The Problem

- Are two elements in the same set?
- Form the union of two sets

Disjoint Sets ADT

- Find(x)
  * returns set identifier
  * Find(x) = Find(y) iff x and y are in the same set

- Union(A, B)
  * Arguments are set identifiers
  * How do we union the sets containing x and y?

- MakeNewSet(item)
  * Tell the ADT to recognize item as an element
An Application

- We love mazes!
- How to generate randomly?

Mazes

Add walls randomly

Random Mazes

- Circuits are bad
- We don’t want to add a wall if it creates a circuit
- How to detect?
Disjoint Sets to the Rescue!

- Elements are grid points
- Continuous walls are the sets
- Only add a new wall if the endpoints are in different sets

DS in Action

```
while (...) {
    (x,y) = ChooseRandomWall();
    u = Find(x);
    v = Find(y);
    if (u != v) {
        AddWall(x,y);
        Union(u,v);
    }
}
```

Tree Representation

- Maintain forest of trees
- Each set is a tree
- The root of a tree is the set identifier
Find

- Find($x$): walk parents of $x$ to the root

Union

- Union($A, B$): join the two trees
- As $A$ and $B$ are already the roots of a tree, this is easy

What Our Trees are Like

```c
struct Node {
    Item t;
};
```
Up Trees

- Much less structure than binary or ordered trees
- Can have large branching factor with no overhead

struct UpTreeNode {
    Item t;
    UpTreeNode *parent;
};

Union

```c
Union(Node *A, Node *B) {
    B->parent = A;
}
```

How much time to Union?

Find

```c
Node *Find(Node *x) {
    while (x->parent) {
        x = x->parent
    }
    return x;
}
```

How much time to Find?
### What's a Bad Case?

```
 a
 b
 c
 d
 e
 f
 g
 h
 i
 j
 k
```

### What's a Good Case?

```
 a b c d e f g h i j k
```

### Improving Union

```cpp
Union(Node *A, Node *B)
{
    B->parent = A;
}
```

How fast does this make Find?
Improving Find

```c
Node *Find(Node *x)
{
    while (x->parent)
        x = x->parent
    return x;
}

Wait—what’s there to improve?
```

Improving Future Finds

```
Node *Find(Node *x)
{
    Node *r = x;
    while (r->parent)
        r = r->parent
    for ( ; x->parent; x = x->parent)
        x->parent = r;
    return r;
}
```

Path-Compression
Analysis

- Worst case times
  - For Union?
  - for Find?

- Amortized Time?

A Tour of Slow Functions

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>64</th>
<th>1024</th>
<th>32768</th>
<th>$2^{20}$</th>
<th>$2^{30}$</th>
<th>$2^{220}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>1</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>log log</td>
<td>0</td>
<td>2.6</td>
<td>3.9</td>
<td>4.3</td>
<td>4.9</td>
<td>4.9</td>
<td>20</td>
</tr>
<tr>
<td>log log log</td>
<td>0</td>
<td>1.4</td>
<td>1.9</td>
<td>2.1</td>
<td>2.3</td>
<td>2.3</td>
<td>4.3</td>
</tr>
<tr>
<td>log$^+$</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

$log^+ x = k \iff x = 2^{2^k}$

Time to do $m$ Union/Finds on $n$ items is $O(m \log^+ n)!$

I’m Just Getting Started

- Ackerman’s Function:
  \[ A(1) = 2 + 1 = 3 \]
  \[ A(2) = 2 \cdot 2 = 4 \]
  \[ A(3) = 2^3 = 8 \]
  \[ A(4) = 2^{2^3} = 65536 \]
  \[ A(5) = 2^{2^{2^3}} \approx 65536 \text{ who cares?} \]

- Inverse Ackerman’s Function:
  \[ \alpha(x) = k \iff A(k) \leq x < A(k + 1) \]

- Time to do $m$ Union/Finds on $n$ items is $O(m \cdot \alpha(n))!$
  
  For all practical purposes, $O(m \cdot 5)$

- $\alpha$ also comes up in:
  - Computation Geometry (Surface Complexity)
  - Combinatorics of Sequences
- Also known as *Union-Find*
- Simple, efficient implementation
  - with *union-by-size* and *path-compression*
- Great asymptotics
- Kind of weird at first glance, but lots of applications