

UMass Lowell Computer Science 91.503

*Algorithms*

Dr. Haim Levkowitz

Fall, 2007

---

***Graph Algorithms:*** Chapters 22-25

***Part 2: minimum spanning trees***

# Minimum Spanning Trees (Ch. 23)

---

- E.g., design of electronic circuitry
- Interconnect  $n$  pins
- Use  $n - 1$  wires, each connecting 2 pins
  - Many different ways
- Most desirable: use least amount of wire

# Another example

---

- Town has set of houses + set of roads
- Road connects 2 and only 2 houses
- Road connecting houses  $u$  and  $v$  has repair cost  $w(u, v)$
- Goal: repair enough (not more) roads, s.t.,
  - Everyone connected (reach every house from all other houses)
  - Total repair cost is minimum
- Model ...

# Model:

---

- Connected, undirected graph  $G = (V, E)$
- $V =$  pins / houses
- $E =$  possible interconnections between pairs of pins / roads between houses
- For each edge  $(u, v) \in E$ 
  - Weight  $w(u, v) =$  cost (wire / road needed) to connect  $u$  and  $v$

# Problem:

---

- Find acyclic subset  $T \subseteq E$ 
  - Connects all vertices
  - With total weight minimized:



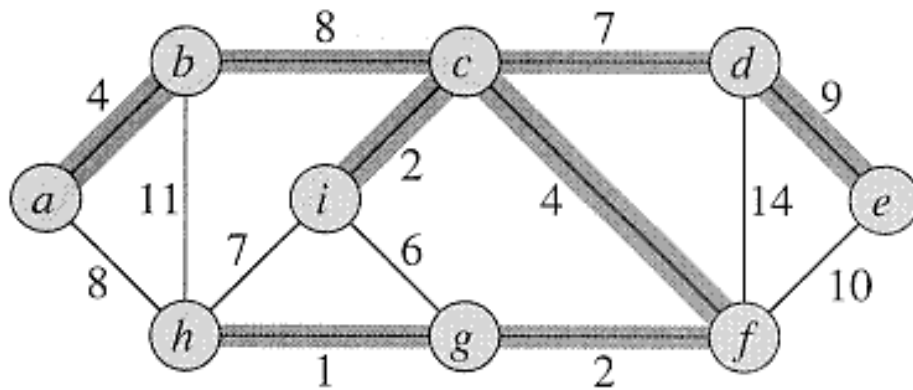
- $T$  acyclic + connects all vertices
- $\rightarrow$  must form a tree: “spanning tree”
  - “Spans”  $G$
- Problem: “minimum-spanning-tree”

# minimum-weight-spanning-tree

---

- “minimum-spanning-tree” = short name
- Not min # edges in  $T$
- All spanning trees have exactly  $|V| - 1$  edges
  - Thm B.2.

# Example



**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.

# Some properties of MST

---

- $|V| - 1$  edges
- No cycles
- Might not be unique



# Building up solution

---

- Build set  $A$  of edges
- Initially,  $A$  has no edges
- As add edge, maintain “loop invariant”
  - $A$  subset of some MST
- Add only edges that maintain invariant
- If  $A$  subset of some MST, edge  $(u, v)$  is *safe* for  $A$  iff  $A \cup \{(u, v)\}$  also subset of some MST
- $\rightarrow$  only add safe edges

# Algorithms

---

- “Generic” – grows spanning tree by adding 1 edge at a time
- Two ways to implement:
  - Kruskal’s
    - Similar to connected-components (Sec. 21.1)
  - Prim’s
    - Similar to Dijkstra’s shortest-path (Sec. 24.3)
- Both:  $O(E \lg V)$  w/ordinary binary heap
- Prim: Fibonacci heap  $\rightarrow O(E + V \lg V)$ 
  - Improvement if  $|V| \ll |E|$
- Both: greedy

# Heaps (refresher)

---

- Max (min)-heap
  - Binary tree (nearly complete)
    - Complete on all level except possibly lowest
  - + array  $A$ 
    - $length[A]$ : # elements in array
    - $heap-size[A]$ : # elements of heap stored in  $A$
  - See Fig. 6.1 p. 128

