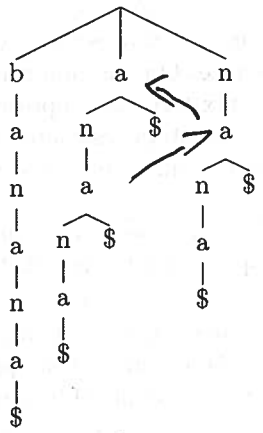
6.854 Dan Ports
Advanced Algorithms drkp@mit.edu
2004/09/22 Collaborators: {sarahl, bhebert, pramook}

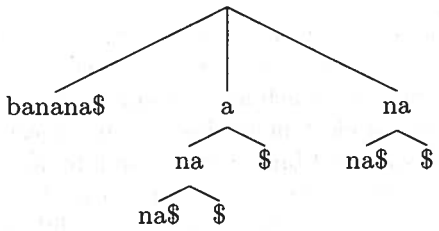
Problem Set 2

P1: 9
P2: 10
P3: 10
P4: 10
39
-1
not enough detail

Problem 1: a)



b)



Problem 2: a) We show that a common suffix tree of the texts $T_1, T_2, \ldots, T_n$ can be built in time $O(|T_1| + |T_2| + \cdots + |T_n|)$.

We begin by appending a unique terminating symbol $\$_i$ to each of the texts T_i. We then concatenate the n texts, and generate a suffix tree from the concatenated string. This tree has size proportional to the length of the concatenated string, i.e. $O(|T_1| + |T_2| + \cdots + |T_n|)$, and requires the same amount of time to build. However, it contains suffixes that cross from one text to another. Note that these suffixes are precisely the suffixes that contain a terminating symbol $\$_i$ in the middle of the string. Therefore, we can eliminate these invalid suffixes by performing a traversal of the tree and eliminating all subtrees of terminator nodes $\$_i$. Though this is clearly not efficient since it creates nodes that will be discarded, it is simple, and provides the desired asymptotic runtime, since generating the suffix tree and traversing it requires time linearly proportional to the size of the tree, which is proportional to the length of the concatenated string, $O(|T_1| + |T_2| + \cdots + |T_n|)$.

b) A string s is a substring of all n texts if it is a prefix of a suffix of each of the n texts. A suffix of each text will end in that string's terminator, $\$_i$. Thus, a string s is a substring of all n texts if and only if the subtree of the node N corresponding to it contains all of the terminators: $\$_i$ for $1 \leq i \leq n$.

c) Given two texts T_1 and T_2, we can find their largest common substring in time $O(|T_1| + |T_2|)$. We begin by computing their common suffix tree as with the algorithm above.

We then augment each node of the suffix tree with two bits of information: whether the subtree

rooted at that node contains the terminator for the corresponding text. From above, we know that every node with such a bit set corresponds to a substring of the corresponding text. Thus, the substrings of both texts will be the ones with both bits set. Observe that a bit will be set at a node if and only if the same bit is set at one of its children, or it is the appropriate terminator. Therefore, we can perform a post-order traversal, visiting each node and setting the bits appropriately to indicate which terminators, if any, are in its subtree. Correctness of this algorithm can be proven by induction on the height of the subtree. The runtime is linear in the size of the tree.

We next observe that the common substrings of the two texts corresponds precisely to the set of nodes with both mark bits set. The longest common substring will correspond to the deepest (doubly-)marked node. In an uncompressed tree, this could be identified easily by performing a breadth-first traversal of the tree. Only a minor modification is required to account for the fact that each edge in a compressed suffix tree may represent more than one character: we weight each edge according to the number of characters associated with it. We then expand the least-weighted paths as in the BFS algorithm; the runtime remains proportional to the number of nodes in the tree, and correctness is obvious.

Each of the steps in this process requires time proportional to the size of the tree, which in a compressed suffix tree is proportional to the length of the texts: $O(|T_1| + |T_2|)$.

Problem 3: a) Suppose that we have just found item x in a binary tree, and wish now to find item y. In order to perform this lookup in time proportional to the length of the path from x to y, we will need to maintain a small amount of information between lookups. We maintain two lists of the ancestors of x: one list of the ancestors whose value is less than x, and one list of those greater than x.

Assume without loss of generality that $y > x$ (the argument is analogous in the $y < x$ case). In this case we consider the list of ancestors whose value is greater than x. We scan backwards across this list (i.e. towards the root) until we reach an ancestor v whose value is greater than y. We then backtrack to the previously scanned element in this list, an ancestor node that we will call w. We then search this subtree (in the standard binary-tree-search manner) for y.

