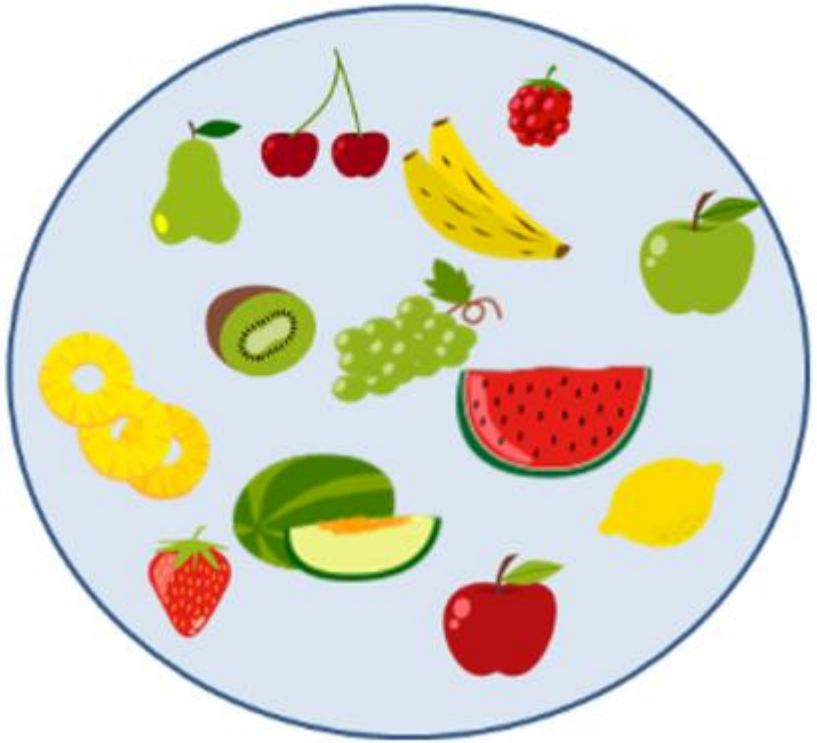
Lecture

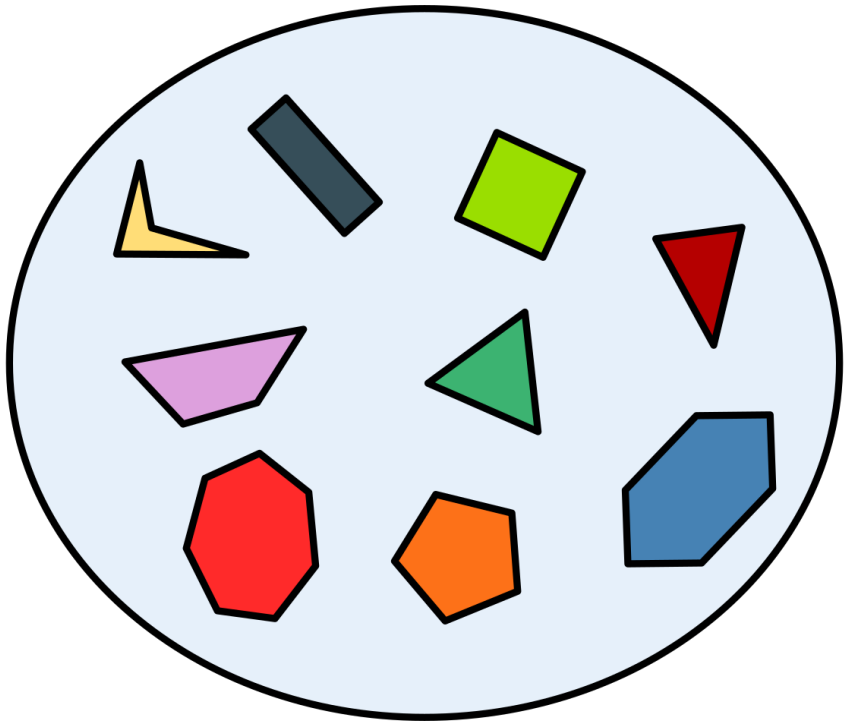
Disjoint sets

Kruskal’s algorithm with Disjoint sets and Activity Selection problems

What Are Disjoint Sets?

