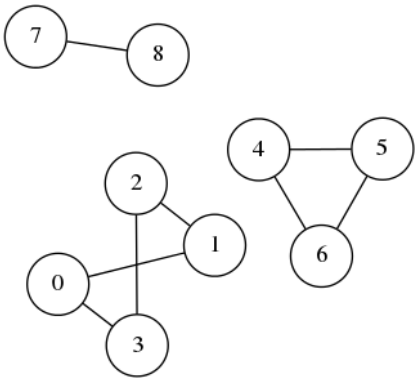
- **Graph:**

  A graph $G = (V, E)$ consists of two sets,

  $V =$ set of vertices (nodes)

  $E =$ set of edges = subset of $(V \times V)$



  - **Connected Graph:**

    An undirected graph is called *connected* if there is a path between every pair of vertices.

    Connected Component:

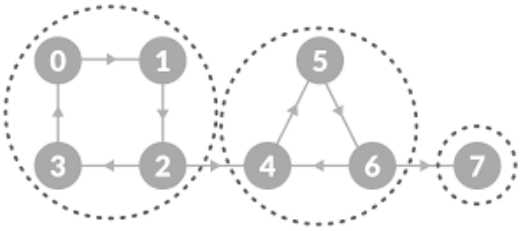
    A *connected component* or simply *component* of an undirected graph is a maximal connected subgraph.

  - **Strongly Connected Graph:**

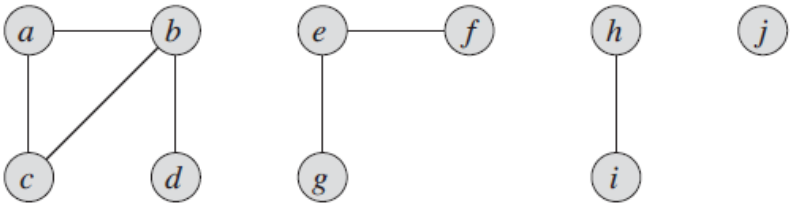
    A directed graph is called strongly connected if there is a directed path between all pairs of vertices.

    Strongly Connected Component:

    A *strongly connected component* or *strong component* of a directed graph is a maximal strongly connected subgraph.



- **Disjoint Set:**
  - A disjoint-set data structure maintains a collection $S = \{S_1, S_2, S_3, \ldots\ldots, S_k\}$ of disjoint dynamic sets.
  - We identify each set by a representative, which is some member of the set.



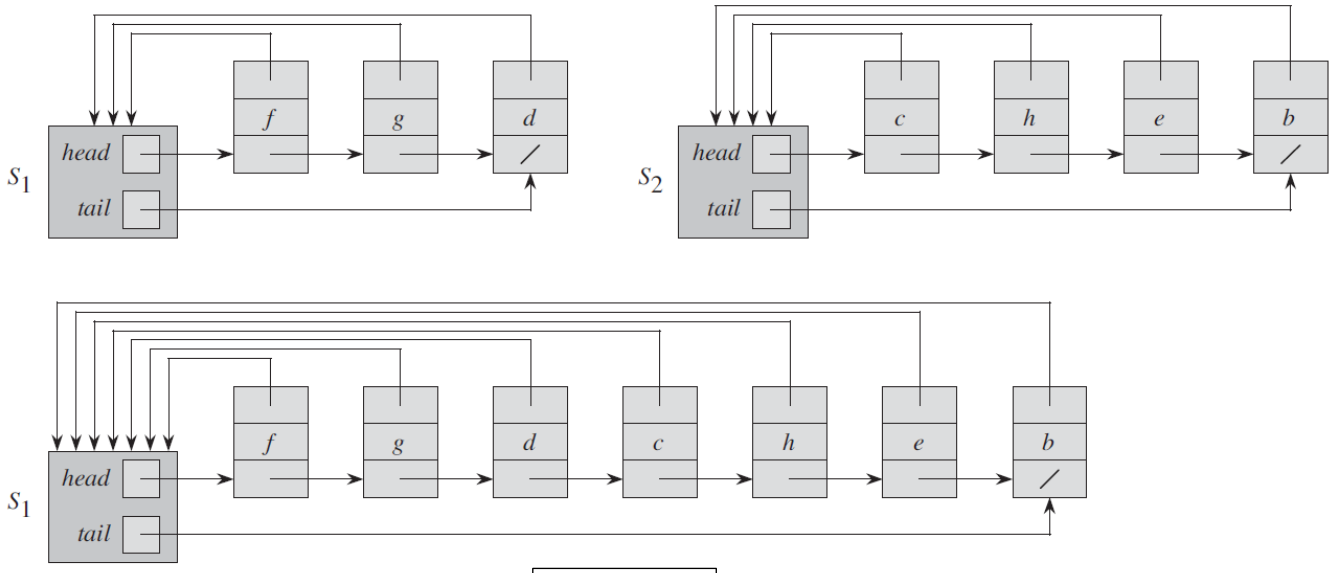
Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$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

- 3 operations:
  1) **MAKE-SET(x)** – creates a new set whose only member is x. x is also the representative of x itself.
  2) **UNION(x, y)** – unites the dynamic sets that contain x and y, say S_x and S_y, into a new set that is the union of these two sets.
  3) **FIND-SET(x)** – returns a pointer to the representative of the set containing x.

