MINIMUM SPANNING TREE

Analysis of Algorithm

Punjab University College of Information Technology (PUCIT)
University of the Punjab, Lahore, Pakistan.

Credit

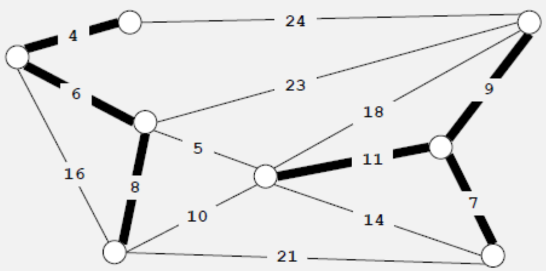
- These notes contain material from Chapter 22 of Cormen, Leiserson, Rivest, and Stein (3rd Edition).
- Minimum Spanning Tree from Algorithms, 4th Edition by Sedgewick, Wayne (http://algs4.cs.princeton.edu/lectures/43MinimumSpanningTrees.pdf)
- http://www.cse.ust.hk/~dekai/271/notes/L07/L07.pdf

Introduction

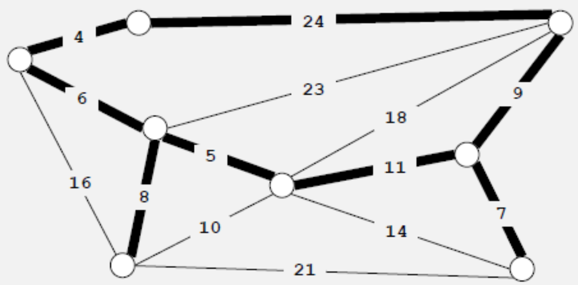
Given: Undirected graph G with positive edge weights (connected).

Definition: A spanning tree of G is a subgraph T that is connected and acyclic.

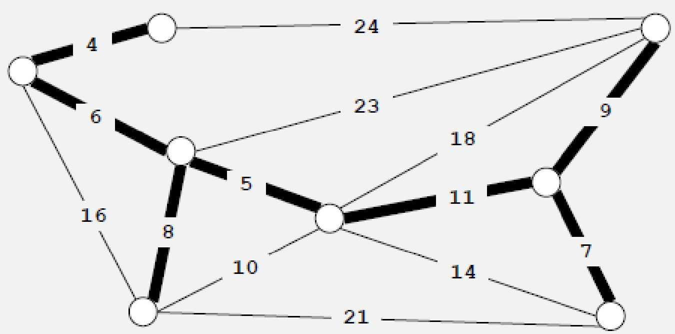
Goal: Find a minimum weight spanning tree.