Two sets A and B are disjoint if they have NO elements in common. ($A \cap B = \varphi$)





What Are Disjoint Sets?

Two sets A and B are disjoint if they have NO elements in common. ($A \cap B = \varphi$)

A= { set of even numbers}
B={set of odd numbers}

What Are Disjoint Set Data Structures?

A disjoint-set data structure maintains a collection
S = {S1, S2,…,Sk} of disjoint dynamic (changing) sets.

Each set has a representative (member of the set).
Each element of a set is represented by an object (x).

Operations Supported By Disjoint Set Data Structures

MAKE-SET(x): creates a new set with a single member pointed to by x.

UNION(x,y): unites the sets that contain common element(s).

FIND-SET(x): returns a pointer to the representative of the set containing x.

Disjoint Set Data Structures

- Keeps track of a set of elements partitioned into a number disjoint subsets
  - two sets are said to be disjoint if they have no elements in common

- Initially, each element *e* is a set in itself:
  - e.g., { $\{e_1\}$, $\{e_2\}$, $\{e_3\}$, $\{e_4\}$, $\{e_5\}$, $\{e_6\}$, $\{e_7\}$}

Operations: Find

- Determine which set a particular element is in
  - Useful for determining if two elements are in the same set

- Each set has a unique name
  - name is arbitrary; what matters is that find(*a*) == find(*b*) is true only if *a* and *b* in the same set
  - one of the members of the set is the "representative" (i.e. name) of the set
  - {{e_3, e_5, e_7, e_1, e_6} , {e_4, e_2, e_8}, {e_9}}

Operations: Find

- Find(x) – return the name of the set containing x.
  - {{e_3, e_5, e_7, e_1, e_6} , {e_4, e_2, e_8}, {e_9}}
  - Find(e_1) = e_5
  - Find(e_4) = e_8

Operations: Union

- Union(x, y) – Combine or merge two sets x and y into a single set
- Before:
- {{e3, e5, e7} , {e4, e2, e8}, {e9}, {e1, e6}}

- After Union(e5, e1):
- {{e3, e5, e7, e1, e6} , {e4, e2, e8}, {e9}}

Application:
Determining the Connected Components of an Undirected Graph

CONNECTED-COMPONENTS(*G*) //computes connected components of a graph

1 for each vertex *v* in the set *V[G]*

2 do MAKE-SET(*v*)

3 for each edge (*u,v*) in the set *E[G]*

4 do FIND-SET(*u*) ≠ FIND-SET(*v*)

5 then UNION(*u,v*)

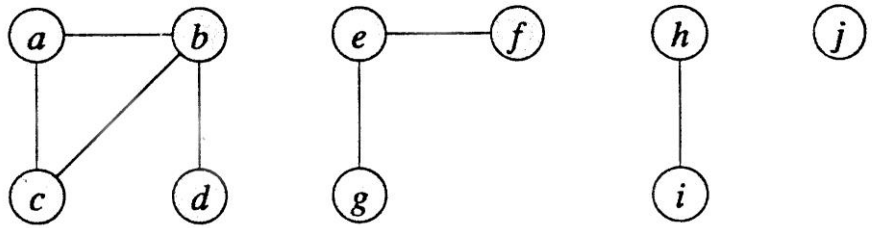
SAME-COMPONENT (*u,v*) //determines whether two vertices are in the same connected component

1 if FIND-SET(*u*) = FIND-SET(*v*)

2 then return TRUE

3 else return FALSE

V[G]= set of vertices of a graph G
E[G]=set of edges of a graph G



(a)

A graph consisting of four connected components

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)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

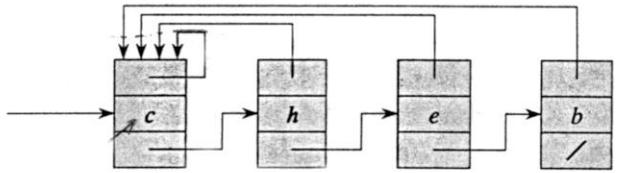
(b)

Figure 22.1 **(a)** A graph with four connected components: $\{a, b, c, d\}$, $\{e, f, g\}$, $\{h, i\}$, and $\{j\}$. **(b)** The collection of disjoint sets after each edge is processed.

