Disjoint Sets

CSE 373
Data Structures & Algorithms
Ruth Anderson
Spring 2008

Today's Outline

- **Admin**:
  - HW #4 due – Thurs 5/03 at 11:59pm
    - Print out of code
    - Write-up
- **Disjoint Sets** (Chapter 8)

Disjoint Set-Definition

- Set
  - A collection of distinct objects (unique in that set)
  - Sorted? Operations?
- Disjoint sets
  - A member of a set is unique among all sets
  - Example: {3,5,7}, {4,2,8}, {9}, {1,6}
- Each set has a unique name, one of its members
  - {3,5,7}, {4,2,8}, {9}, {1,6}
  - Operations?

Union

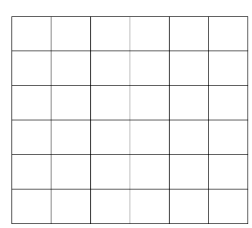
- Union(x,y) – take the union of two sets named x and y
  - {3,5,7} , {4,2,8}, {9}, {1,6}
  - Union(5,1)
    {3,5,7,1,6}, {4,2,8}, {9},
  Or {3,5,7,1,6}, {4,2,8}, {9}

Find

- Find(x) – return the name of the set containing x.
  - {3,5,7,1,6}, {4,2,8}, {9},
  - Find(1) = 5
  - Find(4) = 8

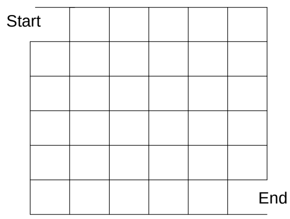
Building Mazes

- Build a random maze by erasing edges.



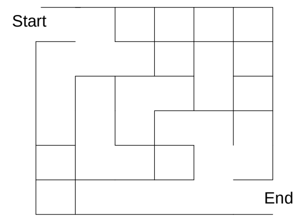
Building Mazes (2)

- Pick Start and End



Building Mazes (3)

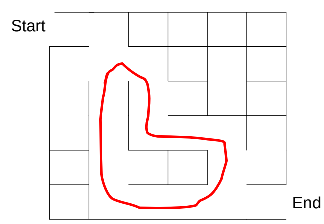
- Repeatedly pick random edges to delete.



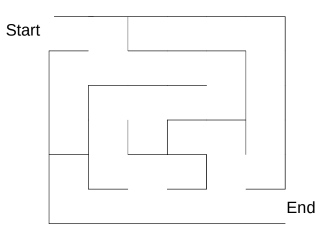
Desired Properties

- None of the boundary is deleted
- Every cell is reachable from every other cell.
- Only one path from any one cell to another (There are no cycles – no cell can reach itself by a path unless it retraces some part of the path.)

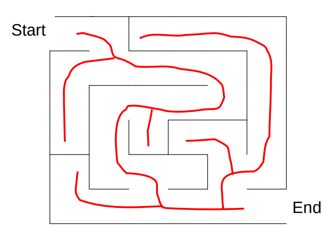
A Cycle



A Good Solution



A Hidden Tree



Number the Cells

We have disjoint sets S ={ {1}, {2}, {3}, {4},… {36} } each cell is unto itself.
We have all possible edges E ={ (1,2), (1,7), (2,8), (2,3), … } 60 edges total.

Start							
	1	2	3	4	5	6	
	7	8	9	10	11	12	
	13	14	15	16	17	18	
	19	20	21	22	23	24	
	25	26	27	28	29	30	
	31	32	33	34	35	36	End

Basic Algorithm

- S = set of sets of connected cells
- E = set of edges
- Maze = set of maze edges (initially empty)

```
While there is more than one set in S {
  pick a random edge (x,y) and remove from E
  u := Find(x);
  v := Find(y);
  if u ≠ v then   // removing edge (x,y) connects previously non-
                  // connected cells x and y - leave this edge removed!
    Union(u,v)
  else            // cells x and y were already connected, add this
                  // edge to set of edges that will make up final maze.
    add (x,y) to Maze
}
All remaining members of E together with Maze form the maze
```


Example Step

Pick (8,14)

Start							
	1	2	3	4	5	6	
	7	8	9	10	11	12	
	13	14	15	16	17	18	
	19	20	21	22	23	24	
	25	26	27	28	29	30	
	31	32	33	34	35	36	End

S
{1,2,7,8,9,13,19}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{14,20,26,27}
{15,16,21}
.
.
{22,23,24,29,30,32
33,34,35,36}

Example

S
{1,2,7,8,9,13,19}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{14,20,26,27}
{15,16,21}
.
.
{22,23,24,29,39,32
33,34,35,36}

Find(8) = 7
Find(14) = 20
→
Union(7,20)

S
{1,2,7,8,9,13,19,14,20 26,27}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{15,16,21}
.
.
{22,23,24,29,39,32
33,34,35,36}

Example

Pick (19,20)

