There are 7 cities, b, c, d, e, f, g, and h.
If there is a path connecting two cities, the pair of cities forms a pair.
(b, e), (h, c), (f, d), (c, e), (g, f), (e, h), and (g, d)
Is there a path connecting any pair of cities?

Union-Find Operation

- Group n elements into a collection of disjoint sets.
- Two operations
  - Find which set a given element belongs to,
  - Union two sets.
- Find connected components, (relation, reflexsive, symmetric, transtive).

Disjoint-Set data structure

- Maintain a collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- Each Set is identified by representative- a member in the set.

Operations

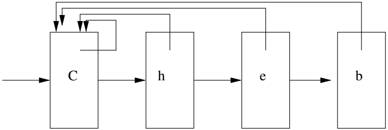
- Make-Set(x), create a new set, x is the only member, x is the representative.
- Union(x, y), Unite the dynamic sets that contains x and y (S_x, S_y). Representative of the union of x and y is some member of $S_x \cup S_y$, S_x and S_y are destroied (removed from S).
- Find-Set(x), return a pointer to the representative of the set containing x.

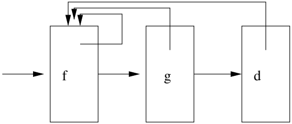
- An application, determine the connected components of an undirected graph.
- Given $G = (V, E)$,
- for each $v \in V[G]$, do Make-Set(v);
- for each edge $(u, v) \in E[G]$, if Find-Set(u) $\neq$ Find-Set(v), then Union(u, v).

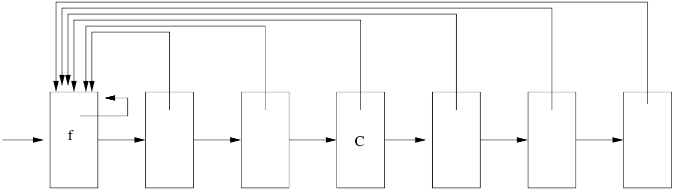
This is a problem, n Make-Set operations, a sequence of m Find-Set operations, and there are at most $n - 1$ Union operations. What is the data structure and what are the time complexcities for the operations.

Linked-List representation

- Head of the list is the representative.
- Every one has a pointer pointing to the representative.
- Make-Set, $\Theta(1)$ time.
- Find, $\Theta(1)$.
- Union is little bit hard.


C
h
e
b


f
g
d


f
C

Union in Linked-List Representation

- Append one to the end of the other.
- Change pointers to the head of the merged list.
- There is a sequence of Unions that cost $\Theta(n^2)$ time if there are $n-1$ union.

A Weighted-Union Heuristic

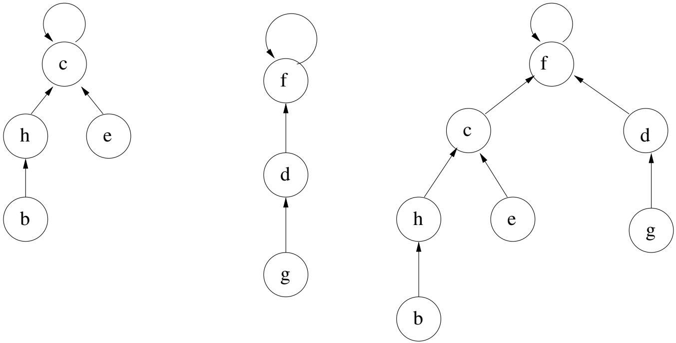
Append the smaller list onto the longer, tie broken arbitrary.

Theorem: Using the linked-list representation of disjoint sets and the weighted-union heuristic, a sequence of m Make-Set, Union, and Find-Set operations, n of Make-Set operations, takes $O(m + n \lg n)$ time.

Proof: Count the number of times that an object's representative pointer was modified.
Consider an object x, first time its pointer was modified, the resulting set has at least 2 members.
2nd time update, resulting set has at least 4 members.
Updating $\lceil \lg k \rceil$ times, the resulting set has k members.
There are at most $\lceil \lg n \rceil$ updates for an object. There are n objects, so Union costs at most $O(n \log n)$ time.
For Make-Set and Find-Set, each one takes $O(1)$ and there are m such operations. Total cost is $O(m + n \lg n)$.

