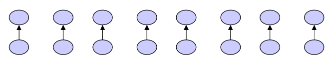
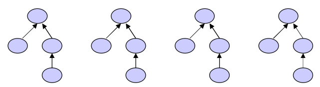
Worst Case for Weighted Union

n/2 Weighted Unions

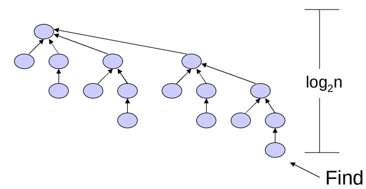


n/4 Weighted Unions



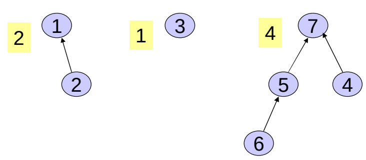
Example of Worst Case (cont')

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 299]

How about Union-by-height?

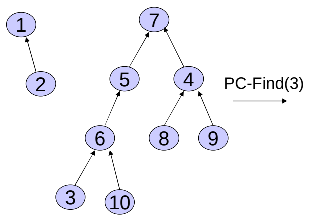
- Can still guarantee O(log *n*) worst case depth

  Left as an exercise! (see Weiss p. 300)

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

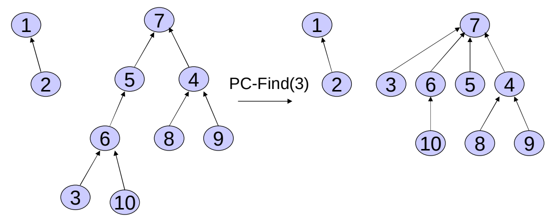
Path Compression

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



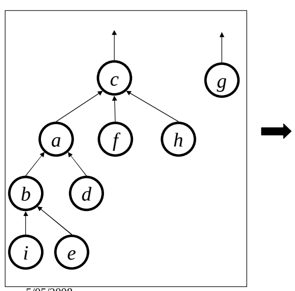
Path Compression

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



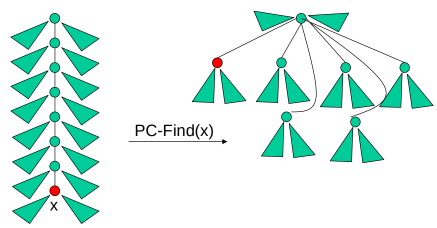
Student Activity

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];

  // Assert: r= the root, up[r] = -1
  if i ≠ r then   // if i was not a root

    temp := up[i];
    while temp ≠ r do // compress path
      up[i] := r;
      i := temp;
      temp := up[temp]

  return(r)
}
```

(New?) runtime for Find:

Interlude: A Really Slow Function

Ackermann's function is a <u>really</u> big function A(x, y) with inverse α(x, y) which is <u>really</u> small

How fast does α(x, y) grow?

α(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!

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.