Disjoint Sets

Keep keys in disjoint sets
Find set containing key
Union two sets
Application: determine connected components of an undirected graph

\[
\text{make each vertex a set}
\]
\[
\text{foreach edge}
\]
\[
\text{union sets containing vertices of edge}
\]

Representation

A ________________ is a data structure \( S = \{S_1, \ldots, S_k\} \), or a collection of disjoint dynamic sets.

Each set has a ________________ element, which never changes unless unioned with another set.

Operations

\[
x = \text{pointer to an object containing some key}
\]
Make-Set(x)
    Create new set $S_x$ with one member x
    Representative of $S_x$ is x
    Disjoint set = Disjoint set + $S_x$

Union(x,y)
    $S_x = \text{set containing } x$
    $S_y = \text{set containing } y$
    $S_u = S_x \cup S_y$
    rep($S_u$) = rep($S_x$) or rep($S_y$) ; or any other object in $S_u$
    Disjoint set = Disjoint set - $S_x$ - $S_y$ + $S_u$

Find-Set(x)
    $S_x = \text{set containing } x$
    return rep($S_x$)

---

**Application**

Finding the connected components of a graph.

**Connected-Components(Graph)**
    foreach v in vertices(Graph)
        Make-Set(v)
    foreach e in edges(Graph)
        (u,v) = e
        if Find-Set(u) $\neq$ Find-Set(v)
            then Union(u,v)

**Same-Component(u,v)**
    if Find-Set(u) = Find-Set(v)
then return True
else return False

---

**Linked-List Representation**

Use linked list to represent set of objects.
Each object contains a pointer to the rep, the key, and a pointer to next.

**Operations**

- **Make-Set(x)**; O(1)
  - rep(x) = x
  - next(x) = NIL

- **Find-Set(x)**; O(1)
  - return rep(x)

- **Union(x, y)**; Θ(size of x)
  - foreach object in rep(x)
    - insert object into y
    - rep(object) = rep(y)
  - remove x

---

**Analysis**

The worst case scenario is:
• Make-Set($x_1$)

• ... 

• Make-Set($x_n$) \[\{\{x_1\}, \{x_2\}, \{x_3\}, \ldots, \{x_n\}\}\]

• Union($x_1, x_2$) \[\{\{x_1 \rightarrow x_2\}, \{x_3\}, \ldots, \{x_n\}\}\]

• Union($x_2, x_3$) \[\{\{x_1 \rightarrow x_2 \rightarrow x_3\}, \ldots, \{x_n\}\}\]

• ...

• Union($x_{q-1}, x_q$) \[\{\{x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \ldots \rightarrow x_n\}\}\]

\[n = \#\text{Make-Set operations}\]
\[m = \#\text{Make-Set, Union, and Find-Set operations}\]
\[m = n + (q - 1) \text{ operations}\]

\[T(m) = \Theta(n) + \sum_{i=1}^{q-1} i\]
\[= \Theta(n + q^2)\]
\[n = \Theta(m) \text{ and } q = \Theta(m)\]

Therefore, \(T(m) = \Theta(m^2)\) and the amortized cost is \(\Theta(m)\) per operation.
Can we do better?

---

**Weighted-Union Heuristic**

**Idea:** Keep track of the number of objects in a set (length of list). Append shorter list to longer list.

**Theorem 22.1**

A sequence of \(m\) operations, \(n\) of which are Make-Set operations, takes \(O(m + n \lg n)\) time.
Proof: Since we only change rep(x) for objects in the shorter list for each Union, and lists start at length=1, then each Union at least doubles the size of x’s list. Thus, we can do at most \( \lceil \log n \rceil \) Unions that require rep(x) changes, and there are n objects.

As a result, there are a total of \( \frac{n}{2} \) changes.

If we add the O(1) costs for the O(m) Make-Set and Find-Set operations, we get \( \frac{3n}{2} \) changes.

---

Disjoint Sets as Forest of Trees

Idea: Represent disjoint sets as a forest of trees.

Object:  

<table>
<thead>
<tr>
<th>key</th>
<th>parent</th>
</tr>
</thead>
</table>

Example: \( S_c = \{c, b, a\} \)

Representatives are roots

\[ x = \text{parent}(x) \]

---

Operations

Make-Set(x)

\[ \text{parent}(x) = x \]

FindSet(x)
Follow parent pointers from x to root
return root

Union(x, y)
    parent(x) = FindSet(y)

Performance same as linked lists, but can do better.

---

**Union by Rank**

In Union, have parent of shallower tree point to other tree.
Maintain rank(x) as an upper bound on the depth of the tree rooted at x.

Make-Set(x)
    parent(x) = x
    rank(x) = 0

Union(x, y)
    repx = FindSet(x)
    repy = FindSet(y)
    if rank(repx) > rank(repy)
    then parent(repy) = repx
    else parent(repx) = repy
        if rank(repx) = rank(repy)
        then rank(repy) = rank(repy) + 1

Can we do even better?
Path Compression

While looking for rep(x) by traversing parent pointers, set each one to the resulting rep(x).

     Click on mouse to advance to next frame.

     Note: Since rank is an ____________ on tree height, path compression need not change ranks.

Pseudocode

FindSet(x)
    if x ≠ parent(x) ; Two-Pass Method
        then parent(x) = FindSet(parent(x))
    return parent(x)

Analysis:

Union by Rank Only:

    $\Theta(mlgn)$
    
    $m = \#\text{operations}$
    
    $n (\leq m) = \#\text{MakeSet operations in m}$

Path Compression Only:

    $\Theta(f \log_{(1+f/n)} n)$ if $f \geq n$
    
    $\Theta(n + f \log n)$ if $f < n$

    $n = \#\text{MakeSet operations}$
    
    There are $\leq n-1$ Unions
    
    $f = \#\text{FindSet operations}$
Analysis

Union by Rank and Path Compression:

O(m * α(m,n)) worst case running time

α(m,n) is inverse of Ackermann’s function A(i,j)

\[ α(m,n) = \min\{i \geq 1 \mid A(i, \lfloor \frac{m}{n} \rfloor) > \lg n\} \]

**Ackermann’s Function A(i,j)**

- \( A(1, j) = 2^j \) for \( j \geq 1 \)
- \( A(i, 1) = A(i-1, 2) \) for \( i \geq 2 \)
- \( A(i, j) = A(i-1, A(i, j-1)) \) for \( i, j \geq 2 \)

<table>
<thead>
<tr>
<th>j=1</th>
<th>j-2</th>
<th>j=3</th>
<th>j=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>2^1</td>
<td>2^2</td>
<td>2^3</td>
</tr>
<tr>
<td>i=2</td>
<td>2^2</td>
<td>2^2</td>
<td>2^2</td>
</tr>
<tr>
<td>i=3</td>
<td>2^2</td>
<td>2^2</td>
<td>2^2</td>
</tr>
</tbody>
</table>

**Note:** \( A(i,j) \) is strictly increasing and \( \lfloor \frac{m}{n} \rfloor \geq 1 \) since \( m \geq n \).

Therefore \( A(4, \lfloor \frac{m}{n} \rfloor) \geq A(4,1) = A(3,2) \)

\( A(3,2) = 2 \) raised to the power 2 16 times \( >> 10^{80} \)

\( 10^{80} = \) the number of atoms in the observable universe

\( α(m, n) = 4 \) for practical uses since \( \lg n \) is typically less than \( 10^{80} \)
Thus, $T(m) = O(m)$.
$O(1)$ amortized cost per operation

Applications