Disjoint-set forest

- Each member points only to its parent.
- A connected component is a rooted tree.
- Parent pointer of the root points to itself.
- Root is the representative.
- Union, change Parent of the root of one set, Find-Set, chacing the Parent link.


c
h
e
b
f
d
g
f
c
d
h
e
g
b

Time Complexity

- Make-Set, each one takes $O(1)$ time.
- Union, each one takes $O(1)$ time.
- Find-Set, time required depending on the distance from the object to the root.
- There exists a sequence of n Union/Find-Set operatine that takes $O(n^2)$ time.

Heuristics to improve the running time

- **Union by rank**: Similar to the weighted-union heuristic, make the root of the tree with fewer nodes point to the root of the tree with more nodes. To ease the analysis, we maintain a **rank** that is an upper bound on the height of the node. Union by rank, the root with small rank is made to point to the root with larger rank during a UNION operation.
- **Path Compression**: During the Find-Set operations, make each node on the find path point directly to the root.

Pseudocode for disjoint-set forest

MAKE-SET(x)
p[x] $\leftarrow x$
$rank[x] \leftarrow 0$

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

FIND-SET(x)
if $x \neq p[x]$
    then $p[x] \leftarrow$ FIND-SET($p[x]$)
return $p[x]$

Link(x, y)
if $rank[x] > rank[y]$
    then $p[y] \leftarrow x$
else $p[x] \leftarrow y$
    if $rank[x] = rank[y]$
        then $rank[y] \leftarrow rank[y] + 1$

Effect of the Heuristics on running time

- If there are f FIND-SET operations, the path-compression heuristic alone gives a worst case running time $\Theta(f \log_{(1+f/n)} n)$ if $f \geq n$ and $\Theta(n + f \lg n)$ if $f < n$.
- If both union by rank and path compression are applied, the wrost case running time is $O(m\alpha(n))$ where $\alpha(n)$ is a *very* slowly growing function.
- In any conceivable application of a disjoint-set data structure, $\alpha(n) \leq 4$, we can view the running time as linear in m in all practical situation.

Functional Iteration

Section 3.2

Notation $f^{(i)}(n)$: function $f(n)$ iteratively applied i times to an intitial value on n.

Let $f(n)$ be a function over reals, for nonnegative integer i,

$$f^{(i)}(n) = \begin{cases} n & \text{if } i = 0 \\ f(f^{(i-1)}(n)) & \text{if } i > 0 \end{cases} \tag{1}$$

If $f(n) = 2n$, then $f^{(i)}(n) = 2^i n$.

For example $f^{(2)}(n) = f(f^{(1)}(n)) = f(f(f^{(0)}(n))) = f(f(n)) = f(2n) = 2^2 n$.

Iterated logrithm function

- $\lg^* n$ denotes iterated logrithm.
- $\lg^{(i)} n$ defined as above with $f(n) = \lg n$.

$$\lg^* n = \min\{i \geq 0 | \lg^{(i)} n \leq 1\} \tag{2}$$

n	$\lg n$	$\lg^* n$
2	$\lg 2 = 1$	$\lg^* 2 = 1$
4	$\lg 4 = 2$	$\lg^* 4 = 2$
16	$\lg 16 = 4$	$\lg^* 16 = 3$
65536	$\lg 65536 = 16$	$\lg^* 65536 = 4$
2^{65536}	$\lg 2^{65536} = 65536$	$\lg^* 2^{65536} = 5$

$\lg^* n$ is a very very slow growing function.
Since the number of observable universe is estimated to be about 10^{80} atoms, and $10^{80} << 2^{65536}$. We rarely encounter an input size n s.t. $\lg^* n > 5$.

Define the function $A_k(j)$

For $k \geq 0$ and $j \geq 1$,

$$A_k(j) = \begin{cases} j+1 & \text{if } k = 0 \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1 \end{cases} \tag{3}$$

where $A_{k-1}^{(j+1)}(j)$ functional-iterated notation,

$$\begin{aligned} A_{k-1}^{(0)}(j) &= j \text{ and} \qquad (4) \\ A_{k-1}^{(i)}(j) &= A_{k-1}(A_{k-1}^{(i-1)}(j)) \text{ for } i \geq 1. \qquad (5) \end{aligned}$$

k the level of function A.