- **Linked-list representation:**

  MAKE-SET(x), FIND-SET(x) require $O(1)$ cost but
  a sequence of *m* MAKE-SET(x), FIND-SET(x), UNION(x, y) operations on *n* objects requires $\Theta(n^2)$ cost.

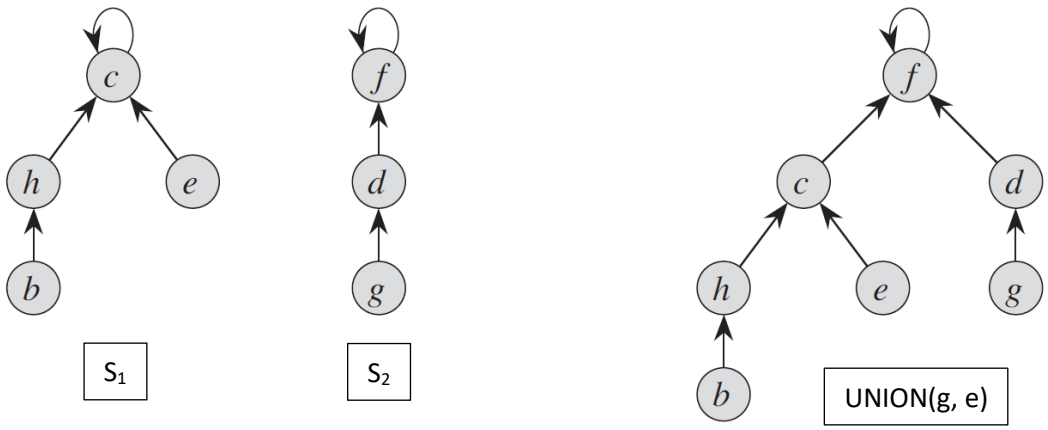
  Weighted-union heuristic – Always append the shorter list onto the longer, breaking ties arbitrarily. A sequence of m MAKE-SET(x), UNION(x, y) and FIND-SET(x) operations on n objects takes $O(m + nlgn)$ cost.



UNION(g, e)

- **Disjoint-set forests:**

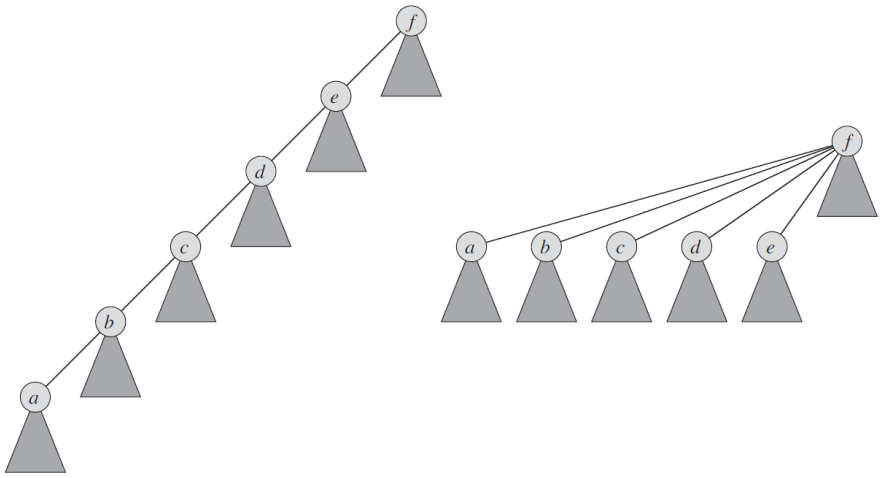
  MAKE-SET(x), UNION(x, y) take $O(1)$ cost and FIND-SET(x) take $O(height)$ cost. n FIND-SET operations will cost $O(n^2)$ cost.



S_1 S_2 UNION(g, e)

To achieve an asymptotically optimal disjoint-set data structure we need to consider two heuristics:

1) **Union by rank:**
   Each node is associated with a **rank**, which is the upper bound on the height of the node (i.e., the height of subtree rooted at the node), then when UNION, let the root with smaller rank point to the root with larger rank.
2) **Path compression:**
   Used in FIND-SET(x) operation, make each node in the path from x to the root directly point to the root. Thus, reduce the tree height.