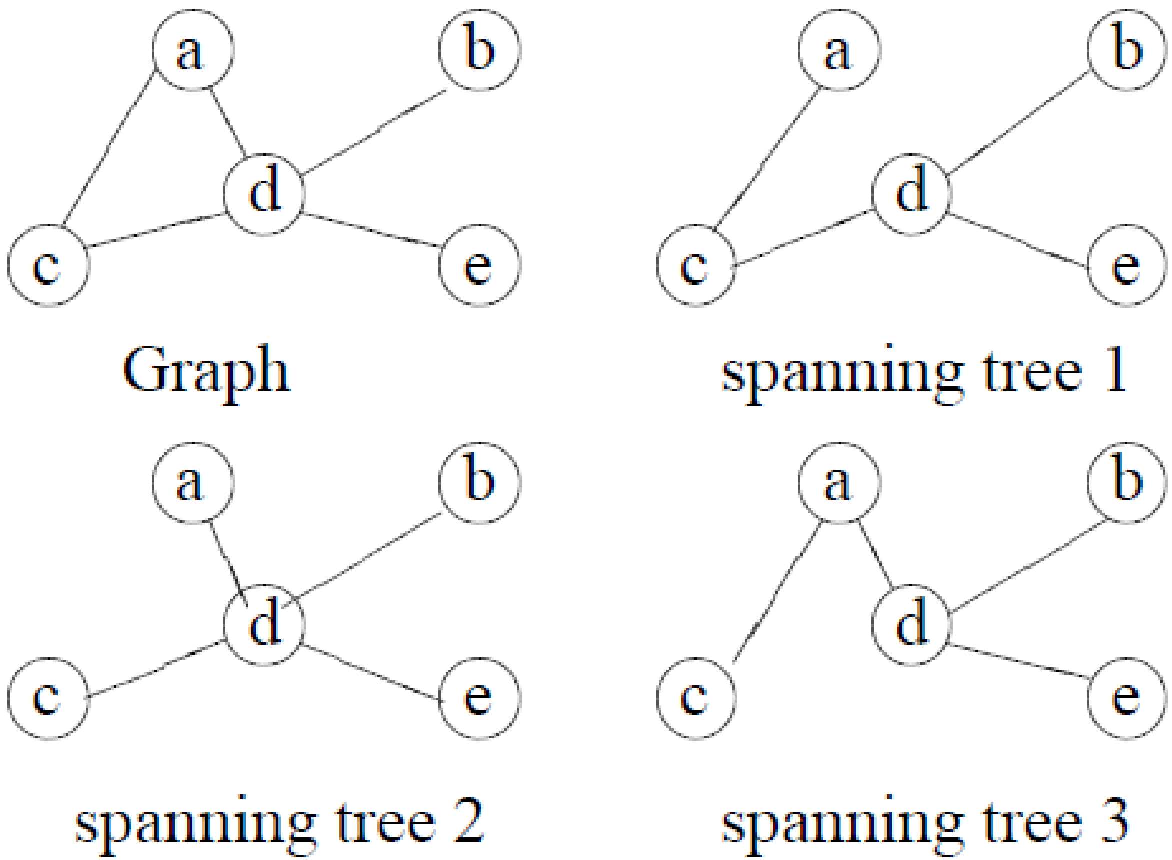
not connected

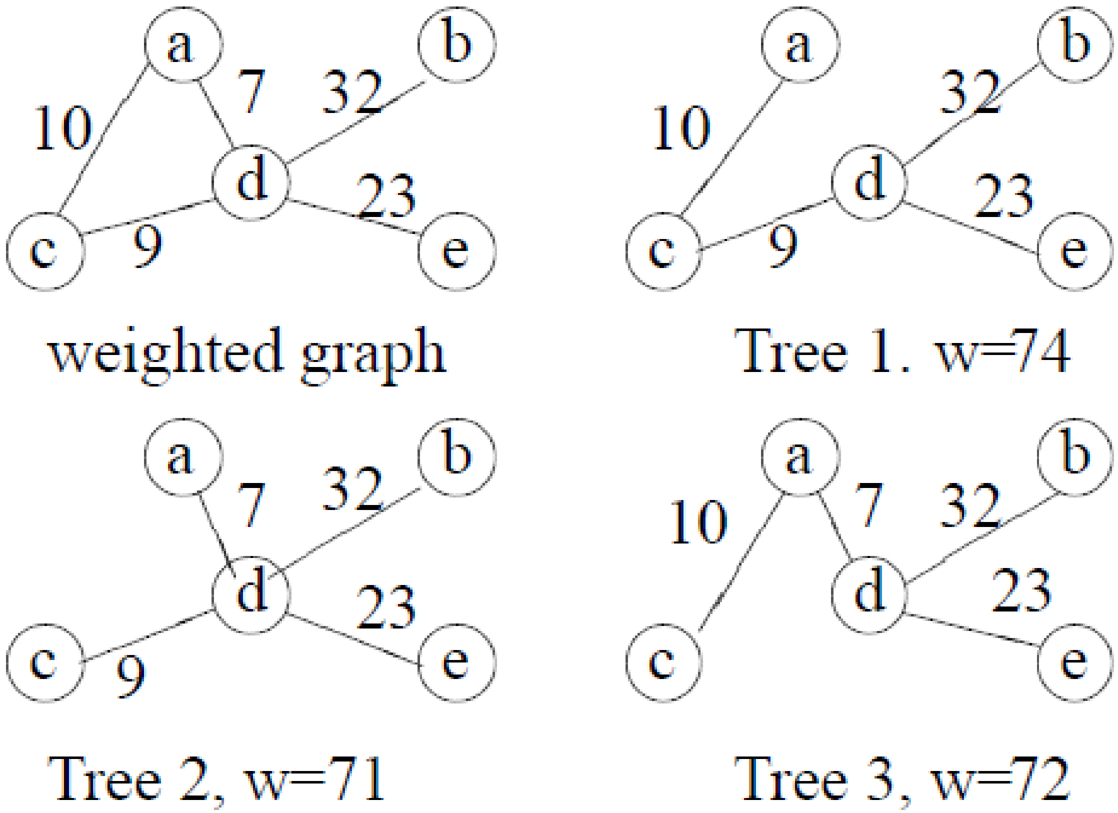


not acyclic