Lemma 2: For any integer $j \geq 1$, we have $A_1(j) = 2j + 1$.
Proof Induction on i to show that $A_0^{(i)}(j) = j + i$.
Base: $A_0^{(0)}(j) = j = j + 0$.
Assume that $A_0^{(i-1)}(j) = j + (i - 1)$.

$$\begin{aligned} A_0^{(i)}(j) &= A_0(A_0^{(i-1)}(j)) && (6) \\ &= A_0(j + (i - 1)) && (7) \\ &= (j + (i - 1)) + 1 && (8) \\ &= j + i. && (9) \end{aligned}$$

Finally

$$
\begin{aligned}
A_1(j) &= A_0^{(j+1)}(j) && (10)\\
&= j + (j+1) && (11)\\
&= 2j + 1 && (12)
\end{aligned}
$$

Lemma 3: For any integer $j \geq 1$, we have
$A_2(j) = 2^{(j+1)}(j+1) - 1$.

Proof Induction on i to show that $A_1^{(i)}(j) = 2^i(j+1) - 1$.

For the base case: $A_1^{(0)}(j) = j = 2^0(j+1) - 1$.

For the inductive step: Assume that $A_1^{(i-1)}(j) = 2^{i-1}(j+1) - 1$.

Then

$$\begin{aligned}
A_1^{(i)}(j) &= A_1(A_1^{(i-1)}(j)) \\
&= A_1(2^{(i-1)}(j+1) - 1) \\
&= 2(2^{i-1}(j+1) - 1) + 1 \\
&= 2^i(j+1) - 2 + 1 \\
&= 2^i(j+1) - 1
\end{aligned}$$

Finally

$$\begin{aligned}
A_2(j) &= A_1^{(j+1)}(j) \\
&= 2^{j+1}(j+1) - 1
\end{aligned}$$

How quickly $A_k(j)$ grows, Look at $A_k(1)$ for level 0, 1, 2, 3, and 4.

$$\begin{aligned}
A_0^{(1)} &= 1+1=2, A_1(1)=2\cdot 1+1=3 \\
A_2(1) &= 2^{1+1}\cdot(1+1)-1=7 \\
A_3(1) &= A_2^{(2)}(1)=A_2(A_2(1))=A_2(7) \\
&= 2^8\cdot 8-1 \\
A_4(1) &= A_3^{(2)}(1)=A_3(A_3(1))=A_3(2047) \\
&= A_2^{(2048)}(2047) \\
&>> A_(2047)=2^{2048}\cdot 2048-1>2^{2048} \\
&= (2^4)^{512}=16^{512}>>10^{80}
\end{aligned}$$

Much greater than the estimated number of atoms in the observable unserve.

The Inverse of $A_k(n)$, $n \geq 0$

$$\alpha(n) = \min\{k : A_k(1) \geq n\} \tag{13}$$

Or $\alpha(n)$ is the lowest level k, s.t. $A_k(1)$ is at least n.

$$\alpha(n) = \begin{cases} 0 & 0 \leq n \leq 2 \\ 1 & n = 3 \\ 2 & 4 \leq n \leq 7 \\ 3 & 8 \leq n \leq 2047 \\ 4 & 2048 \leq n \leq A_4(1) \end{cases}$$

So $\alpha(n) \leq 4$ for all practical purpose.

To show $O(m\alpha(n))$ bounds the running time of the disjoint-Set operation.
Lemma 4: $\forall$ nodes x, we have rank[x]$\leq$rank[p[x]] with strict inequality $x \neq p[x]$; and from then on, rank[x] does not change. The value of rank[p[x]] monotonically increase over time.
Corollary 5: As we follow the path from any node toward a root, the node ranks strictly increase.

Lemma 6: Every node has rank at most $n-1$.
(A weak bound, a tight bound will be $\lfloor \lg n \rfloor$.)
Proof For each node, rank starts 0 (initialization, make a node a set).
It increases only upon $\textsc{Link}$ operation. There are $n-1$ $\textsc{Union}$ so there are at most $n-1$ $\textsc{Link}$ operations.
Each $\textsc{Link}$ either leaves all ranks alone or increase some nodes' rank by 1.
Thus all ranks are at most $n-1$.

Prove the Time Bound

Amortized Analysis

Lemma 7: Suppose we convert a sequence of S' of m' MAKE-SET, UNION, and FIND-SET operations into a sequence S of m MAKE-SET, LINK, and FIND-SET by turing each UNION into two FIND-SET followed by a LINK. The if sequence S runs in $O(m(\alpha n))$ time, sequence S' runs in $O(m'\alpha(n))$ time.
Proof UNION in sequence S' is converted into 3 operations in S. We have $m' \leq m \leq 3m'$, thus $m = O(m')$. An $O(m\alpha(m))$ time bound for S implies $O(m'\alpha(m))$ time bound for S'.