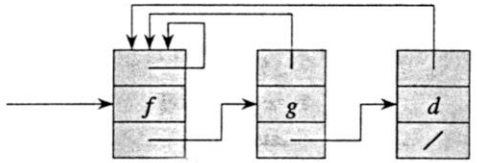
An illustration of how the disjoint sets are computed by CONNECTED-COMPONENTS

Linked-list Representation Of Disjoint Sets

It’s a simple way to implement a disjoint-set data structure by representing each set, in this case set x, and set y, by a linked list.



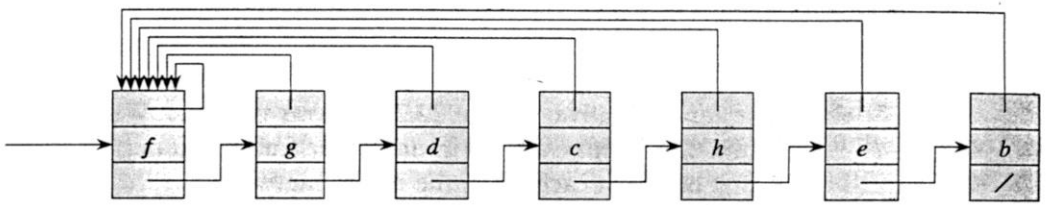
Set x



Set y

(a)

Result of UNION (x,y)



(b)

Disjoint-set Forest Representation

It's a way of representing sets by rooted trees, with each node containing one member, and each tree representing one set.

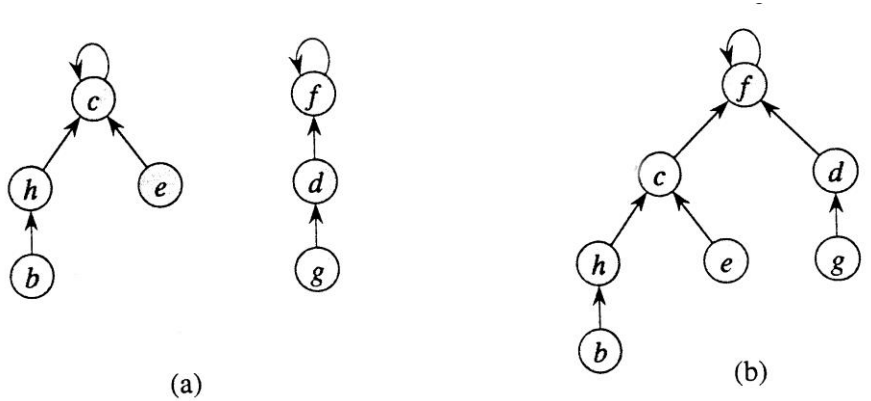
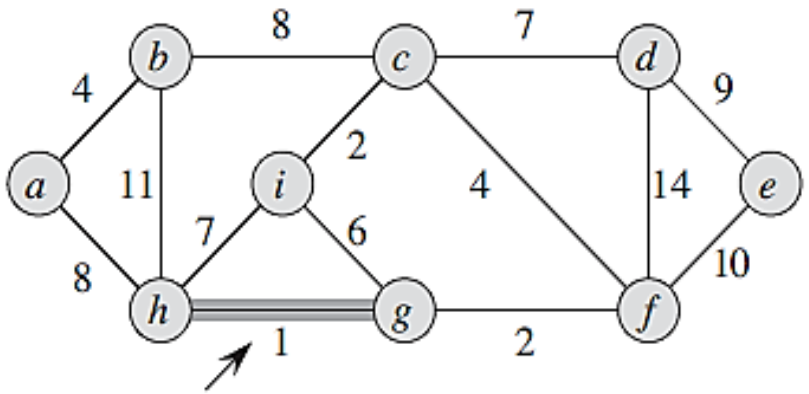


Figure 22.4 A disjoint-set forest. **(a)** Two trees representing the two sets of Figure 22.2. The tree on the left represents the set $\{b, c, e, h\}$, with c as the representative, and the tree on the right represents the set $\{d, f, g\}$, with f as the representative. **(b)** The result of UNION(e, g).

The running time using this representation is linear for all practical purposes but is theoretically superlinear.

