Disjoint Set Union Find
or
How I Learned to Stop Linking
and Love the Array

A poorly named rehash of a
Winter 2002 lecture
Nick Deibel

Implementation

```c
typedef ID int;
ID up[100000];

ID find(Object x)
{
    assert(HashTable.contains(x));
    ID xID = HashTable[x];
    while(up[xID] != -1) {
        xID = up[xID];
    }
    return xID;
}
```

runtime: O(depth) runtime: O(1)

Room for Improvement:
Weighted Union

- Always makes the root of the larger tree the new root
- Often cuts down on height of the new up-tree

Weighted Union Code

```c
typedef ID int;

ID union(Object x, Object y) {
    rx = find(x);
    ry = find(y);
    assert(rx != ry);
    if (weight[rx] > weight[ry]) {
        new runtime of union:
        O(1)
        up[ry] = rx;
        weight[rx] += weight[ry];
    } else {
        new runtime of find:
        O(depth)
        up[rx] = ry;
        weight[ry] += weight[rx];
    }
}
```

Weighted Union Find Analysis

- Finds with weighted union are O(max up-tree height)
- But, an up-tree of height h with weighted union must have at least 2^h nodes

Base case: h = 0, tree has 2^0 = 1 node
Induction hypothesis: assume true for h < h'
and consider the sequence of unions.
Case 1: Union does not increase max height.
Resulting tree still has ≥ 2^h nodes.
Case 2: Union has height h' = 1 + h, where h = height of each of the input trees. By induction hypothesis each tree has ≥ 2^h nodes, so the merged tree has at least 2^h' = 2 * 2^h nodes. QED.
Room for Improvement: Path Compression

- Points everything along the path of a find to the root
- Reduces the height of the entire access path to 1

Path Compression Example

While we're finding e, could we do anything else?

Path compression!

Path Compression Code

```c
ID find(Object x) {
    assert(HashTable.contains(x));
    ID xID = HashTable[x];
    ID hold = xID;
    while(up[xID] != -1) {
        xID = up[xID];
    }
    while(up[hold] != -1) {
        temp = up[hold];
        up[hold] = xID;
        hold = temp;
    }
    return xID;
}
```

runtime: $O(\log n)$

Digression: Inverse Ackermann’s

Let $\log^k n = \log (\log (\log \ldots (\log n)))$

Then, let $\log^* n = \min k$ such that $\log^k n < 1$

How fast does $\log^* n$ grow?

- $\log^* (2) = 1$
- $\log^* (4) = 2$
- $\log^* (16) = 3$
- $\log^* (65536) = 4$
- $\log^* (2^{65536}) = 5$ (a 20,000 digit number!)
- $\log^* (2^{2^{65536}}) = 6$

Complex Complexity of Weighted Union + Path Compression

- Tarjan (1984) proved that $m$ weighted union and find operations with path compression on a set of $n$ elements have worst case complexity $O(m \cdot \log^*(n))$
  - actually even a little better!

- For all practical purposes this is amortized constant time