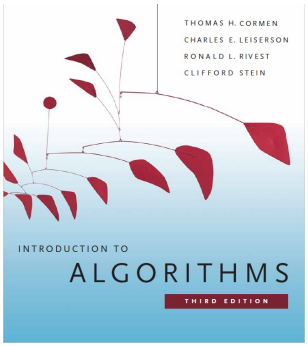
Disjoint Sets

Prepared by Shmuel Wimer
Courtesy of Prof. Dror Rawitz



1

Disjoint-Set Operations

$N = \{1,2,\dots,n\}$ grouped in dynamic sets $\boldsymbol{S} = \{S_1, S_2, \dots, S_k\}$ s.t.

$\forall i \;\; S_i \subseteq N, \;\; \forall i \neq j \;\; S_i \cap S_j = \emptyset$. $r_i \in S_i$ is called **representative**.

Three operations are supported

MAKE(x): for $x \in N$ creates S_x, $|S_x| = 1$, $r_x = x$.

FIND(z): returns $r_x \in S_x$ if $z \in S_x$, or $\emptyset$ if $z \notin N$.

UNION(z, w): returns $S_x \cup S_y$, $r_x = \mathrm{FIND}(z)$ and $r_y = \mathrm{FIND}(w)$, with new representative (r_x or r_y). S_x and S_y are removed from $\boldsymbol{S}$.

2

Example: $N = \{1,2,\dots,8\}$. $\boldsymbol{S}$: $S_2 = \{\mathbf{2}, 6\}$, $S_4 = \{\mathbf{4}, 8\}$, $S_7 = \{1,3,\mathbf{7}\}$.

$\mathrm{FIND}(8) = 4$ since $8 \in S_4$ and $r_4 = 4$.

$\mathrm{UNION}(2,4) \Rightarrow \boldsymbol{S}$: $S_2 = \{\mathbf{2}, 4,6,8\}$, $S_7 = \{1,3,\mathbf{7}\}$.

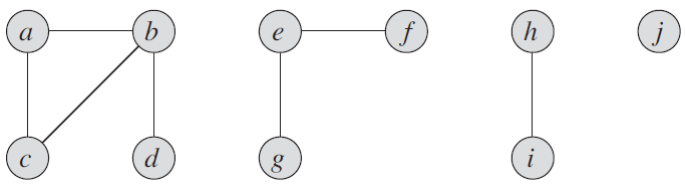
$\mathrm{MAKE}(5) \Rightarrow \boldsymbol{S}$: $S_2 = \{\mathbf{2}, 4,6,8\}$, $S_5 = \{\mathbf{5}\}$, $S_7 = \{1,3,\mathbf{7}\}$.

$\mathrm{FIND}(8) = 2$ since $8 \in S_2$ and $r_2 = 2$.

There are at most n MAKE and $n-1$ UNION operations.

We consider a series of total m MAKE, UNION and FIND operations.

Example: Find the connected components of simple graph $G(V,E)$.



V: $\{a,b,c,d,e,f,g,h,i\}$

E: $\{(a,b),(a,c),(b,d),(e,f),(e,g),(h,i)\}$

G's DS can be adjacency matrix, edge list, or alike.

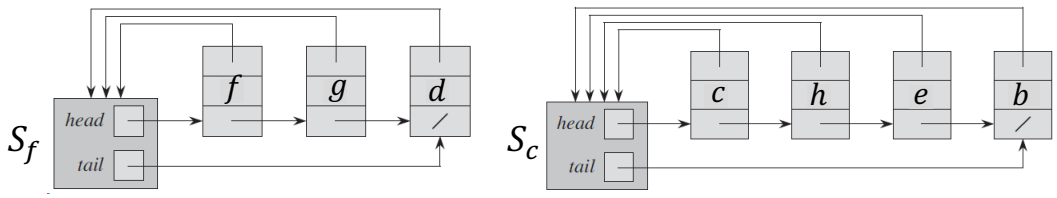
```
CONNECTED−COMPONENT(G(V, E))
    ∀v ∈ V  MAKE(v)  // initially every set is a single vertex
    ∀(u, v) ∈ E
        // check if edge's vertices already in same component
        if FIND(u) ≠ FIND(v)
            UNION(u, v) // merge two component
```

Initially: $S_a = \{a\}, S_b = \{b\}, \ldots, S_j = \{j\}$.

