Minimum Spanning Tree

ITX2010, CSX3003, IT2230

Data Structures and Algorithms, Information Structures

Learning Objectives

Students will be able to:

- Understand what minimum spanning tree is
- Explain how MST grow
- Present how a specific MST Kruskal’s and Prim’s algorithm work.

Chapter Outline

1. Minimum Spanning Tree
   1) Minimum-Spanning-Tree problem
   2) Growing a minimum spanning tree
2. Kruskal's Algorithm
3. Prim's Algorithm

23.1

Minimum Spanning Tree

Minimum Spanning Tree

- In design of electronic circuitry, it is often necessary to make the pins of several components electrically equivalent by wiring them together.

Chapter 23 Minimum Spanning Trees 562

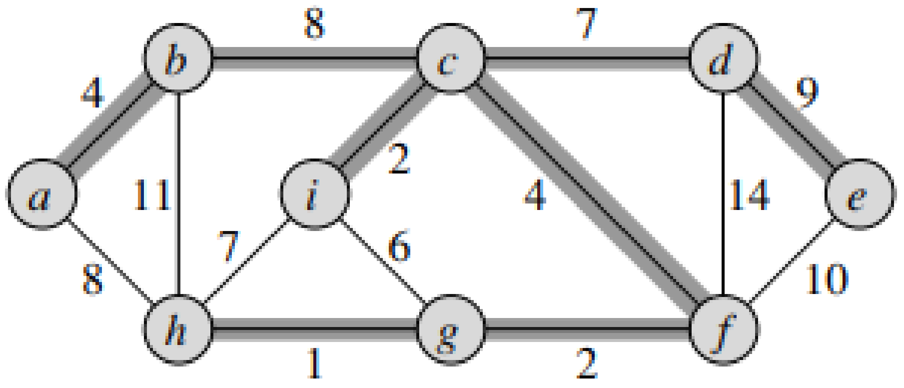


Figure 23.1 A minimum spanning tree for a connected graph. The weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree shown is 37. This minimum spanning tree is not unique: removing the edge (b, c) and replacing it with the edge (a, h) yields another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

Minimum Spanning Tree

- We can model the wiring problem with a connected, undirected graph G =(V, E): [1]
  - V is the set of pin,
  - E us the set of pair of pins (u, v),
  - w(u, v) specifies a weight or cost on an edge,
  - T is an acyclic subset and T $\subseteq$ E that connects all vertices and total weight.

$$w(T) = \sum_{(u,v)=1}^{T} w(u,v)$$

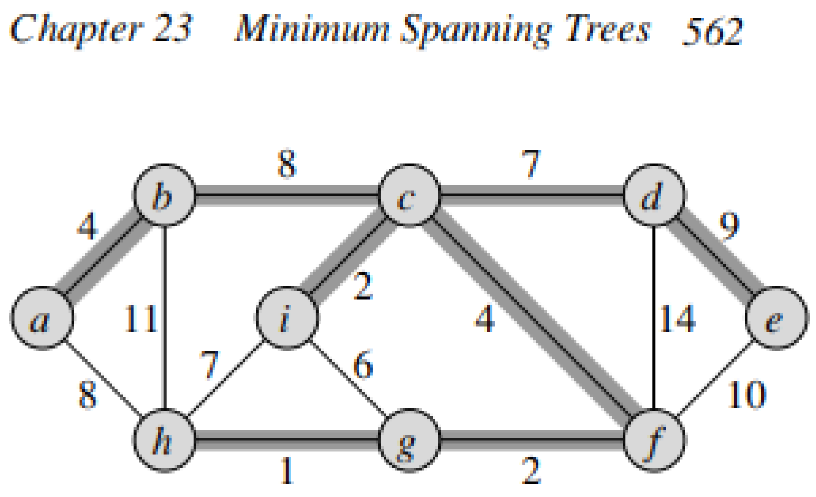


Fig 23.1 Minimum Spanning Tree [1]

Minimum Spanning Tree

- T is acyclic and connects all vertices which must form a tree – called a spanning tree.