# Fibonacci heap

---

- Collection of min-heap trees
- Each node  $x$  :
  - $p[x]$ : pointer to parent
  - $child[x]$ : pointer to any one of children
  - Children linked together in circular, doubly linked list
    - *child list* of  $x$
    - $left[y], right[y]$ : siblings of child  $y$
    - Only child  $\rightarrow left[y] = right[y] = y$
- $\rightarrow$  remove node in  $O(1)$  time
- $\rightarrow$  concatenate (“splice”) 2 lists in  $O(1)$  time

# Generic MST algorithm

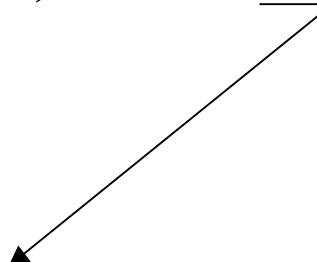
---

- Graph:  $G = (V, E)$
- Weight function:  $w : E \rightarrow \mathbb{R}$
- Greedy approach in generic alg
- Grow one edge at time
- Set of edges:  $A$ 
  - Prior to each iteration, subset of some MST
    - invariant

# Generic-MST( $G, w$ )

---

- $A \leftarrow \emptyset$
- **while**  $A$  does not form a spanning tree
  - **do** find an edge  $(u, v)$  that is safe for  $A$ 
    - $A \leftarrow A \cup \{(u, v)\}$
- **return**  $A$



**Doesn't violate, i.e.,  $A \cup \{(u, v)\}$  also subset of some MST**

**Tricky to find!!!**

# Show that generic alg works

---

- Use loop invariant
- Initialization: empty set trivially satisfies
- Maintenance: only add safe edges
  - $\rightarrow A$  remains subset of some MST
- Termination: All edges added to  $A$  are in some MST
  - $\rightarrow$  when stop
    - $A$  is spanning tree
    - also MST

# Finding safe edges (intuitive)

---

- Edge  $(c, d)$  with lowest weight in graph
- Safe for  $A = \emptyset$ ?
- $S \subset V$  set of vertices includes  $c$  but not  $d$ 
  - $d$  is in  $V - S$
- In MST, must have at least one edge that connects  $S$  with  $V - S$
- Choose one w/ minimum weight
  - $(c, d)$  in example