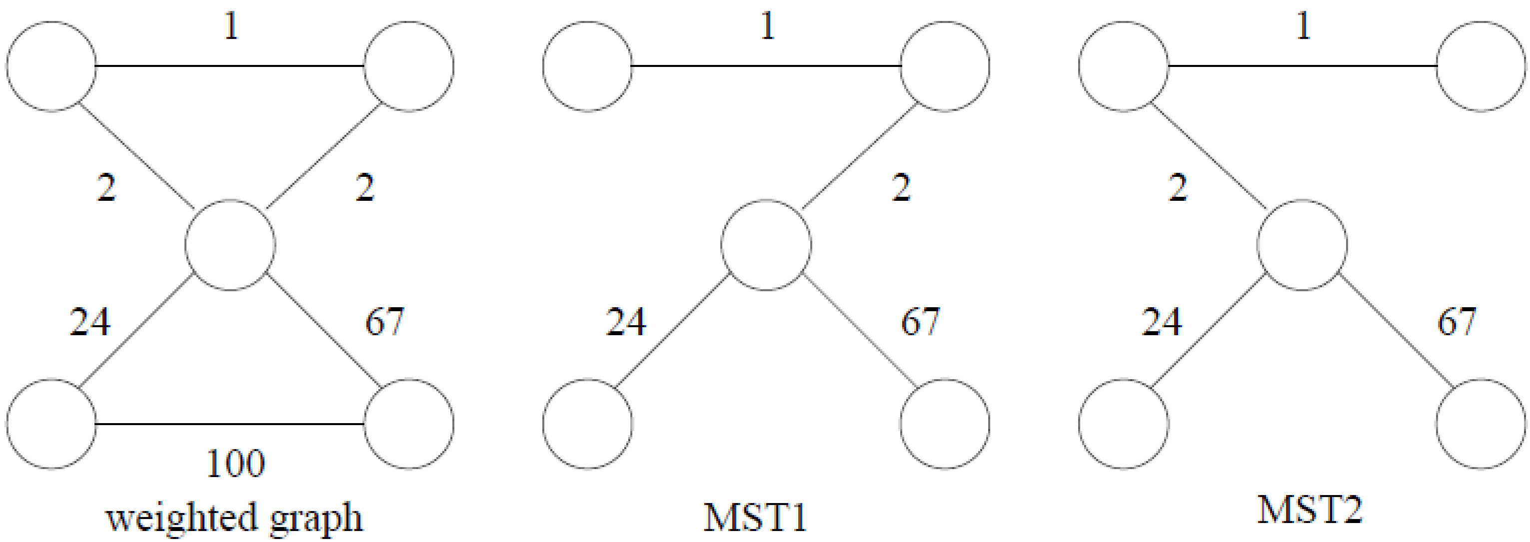
spanning tree T: cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7