- There are two algorithms – used in solving the minimum-spanning-tree problems: [1]
  - Kruskal's algorithm
  - Prim algorithm

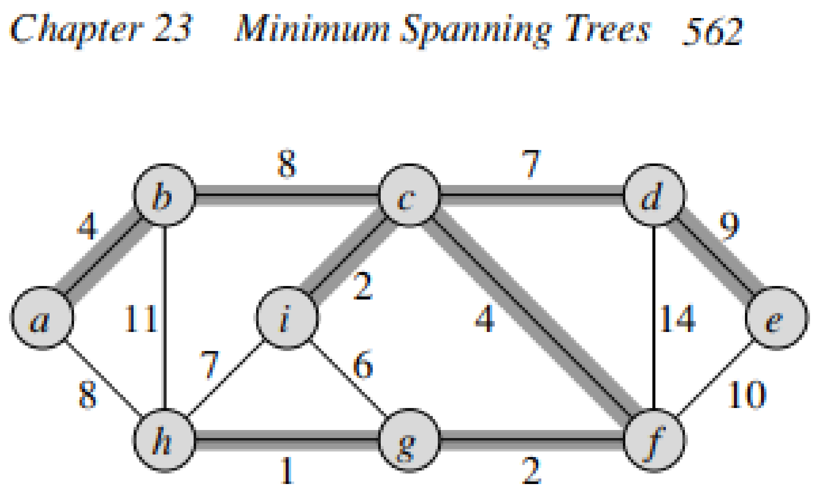


Fig 23.1 Minimum Spanning Tree [1]

Minimum Spanning Tree

- Both greedy algorithm: [1]
  - grow a spanning tree by adding edge at a time.
  - yield a spanning tree with minimum weight.
  - can easily be made to run in time O(E lg V) using ordinary binary heaps.

- By using Fibonacci heaps, Prim algorithm can be speed up to run in time O(E+ V lg V) [1]
  - It is an improvement if |v| is much smaller than |E|.

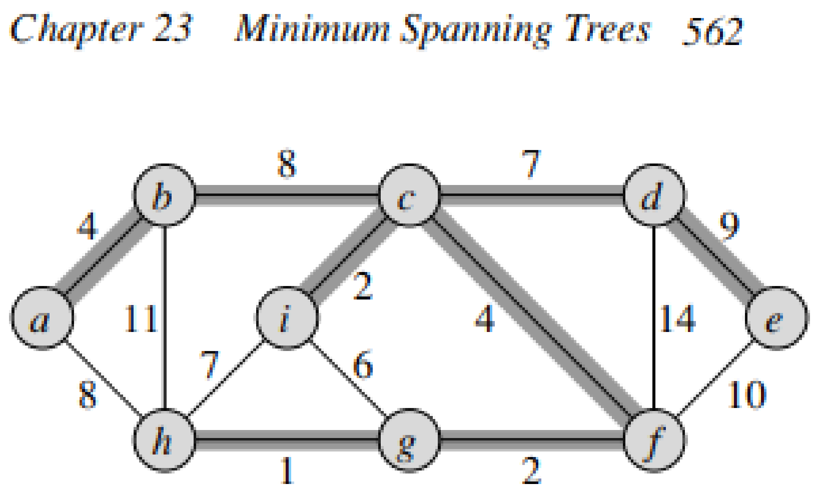


Fig 23.1 Minimum Spanning Tree [1]

Growing a Minimum Spanning Tree

Generic-MST

- A connected, undirected graph G = (V, E) with [1]
  - A weight function w : E → R,
  - A subset of some minimum spanning tree A,

The generic algorithm find MST's maintain a subset A,

- At each step, an edge (u, v) is added to A without violating the invariant
- Find a safe edge for A

23.1 Growing a minimum spanning tree [1] *563*

```
GENERIC-MST(G, w)
1  A ← ∅
2  while A does not form a spanning tree
3      do find an edge (u, v) that is safe for A
4          A ← A ∪ {(u, v)}
5  return A
```

We use the loop invariant as follows:

Initialization: After line 1, the set A trivially satisfies the loop invariant.

Maintenance: The loop in lines 2–4 maintains the invariant by adding only safe edges.

Termination: All edges added to A are in a minimum spanning tree, and so the set A is returned in line 5 must be a minimum spanning tree.