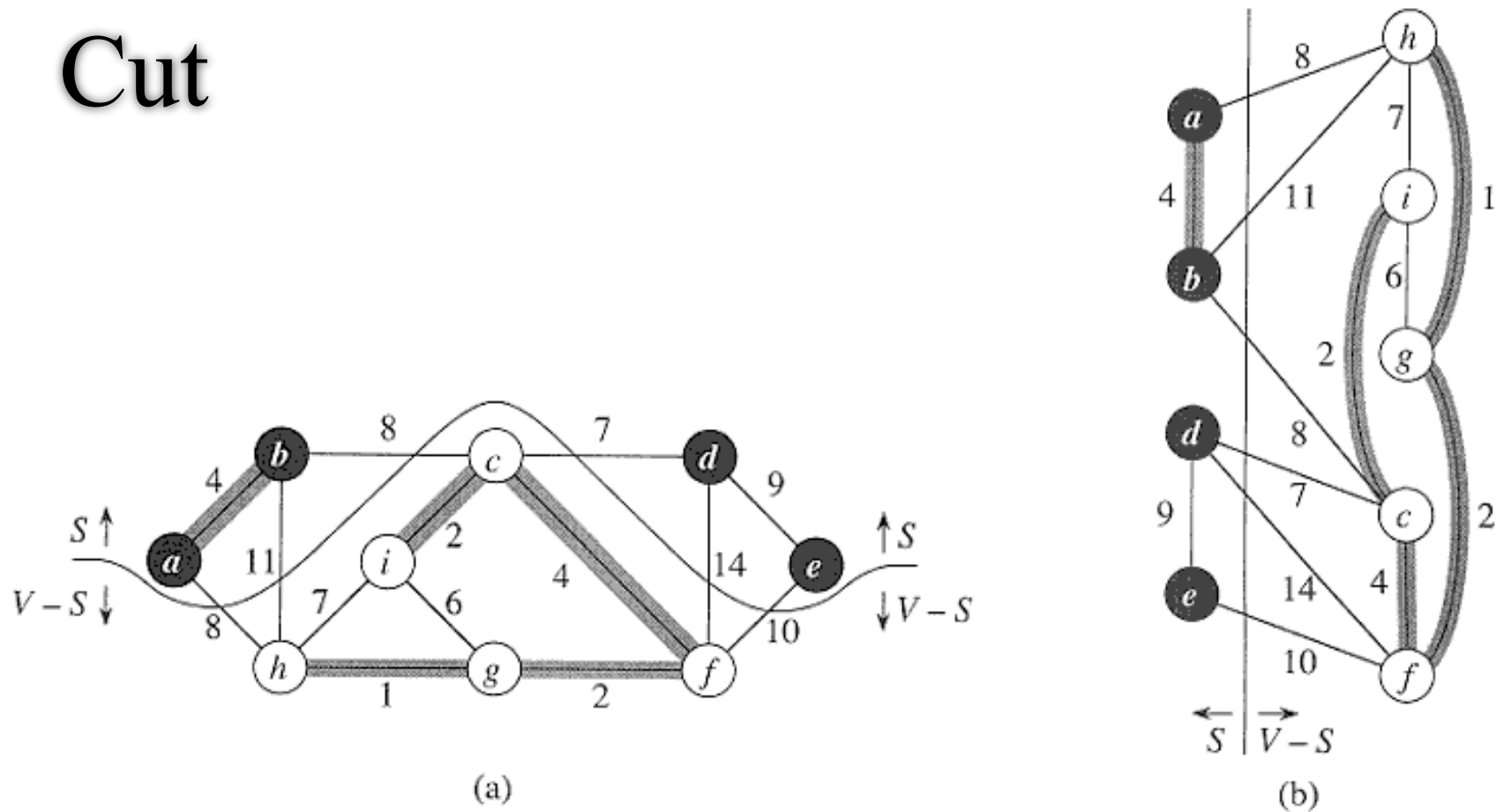


# Recognizing safe edges: a rule (Thm. 23.1)

---

- Definitions
  - *Cut*  $(S, V - S)$  of  $G = (V, E)$  :
    - partition of  $V$
    - Fig. 23.2 ...
  - *Edge*  $(u, v) \in E$  crosses cut  $(S, V - S)$  :
    - 1 endpoint in  $S$
    - Another in  $V - S$
  - *Cut respects* set  $A$  of edges : no edge in  $A$  crosses cut
  - *Light edge* : crossing cut, and weight is minimum of any edge crossing cut
    - Can be more than one
  - *Light edge satisfying* given property: weight min among all edges satisfying same property

