Collaborators

Ben Nelson and I collaborated for this assignment on all questions except question 2. We discussed $5c$ in some detail and discussed the rest without going into too many specifics.

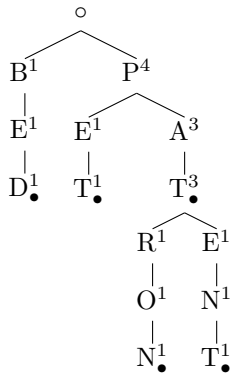
Question 1: Prefix trees revisited ..... **[8]**

Recall the prefix tree data structure for storing a collection of strings (Lecture 2). Suppose we wish to support the following operations: ADD string, DELETE string, QUERY string, COUNT-PREFIX. The last operation takes a string w and returns the number of strings in our collection having w as a prefix.

We saw how to build a data structure that supports the first three operations in time $O(|w|)$, where $|w|$ is the length of the string w being added/deleted/queried. Describe a way to modify the procedures so that COUNT-PREFIX can also be answered in time $O(|w|)$.

A prefix tree is shown below with the words $BED, PET, PAT, PATRON$ and $PATENT$. The root of the tree is denoted by $\circ$. Each node of the tree stores a letter that is part of a word along with a count that denotes the number of words the letter is used in, a marker $\bullet$ when a word ends with the letter and an optional set of pointers to other nodes. A node can have an edge connecting it to another node when the two letters in the nodes are subsequent letters in a word. Although the tree shown does not have directional edges, the edges are to be read as if they were pointing downward. For example, the connected nodes B, E and D after the root show that the word BED has been stored. Each of these letters have a count of 1 to show that they have been used only in one word and the $\bullet$ after D shows that the word BED end with the node D. The letter P has a count of 4 since it is used in words $PET, PAT, PATRON$ and $PATENT$. There is a $\bullet$ marker on the letter T in the middle of the word $PATENT$ since PAT is also stored in the tree and the two words have the first three letters common.

For the $QUERY$ operation, a client of the prefix tree will start at the root and look for connected nodes starting with the first letter of the word being queried and ending with the last letter of the word. If the last node also has a $\bullet$ marker, then it means that the word being queried, is present in the tree. So the word BE will not be found in the tree even though nodes B and E exist after the root, but the word BED will be found.



Question 2: Recurrences, recurrences ..... **[12]**

Solve each of the recurrences below, and give the best $O(\cdot)$ bound you can for each of them. [*Hint:* You might find chapters 12 and 13 of the Lehman, Leighton book (available from the course homepage) useful.]

(a) **[3]** $T(n) = 2T(\sqrt{n}) + 4$.

(b) **[4]** $T(n) = T(n/3) + T(n/2) + \sqrt{n}$.

(c) **[5]** Suppose you have devised a divide-and-conquer algorithm for a certain problem that breaks up a problem of size n into three subproblems of size $n/2$ each, solves them recursively, and then combines the solutions. Suppose the time for breaking up and combination is given by $g(n)$. Suppose in your first attempt, you had a combine step that had $g(n) = n^{1.8}$. Does it make sense to think further

and reduce $g(n)$ to $n^{1.5}$? What about reducing it even further to $g(n) = n \log n$? (Assume you only care about the asymptotic running time of the overall procedure.)

Question 3: Bubble sort ..... **[8]**

Recall the bubble sort algorithm (pseudocode at http://en.wikipedia.org/wiki/Bubble_sort). Recall that the worst case running time is $O(n^2)$.

(a) **[3]** Give an example of an input array $A[]$ that is (a) not sorted to begin with, and (b) the algorithm takes time $O(n)$ on $A[]$.

(b) **[5]** Give an example of an input array on which the algorithm takes time $\Theta(n^{3/2})$.

Question 4: Pecking orders ..... **[10]**

(*Source:* Jeff Erickson's exercises)

Whenever groups of pigeons gather, they instinctively establish a pecking order. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles – for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

(a) **[5]** Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left. [*Hint:* start with two piegons, and consider placing the rest one after another.]

(b) **[5]** Suppose you are given a directed graph representing the pecking relationships among a set of n pigeons. The graph contains one vertex per pigeon, and it contains an edge $i \to j$ if and only if pigeon i pecks pigeon j. Describe and analyze an algorithm to compute a pecking order for the pigeons, as guaranteed by part (a). Your algorithm should run in time polynomial in n.

Question 5: More divide, better run time ..... **[12]**

Moral: as long as the "conquer" step is not too expensive, dividing a problem into smaller sub-problems typically helps.

Consider the problem of multiplying two n-digit integers. We saw in class (Lecture 3, Karatsuba's algorithm) that dividing into two $n/2$ bit numbers and combining appropriately, we can compute the product in time $O(n^{1.585..})$. So we can wonder: can division into three pieces help?

Suppose we have two n-digit integers A, B. Split A into A_1, A_2 and A_3, each having $n/3$ digits (assume n is a power of 3 for this problem). So also, split B into B_1, B_2 and B_3. If the numbers are in base 10, we can write $A = 10^{2n/3}A_1 + 10^{n/3}A_2 + A_3$. Let $p(z)$ be the polynomial $A_1z^2 + A_2z + A_3$, and let $q(z) = B_1z^2 + B_2z + B_3$. Define $r(z) = p(z)q(z)$. The degree of $r(z)$ is 4, so $r(z) = C_1z^4 + C_2z^3 + C_3z^2 + C_4z + C_5$, for some coefficients C_i.

(a) **[2]** Observe that the product of A and B is $r(10^{n/3})$.

(b) **[2]** Observe that once we know the C_i, assuming they are $O(n)$ digits each, $r(10^{n/3})$ can be found in $O(n)$ time.

(c) **[4]** We thus need to find the C_i. The trick is to find the values of $r(z)$ at a few *small* values of z. Specifically, we find $r(z)$ for $z \in \{-2, -1, 0, 1, 2\}$. Show that we can compute each of these $r(z)$ in time $T(n/3) + O(n)$, where $T(n/3)$ is the time needed to multiply two $n/3$ digit numbers. Explain why these values uniquely determine C_i. [*Hint:* use $r = p \cdot q$, and the degree of r.]

(d) **[4]** To find the C_i, write a system of linear equations, and show how these can be solved to obtain C_i in $O(n)$ time.

These observations let us conclude that $T(n) = 5T(n/3) + O(n)$, which results in an overall run time of $O(n^{1.464..})$, which is better than Karatsuba's algorithm. (This can be pushed further, to obtain $n^{1+\epsilon}$, for any constant $\epsilon > 0$.)