Growing a Minimum Spanning Tree

Generic-MST

- A connected undirected graph G = (V, E) :
  - **cut (S, V-S)** is a partition of V,
  - **edge crossing** is an edge that connects a vertex in S to a vertex in V-S
  - **light edge** is the edge crossings with the minimum weight.

564 *Chapter 23 Minimum Spanning Trees* [1]

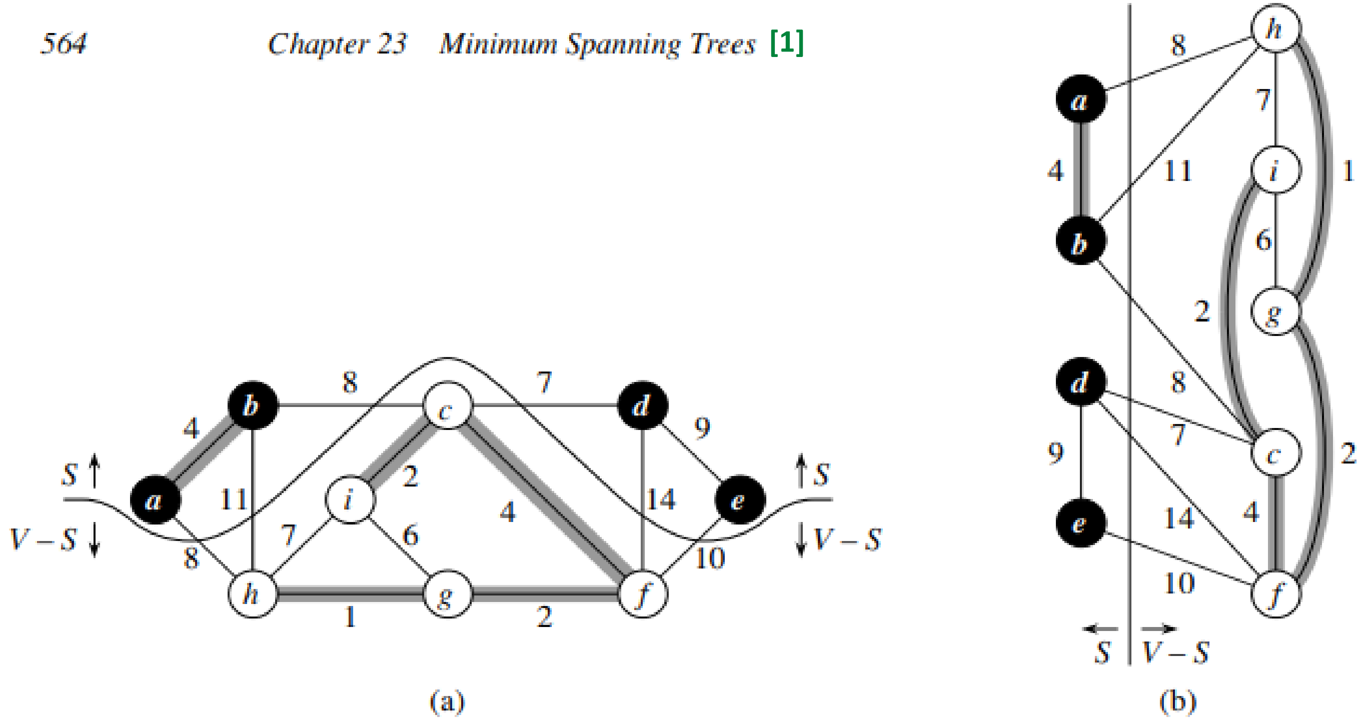


Figure 23.2 Two ways of viewing a cut $(S, V - S)$ of the graph from Figure 23.1. **(a)** The vertices in the set S are shown in black, and those in $V - S$ are shown in white. The edges crossing the cut are those connecting white vertices with black vertices. The edge (d, c) is the unique light edge 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. [1]

Growing a Minimum Spanning Tree

Recognizing safe edges

- An edge crosses the cut if one of its endpoints is in S and the other is in V-S.
- 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.