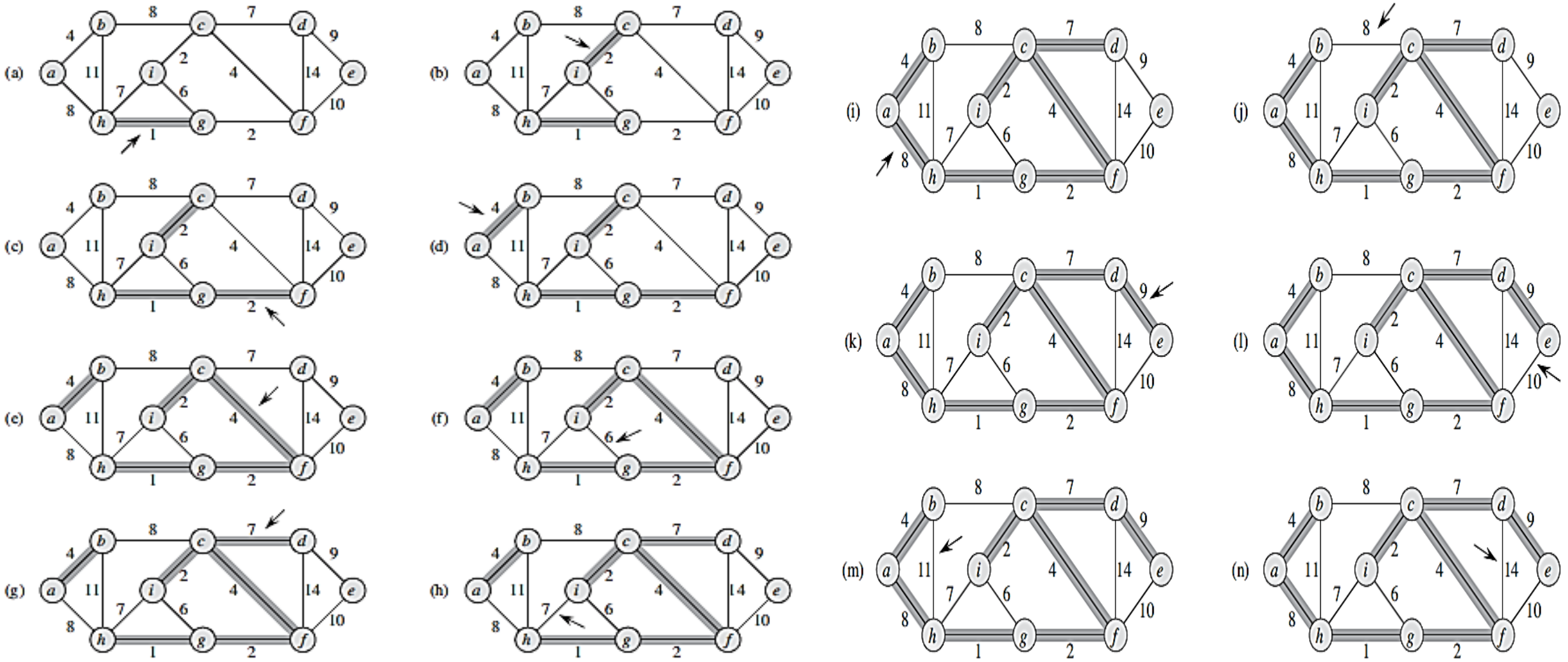
Minimum Cost Spanning Tree: Kruskal's algorithm

```
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 weigh
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

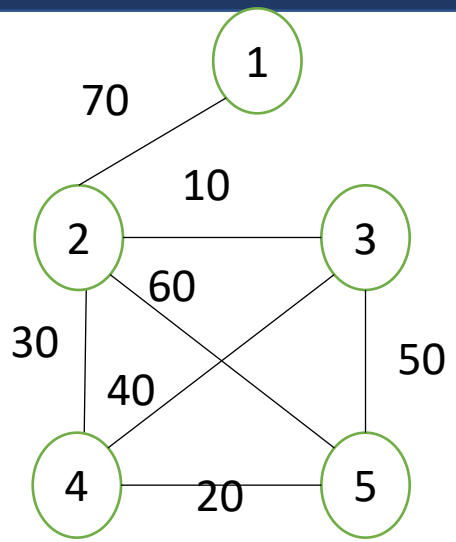


- ✓ Kruskal's algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u,v) of least weight.
- ✓ It uses a disjoint-set data structure to maintain several disjoint sets of elements.
- ✓ Each set contains the vertices in one tree of the current forest.
- ✓ The operation FIND-SET (u) returns a representative element from the set that contains u. It helps in determining whether two vertices u and v belong to the same tree by testing whether FIND-SET (u) equals FIND-SET (v).
- ✓ To combine trees, Kruskal's algorithm calls the UNION procedure.

Kruskal's algorithm: Process

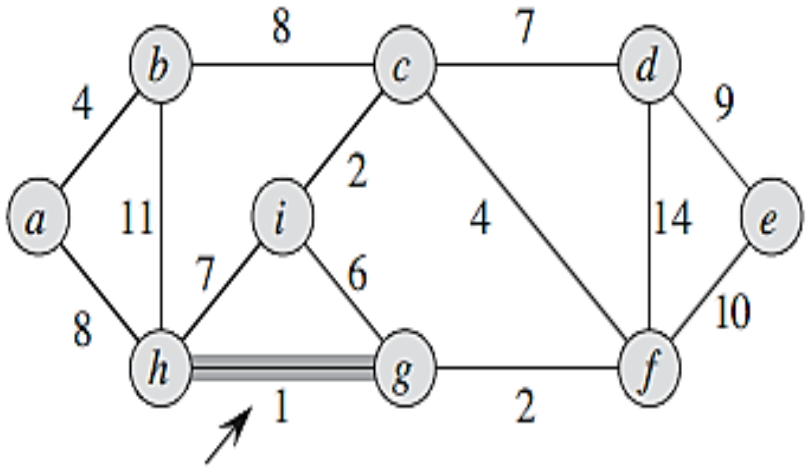


Try for this



Kruskal's algorithm: Example

