Application

- Given a graph, find the connected components



```
CONNECTED-COMPONENTS(G)
for each vertex $v \in G.V$
    MAKE-SET(v)
for each edge $(u, v) \in G.E$
    if FIND-SET(u) $\neq$ FIND-SET(v)
        UNION(u, v)
```

```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```

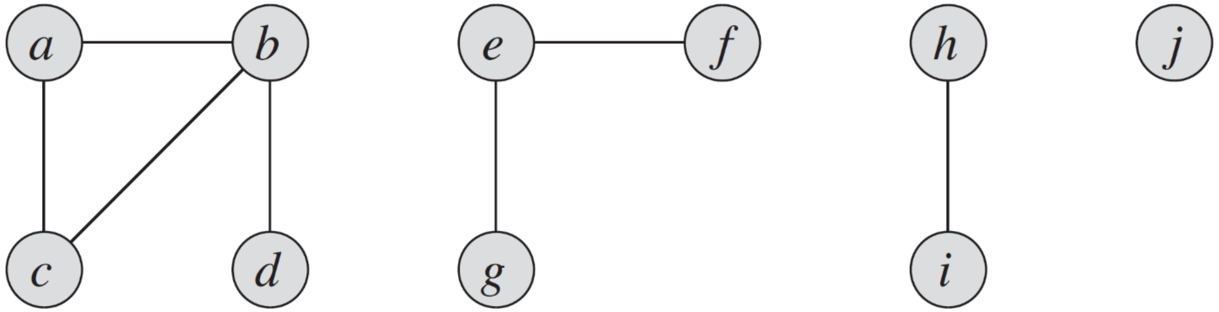


Disjoint sets: Application

- Given a graph, find the connected components

```
CONNECTED-COMPONENTS(G)
for each vertex $v \in G.V$
    MAKE-SET($v$)
for each edge $(u, v) \in G.E$
    if FIND-SET($u$) $\neq$ FIND-SET($v$)
        UNION($u, v$)
```

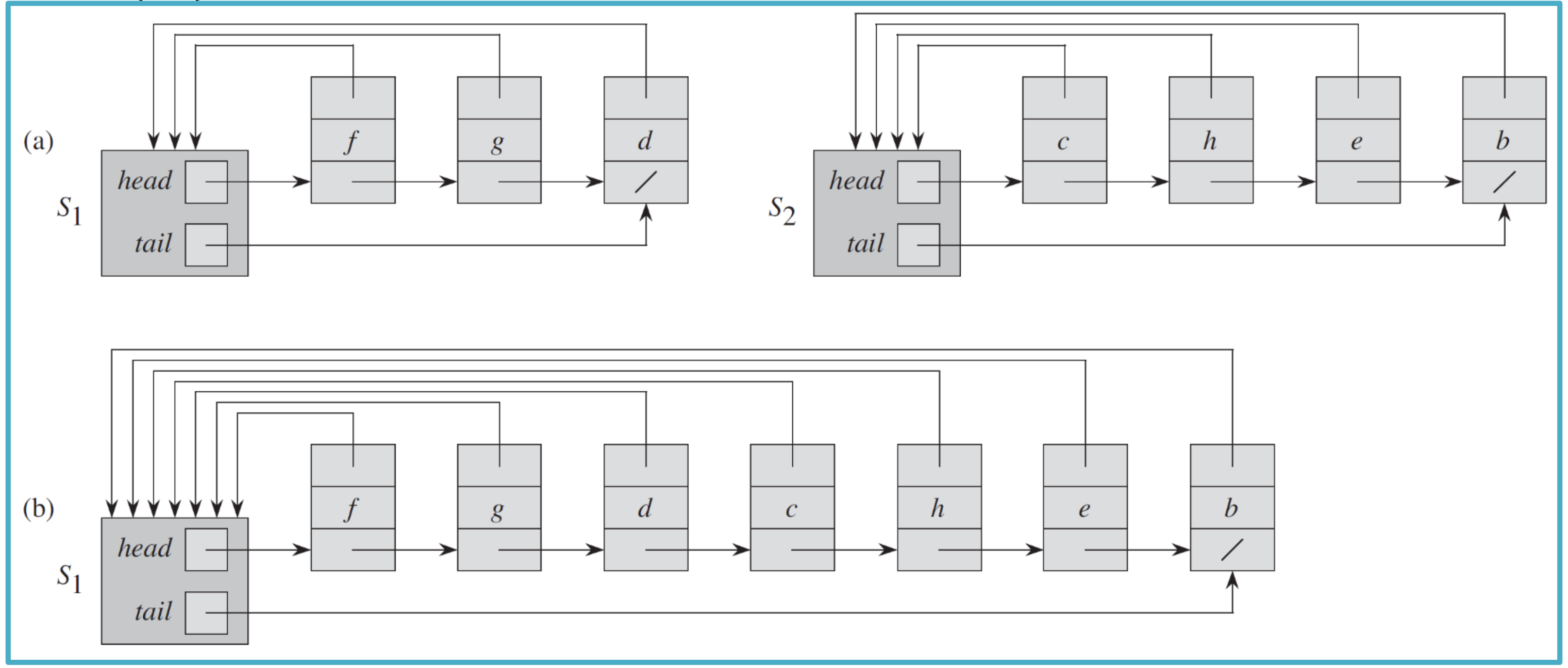
```
SAME-COMPONENT(u, v)
if FIND-SET($u$) == FIND-SET($v$)
    return TRUE
else return FALSE
```



Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

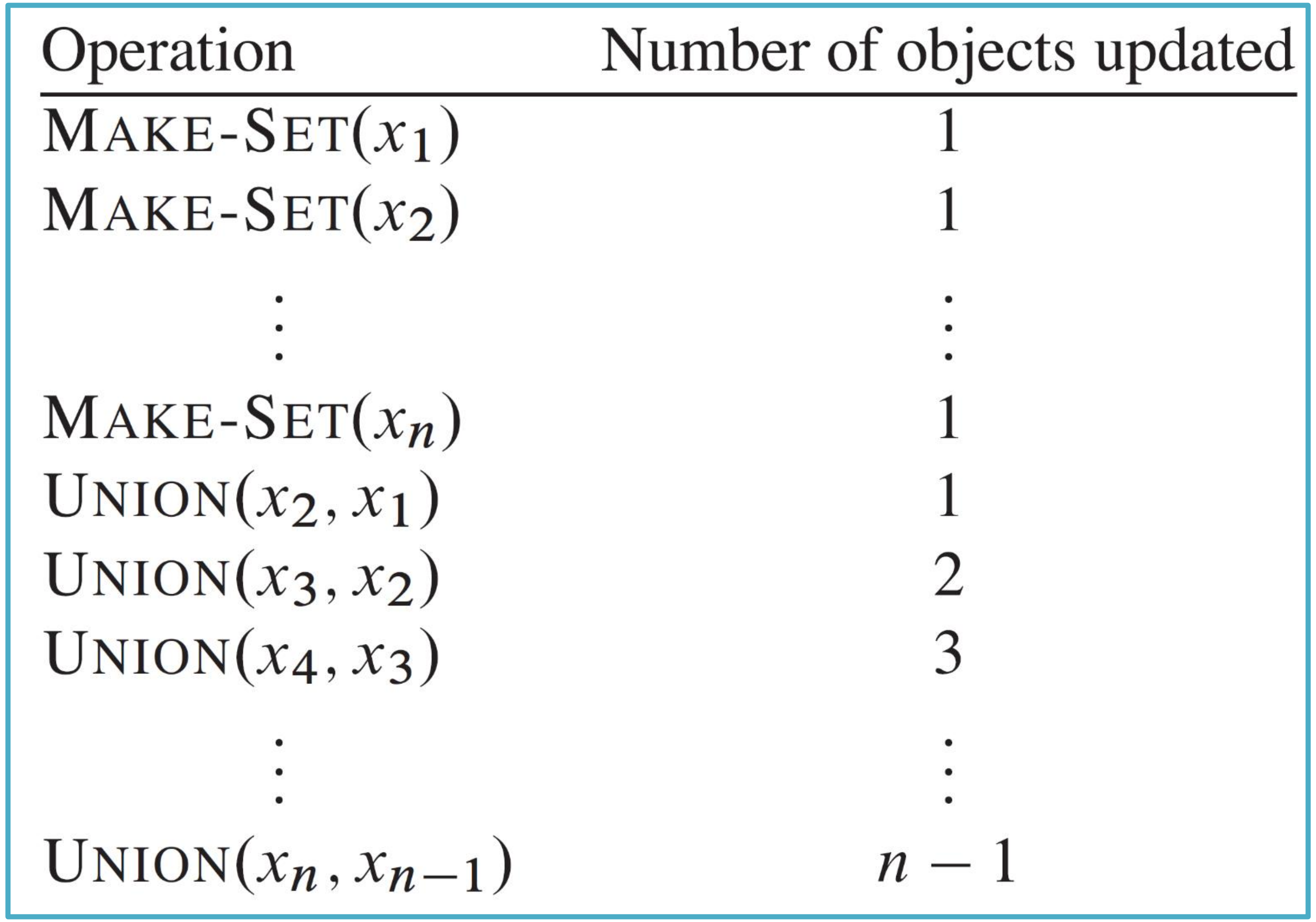
Disjoint sets: Linked lists representation

- **MAKE-SET(x):** makes a new LL, thus $O(1)$.
- **FIND-SET(x):** follow the pointer from x to the set object and returns the pointer that the head node points to as the representative, thus $O(1)$.
