[redacted]

Problem Set 8

Due: 8 am on Thursday, November 11

Problem 8-1 (Optimal Trees) (21+10 points)

In the lecture we looked at the problem of constructing optimal binary search trees with input K. Implicitly, K corresponded to the inorder traversal of the tree T. In this problem, we will look at the problem of applying it to other traversals and other trees. Further, for this problem, we will have the following cost function:

$$Cost(T) = T.root.key + 2 \cdot Cost(T.root.left) + 2 \cdot Cost(T.root.right)$$

and $Cost(\textbf{nil}) = 0, Cost(v) = v.key$ when v is a leaf node. By $T.root.left$ and $T.root.right$ we denote the left and the right subtrees respectively.

(a) (7 points) $K[1, \ldots, n]$ is the input, as defined before, and K corresponds with the inorder traversal of the binary tree T. We will let T to be any binary tree. Now, let $\textsc{Best}[i, j]$ represent the least cost binary tree you can make from $K[i, \ldots, j]$. Note that this tree would have $j - i + 1$ nodes.

- (1 point) Give base case(s) for Best.
- (4 points) Give and justify a recursive formulation for $\textsc{Best}[i, j]$ in terms of the subproblems .
- (2 points) State and justify the running time of an efficient implementation of your algorithm to compute Best.

(b) (7 points) $K[1, \ldots, n]$ is the input, as defined before, but now K corresponds with the postorder traversal of the binary tree T. Note that if T was a binary search tree, then there is exactly one way to define T. Instead, we will let T to be any binary tree. Now, let $\textsc{Best}[i, j]$ represent the least cost binary tree you can make from $K[i, \ldots, j]$ such that the postorder traversal of this tree yields $K[i, \ldots, j]$. Note that this tree would have $j - i + 1$ nodes.

- (1 point) Give base case(s) for Best.
- (4 points) Give and justify a recursive formulation for $\textsc{Best}[i, j]$ in terms of the subproblems .
- (2 points) State and justify the running time of an efficient implementation of your algorithm to compute Best.

(c) (7 points) Now let T be any arbitrary tree, i.e., each node v can have any number of children $w_1, \ldots, w_\ell$ for $\ell \geq 0$. Let us modify the cost function as follows:

$$Cost(T) = T.root.key + \sum_{1 \leq i \leq \ell} 2 \cdot Cost(w_i) \ .$$

By $w_1, \ldots, w_\ell$ we denote the children subtrees. Further, let $K[1, \ldots, n]$ be the postorder traversal of an n-node arbitrary tree. Now, let $\textsc{Best}[i, j]$ represent the least cost arbitrary tree you can make from $K[i, \ldots, j]$. Note that this tree would have $j - i + 1$ nodes.

- (1 point) Give base case(s) for Best.
- (4 points) Give and justify a recursive formulation for $\textsc{Best}[i, j]$ in terms of the subproblems .
- (2 points) State and justify the running time of an efficient implementation of your algorithm to compute Best.

(**Hint**: Your algorithm will run in time $O(n^3)$ and will use $O(n^2)$ in space. In the case of binary trees, you found a root and then found the most optimal left child and the most optimal right child and then attached it to the root. In the case of arbitrary trees, you do not have any restrictions on number of children. So, you could simply choose to incrementally grow the number of children by attaching more and more children to the left.)

(d) (**Extra Credit**)(10 points) We are given $K[1, \ldots, n]$ which corresponds to the postorder traversal of any arbitrary tree. There are no constraints on the number of children a node can have. Let us modify the *Cost* function as follows: The cost of a tree rooted at v with children $w_1, \ldots, w_\ell$

$$Cost(v) = v.key + \sum_{1 \le i \le \ell} (\ell - i + 1) \cdot Cost(w_i) \ .$$

Here $w_1, \ldots, w_\ell$ are the subtrees from left to right. Let $\textsc{Best}[i, j, \ell]$[1] be the best possible constrained tree T with $j - i + 1$ nodes whose postorder traversal lies in $\textsc{Best}[i, j, \ell]$ and where the root of the tree has *exactly* ℓ children. Here, $\ell \in \{0, 1, \ldots, , n\}$.

- (2 points) Give base case(s) for Best.
- (6 points) Give and justify a recursive formulation for $\textsc{Best}[i, j, \ell]$ in terms of the subproblems .
- (2 points) State and justify the running time of an efficient implementation of your algorithm to compute Best. What is the space complexity?

(**Hint**: You begin by doing something similar to the previous part. However, now the subtree rooted at k can have any number of children and we have changed the definition of Best. Keep this in mind when formulating the recursive solution.)