# Cut



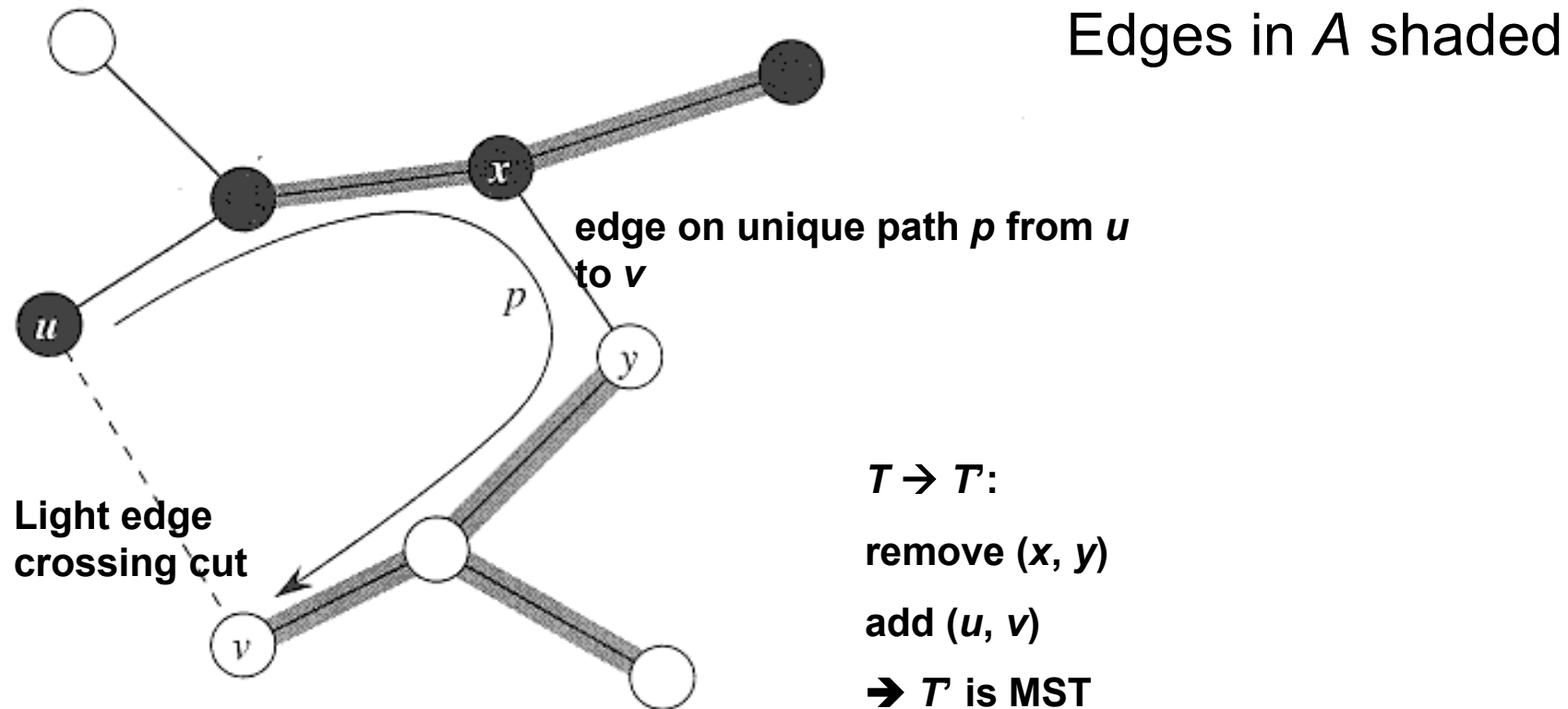
**Figure 23.2** Two ways of viewing a cut  $(S, V - S)$  of the graph from Figure 23.1. (a) The vertices in the set  $S$  are shown in black, and those in  $V - S$  are shown in white. The edges crossing the cut are those connecting white vertices with black vertices. The edge  $(d, c)$  is the unique 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. (b) The same graph with the vertices in the set  $S$  on the left and the vertices in the set  $V - S$  on the right. An edge crosses the cut if it connects a vertex on the left with a vertex on the right.

# Thm. 23.1

---

- Let  $G = (V, E)$  be a connected, undirected graph with a real-value weight function  $w$  defined on  $E$ ,
- Let  $A$  be a subset of  $E$  that is included in some MST 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  $A$
- Proof outline, Fig. 23.3, p. 565 ...

# Proof of Thm. 23.1



**Figure 23.3** The proof of Theorem 23.1. The vertices in  $S$  are black, and the vertices in  $V - S$  are white. The edges in the minimum spanning tree  $T$  are shown, but the edges in the graph  $G$  are not. The edges in  $A$  are shaded, and  $(u, v)$  is a light edge crossing the cut  $(S, V - S)$ . The edge  $(x, y)$  is an edge on the unique path  $p$  from  $u$  to  $v$  in  $T$ . A minimum spanning tree  $T'$  that contains  $(u, v)$  is formed by removing the edge  $(x, y)$  from  $T$  and adding the edge  $(u, v)$ .