- **UNION(x, y):** joining the two LLs is easy but updating pointers back to the set object is expensive, $O(n)$.
  - $O(n)$
  - If n nodes in a graph are connected as one components, then n-1 calls to union are needed → $O(n^2)$



Disjoint sets: Linked lists representation

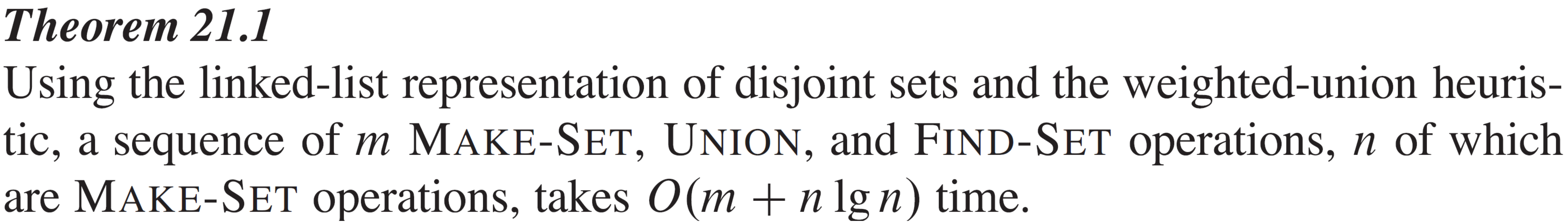
- If n nodes in a graph are connected as one components, then $n-1$ calls to union are needed, thus $\Theta(n^2)$.



Operation	Number of objects updated
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
⋮	⋮
MAKE-SET(x_n)	1
UNION(x_2, x_1)	1
UNION(x_3, x_2)	2
UNION(x_4, x_3)	3
⋮	⋮
UNION(x_n, x_{n-1})	$n-1$

Disjoint sets: Linked lists representation

- Weighted-union heuristic: simply append the shorter list to the longer one.



Theorem 21.1

Using the linked-list representation of disjoint sets and the weighted-union heuristic, a sequence of m MAKE-SET, UNION, and FIND-SET operations, n of which are MAKE-SET operations, takes $O(m + n \lg n)$ time.

Disjoint sets: Disjoint-set forests

- MAKE-SET(x): makes a new tree.
- FIND-SET(x): follow the tree node up to the root.
- UNION(x, y): cause one root to point to the other root.