a
b
d
c
e
Graph
a
b
d
c
e
spanning tree 1
a
b
d
c
e
spanning tree 2
a
b
d
c
e
spanning tree 3



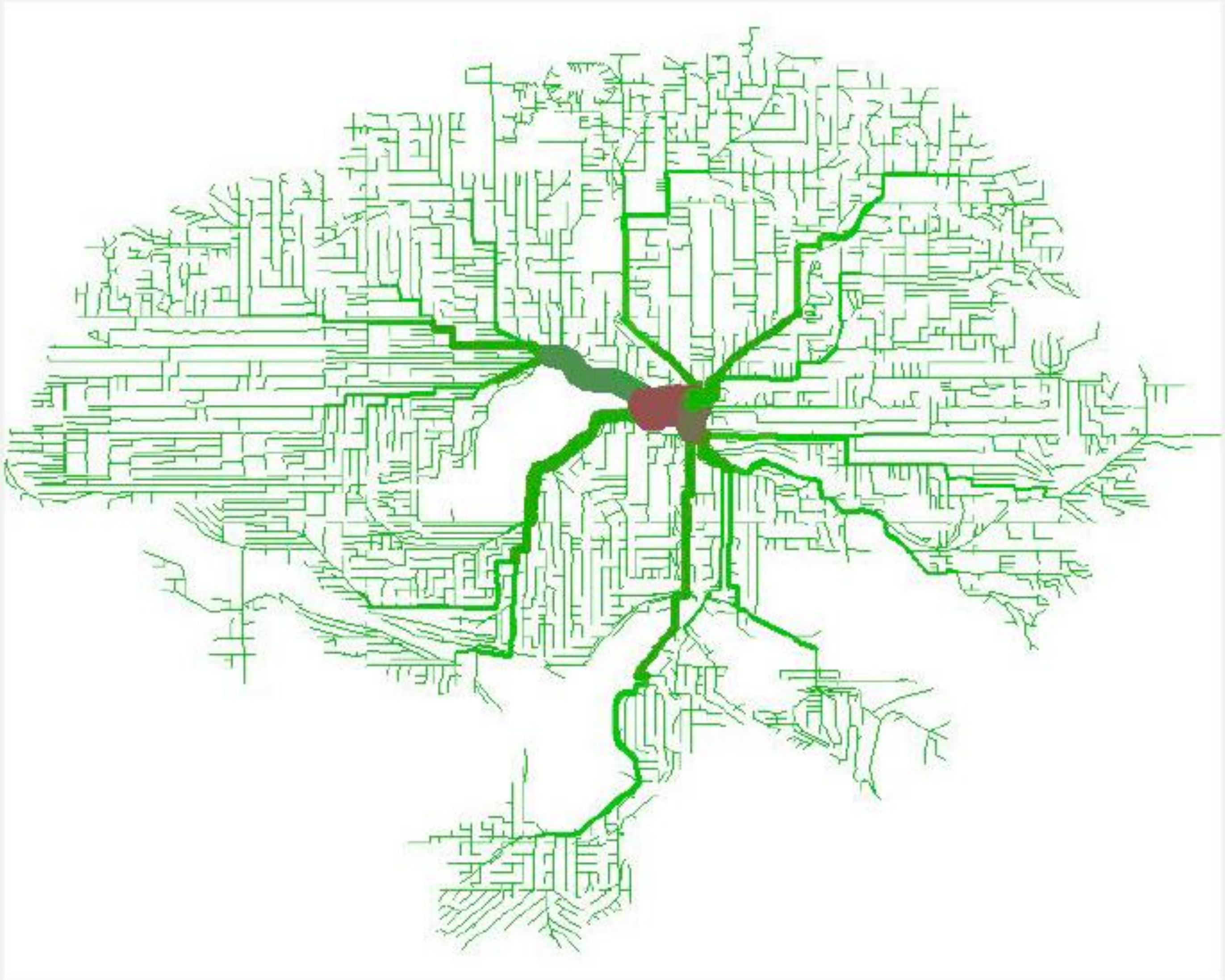
Minimum spanning tree

- The minimum spanning tree may not be unique. However, if the weights of all the edges are pairwise distinct, it is indeed unique.