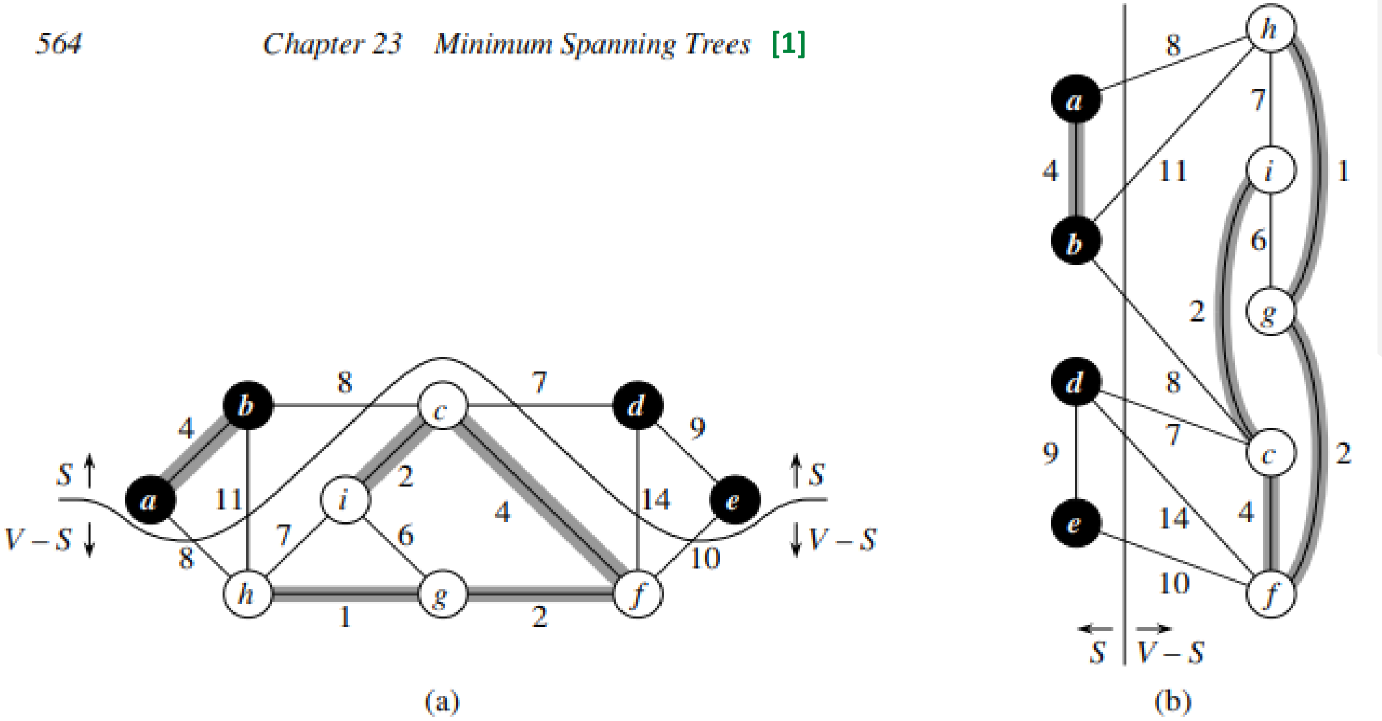


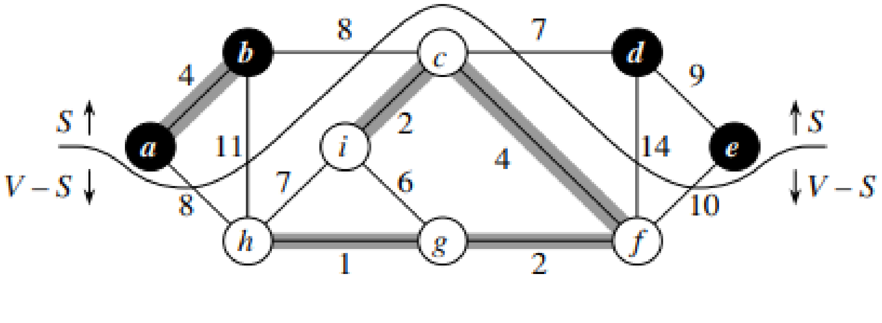
Figure 23.2 Two ways of viewing a cut $(S, V-S)$ of the graph from Figure 23.1. **(a)** The vertices in the set S are shown in black, and those in $V-S$ are shown in white. The edges crossing the cut are those connecting white vertices with black vertices. The edge (d, c) is the unique light edge crossing 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. [1]