- Two heuristics:
  - Union by rank
    - Similar to weight-union
  - Path compression

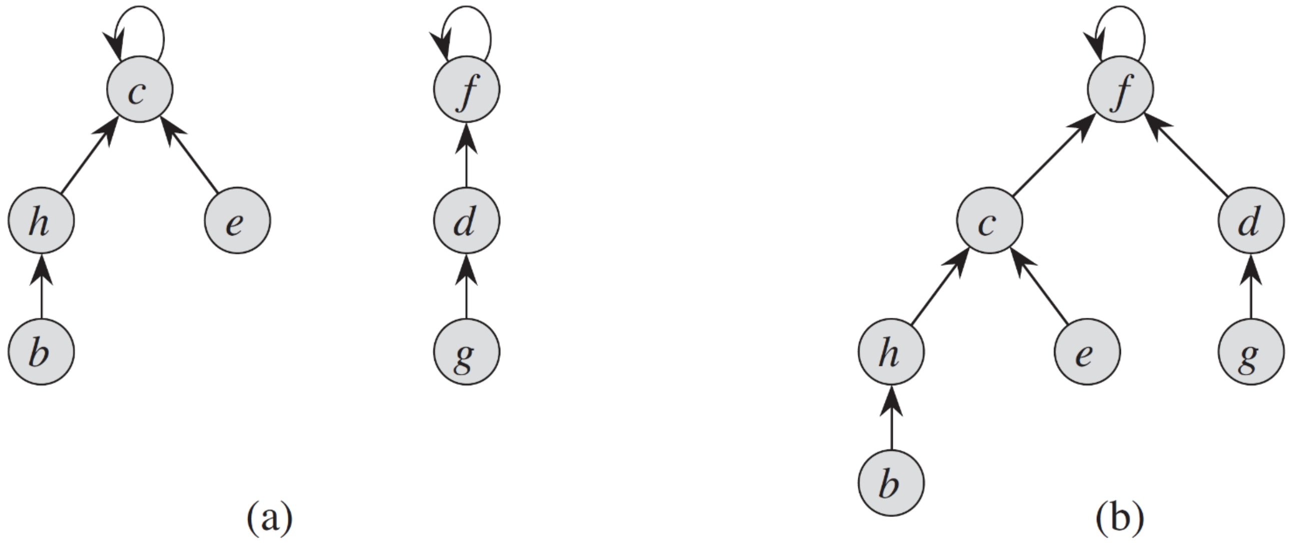


Figure 21.4 A disjoint-set forest. **(a)** Two trees representing the two sets of Figure 21.2. The tree on the left represents the set $\{b, c, e, h\}$, with c as the representative, and the tree on the right represents the set $\{d, f, g\}$, with f as the representative. **(b)** The result of UNION(e, g).

Disjoint sets: Disjoint-set forests

Path compression

```
MAKE-SET($x$)
    $x.p = x$
    $x.rank = 0$

UNION($x, y$)
    LINK(FIND-SET($x$), FIND-SET($y$))

LINK($x, y$)
    if $x.rank > y.rank$
        $y.p = x$
    else $x.p = y$
        if $x.rank == y.rank$
            $y.rank = y.rank + 1$
```

```
FIND-SET($x$)
    if $x \neq x.p$
        $x.p = $ FIND-SET($x.p$)
    return $x.p$
```

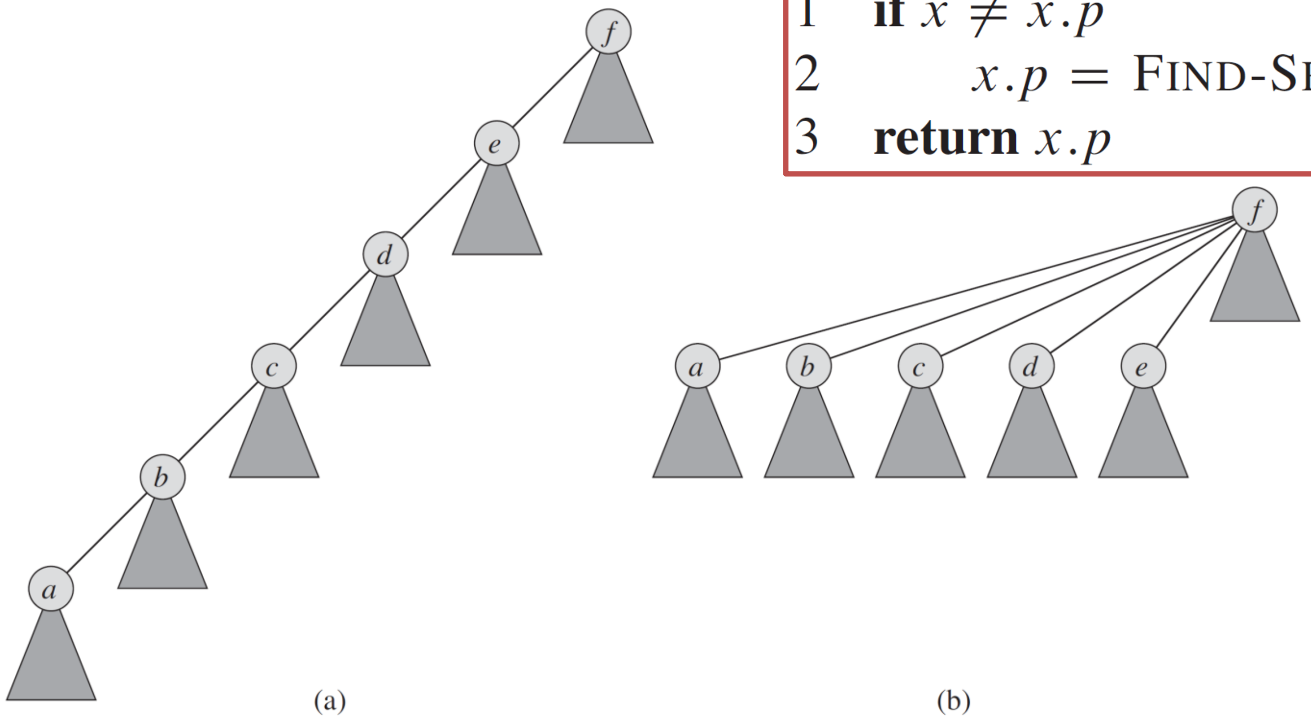


Figure 21.5 Path compression during the operation FIND-SET. Arrows and self-loops at roots are omitted. **(a)** A tree representing a set prior to executing FIND-SET(a). Triangles represent subtrees whose roots are the nodes shown. Each node has a pointer to its parent. **(b)** The same set after executing FIND-SET(a). Each node on the find path now points directly to the root.

