------------------------------ 21 -----------------------------

Data Structure for Disjoint Sets

21.1 Disjoint-set operations

***Disjoint set data structure*:**

1. a data structure maintains a collection $S=\{S_1, S_2, \ldots, S_k\}$ of **disjoint dynamic** sets.

2. Each set is identified by a **representative**, which is some member of the set. In some applications, it doesn't matter which member is used as the representative; we only care that if we ask the representative of a set without modifying the set between the requests, we get the same answer. In other applications, there may be a representative rule for choosing the representative, such as choosing the smallest member in the set.

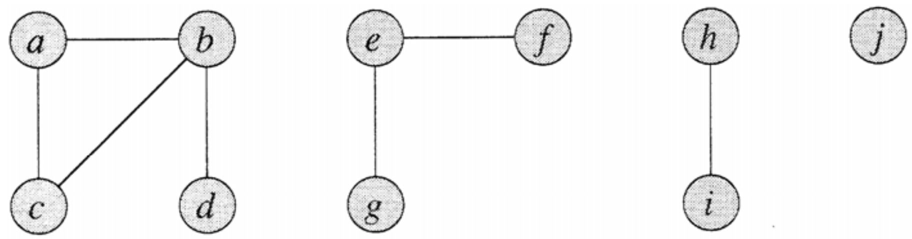
3. The following operations should be supported.

   Make-Set(x): create a new set $\{x\}$.
   Union(x, y): unite the two sets containing x, y.
   Find-Set(x): return a pointer to the
   representative of the set containing x.

* n: number of *Make-Set* operations
m: total number of *Make-Set*, *Union*, and *Find-Set* operation.

* $m \geq n$ and the number of *Union* operations is at most n-1.

An application of disjoint-set data structures



(a)

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

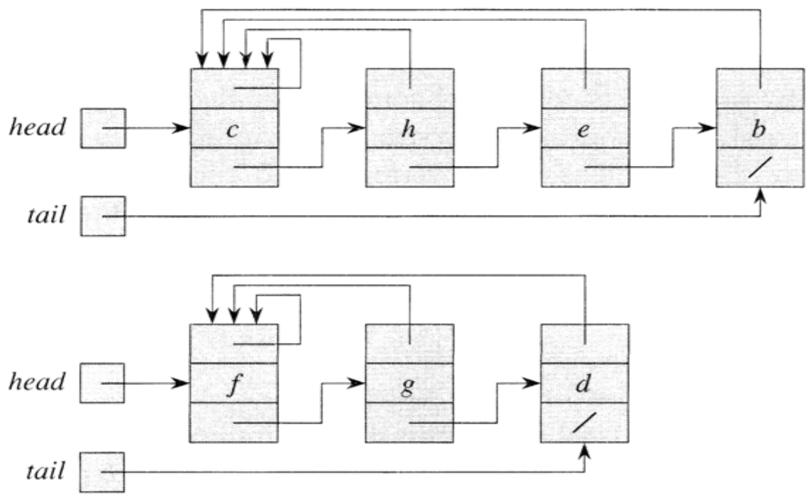
(b)

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ V[G]
     do MAKE-SET(v)
for each edge (u, v) ∈ E[G]
     do if FIND-SET(u) ≠ FIND-SET(v)
           then UNION(u, v)
```