23.1

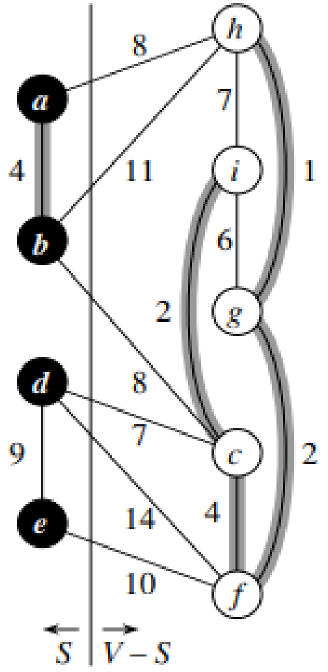
Growing a Minimum Spanning Tree

Theorem 23.1

Theorem 23.1
Let $G = (V, E)$ be a connected, undirected graph with a real-valu
tion w defined on E. Let A be a subset of E that is included in
spanning tree for G, let $(S, V - S)$ be any cut of G that respects
be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for .



(a)



(b)

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. [1]

Growing a Minimum Spanning Tree

Theorem 23.1

23.1 Growing a minimum spanning tree 565

Theorem 23.1 [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 A. [1]

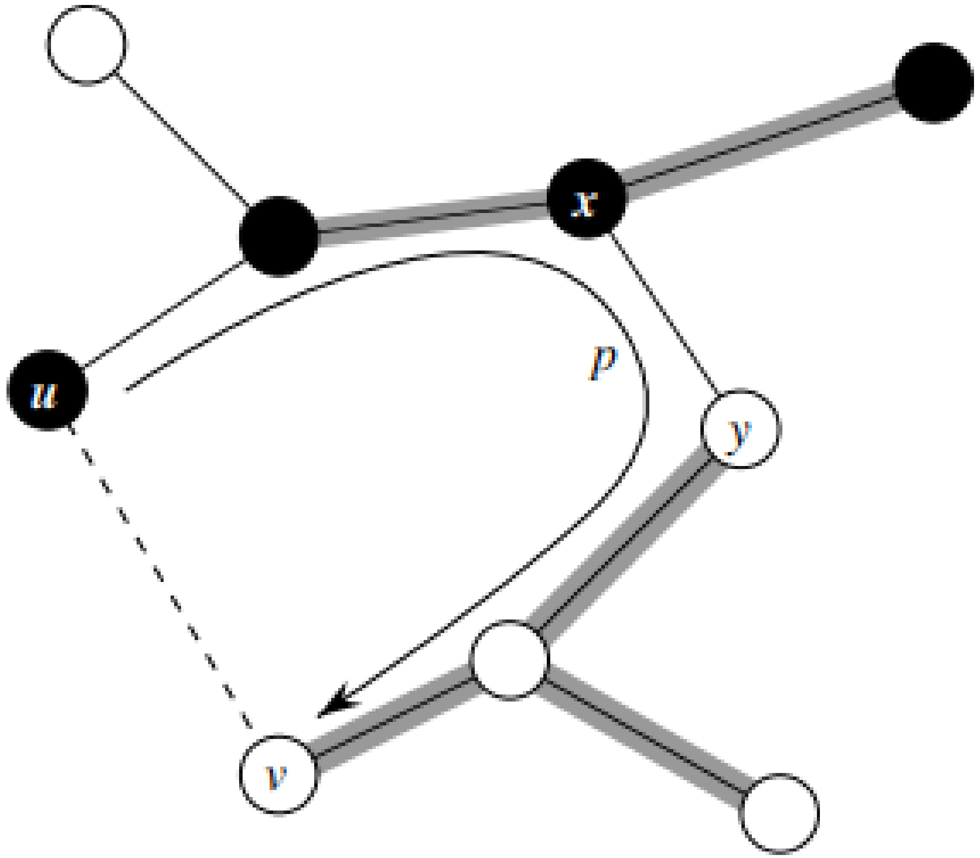


Figure 23.3 The proof of Theorem 23.1. The vertices in S are black, and the vertices in $V -$ white. The edges in the minimum spanning tree T are shown, but the edges in the graph G ai The edges in A are shaded, and (u, v) is a light edge crossing the cut $(S, V - S)$. The edge $(x$ an edge on the unique path p from u to v in T. A minimum spanning tree T' that contains $(u$ formed by removing the edge (x, y) from T and adding the edge (u, v). [1]