Algorithms

– worst case running time for m MAKE-SET, UNION, FIND-SET operations on n objects: $O(m.\alpha(n))$ where $\alpha(n) \leq 4$. So nearly linear in m.

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

```
UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if x.rank > y.rank
2       y.p = x
3  else x.p = y
4       if x.rank == y.rank
5            y.rank = y.rank + 1
```

The FIND-SET procedure with path compression is quite simple:

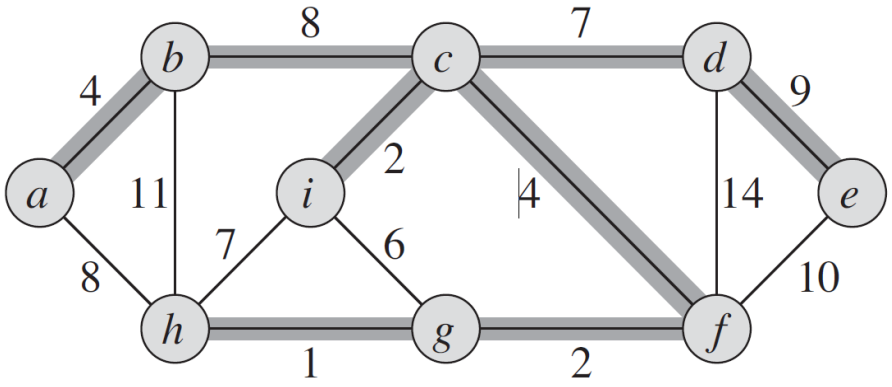
```
FIND-SET(x)
1  if x ≠ x.p
2       x.p = FIND-SET(x.p)
3  return x.p
```

- **Spanning Tree:**

  A tree (i.e. connected, acyclic graph) which contains all the vertices of the graph.

Minimum Spanning Tree:

Spanning tree with the minimum sum of weights.

