COS 528 notes

Janet Suzie Yoon

09/18/2006

3 main problems we will go over in class

1. Union Find Problems, disjoin set union, evaluation of paths on trees
2. Dynamic Trees, link cut trees, topology trees. Applications: max network flow problem and mergable trees
3. Network Flow
   (a) single commodity
   (b) maximum flow
   (c) simple balancing algorithms - very simple algos for solving max flow
   (d) generalize to multi commodity flow

UNION FIND
Given: Collection of items grouped into disjoint sets.
Assume: Initially each of the n items are each in a singleton set. Each set has a name.
Operations:

1. $find(x)$: given element x, returns the name of the set containing x
2. $unite(x)$: COmbines set A and B into a single set named A

We can maintain set names by assuming the name is an interger from 1 to n and have an array data structure pointing to the root element of the tree. We can also get rid of set names altogether.
UNION FIND without set names
Operations:

1. $find(x)$: given element x, return the canonical element of the set containing x

2. *link*(x, y): x, y are canonical elements. COmbine sets containing x and y and designates etierh x or y as canonical elements of the new set

Implementing the operations: Represent each set as a *rooted tree.* The root is the canonical element and the nodes are the elements. Represent the tree using parent pointers. Initially have a coolection o fnodes, each with a self pointer as the parent pointer.

Link: To do a link, have one of the nodes link to another. This is done in $O(1)$ time since we only need to change 1 pointer.

Find: We can potentially build a long path of size n. $find(x)$ takes time proportinal to the length of the path from x to root. This can be up to linera time $O(n)$. Let n be the number of elements and let $m =$ number of finds. Assume $m \geq n$. Worse tmie could be $O(mn)$. ideally we want $O(1)$ time for find. This is not possible, but we can come close.

Smart Linking: There are two types of smart linking

1. linking by size: maintain in each root the number of nodes in its tree. Make the small roote point to the large root. Break ties arbitrarily. Now no path has length $\geq log(n)$. The maximum depth in a tree of n nodes is $log(n)$. Therefore $find(x)$ worst case time is $O(log(n))$. Example: binomial trees.

2. linking by rank: The rank is intended to be an estimate of max tree depth. In general, the rank is an upper bound of the tree depth. Assume we have two trees with rank r and rank s. Singletons have rank of 0. If $r > s$ we have the tree wtih rank s point to the tree with rank r. Note the rank from this link remains r since the max path in s is $s + 1$ and $r > s$ and the max path in r is r. If $r = s$ then link s to r and increase the rank of this link to $r + 1$. Any tree root of rank r has $\geq 2^r$ descendants. We can prove this by induction by the number of links. THis implies the max depth in a tree of k nodoes is $\leq log(k)$.

In the fast find method, the find is constant time but the link isnt

Path Compression: the idea is to avoid long paths. During a find, we run up the root to get the answer. While we run up the path to the root, put direct pointers from the node to the root. We might have a bad find, but it makes up for it in following finds.

Variants of Path Compression: Can we get the benefit of path compression with a single walk?

1. Path Reversal: While walking up a path, have all nodes point to x. This changes the root/canonical element. This doe snot lead to inverse akerman time bound.

2. Path Splitting: As you walk up the path, let it point one step up and then it used to be. This does give the inverse akerman bound. Splits into 2 half-sized paths

3. Path Halving

Given: values on nodes (or edges)
Goal: Combine values along a path (via some binary operations e.g. +, max)
Operations:

1. eval(x): return the combination of values along the path from x to root

2. link(x,y): x, y are roots. Make y the parent of x

Application: Kruskals MST alg
How to verify MST
Given: Graph and tree connect all vertices
Return: Is MST or Not?
for every non tree edges, can we make the tree cheaper by including another edge or not. For every nontree edge, walk along the path and see if the max path is less than MST. This is a path evaluation problem.