Network design

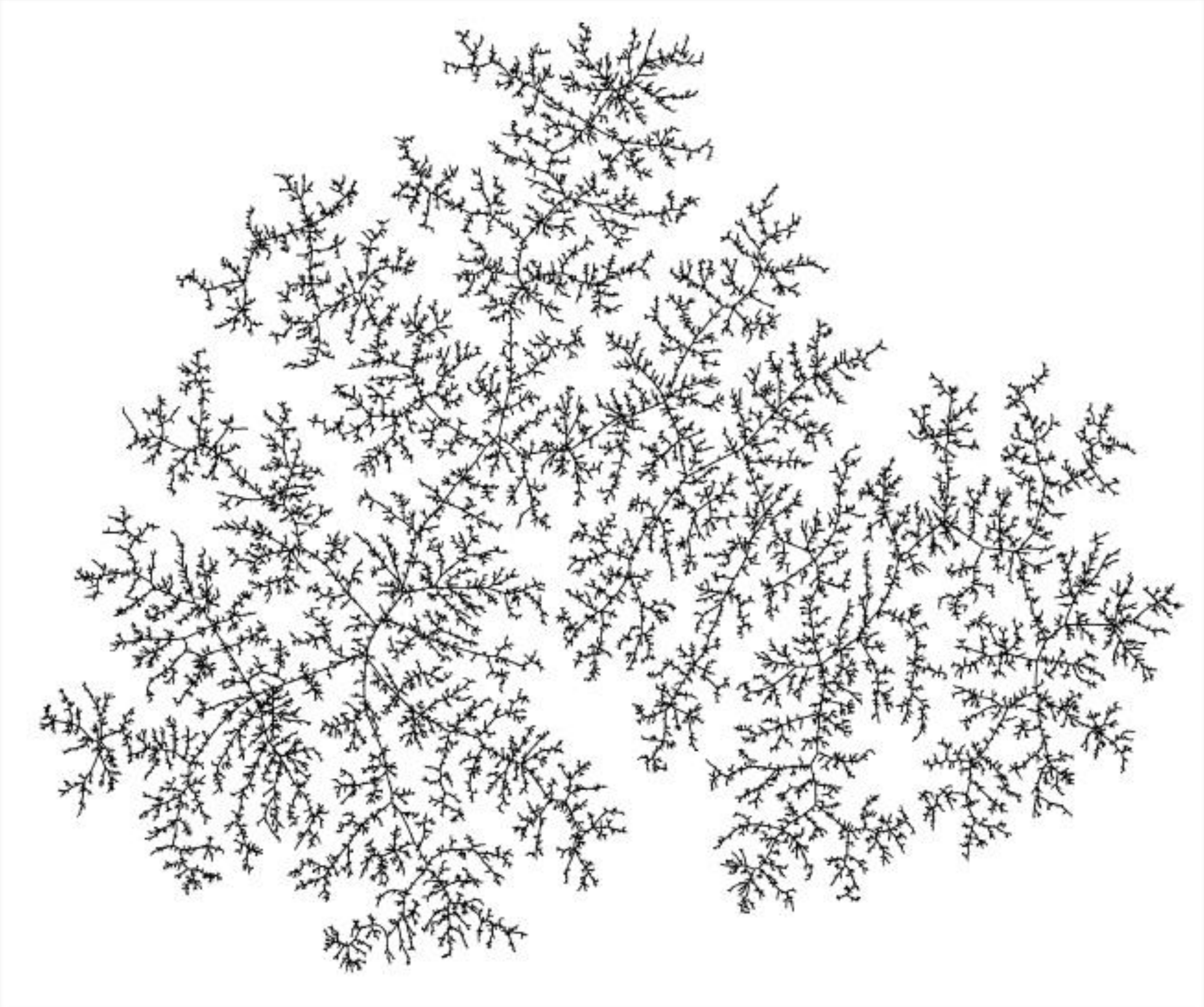
MST of bicycle routes in North Seattle



http://www.flickr.com/photos/ewedistrict/21980840

Models of nature

MST of random graph



http://algo.inria.fr/broutin/gallery.html

Prim’s Algorithm

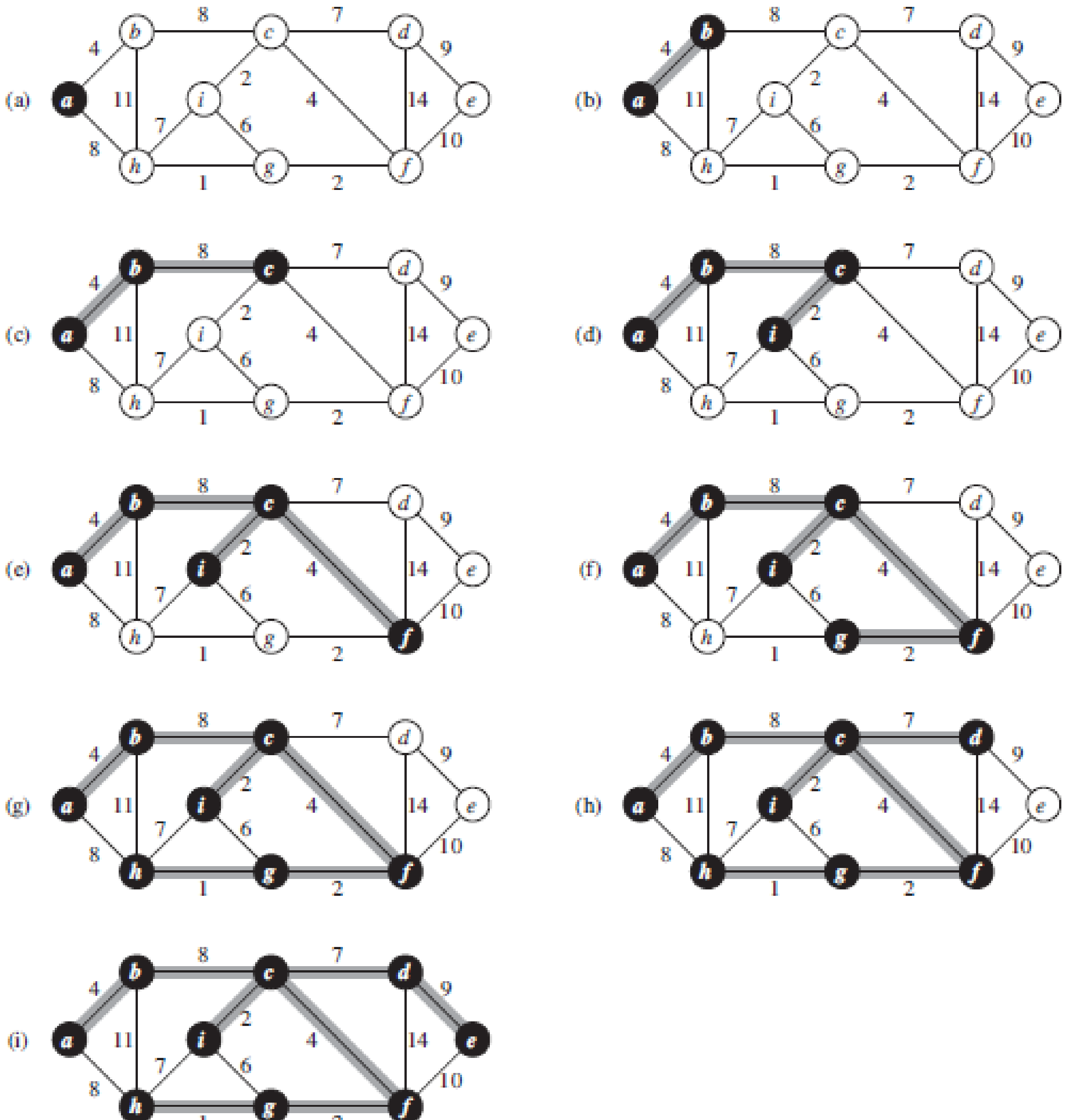
Grow a tree:

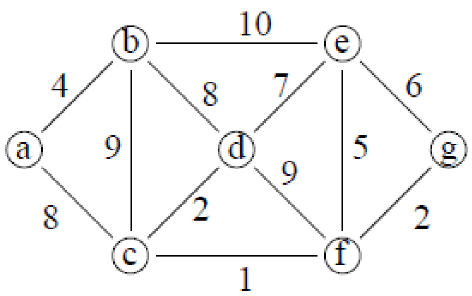
- Start by picking any vertex r to be the root of the tree.
- While the tree does not contain all vertices in the graph find shortest edge leaving the tree and add it to the tree.

Step 0: Choose any element r; set $S = \{r\}$ and $A = \emptyset$. (Take r as the root of our spanning tree.)

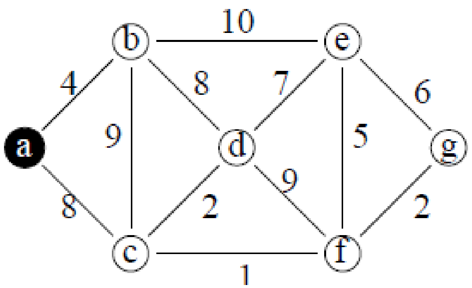
Step 1: Find a lightest edge such that one endpoint is in S and the other is in $V \setminus S$. Add this edge to A and its (other) endpoint to S.

Step 2: If $V \setminus S = \emptyset$, then stop & output (minimum) spanning tree (S, A). Otherwise go to Step 1.


