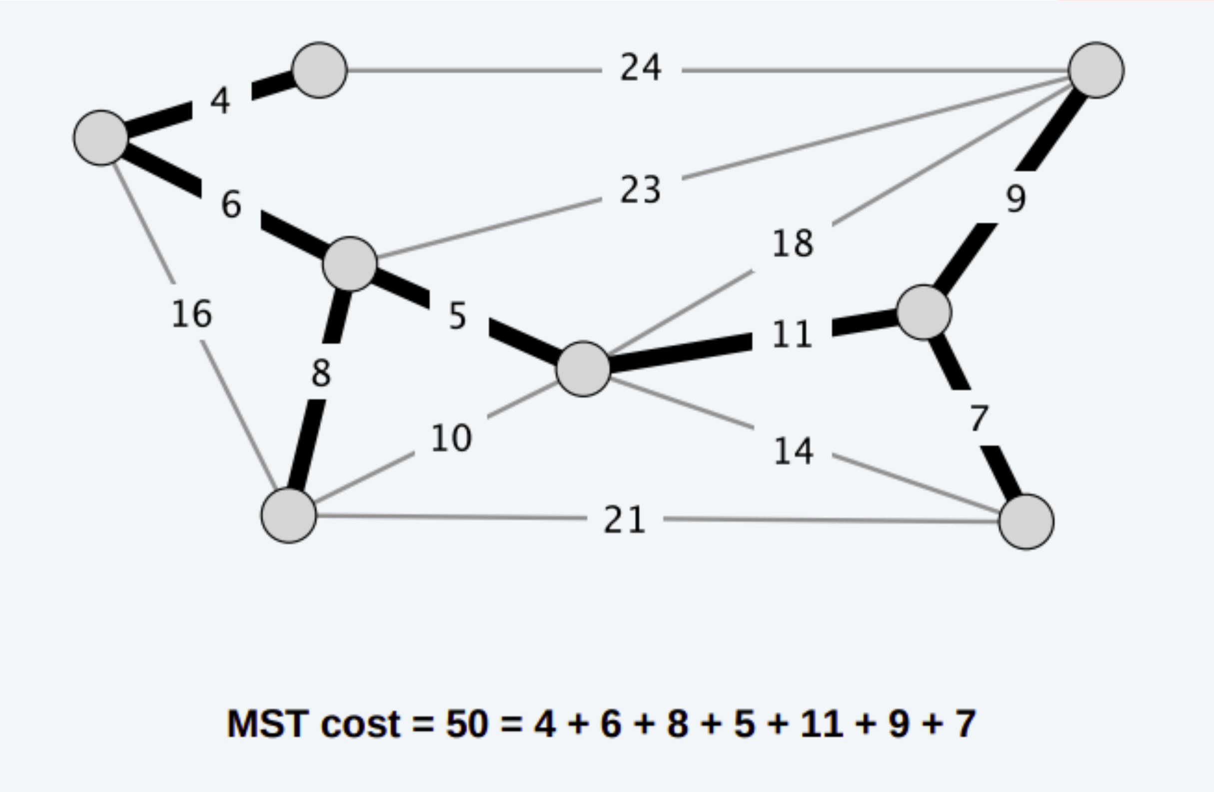
Minimum Spanning Tree

MST (chapter 23)

- Two classic algorithms
  - Kruskal’s
    - Requires Data Structure for Disjoint Sets
  - Prim’s
    - Requires Priority Queue (Heap)



Applications of MST

- Design of networks, including computer networks, telecommunications networks, transportation networks, water supply networks, and electrical grids
- Approximating the traveling salesman problem
- Approximating the multi-terminal minimum cut problem
- Approximating the minimum-cost weighted perfect matching
- Taxonomy
- Cluster analysis
- Etc.