We also briefly note two degenerate cases: if the process of scanning the list backwards never finds a node greater than both x and y, then x and y are on different sides of the root, and y must be found by performing a search from the root, but this is still proportional to the shortest path. If the first ancestor in the list is greater than both x and y, then y is in x's subtree, and the algorithm proceeds correctly by designating x as w.

To show that this is correct, first observe that y is in the left subtree of v, since $v > y$. It cannot be the case that y is above v, because then v would be on the right side of y and x on the left. Next we claim that y is in a subtree of w. Recall that w is the next node along the path from v to x that was greater than x. Thus w is reachable from v by a chain of one or more right-child links. This means that y is in w's subtree unless it is reachable by a left-child link from one of the intermediate nodes, but this is not possible because y is larger than x and x is in w's subtree. We have thus shown that y is in the subtree of w. Performing a search for y on this subtree will therefore locate it correctly.

We now show that this produces the desired runtime. We claim that w is the least common ancestor of x and y. We will prove this claim by showing that x and y are in different subtrees of w. Note that $x < w$, since w is found in the list of ancestors greater than x; thus, x is in w's left subtree. But $y > w$: if it were not so, then $w > y$ and we would have contradicted the definition of v as the lowest ancestor of x satisfying $v > x, y$. This means y is in the right subtree of w. So w is the least common ancestor of x and y, and the shortest path from x to y travels up the chain of parent pointers from x to w, then down to y. Note that the number of operations required to scan the lists backwards from x to find v and w is bounded above by the distance between x and w plus 1, and the time required to find y in the subtree of w is proportional to the depth of y in this subtree. So the runtime is proportional to the length of the shortest path from x to y.

ok good.

(For completeness, we note that we must update the finger ancestor lists to reflect the new sets of ancestors of y, but the process for doing so is standard and does not affect the asymptotic runtime: discard all entries in the path between x and w, and add new entries reflecting the path from w to y.)

b) Let x and y be nodes in a treap, and let d be the number of items between them in sorted order. We show that y can be found from x in time $O(d)$. Without loss of generality, assume the values in the treap are the integers 1 to n, and $x < y$. Then $d = y - x$.

We will use the previous result to obtain this bound, so observe that the shortest path from x to y travels upward from x to the least common ancestor w, then downward to y. Thus, the path from x to y consists of the ancestors of x and the ancestors of y, except for their common ancestors.

We define three classes of indicator random variables. Let X_i be 1 if i is an ancestor of x, Y_i be 1 if i is an ancestor of y, and $Z_i = X_i Y_i$ be 1 if i is an ancestor of both x and y. By the previous result,

$$\mathrm{E}\,[\text{path length}] = \sum_{i=1}^{n} X_i + Y_i - 2Z_i$$

Consider a node z. There are three cases, and we consider them individually.

1. $z < x \leq y$. Then z will be an ancestor of x only if it is inserted with higher priority than all items with values between z and x. So $X_i = \frac{1}{x-z+1}$. z will be an ancestor of y if it is inserted with higher priority than all elements with values between z and y — but then it will also be an ancestor of x. So $Y_i = Z_i = \frac{1}{y-z+1}$.
2. $x < z \leq y$. Then z will be an ancestor of x only if it is inserted with higher priority than all items with values between x and z. So $X_i = \frac{1}{z-x+1}$. z will be an ancestor of y if it is inserted with higher priority than all elements with values between z and y: $Y_i = \frac{1}{y-z+1}$. And it will be an ancestor of both x and y if it is inserted before all values between x and y: $Z_i = \frac{1}{y-x+1}$.
3. $x < y \leq z$. Then z will be an ancestor of y only if it is inserted with higher priority than all items with values between y and z. So $Y_i = \frac{1}{z-y+1}$. z will be an ancestor of x if it is inserted with higher priority than all elements with values between x and z — but then it will also be an ancestor of y. So $X_i = Z_i = \frac{1}{z-x+1}$.

Performing the summation,

7/7

$$\begin{aligned}
\mathrm{E}\,[\text{len}] &= \sum_{i=1}^{n} X_i + Y_i - 2Z_i \\
&= \sum_{i=1}^{x} X_i + Y_i - 2Z_i + \sum_{i=x+1}^{y} X_i + Y_i - 2Z_i + \sum_{i=y+1}^{n} X_i + Y_i - 2Z_i \\
&= \sum_{z=1}^{x} \frac{1}{x-z+1} - \frac{1}{y-z+1} + \sum_{z=x+1}^{y} \frac{1}{z-x+1} + \frac{1}{y-z+1} - 2\frac{1}{y-x+1} \\
&+ \sum_{z=y+1}^{n} \frac{1}{z-y+1} - \frac{1}{z-x+1} \\
&= -2 + H(x) + H(y-x+1) + H(y-x+1) - H(y) + H(y-x+1) \\
&+ H(n-y+1) + -H(n-x+1) + H(y-x+2) \\
&= -2 + H(x) - H(y) + 4H(y-x+1) + H(n-y+1) - H(n-x+1) \\
&= -2 + (H(x) - H(y)) + 4H(y-x+1) + (H(n-y+1) - H(n-x+1) \\
&= O(\log(y-x+1)) = O(\log d)
\end{aligned}$$

So the expected path length is $O(\log d)$. By the previous part, this means we can find y from x in $O(\log d)$ time.