Disjoint sets: Disjoint-set forests

- Union by rank yields $O(m \lg n)$.

- With union by rank + path compression, the worst case running time is $O(m\,\alpha(n))$.

- $\alpha(n)$ is a very slowly growing function.

Minimum Spanning Trees (MST)

- Problem common in many applications
- Given distances between cities, choose which roads to construct in order for all cities to be reachable with minimum construction cost.

	Alexandria	Cairo	Matrouh	Aswan	Assiut	Hurghada
Alexandria	0	220	320	1,080	580	680
Cairo	220	0	450	860	360	450
Matrouh	320	450	0	1,300	800	900
Aswan	1,080	860	1,300	0	500	400
Assiut	580	360	800	500	0	300
Hurghada	680	450	900	400	300	0

Definition

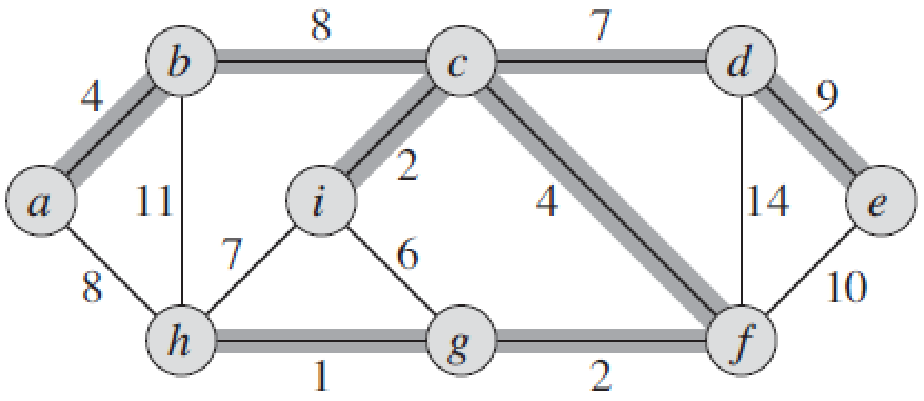


Figure 23.1 A minimum spanning tree for a connected graph. The weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree shown is 37. This minimum spanning tree is not unique: removing the edge (b, c) and replacing it with the edge (a, h) yields another spanning tree with weight 37.

- What is the use of this?!!
  - – In electronic circuit design, we need to wire the electric components together

Definition

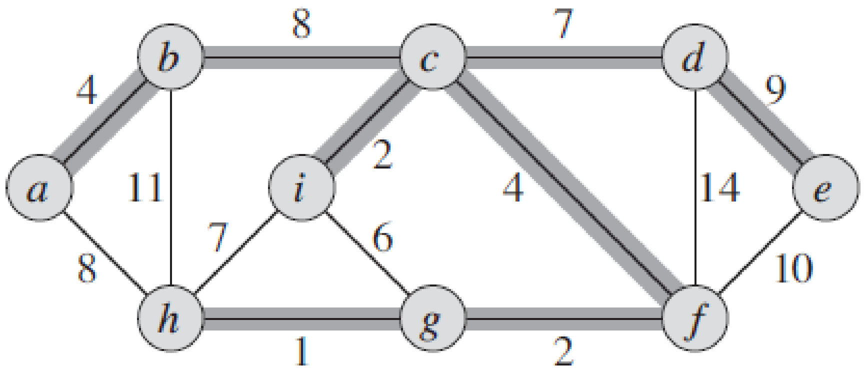
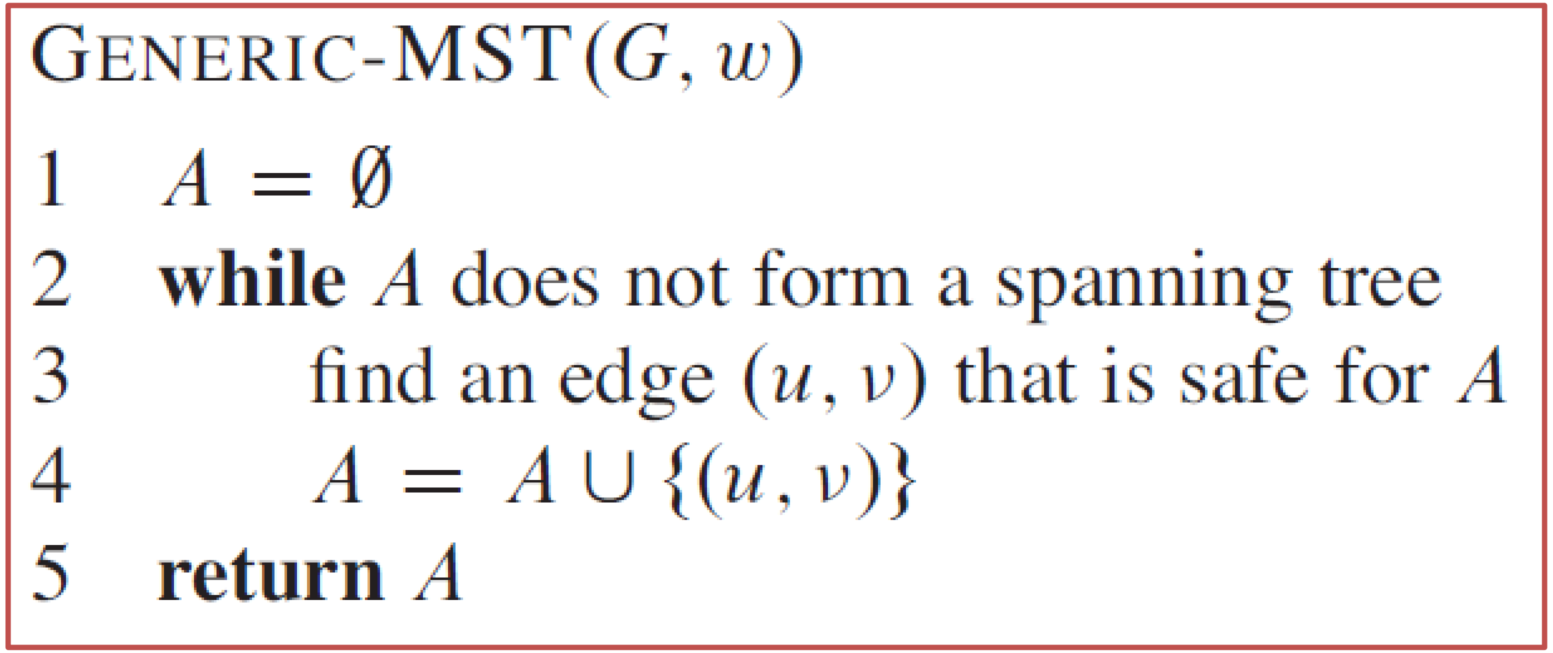


Figure 23.1 A minimum spanning tree for a connected graph. The weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree shown is 37. This minimum spanning tree is not unique: removing the edge (b, c) and replacing it with the edge (a, h) yields another spanning tree with weight 37.