# In Generic-MST

---

- $A$  = forest containing connected components
- Initially each component is single vertex
- Any safe edge merges 2 components into one
  - Each component is a tree
- MST has exactly  $|V| - 1$  edges
  - $\rightarrow$  for loop iterates  $|V| - 1$  times
- I.e., after adding  $|V| - 1$  safe edges
  - Down to one component

# Kruskal's alg

---

- $G = (V, E)$  is connected, undirected, weighted graph,  $w : E \rightarrow \mathbb{R}$
- Start w/ ea. vertex being its own component
- Repeatedly merge 2 components into one
  - Choose light edge that connects them (greedy)
    - Crossing cut between them
- Scan set of edges in monotonically increasing order by weight

- 
- Use disjoint-set data structure to
    - Maintain several disjoint sets of elements
    - Determine if edge connects vertices in different components
  - Make-Set( $v$ ) creates  $|V|$  trees, ea. single vertex
  - Find-Set( $u$ ): return representative element from set that contains  $u$
  - Find-Set( $u$ ) = Find-Set( $v$ )  $\iff u$  &  $v$  belong to same tree
  - Combine trees by Union( $u, v$ )

# MST-Kruskal( $G, w$ )

MST-KRUSKAL( $G, w$ )

1  $A \leftarrow \emptyset$

2 **for** each vertex  $v \in V[G]$

3     **do** MAKE-SET( $v$ )

Create  $|V|$  single vertex trees

4 sort the edges of  $E$  into nondecreasing order by weight  $w$

5 **for** each edge  $(u, v) \in E$ , taken in nondecreasing order by weight

6     **do if** FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )

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

8             UNION( $u, v$ )

9 **return**  $A$

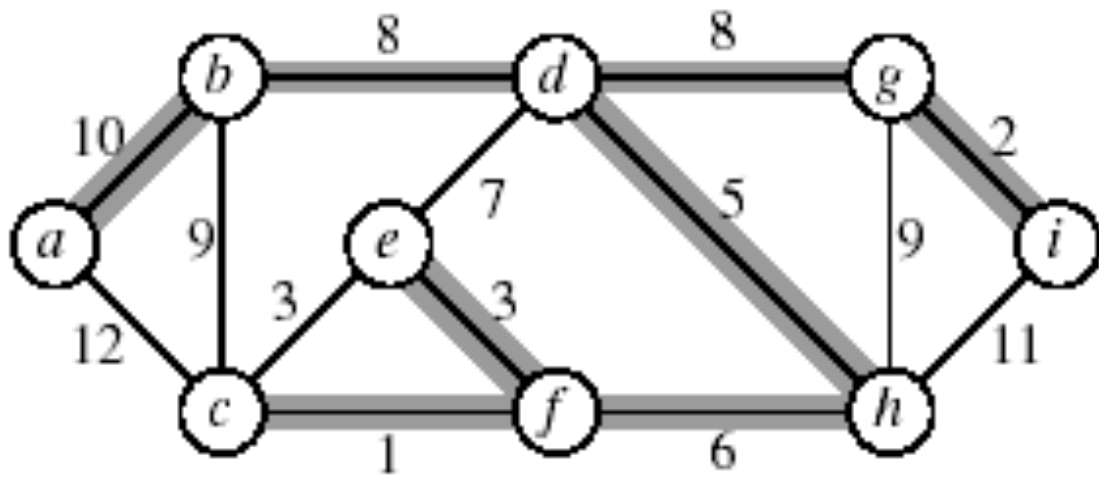
$u, v$ , belong to same tree? If not, add edge (7) + merge (8)