Data Structure for Disjoint Sets

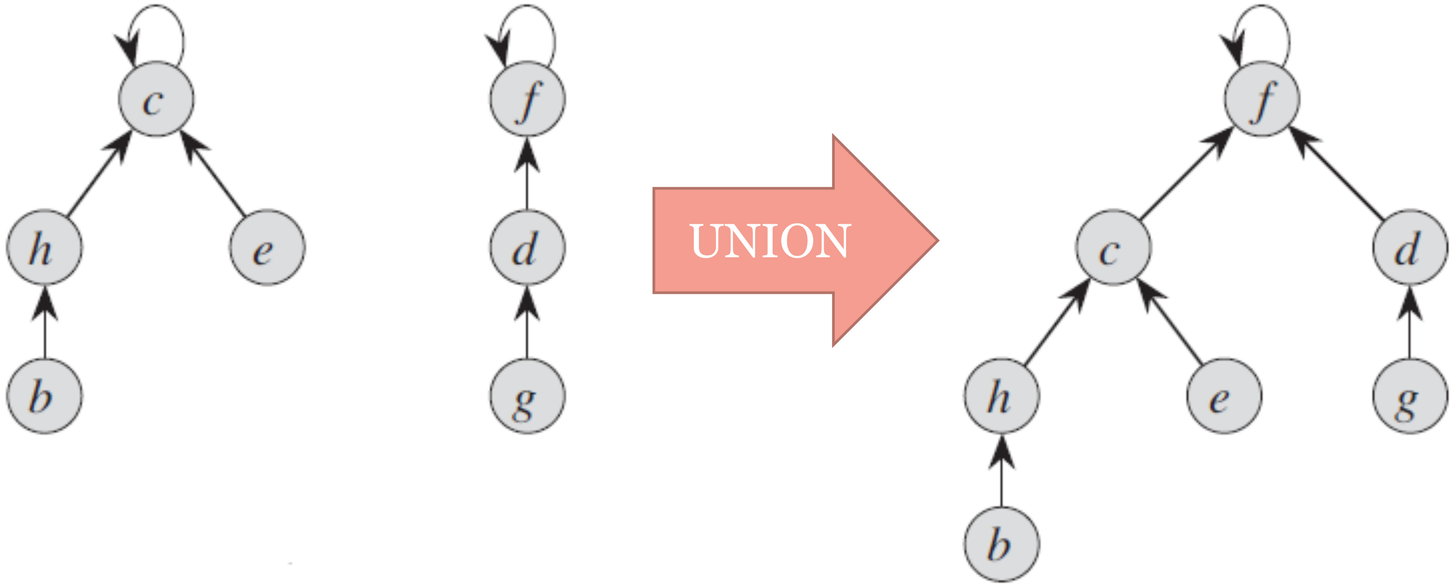
A set is identified by the representative (a member)

Operations

- MAKE-SET(x) creates a new set whose only member.
  - Thus, representative is x.
  - x is not already be in some other set.
- UNION(x, y) unites the dynamic sets that contain x and y, say Sx and Sy, into a new set. The representative of the resulting set is any member.
- FIND-SET(x) returns the representative of the (unique) set containing x.

Disjoint-Set Forests (sec 21.3, page 568)

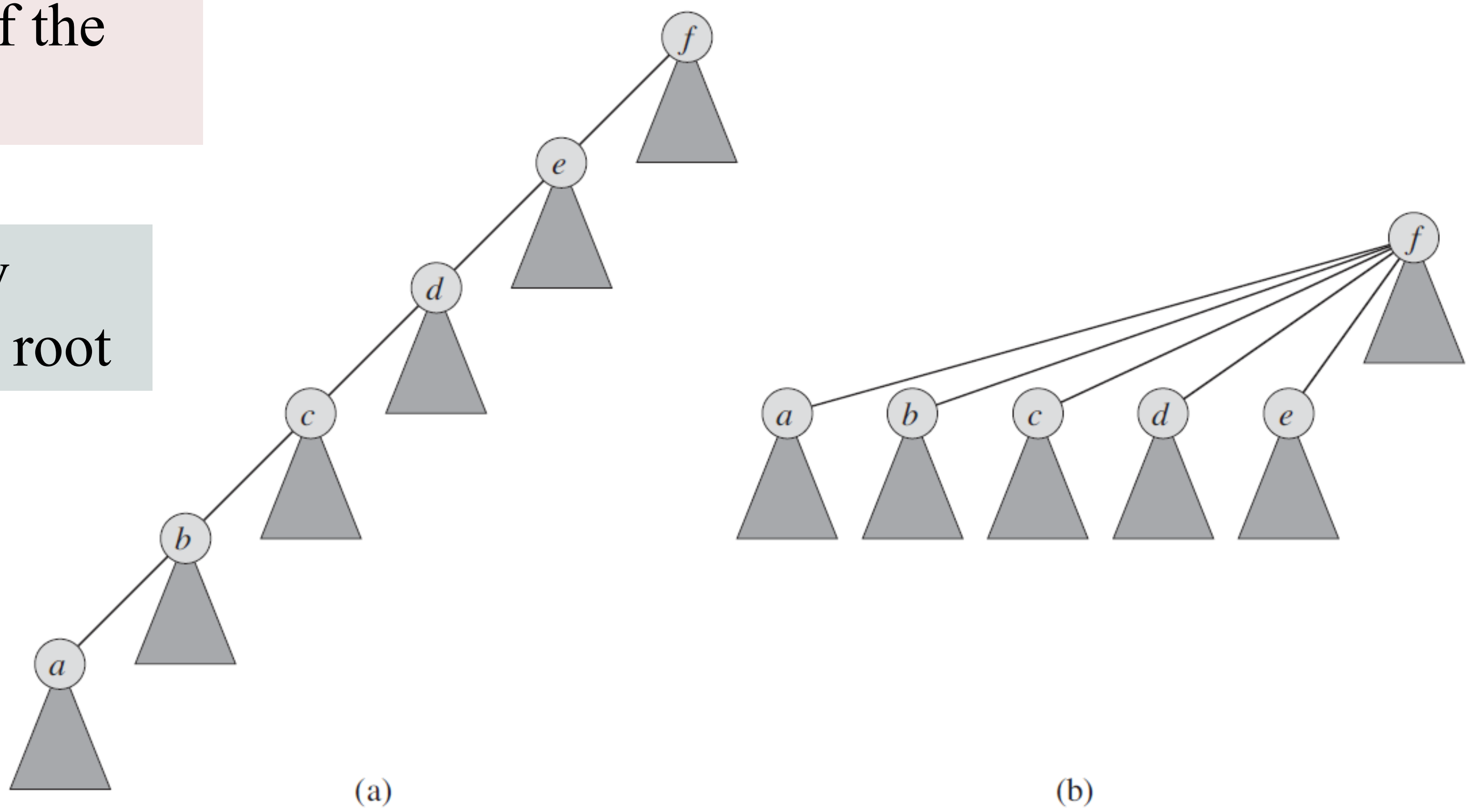
- FIND-SET(b) is c
- FIND-SET(g) is f
- “rank” maintains the approximation of subtree height
  - is not updated by path compression