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



Connected graph

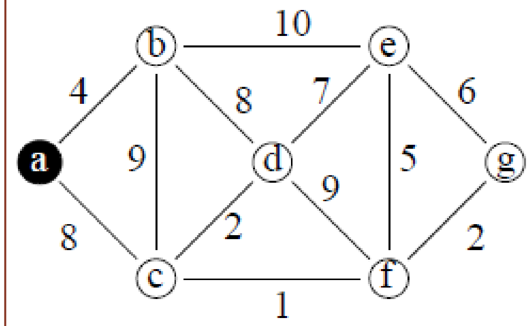


Step 0

S={a}

V \ S = {b,c,d,e,f,g}

lightest edge = {a,b}



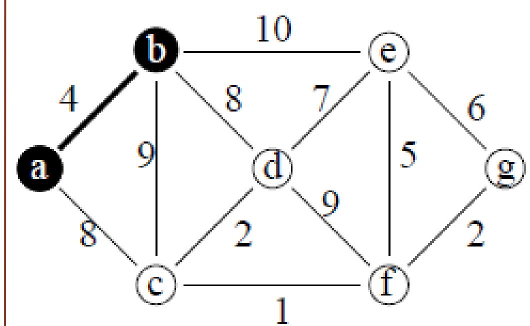
Step 1.1 before

S={a}

V \ S = {b,c,d,e,f,g}

A={}

lightest edge = {a,b}



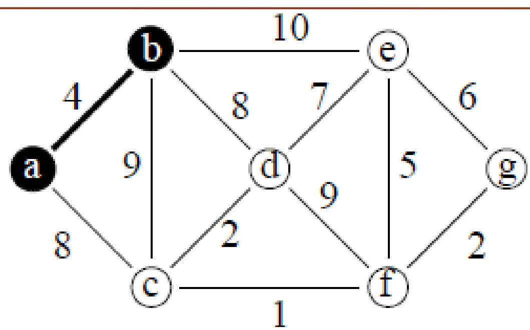
Step 1.1 after

S={a,b}

V \ S = {c,d,e,f,g}

A={{a,b}}

lightest edge = {b,d}, {a,c}



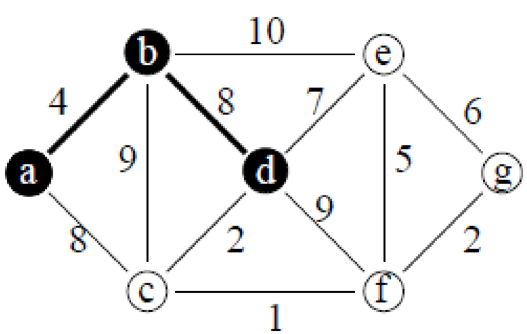
Step 1.2 before

S={a,b}

V \ S = {c,d,e,f,g}

A={{a,b}}

lightest edge = {b,d}, {a,c}



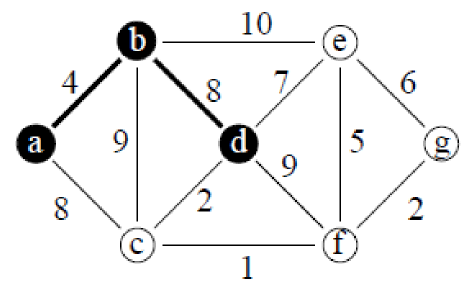
Step 1.2 after

S={a,b,d}

V \ S = {c,e,f,g}

A={{a,b},{b,d}}

lightest edge = {d,c}



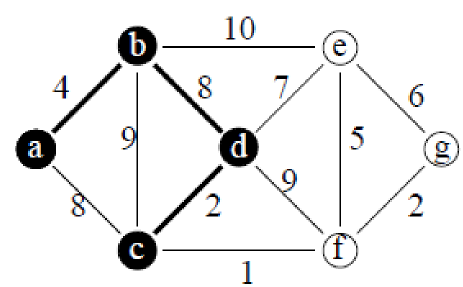
Step 1.3 before

S={a,b,d}

V \ S = {c,e,f,g}

A={{a,b},{b,d}}

lightest edge = {d,c}



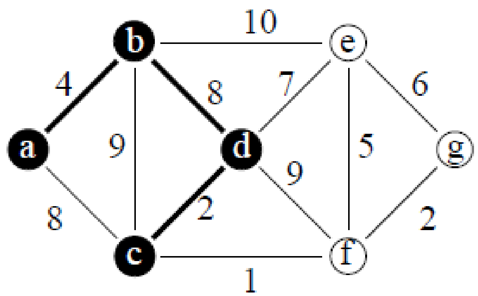
Step 1.3 after

S={a,b,c,d}

V \ S = {e,f,g}

A={{a,b},{b,d},{c,d}}

lightest edge = {c,f}



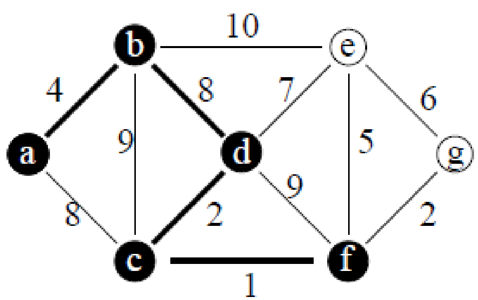
Step 1.4 before

S={a,b,c,d}

V \ S = {e,f,g}

A={{a,b},{b,d},{c,d}}

lightest edge = {c,f}



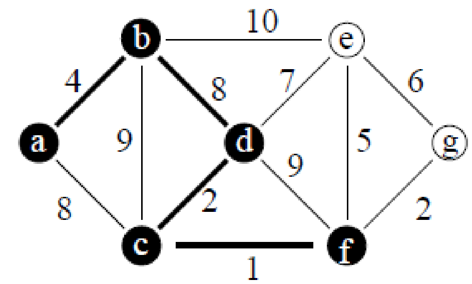
Step 1.4 after

S={a,b,c,d,f}

V \ S = {e,g}

A={{a,b},{b,d},{c,d},{c,f}}

lightest edge = {f,g}



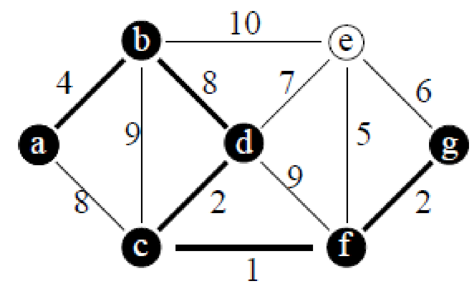
Step 1.5 before

S={a,b,c,d,f}

V \ S = {e,g}

A={{a,b},{b,d},{c,d},{c,f}}

lightest edge = {f,g}



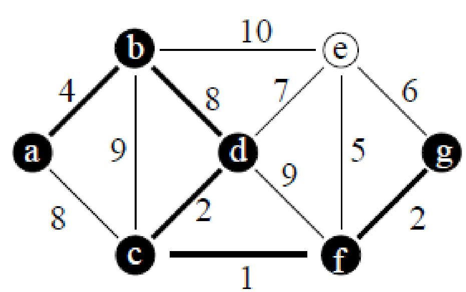
Step 1.5 after

S={a,b,c,d,f,g}

V \ S = {e}

A={{a,b},{b,d},{c,d},{c,f},
{f,g}}

lightest edge = {f,e}



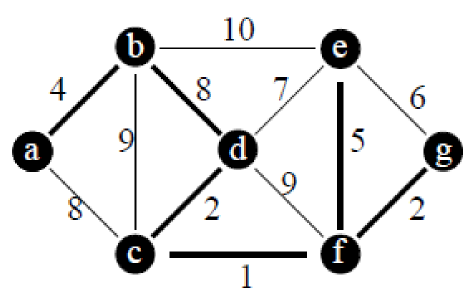
Step 1.6 before

S={a,b,c,d,f,g}

V \ S = {e}

A={{a,b},{b,d},{c,d},{c,f},
{f,g}}

lightest edge = {f,e}



Step 1.6 after

S={a,b,c,d,e,f,g}

V \ S = {}

A={{a,b},{b,d},{c,d},{c,f},
{f,g},{f,e}}

MST completed

Prim's Algorithm (Cont.)

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Prim's Algorithm (Cont.)

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

lines 1–5 in O(V)

While is for O(V)

Extract min O(lgV)

lines 8–11 executes O(E) times

The assignment in line 11 involves an implicit DECREASE-KEY operation on the min-heap, which a binary min-heap supports in O(lg V) time.

Prim's algorithm is O(V lg V + E lgV) = O (E lgV)

More on Page 636

Disjoint Sets Data Structure

- Consider a collection of n people, each of whom belongs to a particular political party. We need a data structure to store the assignment of people to parties. The data structure needs to support just these 2 operations:

- int FIND(int x):
  Given a person, x, returns the leader of x's party.
- void UNION(int x, int y)
  Given two persons, x and y, merges x's and y's parties together under a single leader.

