CSE 326: Data Structures
Topic #11: Disjoint Set ADT (2)

Ashish Sabharwal
Autumn, 2003

Improving Union
Could we do a better job on this union?

Union-by-size: Code

```c
int Union(int x, int y) {
    if (up[x] == -1 && up[y] == -1) { // -1, this algorithm is in trouble
        up[y] = x;
        size[x] += size[y];
        return;
    } else { // new runtime for Union();
        up[x] = y;
        size[y] += size[x];
        return;
    }
}
```

Union-by-size: Find Analysis

- Complexity of Find: $\Theta(\text{max node depth})$
- All nodes start at depth 0
- Node depth increases
  - Only when it is part of smaller tree in a union
  - Only by one level at a time
- Result: tree size doubles when node depth increases by 1

Find runtime = $\Theta(\text{node depth}) =
\text{runtime for } m \text{ finds and } n-1 \text{ unions } =
```

Nifty Storage Trick

- Use the same array representation as before
- Instead of storing $-1$ for the root, simply store $-\text{size}$

[Read section 8.4, page 276]

How about Union-by-height?

- Can still guarantee $\Theta(\log n)$ worst case depth

Left as an exercise!
(will probably appear in Homework #3)

- Problem: Union-by-height doesn’t combine very well with the new find optimization technique we’ll see next
Improving Find

While we’re finding $f$, could we do anything else?

*Hint*: think splay trees…

Path Compression!

Recall: it need not be a binary tree!

Path Compression: Code

```c
int Find(Object x) {
    // x better be in // the set!
    int xID = hTable[x];
    int i = xID;
    // Get the root for // this set
    while(up[xID] != -1) {
        temp = up[i];
        up[i] = xID;
        i = temp;
    }
    // Change the parent for // all nodes along the path
    while(up[i] != -1) {
        temp = up[i];
        up[i] = xID;
        i = temp;
    }
    return xID;
}
```

(New?) runtime for Find:

Interlude: A Really Slow Function

Ackermann’s function is a really big function $A(x, y)$ with inverse $\alpha(x, y)$ which is really small

How fast does $\alpha(x, y)$ grow?

- $\alpha(x, y) = 4$ for $x$ far larger than the number of atoms in the universe ($2^{300}$)
- $\alpha$ shows up in:
  - Computation Geometry (surface complexity)
  - Combinatorics of sequences

A More Comprehensible Slow Function

$log^* x = \text{number of times you need to compute log to bring value down to at most 1}$

E.g. $log^* 2 = 1$

$log^* 16 = log^* 2^4 = 3$ \hspace{1cm} (log log 16 = 1)

$log^* 65536 = log^* 2^{2^{16}} = 4$ \hspace{1cm} (log log log 65536 = 1)

$log^* 2^{65536} = \ldots \ldots = 5$

Take this: $\alpha(m, n)$ grows even slower than $log^* n$ !!

Complex Complexity of Union-by-Size + Path Compression

Tarjan proved that, with these optimizations, $p$ union and find operations on a set of $n$ elements have worst case complexity of $O(p \cdot \alpha(p, n))$

For all practical purposes this is amortized constant time: $O(p \cdot 4)$ for $p$ operations!

- Very complex analysis – worse than splay tree analysis etc. that we skipped!
- Tarjan is also the (very smart) splay tree guy