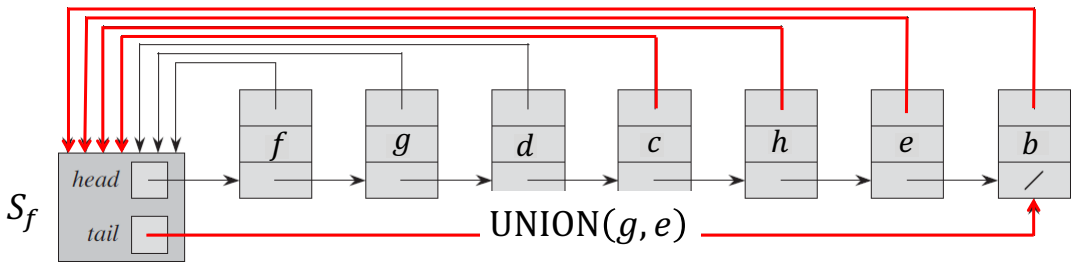
Output: $S_a = \{a, b, c, d\}, S_e = \{e, f, g\}, S_h = \{h, i\}, S_j = \{j\}$.

```
SAME−COMPONENT(u, v)
    return  FIND(u) == FIND(v)
```


Linked List Representation



MAKE and FIND require $O(1)$ time.



Naive UNION implementation takes $O(n)$ due to pointer updates.

operation	# updates
$\text{MAKE}(x_1)$	1
$\vdots$	$\vdots$
$\text{MAKE}(x_n)$	1
$\text{UNION}(x_2, x_1)$	1
$\vdots$	
$\text{UNION}(x_i, x_{i-1})$	i-1
$\vdots$	
$\text{UNION}(x_n, x_{n-1})$	n-1

Given n objects $\{x_1, x_2, \dots, x_n\}$, the following $2n-1$ operation sequence results in

$\sum_{i=1}^{n} 1 + \sum_{i=1}^{n-1} i = \Theta(n^2)$

total number of object updates, hence $\Theta(n^2)$ run time, with $\Theta(n)$ amortized time per operation.

Weighted-Union Heuristic

Weighted-union heuristic append always shorter list to longer, resulting in fewer object updates.

Each set stores its size, which upon UNION is $O(1)$ time to update.

Theorem 1: Using linked list DS with weighted-union heuristic, sequence of total m MAKE, UNION, and FIND, n of which are MAKE, takes $O(m + n \log n)$ time.

Proof: Consider the pointer updates of an object x along the entire sequence of m operations.

UNION updates x's pointer only if it belongs to a set not larger than the counterpart, case where x's set size is at least **doubled**.

Since there are n objects, size doubling (and hence pointer update) can occur at most $\lceil \log n \rceil$ times.

Hence UNION time is bounded by $O(n \log n)$ plus $O(n)$ due to list concatenations.

Since each MAKE and FIND takes $O(1)$, the time for the entire sequence of m operations is $O(m)$.

The total time is thus $O(m + n \log n)$. ■

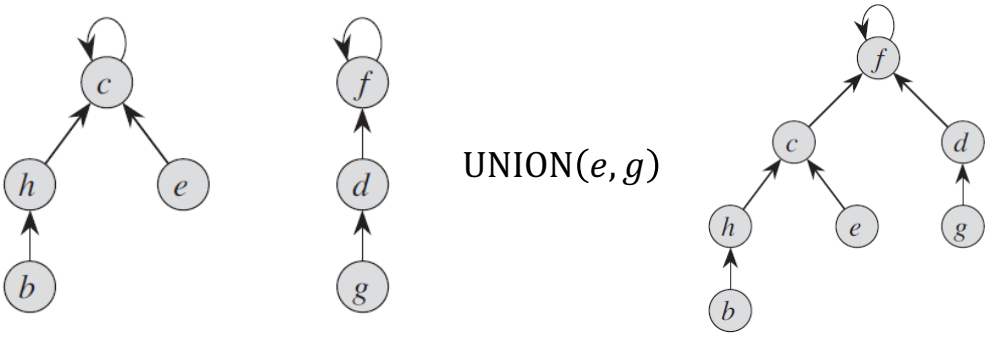
Disjoint-Set Forests

Run time speedup is achieved by rooted trees, with each node containing one member and each tree representing one set.

Each member points only to its parent. The root contains the set's representative and it is its own parent.

The speedup is enabled by two heuristics: **union by rank** and **path compression**.