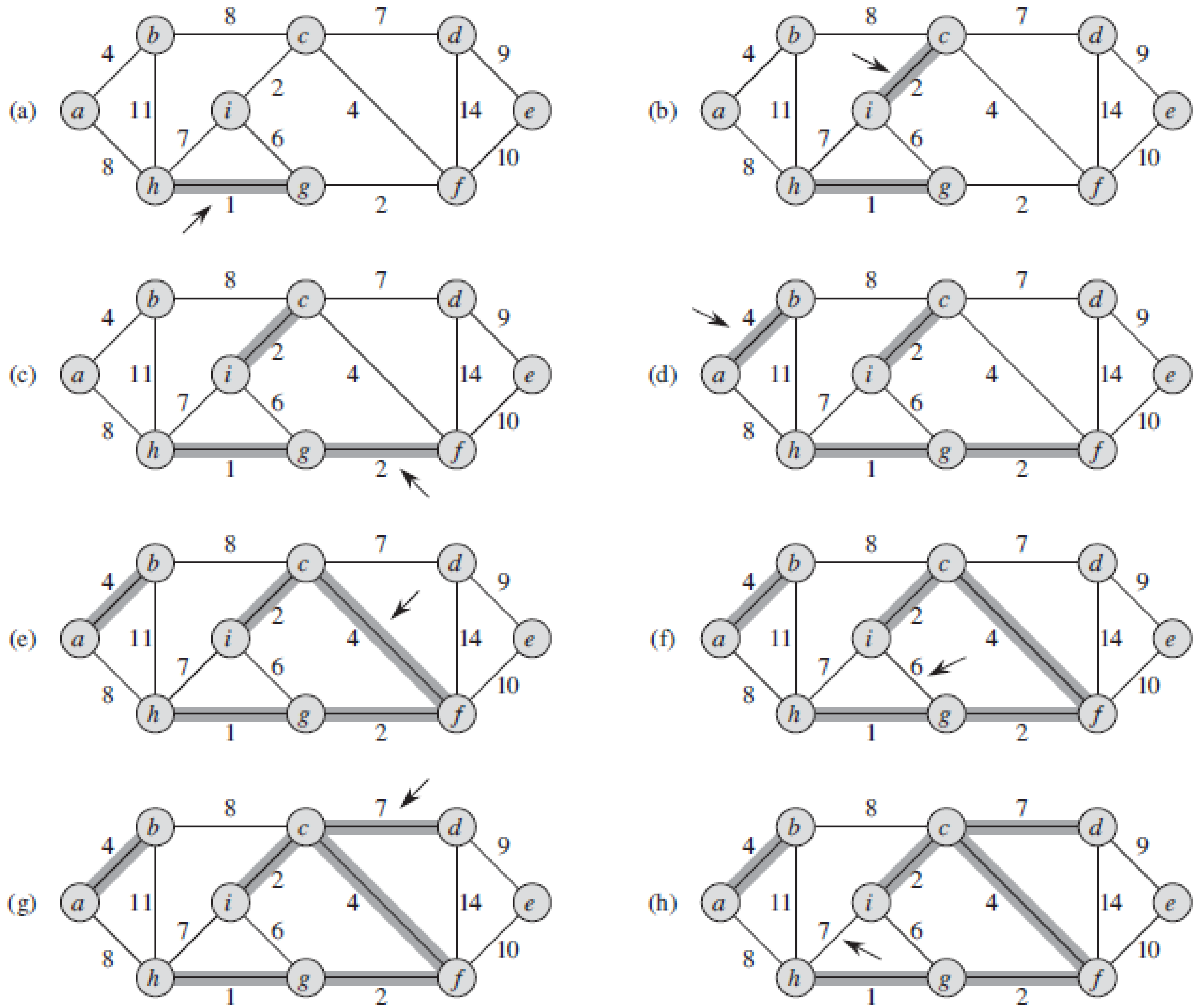
Example 1: Slow UNION/FIND

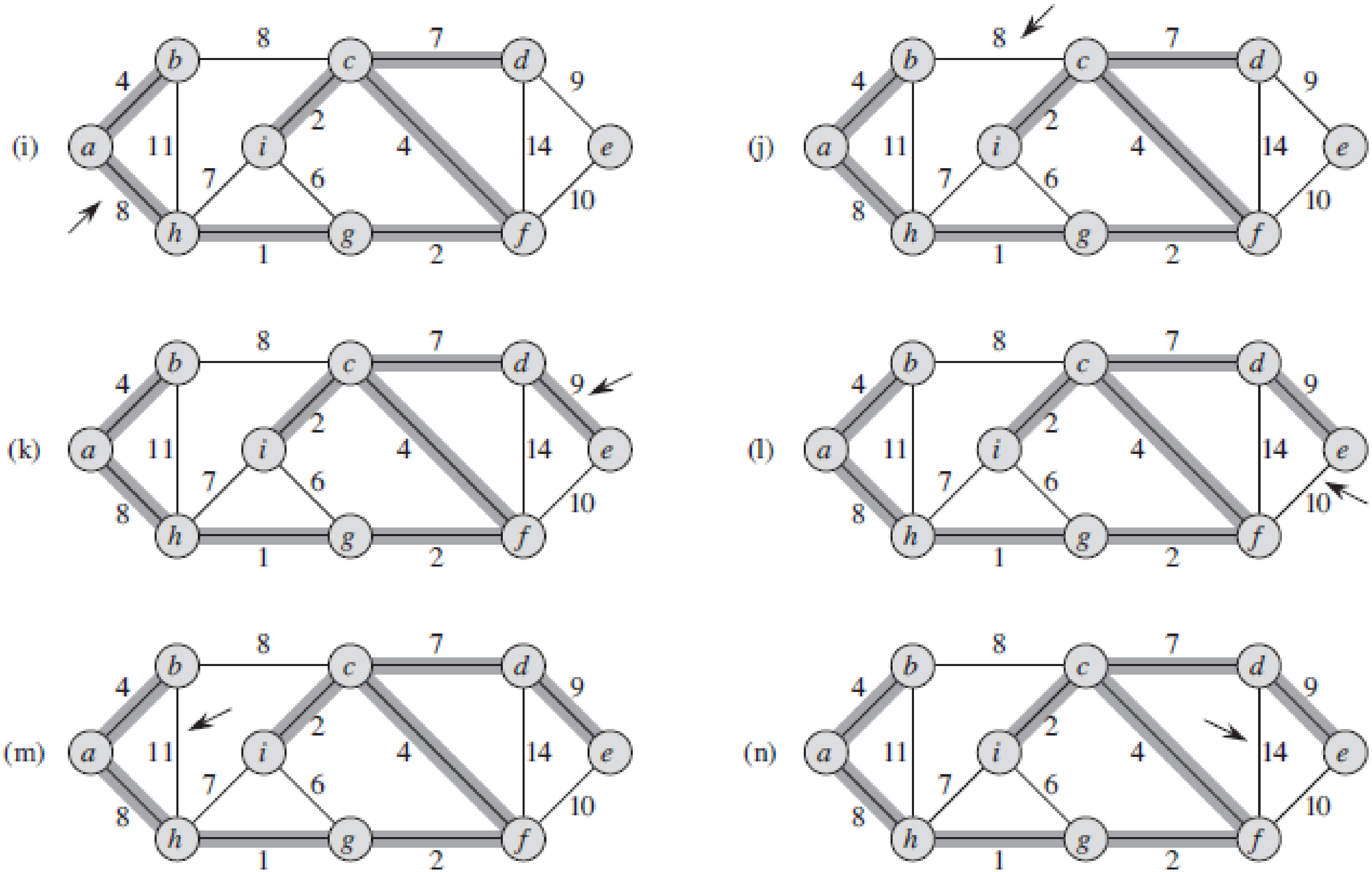
```
int FIND( int x ) {
    if( uf[x] == x ) return x;  // x is the leader
    return FIND( uf[x] );
}

void UNION( int x, int y ) {
    uf[FIND( x )] = FIND( y );
}
```

Kruskal’s Algorithm

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


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



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

Kruskal’s Algorithm Cont.

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

Kruskal algorithm is O (E lg E)

More on Page 633