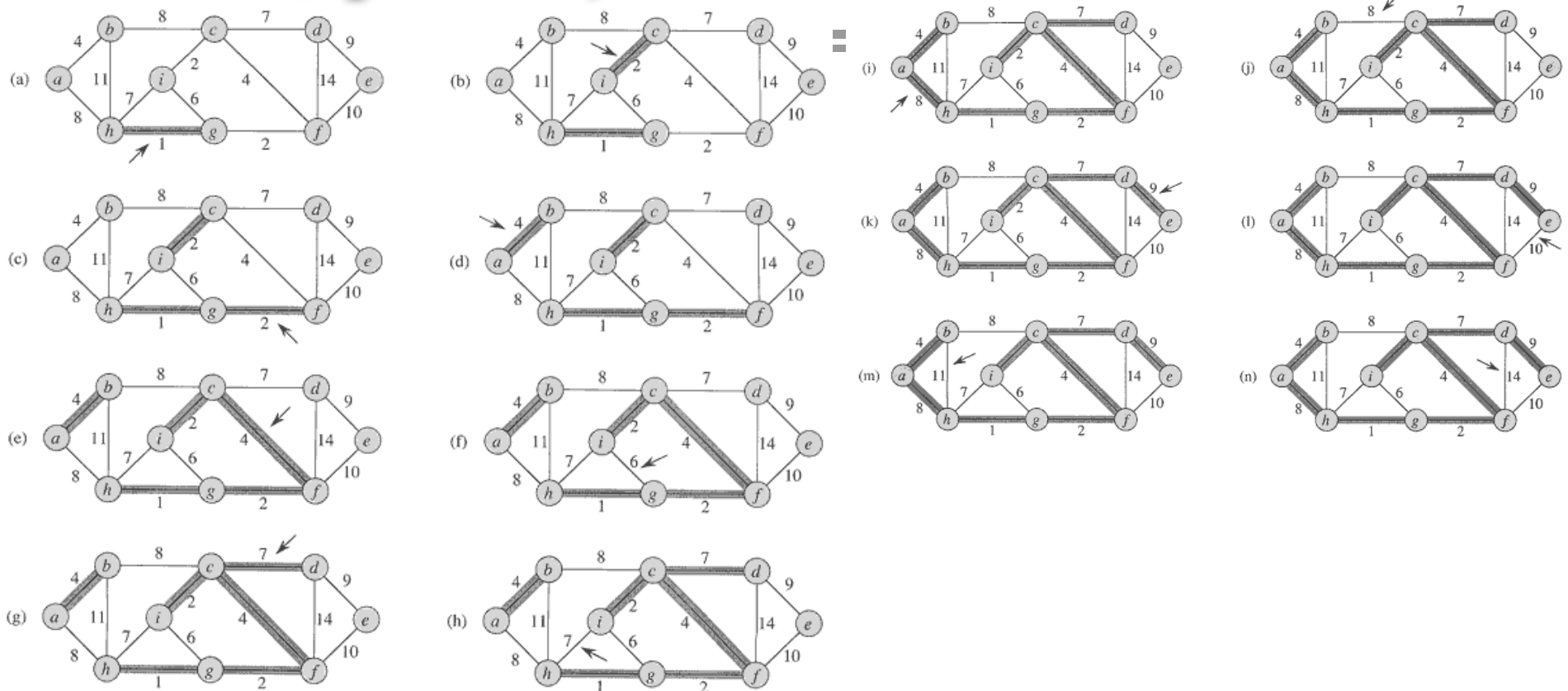
# Kruskal's alg's execution: example

---

- $(c, f)$  : safe
- $(g, i)$  : safe
- $(e, f)$  : safe
- $(c, e)$  : reject
- $(d, h)$  : safe
- $(f, h)$  : safe
- $(e, d)$  : reject
- $(b, d)$  : safe
- $(d, g)$  : safe
- $(b, c)$  : reject
- $(g, h)$  : reject
- $(a, b)$  : safe
- At this point, we have only one component, so all other edges will be rejected.



# Kruskal's alg's execution (Fig. 23.4, p. 568)



**Figure 23.4** The execution of Kruskal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest  $A$  being grown. The edges are considered by the algorithm in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees.

# Kruskal running time

---

- Depends on implementation of disjoint-set data structure
  - See section 21.3 for asymptotically fastest implementation
    - (union-by-rank + path-compression heuristics)