Union by Rank and Path Compression

When Union, make the root with smaller rank become child of the root with larger rank

When Find-set, update every parent on the find-path to be root



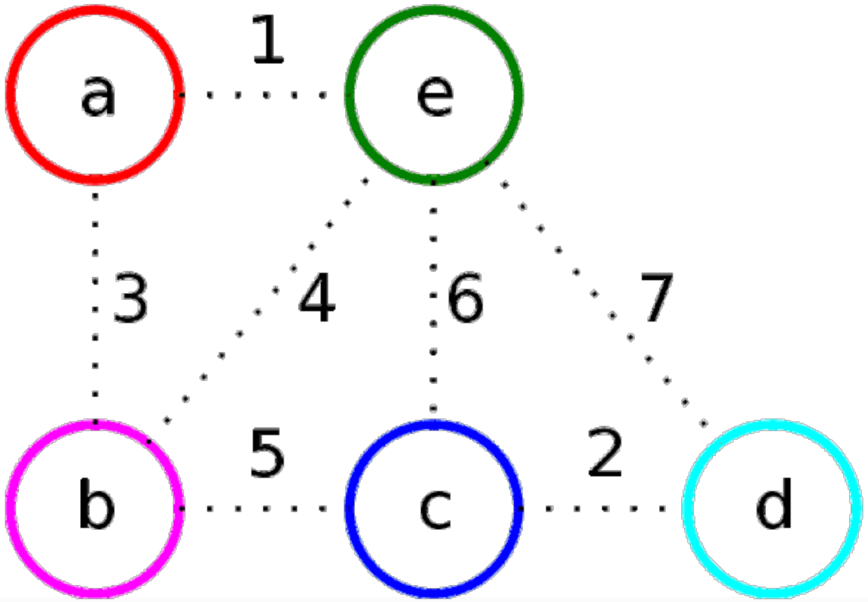
Kruskal's MST

```
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 tree may not contain cycle.
- Disjoint sets are utilized for cycle detection

Kruskal's MST

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



Prim's MST

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ Ø
    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)
```

Priority Queue is utilized for vertex selection as the priority (value of key) of each vertex changes dynamically

Prim’s MST with simple heap

- Simple heap does not support changing value of node’s key.
- A vertex may be inserted multiple times with different values of key.
- Once a vertex is extracted, subsequent copy will be ignored.

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    MST[u] = False
    MinKey[u] = ∞
s.vertex = r
s.key = 0
s.π = NIL
Insert(PQ, s)
while PQ ≠ ∅
    t = Extract-Min(PQ)
    u = t.vertex
    if MST[u] = False
        MST[u] = True
        for each v ∈ G.Adj[u]
            if MST[v] = False and w(u,v) < MinKey[v]
                s.vertex = v
                s.key = MinKey[v] = w(u,v)
                s.π = u
                Insert(PQ, s)
```

Prim’s MST

SET: { }