Disjoint-set forests is asymptotically optimal disjoint-set DS, achieving almost linear run-time (shown later).



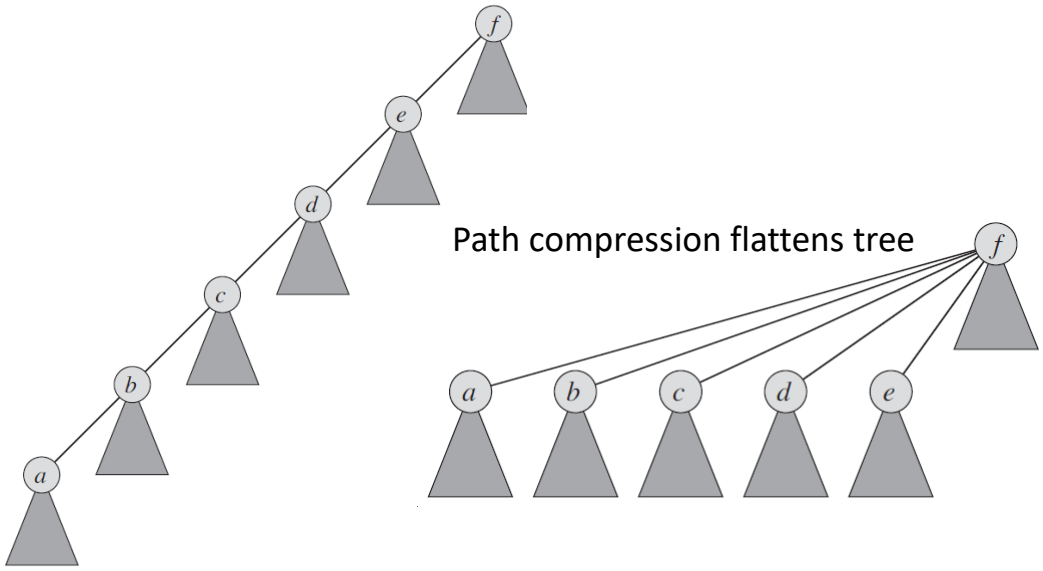
MAKE creates one-node tree, setting its rank to 0.

FIND follows pointers up to root.

UNION causes pointer of one tree to point to other's root. Poor implementation may create a tree that is linear chain of n nodes.


```
UNION(x, y)   // one root points to the other
    LINK (FIND(x), FIND(y))
```

```
LINK(x, y)   // implements union by rank
    if x.rank > y.rank   // rank bounds tree depth
        y.p = x   // connect shallower tree to deeper
    else x.p = y
        if x.rank == y.rank   // only equal ranks increases
            y.rank = y.rank + 1   union's root rank
```




```
FIND (x)  // simultaneous find and path compression
    if x ≠ x.p  // check for root
        x.p = FIND(x.p)  // recursive upward pass
    return x.p  // return the root which points to itself
```

FIND is two-pass method. As it recur, it makes one pass upwards to find the root.

As the recursion unwinds, it makes a second pass back downwards to update each node to point directly to the root.

Lemma 2: Number of nodes in a tree T of height h satisfies $|T| \geq 2^h$.

Proof: By induction on series of operations.

Base: Let x and y be single node trees created by MAKE, where $x.rank = y.rank = 0$. Then $|T| = 1 \geq 2^0$ indeed.

Subsequent MAKEs and FINDs make no change until first UNION.

UNION(x, y) results in the smallest tree of height 1, yielding $y.rank = 1$. Then $|T| = 2 \geq 2^1$ indeed.

Induction: after i operations there is $|T_1| \geq 2^{h_1}$ and $|T_2| \geq 2^{h_2}$.

Subsequent MAKEs do not change depth, whereas subsequent FINDs may only flatten T_1 and T_2, i.e., reducing h_1 and h_2.

Upon UNION by rank $T = T_1 \cup T_2$, $|T| = |T_1| + |T_2| \geq 2^{h_1} + 2^{h_2}$.

If $h_1 = h_2$, $h(T) = h_1 + 1 \Rightarrow |T| \geq 2^{h_1} + 2^{h_2} = 2^{h_1+1} = 2^{h(T)}$.