Potential Function

- $\phi_q(x)$: a potential assigned to each node x in the disjoint-set forest after q operations.
- $\Phi_q = \sum_x \phi(x)$: the potential for the entire forest after q operations.
- The forest is empty prior the first operation, $\Phi_0 = 0$.
- $\phi_q(x)$: if x is a tree root after the qth operation or if $rank[x] = 0$, then $\phi_q(x) = \alpha(n) \cdot rank[x]$.
- If x is not a root and $rank[x] \geq 1, \ldots$

x is not a root and $rank[x] \geq 1$

Define 2 auxiliary functions on x,

$$level(x) = \max\{k : rank[p[x]] \geq A_k(rank[x])\} \tag{14}$$

$$iter(x) = \max\{i : rank[p[x]] \geq A^{(i)}_{level(x)}(rank[x])\} \tag{15}$$

$$level(x) = \max\{k : rank[p[x]] \geq A_k(rank[x])\}$$

$level(x)$: Greatest level k for which A_k, applied to x's rank, is no greater than x's parent's rank.

Claim 1 $0 \leq level(x) < \alpha(n)$

proof

$$\begin{aligned} rank[p[x]] &\geq rank[x] + 1 \text{ by Lemma 4} \\ &= A_0(rank[x]) \text{ by definition of } A_0(j) \end{aligned}$$

That implies $level(x) \geq 0$.

$$\begin{aligned} A_{\alpha(n)}(rank[x]) &\geq A_{\alpha(n)(1)} \\ &\geq n \\ &> rank[p[x]] \end{aligned}$$

That implies $level(x) < \alpha(n)$.

$$iter(x) = \max\{i : rank[p[x]] \geq A^{(i)}_{level(x)}(rank[x])\}$$

$iter(x)$: The largest number of times we can iteratively apply $A_{level(x)}$, applied to x's rank before we get a value greater than x's parent's rank.

Claim 2: $1 \leq iter(x) \leq rank[x]$

Proof:

$$\begin{aligned} rank[p[x]] &\geq A_{level(x)}(rank[x]) && \text{by definition of } level(x) \\ &= A^{(1)}_{level(x)}(rank[x]) && \text{by definition of functional iteration} \end{aligned}$$

So $iter(x) \geq 1$.

$$\begin{aligned} A^{(rank[x]+1)}_{level(x)}(rank[x]) &= A_{level(x)+1}(rank[x]) && \text{by definition of } A_k(j) \\ &> rank[p[x]] && \text{by definition of } level(x) \end{aligned}$$

which implies $iter(x) \leq rank[x]$.

Potential of node x after q operations

$$\phi_q(x) = \begin{cases} \alpha(n) \cdot rank[x] & \text{if } x \text{ is a root or } rank[x] = 0 \\ (\alpha(n) - level(x)) \cdot rank(x) - iter(x) & \text{if } x \text{ is not a root and } rank[x \end{cases} \tag{16}$$

Lemma 8 For every node x, and for all operation count q,

$$0 \leq \phi_q(x) \leq \alpha(n) \cdot rank[x]$$

Proof If x is a root or $rank[x] = 0$, $\phi_q(x) = \alpha(n) \cdot rank[x]$.
If x is not a root and $rank[k] \geq 1$:
To obtain lower bound on $\phi_q(x)$, we maximize $level[x]$ and $iter(x)$. Since $level(x) \leq \alpha(n) - 1$ (Claim 1) and $iter(x) \leq rank[x]$ (Claim 2)

$$\begin{aligned} \phi_q(x) &\geq (\alpha(n) - (\alpha(n) - 1)) \cdot rank[x] - rank[x] \\ &= rank[x] - rank[x] \\ &= 0. \end{aligned}$$

To obtain an upper bound on $\phi_q(x)$, we minimize $level(x)$ and $iter(x)$. Since $level(x) \geq 0$ (Calim 1), and $iter(x) \geq 1$ (Claim 2), we have

$$\begin{aligned} \phi_q(x) &\leq (\alpha(n) - 0) \cdot rank[x] - 1 \\ &= \alpha(n) \cdot rank[x] - 1 \\ &< \alpha(n) \cdot rank[x] \end{aligned}$$