Problem 4: **a)** Let X_i be the random variable defined as the maximum of w_i independent samples from the uniform distribution over the interval $[0,1]$. Note that (for $x \in [0,1]$) X_i is less than x if and only if each of the w_i independent samples are less than x. Thus,

$$\Pr[X_i < x] = \Pr[\max\{w_i \text{ samples}\} < x] = x^{w_i}$$

Let Y be a random variable distributed uniformly over the domain $[0,1]$, and define

$$Z_i \stackrel{\text{def}}{=} Y^{\frac{1}{w_i}}$$

so that

$$\begin{aligned}\Pr[Z_i < x] &= \Pr\left[y^{\frac{1}{w_i}} < x\right] \\ &= \Pr[Y < x^{w_i}] \\ &= \Pr[X_i < x]\end{aligned}$$

Thus, X_i and Z_i have the same distribution. So instead of generating a value for X_i, which requires sampling w_i independent random variables, we can instead obtain the same result by generating a value for Z_i, which is a single random variable.

b) We begin with a lemma:

Lemma 1. *If $H(n)$ is the nth harmonic number, then*

$$\frac{a}{x+a} \leq H(x+a) - H(x)$$

Proof.

$$\begin{aligned}\frac{a}{x+a} &= \sum_{i=1}^{a} \frac{1}{x+a} \\ &\leq \sum_{i=1}^{a} \frac{1}{x+i} \\ &\leq \sum_{i=1}^{x+a} \frac{1}{i} - \sum_{i=1}^{x} \frac{1}{i} = H(x+a) - H(x)\end{aligned}$$

□

Without loss of generality, assume that the values of the elements x_i are ordered in increasing order by their subscripts, i.e. $x_1 \leq x_2 \leq \cdots \leq x_n$. To obtain the expected access time of element x_i, we first consider the set $Q^-_{x_i}$ of elements that are ancestors of x_i and have values less than x_i; the procedure for the set $Q^+_{x_i}$ is analogous.

Lemma 2. *If x_i an element with weight w_i, and $x_j, x_k, x_l, \ldots$ are elements with weights $w_j, w_k, w_l, \ldots$ respectively, the probability that x_i has the highest priority value of the set is $\frac{w_i}{w_i+w_j+w_k+w_l+\cdots}$.*

Proof. The priority value of x_i is the maximum of w_i samples from the uniform $[0,1]$ distribution. Therefore, it has the highest priority value of the set only if one of its w_i samples has the highest priority value of the total $w_i + w_j + w_k + w_l + \cdots$ samples. □

Let x_h be some element. x_h can only be in $Q^-_{x_i}$ if $x_h < x_i$, of course. Moreover, there can be no element x_j where $x_h < x_j < x_i$ and the priority value of x_j is greater than the priority value of x_h. That is, x_h must have the highest priority value of the elements $x_{h+1} \cdots x_i$. By Lemma 2, this happens with priority $\frac{w_h}{w_h + w_{h+1} + \cdots + w_i}$. Summing over the elements x_h, we find that

$$\mathrm{E}\left[|Q^-_{x_i}|\right] = \sum_{h=1}^{i} \Pr\left[x_h \in Q^-_{x_i}\right] = \sum_{h=1}^{i} \frac{w_h}{w_h + w_{h+1} + \cdots + w_i}$$

Applying Lemma 1,

$$\mathrm{E}\left[|Q^-_{x_i}|\right] = \sum_{h=1}^{i} H(w_h + w_{h+1} + \cdots + w_i) - H(w_{h+1} + \cdots + w_i)$$

which telescopes to

$$\mathrm{E}\left[|Q^-_{x_i}|\right] = H(w_h + w_{h+1} + \cdots + w_i) - H(w_i) = H(W) - H(w_i)$$

Asymptotically, this shows that

$$\mathrm{E}\left[|Q^-_{x_i}|\right] = O(\log W - \log w_i) = O(\log \frac{W}{w_i})$$

By symmetry, inverting the direction of the inequalities, we can obtain the same bound on $|Q^+_{x_i}|$, and hence on $|Q_{x_i}|$. This is the depth of the node x_i and so the expected access time for x_i is $O(1 + \log \frac{W}{w_i})$.