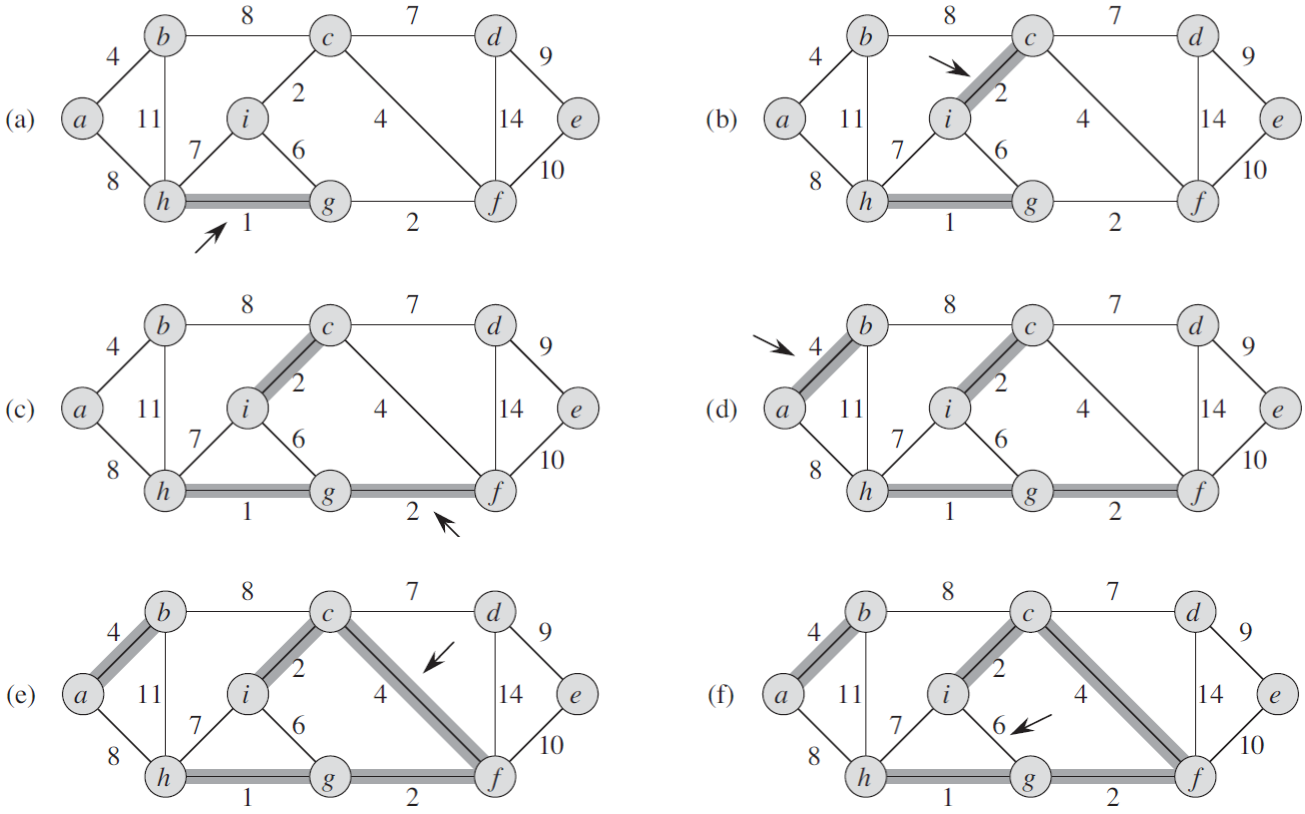


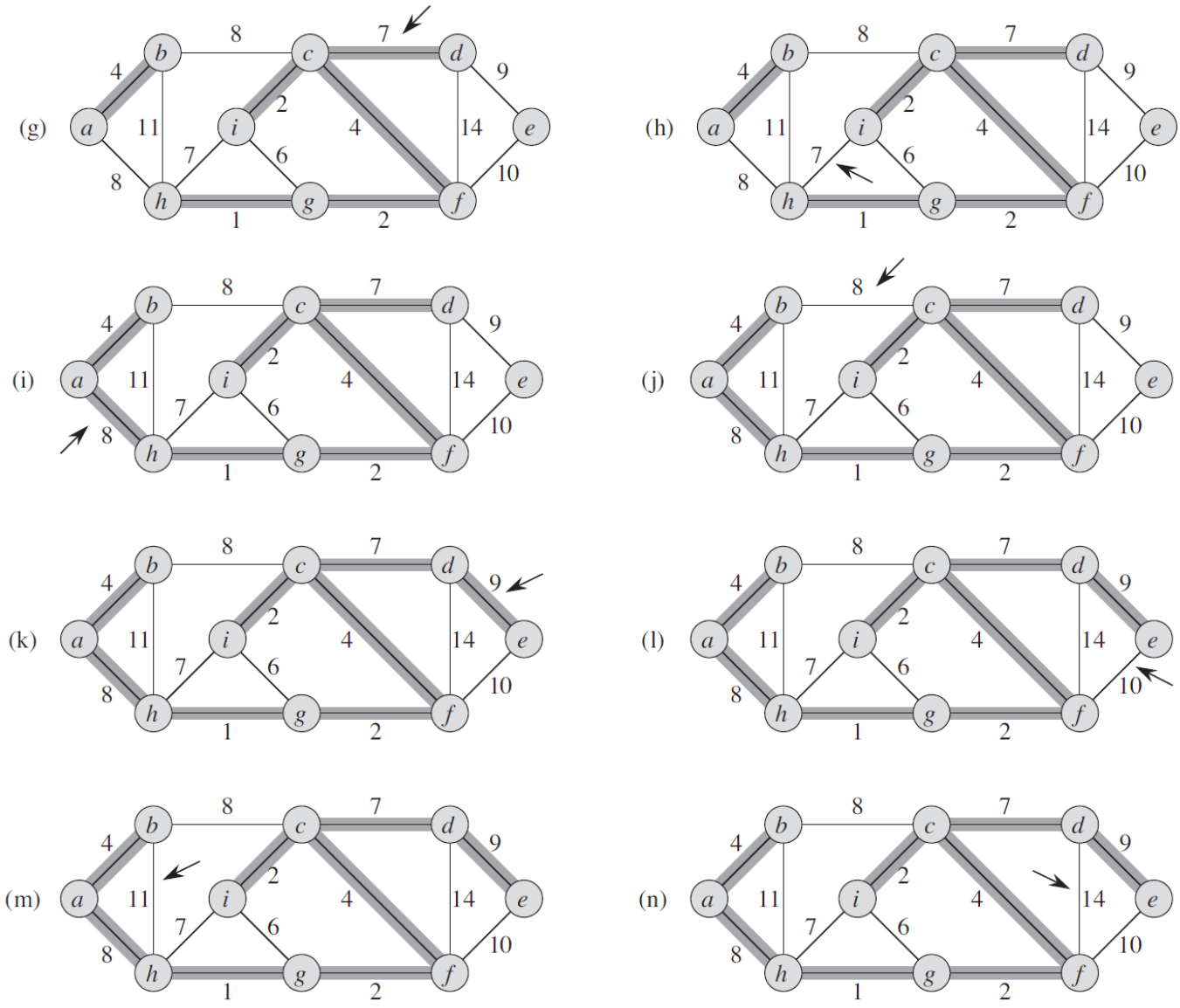
Algorithms:

2 greedy approaches:

- Kruskal's algorithm
- Prim's algorithm

1) **Kruskal's Algorithm:** $O(|E| \lg |V|)$





MST-KRUSKAL(G, w)

```
1  A = ∅
2  for each vertex v ∈ 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) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```