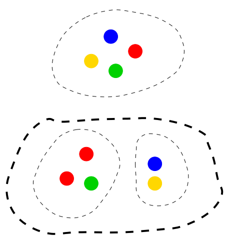
14—Disjoint Sets

§9.1, §11.3

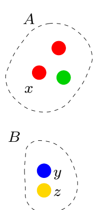
May 14, 2002

The Problem



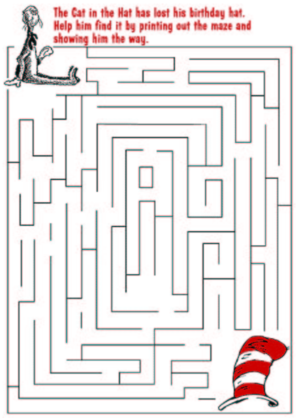
- Are two elements in the same set?
- Form the *union* of two sets

Disjoint Sets ADT



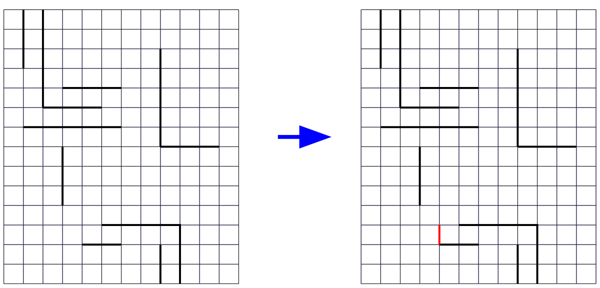
- Find(x)
  - ⋆ returns *set identifier*
  - ⋆ Find(x) $=$ Find(y) iff x and y are in the same set
- Union(A, B)
  - ⋆ Arguments are *set identifiers*
  - ⋆ How do we union the sets containing x and y?
- MakeNewSet(*item*)
  - ⋆ Tell the ADT to recognize *item* as an element

An Application



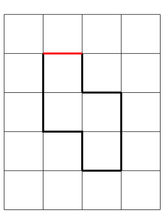
- We love mazes!
- How to generate randomly?

Mazes



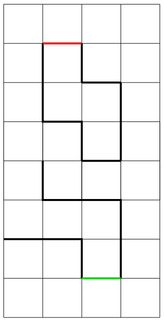
Add walls randomly

Random Mazes



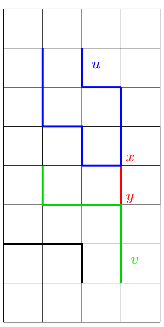
- *Circuits* are bad
- We don't want to add a wall if it creates a circuit
- How to detect?

Disjoint Sets to the Rescue!



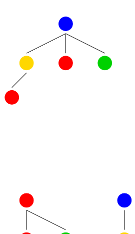
- Elements are *grid points*
- Continuous walls are the sets
- Only add a new wall if the endpoints are in different sets

DS in Action



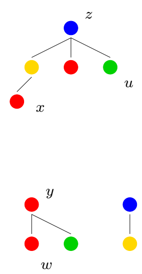

```
while (...) {
  (x,y) = ChoseRandomWall();
  u = Find(x);
  v = Find(y);
  if (u != v) {
    AddWall(x,y);
    Union(u,v);
  }
}
```


Tree Representation



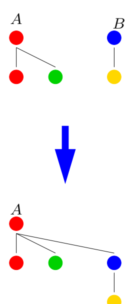
- Maintain *forest* of *trees*
- Each *set* is a *tree*
- The *root* of a tree is the *set identifier*

Find



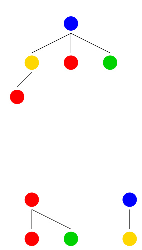
- Find(x): walk parents of x to the root

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

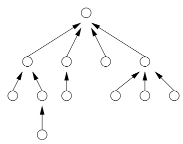


```
struct Node {
  Item t;




};
```

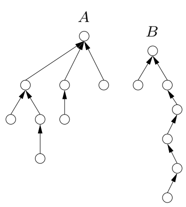
Up Trees



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

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

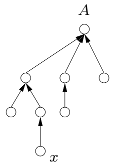
Union




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

How much time to `Union`?

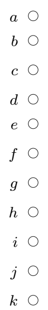
Find



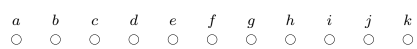

```
Node *Find(Node *x)
{
  while (x->parent)
    x = x->parent
  return x;
}
```

How much time to `Find`?

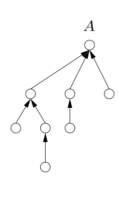
What's a Bad Case?

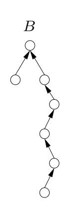


What's a Good Case?



Improving Union

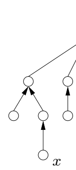





```
Union(Node *A, Node *B)
{
  B->parent = A;
}
```

How fast does this make `Find`?

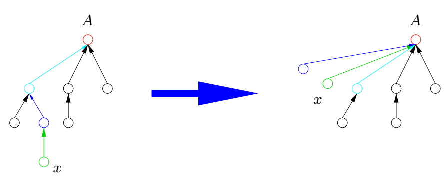
Improving Find




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

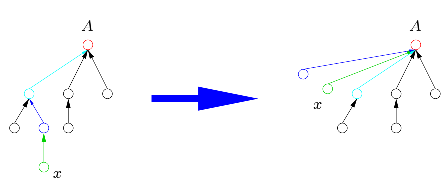
Wait—what's there to improve?

Improving Future Finds



Path-Compression

Path-Compression




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

Analysis

- Worst case times
  - ⋆ For `Union`?

  - ⋆ for `Find`?

- Amortized Time?

A Tour of Slow Functions

	2	64	1024	32768	2^{20}	2^{30}	$2^{2^{20}}$	$2^{2^{2^{20}}}$
log	1	6	10	15	20	30	2^{20}	$2^{2^{20}}$
log log	0	2.6	3.9	4.3	4.9	4.9	20	2^{20}
log log log	0	1.4	1.9	2.1	2.3	2.3	4.3	20
$\log^*$	1	3	3	3	4	4	5	6

$$\log^* x = k \Leftrightarrow x = \left. 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \right\} k$$

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

I'm Just Getting Started

- Ackerman's Function:

$$\begin{aligned} A(1) &= 2+1=3 \\ A(2) &= 2 \cdot 2 = 4 \\ A(3) &= 2^3 = 8 \\ A(4) &= \left. 2^{2^{2^{2}}} \right\} 4 = 65536 \\ A(5) &= \left. 2^{2^{\cdot^{\cdot^{2}}}} \right\} 65536 = \text{ who cares?} \end{aligned}$$

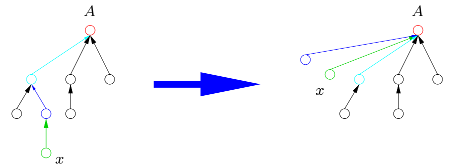
- Inverse Ackerman's Function:

$$\alpha(x) = k \Leftrightarrow A(k) \leq x < A(k+1)$$

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

- α also comes up in:
  - Computation Geometry (Surface Complexity)
  - Combinatorics of Sequences

DS Summary



- 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