Assignment 3

Algorithms: Correctness and Complexity

Due Nov 6th, 2014 in class.

Q0 [10]. We can model a data network as directed graph where switches, routers, computers etc. are nodes and data links are edges. Each edge is associated with a currently available bandwidth. We wish to set up a virtual circuit from node s to node f that has a bandwidth of 1Gbps. I.e., we need a path from s to f whose weakest link (i.e. minimal bandwidth link) is at least 1Gbps. In fact, to avoid creating bottle necks, we would like to find a route whose weakest link is as large as possible. Design an algorithm that takes as input a simple directed graph $G = (V, E)$, two nodes $s, f \in V$ and a bandwidth function $w : E \to \mathbb{R}^+$, and that prints out every simple path from s to f and the weight of its weakest link.

Q1 [5] A tree over a set A is either

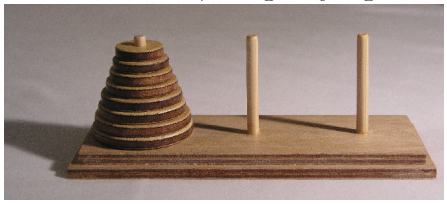
- Leaf()
- Branch(t, a, u) where a is in A and t and u are trees over A.

The fringe of a tree is given by

- fringe(Leaf()) = []
- fringe(Branch(t, a, u)) = fringe(t)^[a]^fringe(u)

Write a procedure that takes a finite sequence s over A and prints all trees with s as its fringe.

Q2 [5] The towers of Hanoi consists of 3 stacks of disks. There are n disks in total, each a different size from the others; disk 0 is the smallest; disk 1 is larger than disk 0; and so on. A **legal** configuration is one where each disk is on some stack and, on each stack, each disk is only above larger disks. A **legal** move moves the top disk of one stack to the top of another stack that is either empty or is topped by a larger disk. Design an algorithm that starts in *any legal configuration* and moves all disks to one tower, using only legal moves.



Q3 [10]. A priority queue is a data structure that holds a collection of values (say strings). There is an operation to add a value and an operation to report and remove the smallest value currently in the collection. Show that for any implementation of a priority queue which only accesses its data by copying it and comparing it, the worst-case time complexity of at least one of these operations is $\Omega(\log n)$.

Hint: You might want to use the result we will look at in Tuesday's class that sorting (under the same restrictions) requires $\Omega(n \log n)$ time.

Bonus [5] Suppose s is an array of n things and p is an array of n integers representing a permutation (i.e. the value of p is a one-one onto function from $\{0, ..n\}$ to $\{0, ..n\}$). Each item of p indicates the rank of the corresponding item of s. Write a procedure to sort s according to rank.

procedure sortByRank(p : array $\langle \mathbb{N} \rangle$,var s : array $\langle T \rangle$)
  precondition p is a permutation of $\{0, ..p.\text{length}\}$ and $p.\text{length} = a.\text{length}$
  changes s
  postcondition $\forall i \in \{0, ..p.\text{length}\} \cdot s_0(i) = s(p_0(i))$

For example, if p and s are initially

s	bob	doug	alice	clara
p	1	3	0	2

then the final value s should be

s	alice	bob	clara	doug

and it doesn't matter what the final value of p is.

Implement this procedure in time $\Theta(n)$, where n is the length of p. Do not use any temporary arrays. Explain why the time is linear.