```
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
```



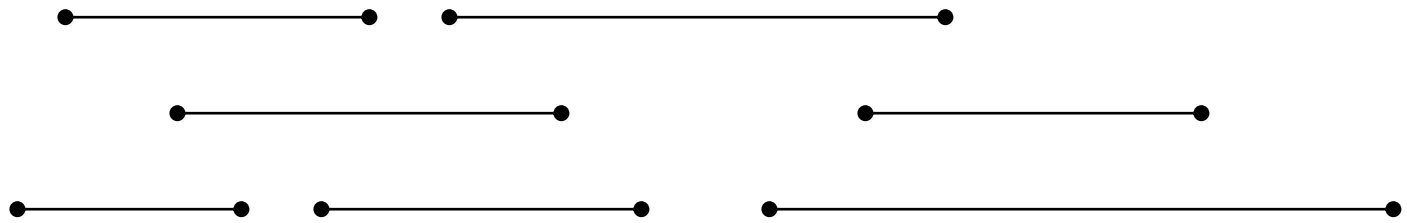
Kruskal's algorithm: Complexity Analysis

```
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
```

Time Complexity = O(ElogV)

The activity selection problem

- Problem: n activities, S = {1, 2, …, n}, each activity i has a start time s_i and a finish time f_i, $s_i \leq f_i$.
- Activity i occupies time interval $[s_i, f_i]$.
- i and j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$.
- The problem is to select a maximum-size set of mutually compatible activities



The activity selection problem

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

The solution set = {1, 4, 8, 11}

Algorithm:

Step 1: Sort f_i into nondecreasing order. After sorting, $f_1 \leq f_2 \leq f_3 \leq \ldots \leq f_n$.

Step 2: Add the next activity i to the solution set if i is compatible with each in the solution set.

Step 3: Stop if all activities are examined. Otherwise, go to step 2.

Time complexity: O(nlogn)

The activity selection problem

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

Solution = {1, 4, 8, 11}

i	s_i	f_i	accept
1	1	4	Yes
2	3	5	No
3	0	6	No
4	5	7	Yes
5	3	8	No
7	6	10	No
8	8	11	Yes
9	8	12	No
10	2	13	No
11	12	14	Yes

The activity selection problem

i	1	2	3	4	5	6	7	8			
s_i	5	3	3	5	3	2	6	8			
f_i	9	5	4	7	8	7	10	15			

Acknowledgements

Information and diagrams of disjoint sets from Cormen, et al, Chapter 22