23.2

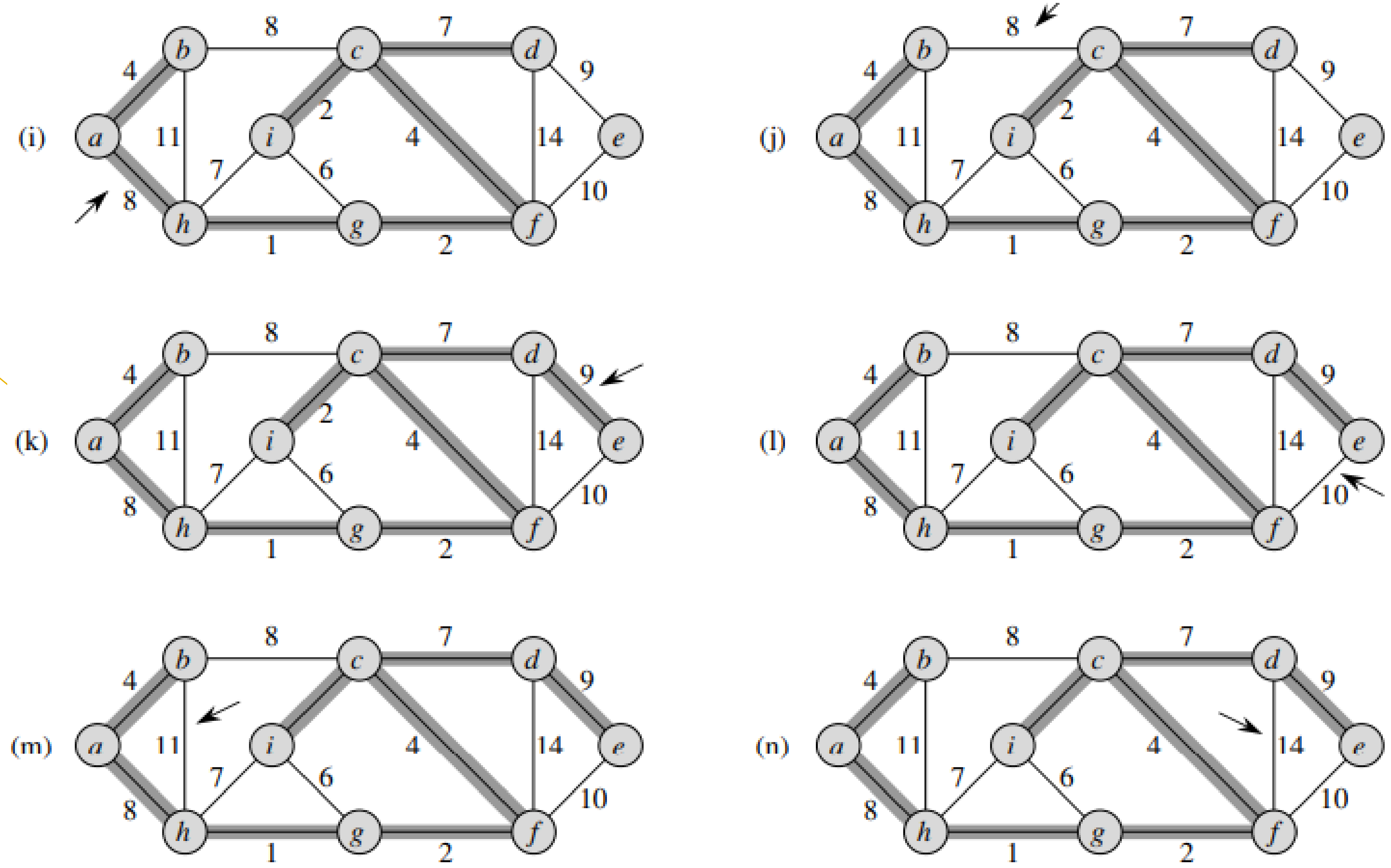
Kruskal's Algorithm

Kruskal's Algorithm

- A minimum –spanning-tree algorithm use a specific rule to determine a safe edge of GENERIC-MST. [1]
- In Kruskal's algorithm, [1]
  - The set A is a growing forest that finds all edges that connect any two trees – C_1 and C_2, with the minimum weight.
  - The safe edge added to A is always a lest-weight edge in the graph that connects two distinct components.
- Let C_1 and C_2 denote the two trees that are connected by (u, v): [1]
  - The (u, v) must be a light edge connecting C_1 to some other tree and safe edge for C_1.
- It uses a disjoint-set to maintain several disjoint sets of elements. [1]

Kruskal’s Algorithm

23.2 The algorithms of Kruskal and Prim [1] *569*



MST-KRUSKAL(G, w) [1]

```
A ← ∅
for each vertex v ∈ V[G]
    do MAKE-SET(v)
sort the edges of E into nondecreasing order by weight w
for each edge (u, v) ∈ E, taken in nondecreasing order by weight
    do if FIND-SET(u) ≠ FIND-SET(v)
          then A ← A ∪ {(u, v)}
               UNION(u, v)
return A
```

Kruskal's Algorithm

- The running time of Kruskal's algorithm for a graph G= (V, E) depends on the implementation of the disjoint-set data structure. [1]
- For the disjoint-set-forest implementation, [1]
  - Line 1 takes O(1) time,
  - Line 4 is O(E lg E),
  - Line 5-8 perform O(E) FIND_SET and UNION operations,
  - Along with the |V| MAKE-SET operations, these take a total of O((V+E) α (V)) time.
- The total running time is O(E lg E).
  - If $|E| < |V|^2$ and we have lg |E| = O(lg V), then O(E lg V)

MST-KRUSKAL(G, w) [1]

```
1  A ← Ø
2  for each vertex v ∈ V[G]
3       do 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       do if FIND-SET(u) ≠ FIND-SET(v)
7             then A ← A ∪ {(u, v)}
8                  UNION(u, v)
9  return A
```

23.3

Prim’s Algorithm

Prim's Algorithm

- Like Kruskal's algorithm, it us a special case of the generic minimum-spanning-tree algorithm.

- In Prim's algorithm, [1]
  - It has the property that the edges in the set A always form a single tree.
  - The tree starts from an arbitrary root vertex r and grows until the tree spans all the vertex in V
  - This rule adds only edges that are sage for A to form a minimum spanning tree.

23.3

```
MST-PRIM(G, w, r)   [1]
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)
```