# Analysis

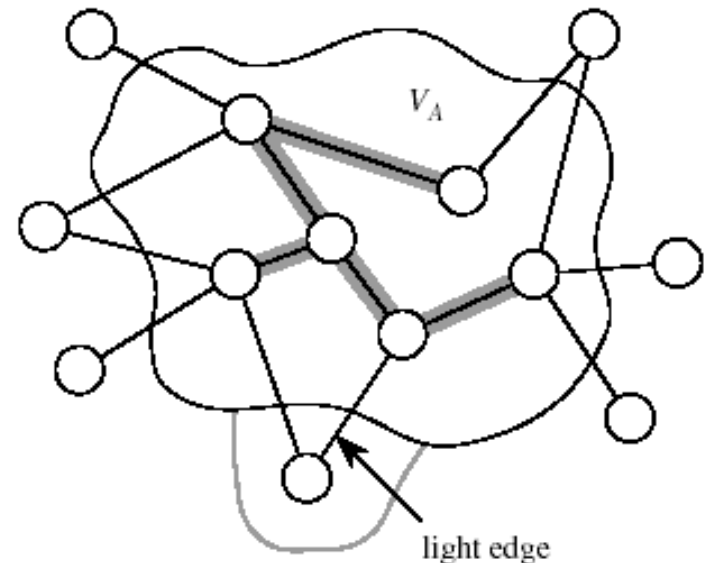
---

- Initialize  $A$ :  $O(1)$
- First for loop:  $|V|$  MAKE-SETs
- Sort  $E$ :  $O(E \lg E)$
- Second for loop:  $O(E)$  FIND-SETs and UNIONs

- 
- Assuming implementation of disjoint-set data structure, w/ union-by-rank and path-compression:
  - $O((V + E) \alpha(V)) + O(E \lg E)$ 
    - $\alpha$  : very slowly growing function (Sec. 21.4)
  - $G$  connected  $\rightarrow |E| \geq |V| - 1$ 
    - $\rightarrow O(E \alpha(V)) + O(E \lg E)$
  - $\alpha(|V|) = O(\lg V) = O(\lg E)$
  - $\rightarrow$  total time is  $O(E \lg E)$
  - $|E| < |V|^2 \rightarrow \lg |E| = O(2 \lg V) = O(\lg V)$
  - $\rightarrow O(E \lg V)$
  - If edges already sorted  $O(E \alpha(V))$  – almost linear

# Prim's algorithm

- Builds one tree, so  $A$  always a tree
- Starts from arbitrary “root”  $r$
- 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  $A$



[Edges of  $A$  are shaded.]

# How to find the light edge quickly?

---

- Use priority queue  $Q$ :
  - Each object = vertex in  $V - V_A$
  - Key of  $v$  = minimum weight of any edge  $(u, v)$ , where  $u \in V_A$
  - $\rightarrow$  vertex returned by EXTRACT-MIN =
    - $v$  s.t. there exists  $u \in V_A$  and
    - $(u, v)$  is light edge crossing  $(V_A, V - V_A)$
  - Key of  $v = \infty$  if  $v$  not adjacent to any vertices in  $V_A$