```
SAME-COMPONENT(u, v)
if FIND-SET(u) = FIND-SET(v)
   then return TRUE
   else return FALSE
```

21.2 Linked-list representation

(First object in a list is the representative.)

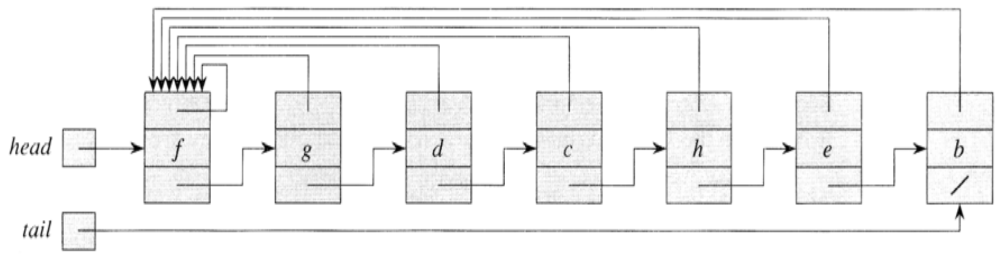


(a)

* Make-Set, Find-Set: $O(1)$ time.

A simple implementation of Union(*x*, *y*)

(Appending the first list onto the second)



(b)

* $O(n^2)$ time for m=2n-1 operations.

Operation	Number of objects updated
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
⋮	⋮
MAKE-SET(x_n)	1
UNION(x_1, x_2)	1
UNION(x_2, x_3)	2
UNION(x_3, x_4)	3
⋮	⋮
UNION(x_{n-1}, x_n)	$n-1$

* Thus, the *amortized time* of each operation is $O(n)$.

A weighted-union heuristic

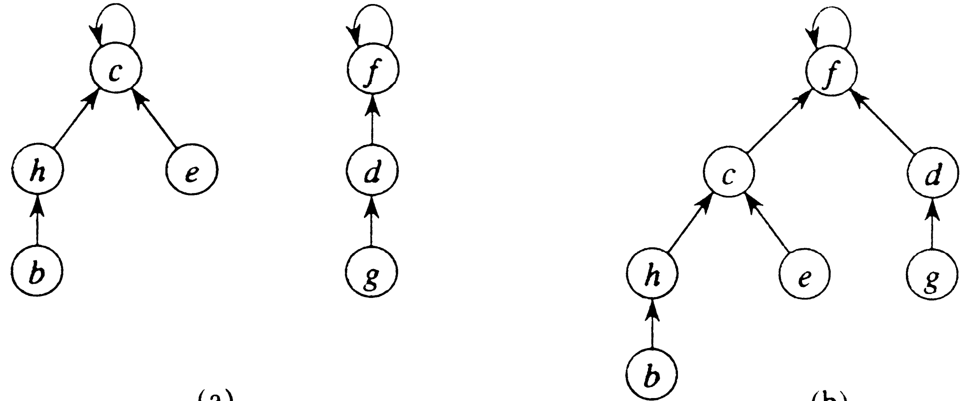
1. Each representative stores the length of the list.
2. Append the smaller list onto the longer.

Theorem 21.1: Using the weighted-union heuristic, a sequence of m *Make-Set*, *Union*, and *Find-Set* operations takes $O(m+n\lg n)$ time, where n is the number of *Make-Set* operations.

Proof: After Make-Set(x) is performed, the list containing x has only one element. At the first time x's representative pointer is updated, the list containing x has at least two elements. Continuing on, we observe that after the k-th time x's representative is updated, the list containing x has at least 2^k elements. Since $k=O(\lg n)$, the time for all *Union* operations is at most $O(n\lg n)$. The time for each *Make-Set* and *Find-Set* operation is $O(1)$. Thus, the theorem holds. Q.E.D.

21.3 Disjoint-set forests

(The root of a tree is the representative.)



(a) (b)

Make-Set(x): $O(1)$ time

Find-Set(x): $O(h)$ time, h is the height of the tree containing x. (**Find path: $x \to$ root**)

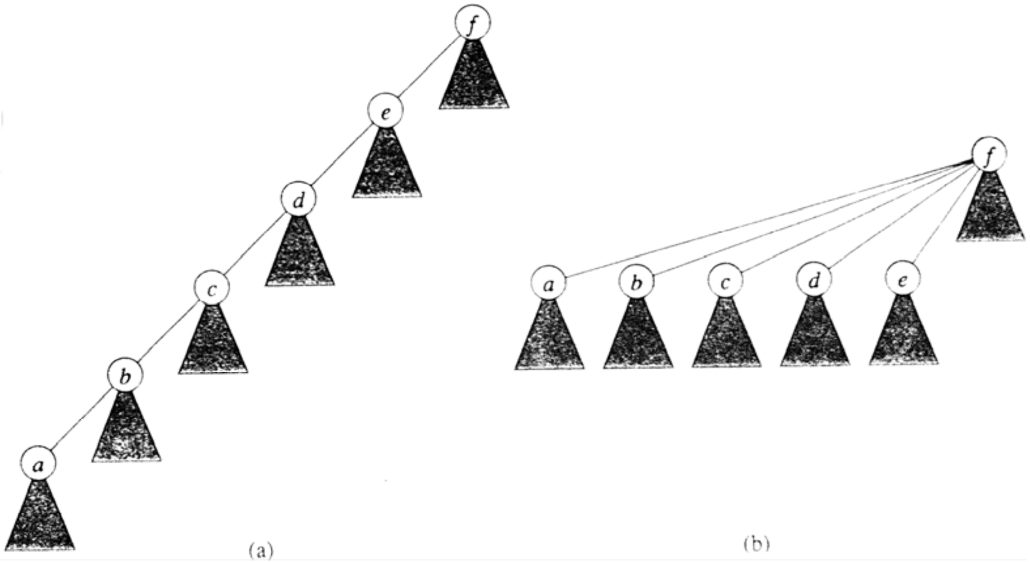
Union(x, y): The root of x points to the root of y. $\to O(h)$ time.

Heuristics to improve the running time

1. ***Union by rank*:** the root of the smaller tree points to the root of the larger tree (according to heights).
   ***rank*[*x*]:** height of x (number of edges in the longest path between x and a descendant leaf)

2. ***Path compression*:** During a Find-Set(x) operation, make each node on the find path point to the root. (It will not change any rank.)

Example: Find-Set(a)



Pseudo-code for disjoint-set forests