If $h_1 > h_2$, $h(T) = h_1 \Rightarrow |T| \geq 2^{h_1} + 2^{h_2} > 2^{h_1} = 2^{h(T)}$. ■

Corollary: The height of trees is bounded by $O(\log n)$. (HW)

Corollary: Time complexity of any operation is bounded by $O(\log n)$, and a series of m operation is bounded by $O(m \log n)$.

Analysis of Union by Rank and Path Compression

Root:

- MAKE creates root with rank 0
- UNION may increase root's rank by 1.

Internal node:

- Never turns to be a root
- Its rank never changed
- Path compression may reduce its subtree height
- The rank of its root is non decreasing

Rank is an upper bound of subtree depth

Lemma 3: If u is the parent of v, then $rank(v) \leq rank(u)$, with strict inequality if $u \neq v$.

Proof: By induction on the series of operations on underlying DS.

Base: First operations $\mathrm{MAKE}(u)$ and $\mathrm{MAKE}(v)$, yields single node DS with rank 0.

Subsequent MAKEs and FINDs make no change until first UNION.

$\mathrm{UNION}(u, v)$ makes u parent of v, $rank(v) = 0$ and $rank(u) = 1$.

Induction: $rank(v) \leq rank(u)$ holds after ith operation.

The $(i+1)$th operation can be any of:

MAKE does not affect the $rank(v) \leq rank(u)$ in any way.

UNION connects root of smaller rank to higher one, and if equal, the parent's rank is increased by 1.

FIND employs path compression, where descendant is connected to its ancestor root, which by induction its rank is higher. ■

We shall prove that a sequence of m disjoint-set operations on n objects takes $O\big(m \cdot \alpha(n)\big)$ time, where $\alpha(n)$ grows very slowly.

$\alpha(n)$ counts how many times $\log$ is applied until

$$\log(\log(\cdots \log(n) \cdots)) \leq 1.$$

Formally,

$$\alpha(n) \triangleq \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \alpha(\log n) & \text{if } n > 1 \end{cases}$$

For argument $n = 2^{2^{\cdot^{\cdot^{2^2}}}}$, $\alpha(n)$ is number of 2s in the tower.

$\alpha(2) = \alpha(2^1) = 1 + \alpha(1) = 1.$

$\alpha(4) = \alpha(2^2) = 1 + \alpha(2) = 2.$

$\alpha(16) = \alpha(2^4) = \alpha\left(2^{2^2}\right) = 1 + \alpha(4) = 3.$

$\alpha(65536) = \alpha(2^{16}) = \alpha\left(2^{2^{2^2}}\right) = 1 + \alpha(16) = 4.$

$\alpha(2^{65536}) = \alpha\left(2^{2^{2^{2^2}}}\right) = 1 + \alpha(65536) = 5.$

Lemma 4: A forest of n nodes has at most $\lfloor n/2^r \rfloor$ nodes of rank r.

Proof: Let a root x change its rank from $r-1$ to r upon UNION, and let T_x be the implied tree. Mark x.

Since rank bounds height, from lemma 2 there is $|T_x| \geq 2^r$, where rank of all T_x's internal nodes is smaller than r.

The rank of T_x's internal nodes never change, and x rank can only increase. ⇒ Node is marked at most once.

⇒ Markers of rank r occur at most $\lfloor n/2^r \rfloor$ times. ■

Theorem 5: Total run time of m operations of which n are MAKE is $O\big(m \cdot \alpha(n)\big)$.

Proof: By amortized analysis. For $k = \alpha(n)$, we define $\alpha^{-1}(k) = n$.

Divide the nodes into sets S_i as follows:

$S_0 = \{x \mid x.rank \in [0, 2^0 - 1]\}$, i.e., $x.rank = 0$.

$S_1 = \{x \mid x.rank \in [2^0, 2^1 - 1]\}$, i.e., $x.rank = 1$.

$S_2 = \{x \mid x.rank \in [\alpha^{-1}(1), \alpha^{-1}(2) - 1]\}$, i.e., $x.rank = 2,3$.

$S_3 = \{x \mid x.rank \in [\alpha^{-1}(2), \alpha^{-1}(3) - 1]\}$, i.e., $x.rank = 4,5, \ldots 15$.

And in general

$S_i = \{x \mid x.rank \in [\alpha^{-1}(i-1), \alpha^{-1}(i) - 1]\}$.

No rank exceeds $n \Rightarrow$ total number of sets does not exceed $\alpha(n)$.