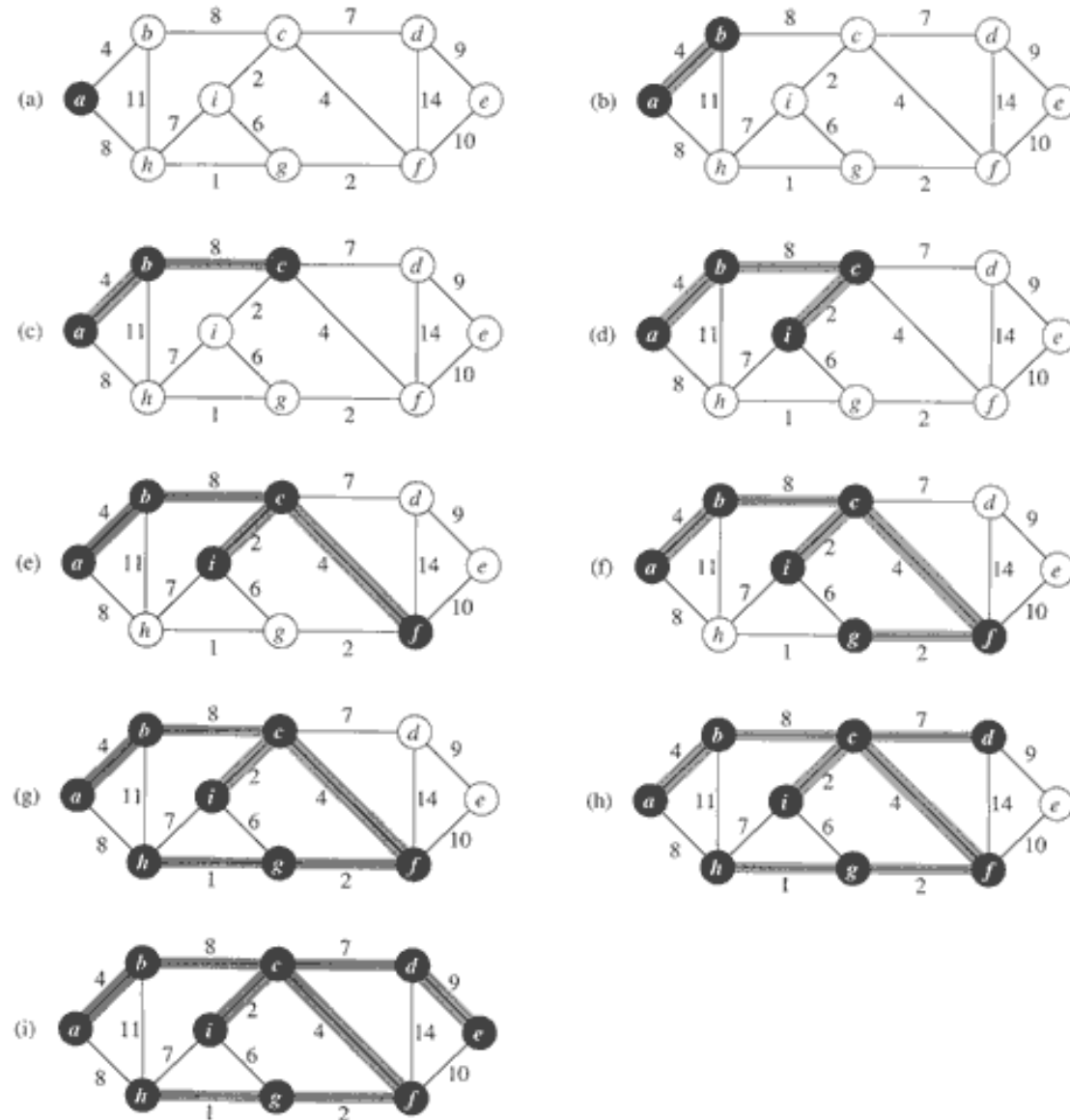
- 
- Edges of  $A$  will form rooted tree w/ root  $r$ 
    - $r$  given as input to algorithm, but can be any vertex
    - Each vertex knows its parent in tree by attribute  $\pi[v]$  = parent of  $v$
    - $\pi[v] = \text{NIL}$  if  $v = r$  or  $v$  has no parent
    - As algorithm progresses
      - $A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$
    - At termination,  $V_A = V \rightarrow Q = \emptyset$
    - $\rightarrow$  MST is  $A = \{(v, \pi[v]) : v \in V - \{r\}\}$



MST-PRIM( $G, w, r$ )      **MST-Prim**( $G, w, r$ )

```
1  for each  $u \in V[G]$ 
2      do  $key[u] \leftarrow \infty$ 
3       $\pi[u] \leftarrow \text{NIL}$ 
4   $key[r] \leftarrow 0$ 
5   $Q \leftarrow V[G]$ 
6  while  $Q \neq \emptyset$ 
7      do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
8      for each  $v \in Adj[u]$ 
9          do if  $v \in Q$  and  $w(u, v) < key[v]$ 
10             then  $\pi[v] \leftarrow u$ 
11              $key[v] \leftarrow w(u, v)$ 
```

# Prim's alg's execution



**Figure 23.5** The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is *a*. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (*b*, *c*) or edge (*a*, *h*) to the tree since both are light edges crossing the cut.

# Analysis

---

- Homework assignment (no submit):
  - Assume  $Q$  is a binary heap

# Minimum Spanning Trees

## ■ Review problem:

- For the undirected, weighted graph below, show 2 different Minimum Spanning Trees. Draw each using one of the 2 graph copies below. Thicken an edge to make it part of a spanning tree. What is the sum of the edge weights for each of your Minimum Spanning Trees?