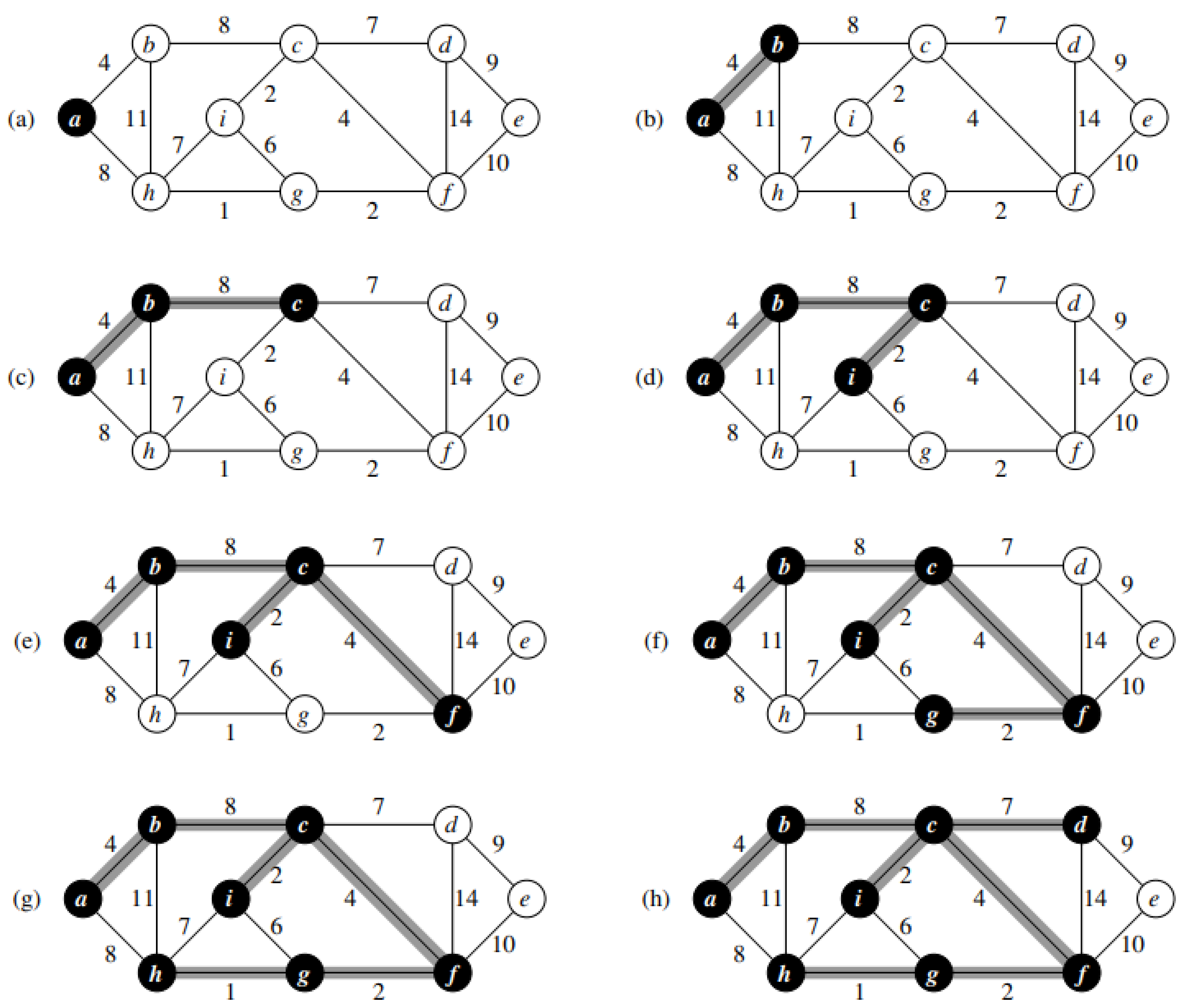
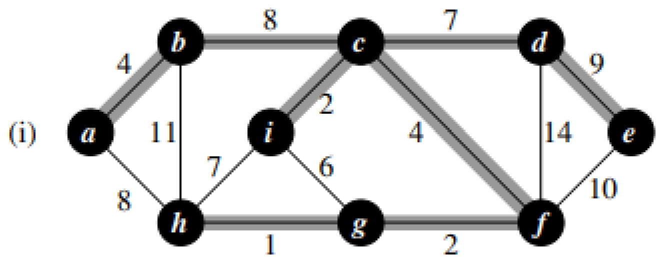


Figure 23.5 The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is a. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing 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. [1]



Prim's Algorithm

- The running time of Prim's algorithm for a graph G= (V, E) depends on how we implement the min-priority queue. [1]

- If Q is implemented as a binary min-heap, [1]
  - Line 1-5 take O(V) time by using BUILD-MIN-HEEAP,
  - While loop takes O(V lg V) times due to each EXTRACT-MIN operation takes O(lg V) time,
  - For loop in line 8-11 is executed O(E) times.
- The total running time is O(V lg V +E lg V) = O(E lg V).
  - If we use a Fibonacci heap to implement the min-priority queue, it will be improved to O(E + V lg V).

MST-PRIM(G, w, r) [1]

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

References

Texts | Integrated Development Environment (IDE)

[1] Introduction to Algorithms, Second Edition, Thomas H. C., Charles E. L., Ronald L. R., Clifford S., The MIT Press, McGraw-Hill Book Company, Second Edition 2001.

[2] https://www.cs.usfca.edu/~galles/visualization/