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

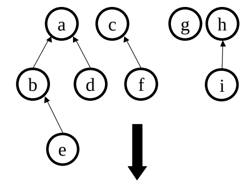
A poorly named rehash of a
Winter 2002 lecture

Nick Deibel

Nifty storage trick

A forest of up-trees can easily be stored in an array.

Also, if the node names are integers or characters, we can use a very simple, perfect hash.



	0 (a)	1 (b)	2 (c)	3 (d)	4 (e)	5 (f)	6 (g)	7 (h)	8 (i)
up-index:	-1	0	-1	0	1	2	-1	-1	7

Implementation

```
typedef ID int;
ID up[10000];

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

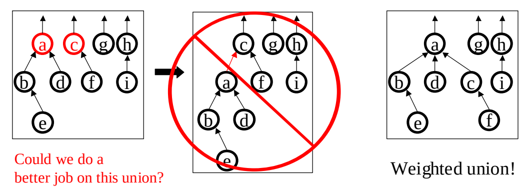
runtime: O(depth)

```
ID union(Object x, Object y)
{
  ID rootx = find(x);
  ID rooty = find(y);
  assert(rootx != rooty);
  up[y] = x;
}
```

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



Could we do a better job on this union?

Weighted union!

Weighted Union Code

```
typedef ID int;

ID union(Object x, Object y) {
  rx = Find(x);
  ry = Find(y);
  assert(rx != ry);
  if (weight[rx] > weight[ry]) {
    up[ry] = rx;
    weight[rx] += weight[ry];
  }
  else {
    up[rx] = ry;
    weight[ry] += weight[rx];
  }
}
```

new runtime of union: O(1)

new runtime of find: O(depth)

Weighted Union Find Analysis

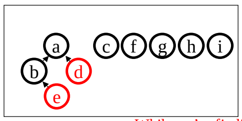
- Finds with weighted union are O(max up-tree height)
- But, an up-tree of height h with weighted union must have at least 2^h nodes

> 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 $\geq 2^h$ nodes.
> Case 2: Union has height $h' = 1+h$, where h = height of each of the input trees. By induction hypothesis each tree has $\geq 2^{h'-1}$ nodes, so the merged tree has at least $2^{h'}$ nodes. QED.

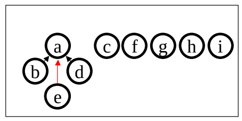
- $\therefore$, $2^{\text{max height}} \leq n$ and max height $\leq \log n$
- So, find takes $O(\log n)$

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



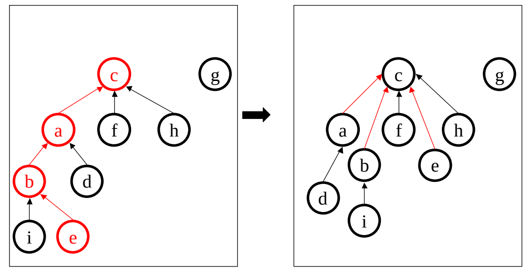
While we're finding ***e***, could we do anything else?



Path compression!

Path Compression Example

find(e)



Path Compression Code

```
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 $\log^{(k)} n = \underbrace{\log (\log (\log \ldots (\log n)))}_{k \text{ logs}}$

Then, let log* n = minimum *k* such that $\log^{(k)} n \le 1$

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 commpression 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