```
MAKE-SET(x)
  p[x] ← x
  rank[x] ← 0

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

LINK(x, y)
  if rank[x] > rank[y]
     then p[y] ← x
     else  p[x] ← y
           if rank[x] = rank[y]
              then rank[y] ← rank[y] + 1
```



```
FIND-SET(x)
  if x ≠ p[x]
     then p[x] ← FIND-SET(p[x])
  return p[x]
```

Effect of the heuristics on the running time

1. If only Union-by-rank is used, it can be easily shown that $O(m\lg n)$ time is required.

2. If only path-compression is used, it can be shown (not proved here) that the running time is
$$\Theta(n + f \cdot (1 + \log_{2+f/n} n)),$$
where *n* is the number of *Make-Set* operations and *f* is the number of *Find-Set* operations.

3. When both heuristics are used, the worst-case running time is $O(m\alpha(n))$, where $\alpha(n)$ is the very slowly growing inverse of Ackermann's function. Since $\alpha(n) \le 4$ for any conceivable application, we can view the running time as linear in *m* in all practical situations.

Ackermann’s function and its inverse

* Let $g(i) = \left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\} i$ be a repeated exponentiation.

  (e.g., $g(4) = 2^{2^{2^{2^{2}}}}$.)

* The function $\lg^* n = \min\{i \geq 0: \lg^{(i)} n \leq 1\}$ is

  essentially the inverse of $g(i)$. (e.g., $\lg^* 2^{2^{2^{2^{2}}}} = 5$.)

* The Ackermann’s function: for integer $i, j \geq 1$,

$$
\begin{aligned}
A(1, j) &= 2^j && \text{for } j \geq 1,\\
A(i, 1) &= A(i-1, 2) && \text{for } i \geq 2,\\
A(i, j) &= A(i-1, A(i, j-1)) && \text{for } i, j \geq 2,
\end{aligned}
$$

	$j = 1$	$j = 2$	$j = 3$	$j = 4$
$i = 1$	2^1	2^2	2^3	2^4
$i = 2$	2^2	2^{2^2}	$2^{2^{2^2}}$	$2^{2^{2^{2^2}}}$
$i = 3$	2^{2^2}	$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}16$	$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}16$	$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}16$

$i = 4$ **>>10^{80}**

* Note that $A(2, j) = \left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\} j = g(j)$ for all $j \geq 1$.
  Thus, $A(i, j) \geq g(j)$ for $i \geq 2$.

* The inverse of Ackermann’s function:

$$\alpha(n) = \min\{i \geq 1: A(i, 1) > \lg n\}.$$

* $A(4, 1) = A(3, 2) = g(16) >> 10^{80}$.

* Since $A(4, 1) >> 10^{80}$, we have $\alpha(n) \leq 4$ for all practical cases (unless $\lg n > 10^{80}$).

* $\lg^* n \leq 5$ for all practical cases (unless $n > 2^{65536}$).

* Since $A(i, 1) \geq g(i)$ for $i \geq 4$, $\alpha(n) = O(\lg^* n)$.

Homework: Ex. 21.4-4, Prob. 21-1, 21-3.