Start							
	1	2	3	4	5	6	
	7	8	9	10	11	12	
	13	14	15	16	17	18	
	19	20	21	22	23	24	
	25	26	27	28	29	30	
	31	32	33	34	35	36	End

S
{1,2,7,8,9,13,19
14,20,26,27}
{3}
{4}
{5}
{6}
{10}
{11,17}
{12}
{15,16,21}
.
.
{22,23,24,29,39,32
33,34,35,36}

Example at the End

S
{1,2,3,4,5,6,7,… 36}

Start							
	1	2	3	4	5	6	
	7	8	9	10	11	12	
	13	14	15	16	17	18	
	19	20	21	22	23	24	
	25	26	27	28	29	30	
	31	32	33	34	35	36	End

E
Maze

Implementing the DS ADT

- *n* elements,
  Total Cost of: *m* finds, ≤ *n*-1 unions

can there be more unions?

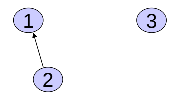
- Target complexity: $O(m+n)$
  i.e. $O(1)$ amortized

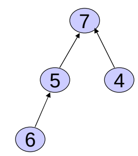
- $O(1)$ worst-case for find as well as union would be great, but…
  Known result: both find and union *cannot* be done in worst-case $O(1)$ time

Up-Tree for Disjoint Union/Find

Initial state:

After several Unions:

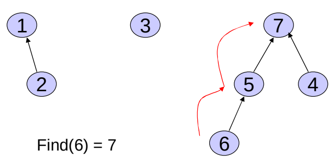




Roots are the names of each set.

Find Operation

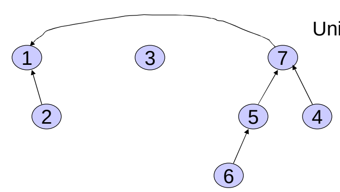
Find(x) - follow x to the root and return the root



Union Operation

Union(x,y) - assuming x and y are roots, point y to x.

Union(1,7)

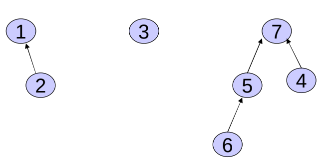


Simple Implementation

- Array of indices

	1	2	3	4	5	6	7
up	0	1	0	7	7	5	0

Up[x] = 0 means x is a root.



Implementation

```
int Find(int x) {

  while(up[x] != 0) {
    x = up[x];
  }

  return x;
}
```

```
void Union(int x, int y) {
  up[y] = x;
}
```

runtime for Union():

runtime for Find():

runtime for m Finds and n-1 Unions:

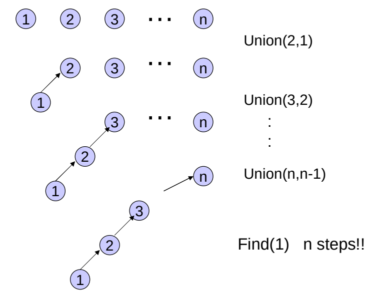
Now this doesn’t look good ☹

Can we do better? *Yes!*

1. Improve union so that *find* only takes $\Theta(\log n)$
   - Union-by-size
   - Reduces complexity to $\Theta(m \log n + n)$

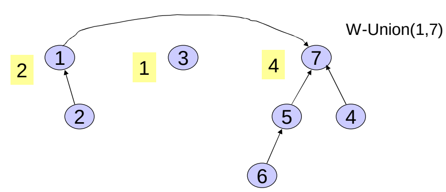
2. Improve find so that it becomes even better!
   - Path compression
   - Reduces complexity to <u>almost</u> $\Theta(m + n)$

A Bad Case

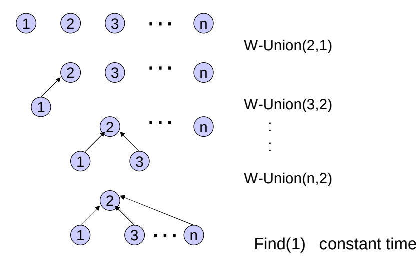


Weighted Union

- Weighted Union
  - Always point the *smaller* (total # of nodes) tree to the root of the larger tree



Example Again

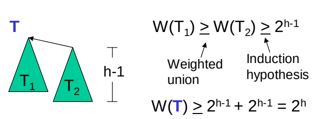


Analysis of Weighted Union

With weighted union an up-tree of height h has weight *at least* 2^h.

- Proof by induction
  - **<u>Basis</u>**: h = 0. The up-tree has one node, $2^0 = 1$
  - **<u>Inductive step</u>**: Assume true for all h’ < h.

Minimum weight up-tree of height h formed by weighted unions



Analysis of Weighted Union (cont)

Let T be an up-tree of weight n formed by weighted union. Let h be its height.

$n \geq 2^h$

$\log_2 n \geq h$

- Find(x) in tree T takes O(log n) time.
  - Can we do better?