Lemma 9 Let x be a node that is not a root, suppose that qth operation is either a LINK or FIND-SET. Then after the qth operation, $\phi_q(x) \leq \phi_{q-1}(x)$.
Moreover, if $rank[x] \geq 1$ and either $level(x)$ or $iter(x)$ changes due to the qth operation, then $\phi_q(x) \leq \phi_{q-1}(x) - 1$. That is x's potential cannot increase, and if it has positive rank and either $level(x)$ or $iter(x)$ changes, then x's potential drops by at least 1.
Proof Since x is not a root, qth operation does not change $rank[x]$, and n does not change after the initial n MAKE-SET operations, $\alpha(n)$ does not change wither.
If $rank[x] = 0$, then $\phi_q(x) = \phi_{q-1}(x) = 0$.
If $rank[x] \geq 1$: $level(x)$ monotonically increase
If qth operation does not change $level(x)$, $iter(x)$ either increases or remains unchanged.
If both $level(x)$ and $iter(x)$ are unchanged, then $\phi_q(x) = \phi_{q-1}(x)$.
If $level(x)$ is unchanged and $iter(x)$ increase, $iter(x)$ increases by at least 1, $\phi_q(x) \leq \phi_{q-1}(x) - 1$.
If qth operation increases $level(x)$, it increases by at least 1. Thus $(\alpha(n) - level(x)) \cdot rank[x]$ drops by at least $rank[x]$.
Since $level[x]$ increased, the value of $iter(x)$ might drop. The drop

Lemma 10 The amortized cost of each MAKE-SET operation is $O(1)$.

Proof Suppose the qth operation is MAKE-SET(x). This operation creates node x with rank 0, so that $\phi_q(x) = 0$. No other ranks or potentials change, so $\Phi_q = \Phi_{q-1}$.

The actual cost is of the MAKE-SET operations is $O(1)$.

Lemma 11 The amortized cost of each LINK operation is $O(\alpha(n))$.

Proof The qth operation is LINK. The actually cost is the LINK operation is $O(1)$. Wlog that the LINK makes y the parent of x. The nodes whose potentials may change are x, y, and the children of y.

To show that the only node whose potential can increase due to the LINK is y, and the increase is at most $\alpha(n)$.

- By Lemma 9, any node of y's children cannot have its potential increase due to the LINK.
- x was a root before qth operation, $\phi_{q-1}(x) = \alpha(n) \cdot rank[x]$. If $rank[x] = 0$, then $\phi_q(x) = \phi_{q-1}(x) = 0$. Otherwise $\phi_q(x) = (\alpha(n) - level(x)) \cdot rank[x] - iter(x) < \alpha(n) \cdot rank[x]$.
- y was the root, $\phi_{q-1}(y) = \alpha(n) \cdot rank[y]$. y is still a root after the LINK operation. After the LINK operation, y's rank increases by 1 or remains the same. Thus either $\phi_q(y) = \phi_{q-1}(y)$ or $\phi_q = \phi_{q-1}(y) + \alpha(n)$.

The amortized cost is $O(1) + O(\alpha(n)) = O(\alpha(n))$.

Lemma 12 The amortized cost of each FIND-SET operation is $O(\alpha(n))$.

Proof qth operation is FIND-SET and that the find path contains s nodes. The actual cost is $O(s)$.

We show that

1. no node's potential increases due to the Find-Set and
2. at least $\max(0, s - (\alpha(n) + 2))$ nodes on the find path have their potential decrease by at least 1.

"No node's potential increase" $\forall$ nodes except the root, by Lemma 9.

If x is the root, then its potential is $\alpha(n) \cdot rank[x]$, which does not change.

To show that at least $\max(0, s - (\alpha(n) + 2))$ node have their potential decrease by at least one.

x: a node on the find path that $rank[x] > 0$ and x is followed somewhere on the find path by another node y that is not a root, where $level(y) = level(x)$ just before the FIND-SET operation.

At most $\alpha(n) + 2$ nodes on the path do not satisfy the constraints on x. The first node has rank 0, the last node is the root, and the last node w on the path for which $level(w) = k$,

Theorem 13 A sequence of m MAKE-SET, UNION, and FIND-SET operations, n of which are MAKE-SET operations, can be performed on a disjoint-set forest with union by rank and path compression in worst-case time $O(m\alpha(n))$.

Proof By Lemmas 7, 10, 11, and 12.