By lemma 4, the number of nodes which ranks are in S_k (ranks in $[\alpha^{-1}(k-1), \alpha^{-1}(k) - 1]$) is bounded by

(1) $\sum_{j=\alpha^{-1}(k-1)}^{\alpha^{-1}(k)-1} \lfloor n/2^j \rfloor < \frac{n}{2^{\alpha^{-1}(k-1)}} \sum_{j=0}^{\infty} 2^{-j} = \frac{2n}{2^{\alpha^{-1}(k-1)}} = \frac{2n}{\alpha^{-1}(k)}$.

The total time consumed by the operations can be associated to the traversed links.

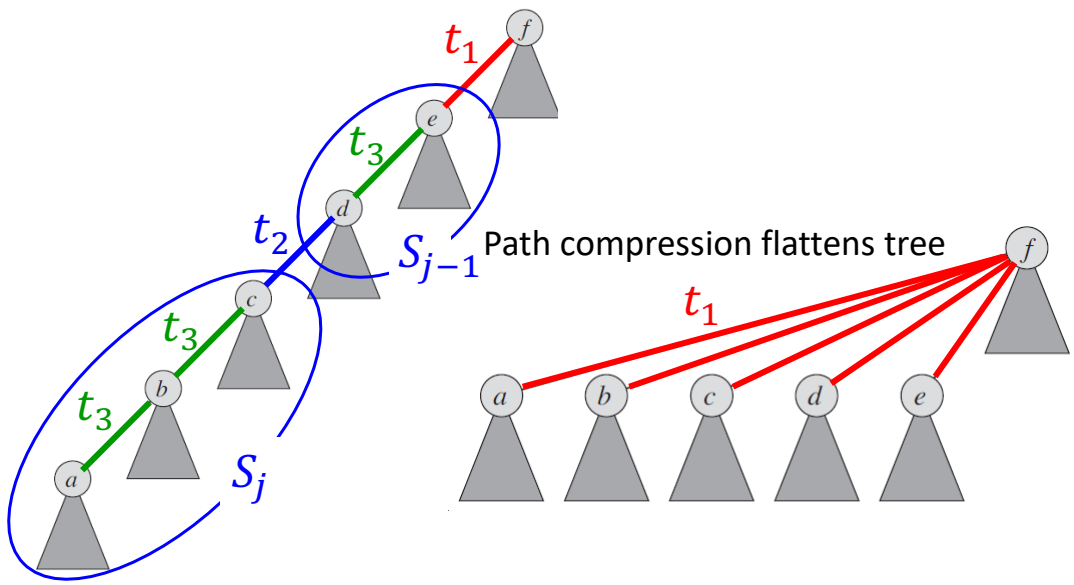
Consider the links traversed in all FINDs among the m operations. MAKE and UNION anyway contribute totally $O(n)$ time.

t_1 = # links traversed to a root

t_2 = # links traversed across the sets $S_0, S_1, \dots, S_{\alpha(n)}$

t_3 = # links traversed within the sets $S_0, S_1, \dots, S_{\alpha(n)}$

The total complexity of the m operations is $O(t_1 + t_2 + t_3)$.



In path compression of FIND one link is connected directly to root. Hence, $t_1 = O(m)$.

In the worst case any operation traverses across all $S_0, S_1, \dots, S_{\alpha(n)}$, yielding a bound $t_2 = O\big(m \cdot \alpha(n)\big)$.

To account the links within S_k , $1 \le k \le \alpha(n)$, consider the path $v_0 \to v_1 \to v_2 \cdots \to v_p$.

By lemma 3 their rank is strictly increasing, hence the paths length is bounded by $(\alpha^{-1}(k) - 1) - \alpha^{-1}(k-1) < \alpha^{-1}(k)$.

Hence

$$t_3 \le \sum_{k=0}^{\alpha(n)} \sum_{v \in S_k} \alpha^{-1}(k) \overset{(1)}{=} \sum_{k=0}^{\alpha(n)} \alpha^{-1}(k) \frac{2n}{\alpha^{-1}(k)}$$

$$= 2n \cdot \alpha(n) = O\big(n \cdot \alpha(n)\big).$$

Summing t_1, t_2 and t_3, there is

$$t_1 + t_2 + t_3 \le O(m) + O\big(m \cdot \alpha(n)\big) + O\big(n \cdot \alpha(n)\big)$$

$$= O\big(m \cdot \alpha(n)\big). \ \blacksquare$$