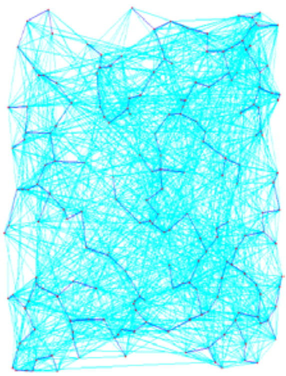


Finish Graphs

Lecture 24
CS211 – Fall 2005

Announcements

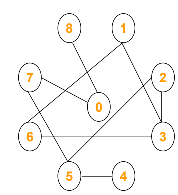
- Paul Chew’s office hour for today (Thursday, Nov 17) is cancelled

Graph Overview

- Graph Definitions
  - Directed graph (digraph)
  - Undirected graph
  - Directed acyclic graph (dag)
  - Paths & cycles
- Graph Properties
  - Graph coloring
  - Planarity
  - Bipartite graphs
- Graph Implementations
  - Adjacency matrix
  - Adjacency lists
- Graph Searching
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
- Graph Algorithms
  - Single-source shortest paths
    - Dijkstra’s Algorithm
  - Minimum spanning tree (MST)
    - Prim’s Algorithm
    - *Kruskal’s Algorithm*

New Problem: Connected Components

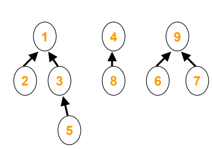
- Given a set of edges (p,q), *quickly* determine if some p’ and q’ are in the same *connected component*
  - Def: Two vertices are in the same *connected component* if there is a path between them
- Example:
  - Given edges (1,3) (2,3) (5,4) (6,3) (7,5) (1,6) (7,0) (0,8) (5,2)
  - Are 4 and 6 in the same component?
- How can a computer resolve this for a large set?



Union and Find

- We break this problem into two operations
  - Union: Combine two sets
  - Find: Given an item, determine the “name” of the set that contains it
- Many applications
  - Checking components of a dynamic graph
  - Computers in a network: Can p communicate with q?
  - Minimum Spanning Trees

Union/Find using Reverse Trees

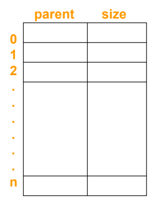


The root is the “name” of the set

- Find
  - Follow links to root
  - Time O(n) in the worst case
- Union
  - Link root of one tree to the other
  - Time O(1) in the worst case

An Improvement: Union by Size

- Note: Every union takes one tree and moves everything in it one step farther from the root
- Idea: Make the *smaller* tree be the one that moves down
- Can show
  - Time for union is O(1)
  - Time for find is O(log n)
- Implement using arrays
- Initially, all items have no parent and size 1



Union-by-Size Lemma

<u>Lemma</u>

A tree with height h contains at least 2^h nodes

<u>Proof</u>

- The only way in which a node can change its level is when it is within the *smaller* of two trees participating in a union
- Thus, when any node *x* drops a level, the tree that it is within doubles in size (or more)
- If a node is at level h then it is within a tree of size at least 2^h

<u>Corollary</u>

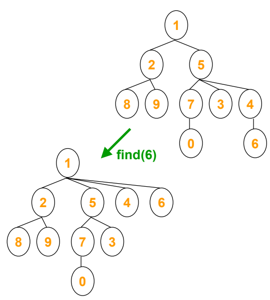
Worst-case time for find is O(log n) where n is the total number of items

<u>Proof</u>

- The largest possible tree contains n nodes, so the deepest node is at level *log n*

Union-by-Size + Path Compression

- Idea: Every time we "find" something, we update every item we touch so that it points at the root
  - This is *almost* free since we have to touch these items anyway
  - Intuition: next time we find one of these items it will be quicker
- Does this help?



Yes, It Helps

<u>Theorem</u> (Tarjan)

Using weighted union and path compression, a sequence of n union/find operations takes time O(n α(n))

- The function α(n) is the inverse of *Ackerman's function* and it grows <u>*very*</u> slowly

<u>Definition</u> (Ackerman's function)

$$A(p,q) = \begin{cases} 2q & \text{if } p = 0 \\ 0 & \text{if } q=0,\ p>0 \\ 2 & \text{if } q=1,\ p>0 \\ A(p-1, A(p, q-1)) & \text{if } q>1,\ p>0 \end{cases}$$

This definition is a bit different from the text's version, but both have similar properties

Ackerman's Function

- $A(0,q) = 2 + \ldots + 2 = 2q$
- $A(1,q) = 2 * \ldots * 2 = 2^q$
- $A(2,q) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$
  (a height-q stack of 2's)
- Thus $A(2,4) = 2^{16} = 65{,}536$
- Each level does the operation from the previous level q times
- What is A(3,4)?
- So A(4,4) must be <u>*extremely*</u> large

Definition for α(n)

<u>Definition</u> (*inverse Ackerman's function*)

α(n) = least x such that $A(x,x) \geq n$

Note that $\alpha(n) \leq 4$ for any integer n that we are <u>*ever*</u> likely to encounter

Union/Find Analysis

Theorem (Tarjan)
Using weighted union and path compression, a sequence of n union/find operations takes time $O(n\ \alpha(n))$

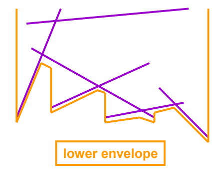
- Note that $\alpha(n) \leq 4$ for any integer n that we are *ever* likely to encounter

- Is the $\alpha(n)$ factor really necessary?
  - Yes: Tarjan showed a *lower* bound of $\Omega(n\ \alpha(n))$ for union/find
  - Claim: the inverse Ackerman's function is *not* just an artifact of this one problem

Lower Envelope of Line Segments

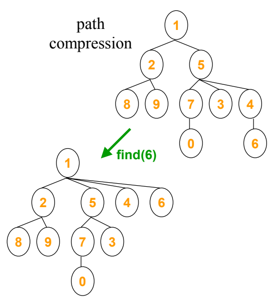
- Given n line segments in the plane, what is the worst-case complexity of their *lower envelope*?

$\Theta(n\ \alpha(n))$



Union/Find Summary

- Operations
  - Union: Combine two sets
  - Find: Given an item, determine the "name" of the set that contains it
- Use reverse trees
  - Each item points at its parent
  - The root is the "name" of the set
- Union-by-Size
  - Always make the larger tree be the root
- Path Compression
  - Every time we "find" something, we update every item we touch so that it points at the root
- Result
  - n operations take time $O(n\ \alpha(n))$



Two MST Algorithms (Both Greedy)

Kruskal's Algorithm

- Choose the shortest edge e such that
  - e is not yet processed
  - e does not make a cycle
    - We use Union/Find to check this

Prim's Algorithm

- Choose the shortest edge e such that
  - e touches the tree
  - e touches a vertex not in the tree

Kruskal's MST Algorithm

```
KruskalMST(G):
    E = edges of G;
    forest = empty;
    do
        <u,v> = least cost edge of E;
        E = E - <u,v>;
        if (u and v in different trees)
            forest = forest ∪ <u,v>;
    while (E is nonempty);
    return forest
```

- Can sort the edges initially (or can use a PQ)
- Use Union/Find to check for different trees and to combine trees
- Total worst-case time: $O(m \log m)$ when using adjacency lists
- Time is $O(n^2 + m \log m)$ for adjacency matrix