

Announcements:

- Midterm exam 2 should be graded and returned in section this week (we hope).
- mp6 is available, due 4/15, 11:59p (midnightish)
- <http://bachman.cs.uiuc.edu:8080> (prizes for most useful feedback!)

Today:

Disjoint Sets

Graphs

First an animation (not great):

<http://www.cs.unm.edu/~rlpm/499/uf.html>

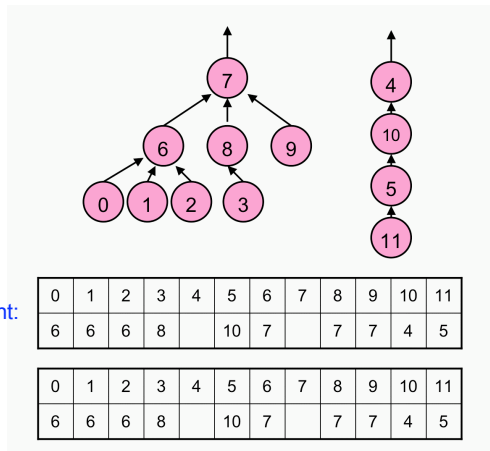
write down functions union, find

naïve: union $O(n)$, find $O(1)$

smart: union $O(1)$, find $O(\log n)$ worst case

can we do better?

Smart unions:



demo pointers on picture, one at a time

need to maintain info about height/size of our structures...

there's a slight complexity with maintaining height.

GUARANTEE $O(\log n)$ height tree

can we improve?

yes, the other time we ever manipulate these trees is during a find
(upward traversal) can we improve height THEN?

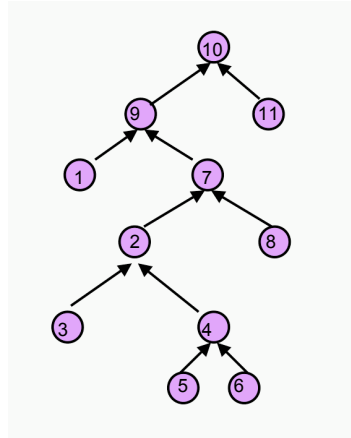
Smart unions:

```
int DS::Find(int i) {  
    if (s[i] < 0) return i;  
    else return Find(s[i]);  
}
```

```
void DS::UnionBySize(int root1, int root2) {  
    int newSize = s[root1]+s[root2];  
    if (isBigger(root1,root2)) {  
        s[root2]= root1;  
        s[root1]= newSize;  
    }  
    else {  
        s[root1] = root2;  
        s[root2]= newSize;  
    }  
}
```

union by height a little more complex, because heights are only updated when there is a tie... plus there is the 0 issue...

Path Compression:



suppose I have a tree that looks like this, and I do Find(4).

show compression on the way up.

get rid of old pointer, replace with new.

draw resulting tree!!!

HOW DO YOU IMPLEMENT THAT?

Path Compression:

```
int DS::Find(int i) {  
    if (s[i] < 0) return i;  
    else return Find(s[i]);  
}
```

```
void DS::UnionBySize(int root1, int root2) {  
    int newSize = s[root1]+s[root2];  
    if (isBigger(root1,root2)) {  
        s[root2]= root1;  
        s[root1]= newSize;  
    }  
    else {  
        s[root1] = root2;  
        s[root2]= newSize;  
    }  
}
```

no one knows average case performance of this... I'm going to tell you worst case on next slide. guesses? (we already have $O(\log n)$ from smart unions.

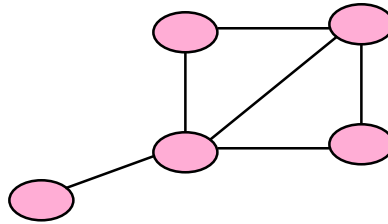
Analysis:

before I can tell you analysis, I have to tell you about a new function...
 $\log^* n$

show that $\log^* 2^{65536} \rightarrow 65536 \rightarrow 16 \rightarrow 4 \rightarrow 2 \rightarrow 1$ (5 steps) = 5

MAIN RESULT: any sequence of m union and find operations results
in worst case running time of $O(m \log^* n)$, where n is the number of
items. (actually, there is an even better result...

Graphs: Mathematically represented as a pair of sets (V,E) , where V is the collection of “vertices” and E is the collection of “edges.”



Applications: http://www.aisee.com/graph_of_the_month/archive.htm

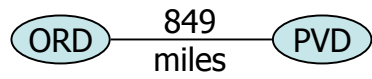
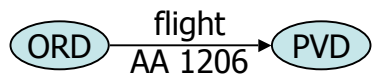
we require more structure than this.

V stored as a collection of vertex objects, contains data (show city labels on graph)

E stored as a collection of edge objects, contain data (show distances on graph)

additional structure: adjacencies edge **defined** by (u,v)

Graphs: directed and undirected edges

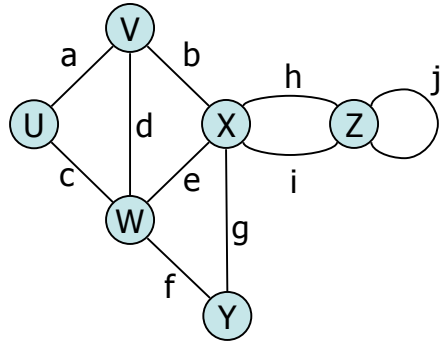


directed edges (graph with directed edges is a directed graph)

ordered pair vs unordered $(uv) \neq (vu)$ in ordered pairs

undirected can be implemented as two directed edges.

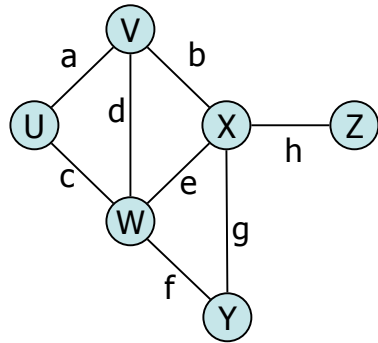
Graphs: more terminology



endpoints of edge b are
edges are incident on vertex v
the degree of vertex x is
are adjacent vertices
are multi-edges
is a self-loop

more terminology
“endpoints” of edge b

Graphs: yet more terminology



A _____ is an alternating sequence of edges and vertices that starts and ends at vertices.

A _____ is a _____ in which each vertex appears at most once.

path:

cegfd

notice that sequence of vertices implicitly gives edges

simple path:

abh