- How to write it as a definition for the problem?
  – Find an acyclic subset $T \subseteq E$ that connects all the vertices with minimum $w(T) = \sum_{(u,v)\in T} w(u, v)$

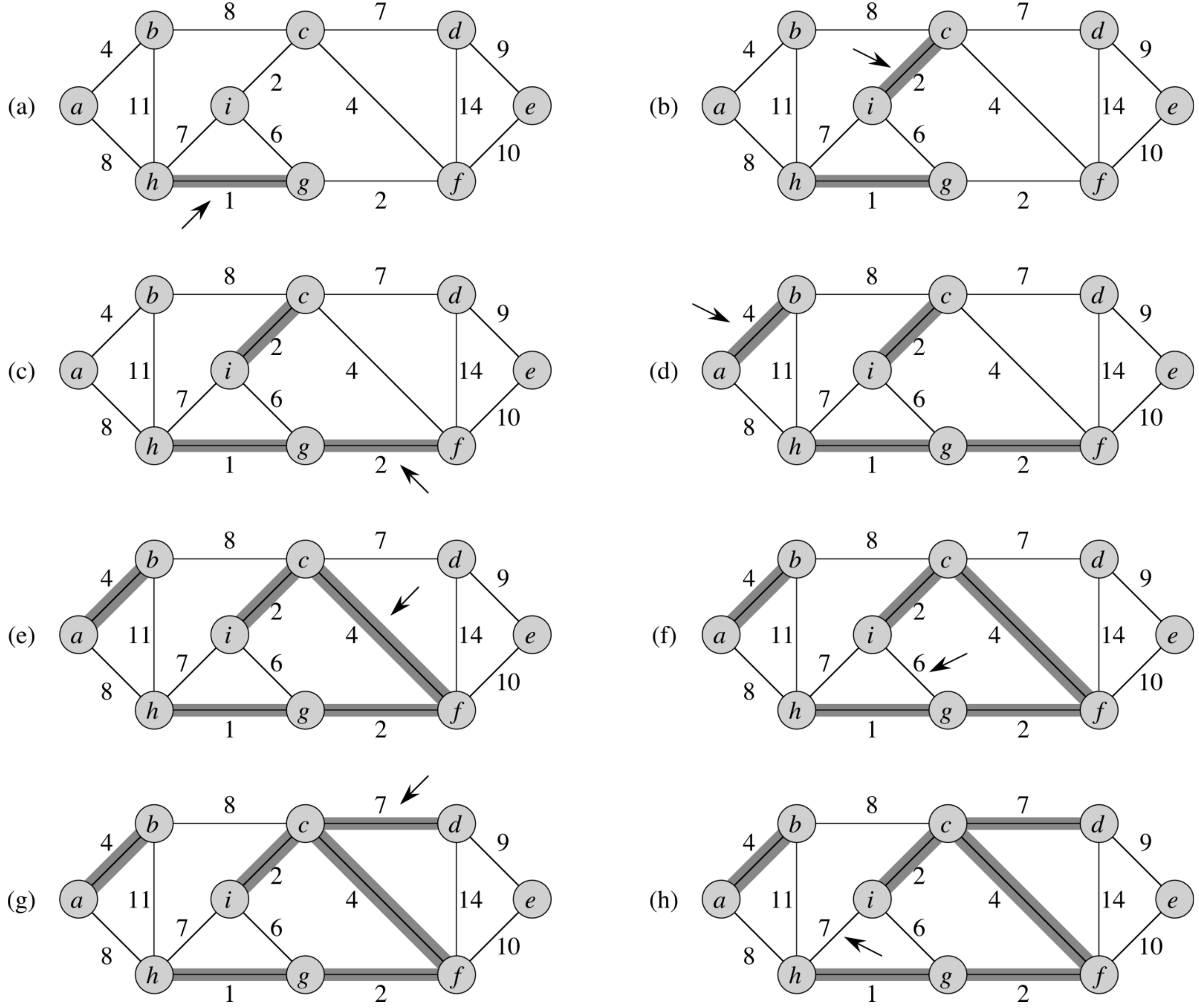
Main concept



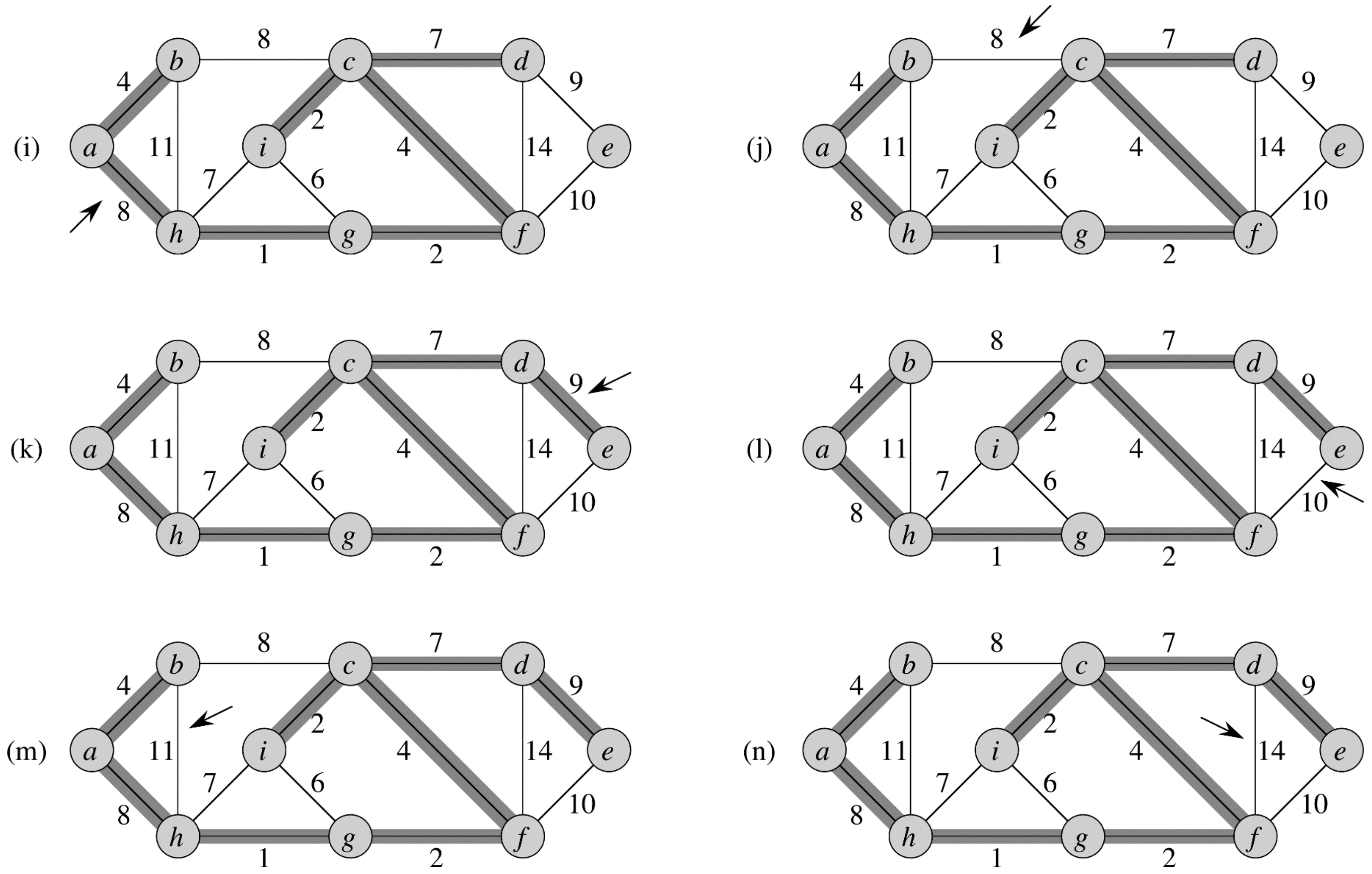
```
GENERIC-MST$(G, w)$
$A = \emptyset$
while $A$ does not form a spanning tree
    find an edge $(u, v)$ that is safe for $A$
    $A = A \cup \{(u, v)\}$
return $A$
```

- Follows which approach??
  - – Greedy approach

Kruskal’s algorithm



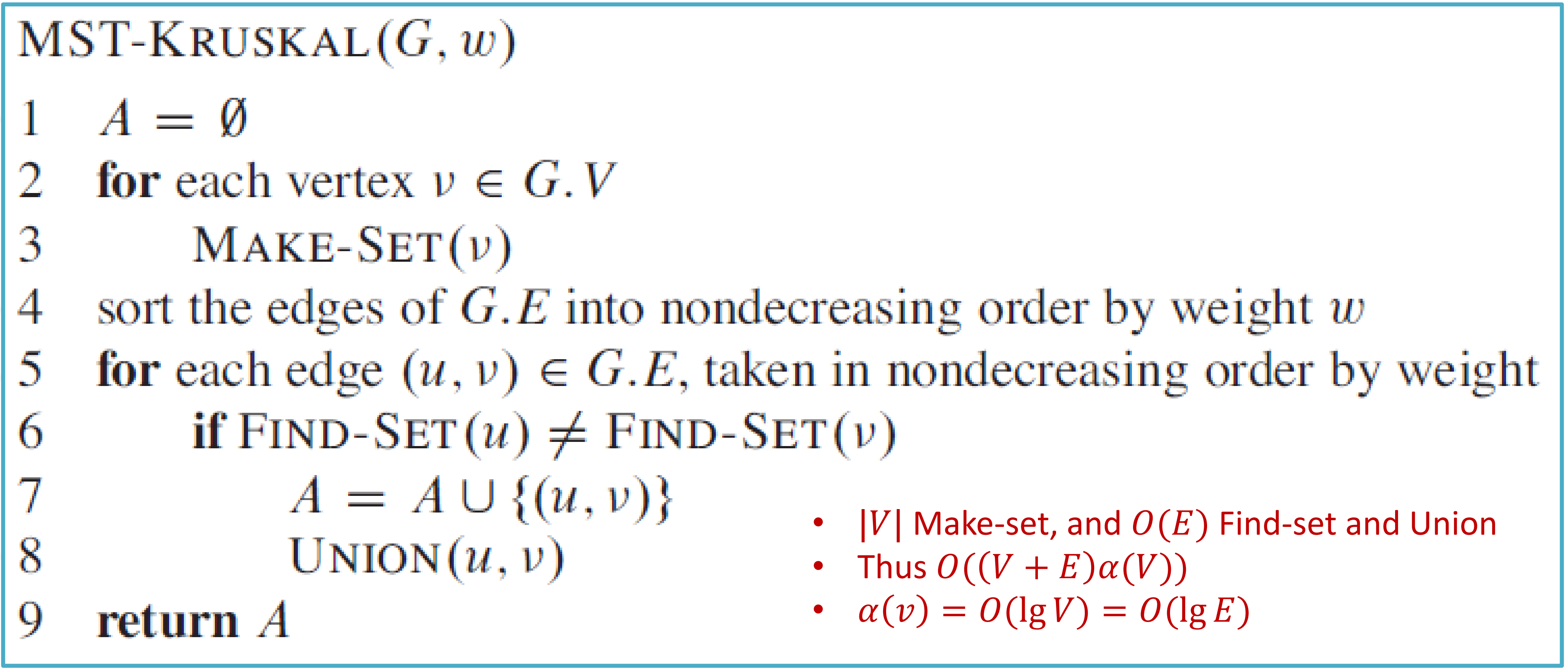
