Disjoint Sets II
Chapter 8 in Weiss

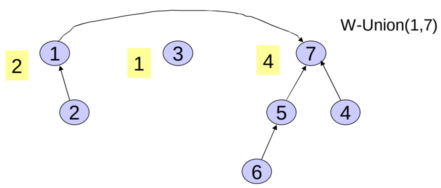
CSE 326
Data Structures
Ruth Anderson

Today's Outline

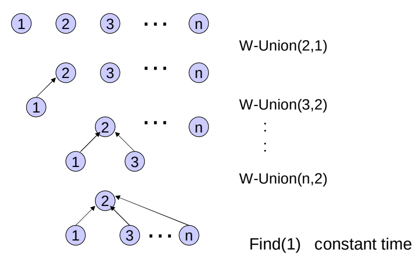
- **Announcements**
  - **Project 3 partner selection due Mon Feb 22 by 11pm, DO NOT WAIT UNTIL THEN TO START!**
  - **Written Homework #6 due Friday 2/26**
- **Today's Topics:**
  - **Disjoint Sets**
  - **Sorting**

Weighted Union/Union by Size

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



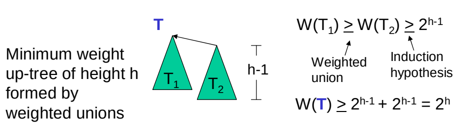
Example Again



Analysis of Weighted Union

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

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



Analysis of Weighted Union (cont)

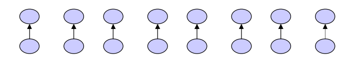
Let T be an up-tree of weight n formed by weighted union. Let h be its height.

$$n \geq 2^h$$
$$\log_2 n \geq h$$

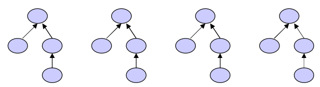
- Find(x) in tree T takes O(log n) time.
  - Can we do better?

Worst Case for Weighted Union

n/2 Weighted Unions

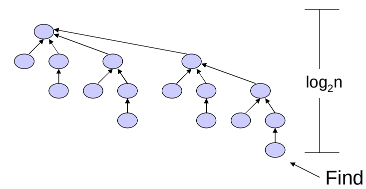


n/4 Weighted Unions



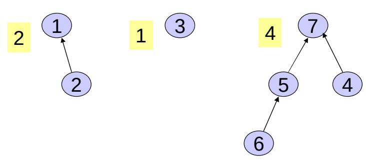
Example of Worst Cast (cont')

After n/2 + n/4 + …+ 1 Weighted Unions:



If there are $n = 2^k$ nodes then the longest path from leaf to root has length k.

Array Implementation



	1	2	3	4	5	6	7
up	-1	1	-1	7	7	5	-1
weight	2		1				4

Weighted Union

```
W-Union(i,j : index){
  //i and j are roots
  wi := weight[i];
  wj := weight[j];
  if wi < wj then
    up[i] := j;
    weight[j] := wi + wj;
  else
    up[j] :=i;
    weight[i] := wi +wj;
}
```

new runtime for Union():

new runtime for Find():

runtime for m finds and n-1 unions =

Nifty Storage Trick

- Use the same array representation as before
- Instead of storing **`-1`** for the root, simply store **`-size`**

[Read section 8.4, page 276]

How about Union-by-height?

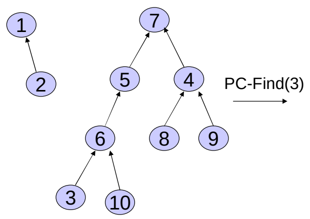
- Can still guarantee O(log *n*) worst case depth

  Left as an exercise!

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

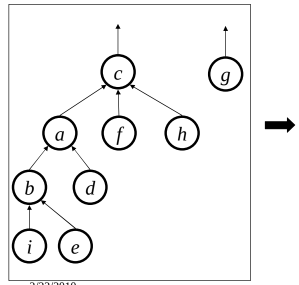
Path Compression

- On a Find operation point all the nodes on the search path directly to the root.



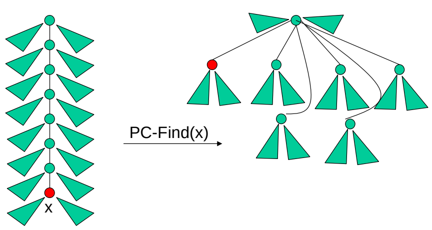
Student Activity

Draw the result of Find(e):



➡

Self-Adjustment Works



Path Compression Find

```
PC-Find(i : index) {
  r := i;
  while up[r] ≠ -1 do //find root//
    r := up[r];
  if i ≠ r then  //compress path//
    k := up[i];
    while k ≠ r do
      up[i] := r;
      i := k;
      k := up[k]
  return(r)
}
```


Path Compression: Code

```
int Find(Object x) {
  // x had better be in
  // the set!
 int xID = hTable[x];
 int i = xID;

  // Get the root for
  // this set
  while(up[xID] != -1)
  {
    xID = up[xID];
  }

  // 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})

α shows up in:

- Computation Geometry (surface complexity)
- Combinatorics of sequences

A More Comprehensible Slow Function

log* x = number of times you need to compute log to bring value down to at most 1

E.g. log* 2 = 1

log* 4 = log* 2^2 = 2

log* 16 = log* 2^{2^2} = 3 (log log log 16 = 1)

log* 65536 = log* $2^{2^{2^2}}$ = 4 (log log log log 65536 = 1)

log* 2^{65536} = ………….. = 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!

Disjoint Union / Find with Weighted Union and PC

- Worst case time complexity for a W-Union is O(1) and for a PC-Find is O(log n).
- Time complexity for $m \geq n$ operations on n elements is O(m log* n) where log* n is a very slow growing function.
  - Log * n < 7 for all reasonable n. Essentially constant time per operation!
- Using "ranked union" gives an even better bound theoretically.

Amortized Complexity

- For disjoint union / find with weighted union and path compression.
  - average time per operation is essentially a constant.
  - worst case time for a PC-Find is O(log n).
- An individual operation can be costly, but over time the average cost per operation is not.