Kruskal’s algorithm



- Each iteration: a) have a forest and b) add the least-weight safe edge connecting two different components

Kruskal’s algorithm

- Algorithm:



```
MST-KRUSKAL$(G, w)$
1  $A = \emptyset$
2  for each vertex $v \in G.V$
3      MAKE-SET$(v)$
4  sort the edges of $G.E$ into nondecreasing order by weight $w$
5  for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6      if FIND-SET$(u) \neq$ FIND-SET$(v)$
7          $A = A \cup \{(u, v)\}$
8          UNION$(u, v)$
9  return $A$
```

$O(1)$

$O(E \log E)$

$O(E \log E)$ Lines 2-3 + 5-8

- $|V|$ Make-set, and $O(E)$ Find-set and Union
- Thus $O((V + E)\alpha(V))$
- $\alpha(v) = O(\lg V) = O(\lg E)$

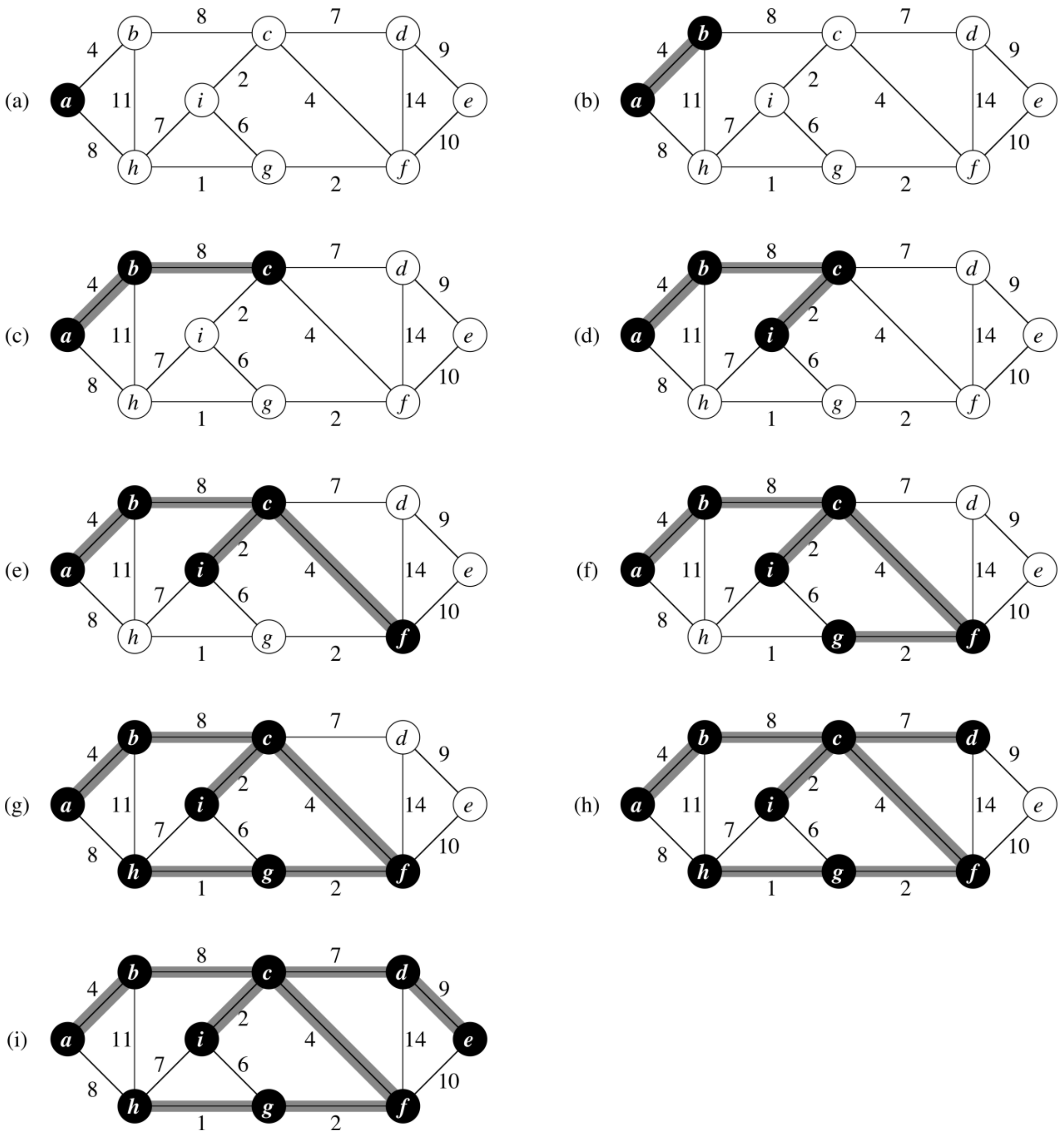
- Complexity: $O(E \log E) = O(E \log V)$

Prim’s algorithm

During each iteration:

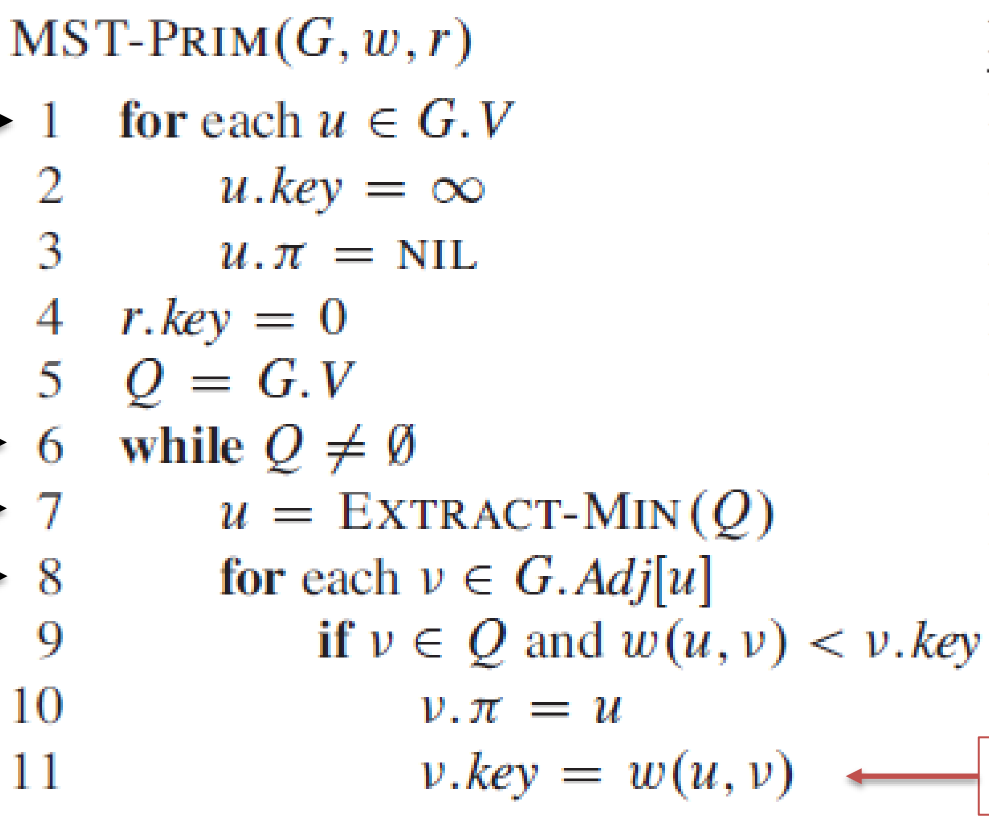
a) have a tree

b) add the least-weight safe edge connecting the tree to vertex not in tree



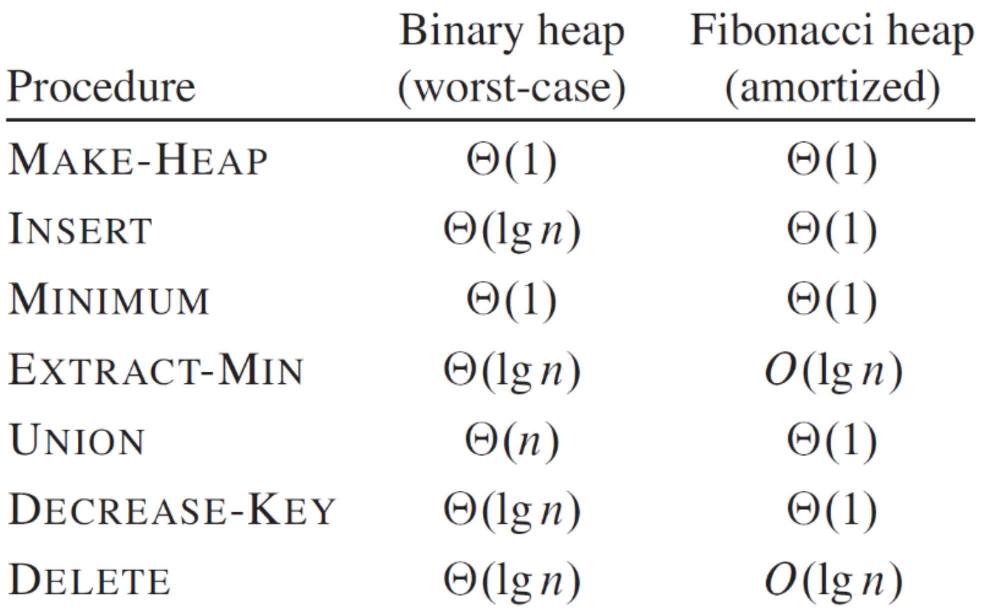
Prim’s algorithm

- Algorithm:



```
MST-PRIM$(G, w, r)$
$O(V)$ →  1  for each $u \in G.V$
          2      $u.key = \infty$
          3      $u.\pi = \text{NIL}$
          4  $r.key = 0$
          5  $Q = G.V$
$O(V)$ →  6  while $Q \neq \emptyset$
$O(\log V)$ →  7      $u = \text{EXTRACT-MIN}(Q)$
$O(E)$ →  8      for each $v \in G.Adj[u]$
Lines 6 – 8   9          if $v \in Q$ and $w(u, v) < v.key$
         10              $v.\pi = u$
$O(\log V)$ → 11              $v.key = w(u, v)$
```

Embedded Decrease-Key



Procedure	Binary heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

- Complexity: $O(V \log V + E \log V) = O(E \log V)$
  - – Using Fibonacci heaps: $O(E + V \log V)$