Problem 8-2 (More with LCS) (9+10 points)

In this problem we will look at variations of the LCS problem. Let $X[1, \ldots, m]$ and $Y[1, \ldots, n]$ be arrays of n characters over the alphabet Σ. We can then assign a cost function $C : \Sigma \times \Sigma \to \mathbb{R}^+$ which assigns a cost of an alignment. Then, we can find the cost of a subsequence (not necessarily

[1] We still need this three dimensional formulation because the coefficient i is contingent on the child position.

common) of length k between X and Y of length k is defined by indices $1 \leq i_1 < i_2 < \ldots < i_k \leq m$ and $1 \leq j_1 < j_2 < \ldots < j_k \leq n$ for $\ell = 1, \ldots, k$ as $\sum_{\ell=1}^{k} C(X[i_\ell], Y[j_\ell])$. You may assume that the cost function is 0 if either one of the characters is empty.

Consider the following optimization problem: Find, for the best k, a k-length subsequence of A and B which maximizes the cost of the subsequence. This problem is called maximum cost subsequence problem (MCS)

(a) (2 points) This problem can be viewed as a generalization of the LCS problem. State the cost function that shows that the LCS problem is a specific instance of the maximum cost subsequence problem, i.e., if we had an algorithm to solve the MCS problem, we can instantiate this cost function, to get the LCS of X and Y instead. Briefly justify your solution.

(b) (7 points) Let $\textsc{Best}[i, j]$ denote the best possible cost between $X[1, \ldots, i]$ and $Y[1, \ldots, j]$.

- (1 point) Give base case(s) for Best.
- (4 points) Give and justify a recursive formulation for $\textsc{Best}[i, j]$ in terms of the subproblems.
- (2 points) State and justify the running time of an efficient implementation of your algorithm to compute Best.

(c) (**Extra Credit**)(10 points) Consider the following cost of a subsequence:

$$\sum_{\ell=1}^{k} C(X[i_\ell], Y[j_\ell]) - k^2 \ .$$

Clearly define the dimension of Best and the value it stores. Present a recursive formula with base cases for Best that solves the maximum cost subsequence problem with the subsequence as defined in this part. Do not forget to state and justify the running time of an efficient implementation of your algorithm to compute Best. (**Hint**: This is another instance of dimensional lifting. Go from two dimensions to three dimensions.)

Problem 8-3 (Enumerating LCS) (21 points)

Let us go back to original LCS problem. Recall that an LCS of length k of two strings $X[1, \ldots, m]$ and $Y[1, \ldots, n]$ is given by indices $1 \leq i_1 < i_2 < \ldots < i_k \leq m$ and $1 \leq j_1 < j_2 < \ldots < j_k \leq n$ such that $X[i_\ell] = Y[j_\ell]$ for $\ell = 1, \ldots, k$. The subsequence is given by $X[i_1]||\ldots||X[i_\ell]$ where $||$ denotes concatenation. For all parts of this problem you may assume that the matrix c where $c[i, j]$ contains the length of the LCS of $X[1, \ldots, i]$ and $Y[1, \ldots, j]$ has been correctly computed and is available.

In this problem, we will begin by looking at two versions of the LCS problem - the set version and the count version.

Set Version. The set version of the LCS problem is denoted by the set $L[i, j]$ which is the set of all LCS of $X[1, \ldots, i]$ and $Y[1, \ldots, j]$. For example, when $X = [ABCBDAB], Y = [BDCABA]$ we have that the length of LCS is 4 but $LCS(X, Y) = \{BDAB, BCAB, BCBA\}$. In this problem, LCS is the set of all subsequences, and not the indices themselves. We use $LCS(X, Y)$ to denote

the set of all subsequences of strings X and Y while $L[i,j]$ is our memoized notation. $L[i,j] = LCS(X[1,\ldots,i], Y[1,\ldots,j])$.

(a) (4 points) Prove that $LCS(X'||b, Y'||b) = LCS(X', Y')||b$ where $||$ denotes the concatenation operation of two strings and $LCS(X', Y')||b$ denotes the concatenation of b to every string in $LCS(X', Y')$. Here, $LCS(X, Y)$ denotes the set of all LCS of X and Y. (**Hint**: Prove that $LCS(X'||b, Y'||b) \subseteq LCS(X', Y')||b$ and $LCS(X', Y')||b \subseteq LCS(X'||b, Y'||b)$)

(b) (5 points) Present a recursive formula for $L[i,j]$. You will use set operations to combine your results. For simplicity, when $ij = 0$, you are given that $L[i,j] = \emptyset$. To this end, you will use your result from part (a) for the case when $X[i] = Y[j]$. As for the case when $X[i] \neq Y[j]$, you will have to look at three sub-cases. Do not forget to justify your answer. (**Hint**: Your recursive formula's cases will take into account the values in c.)

Cardinality Version. Let $\lambda(i,j)$ denote the cardinality of the set $L[i,j]$, i.e., $\lambda[i,j] = |L[i,j]|$. λ is a useful estimate for identifying the running time an implementation to compute $L[i,j]$. Indeed, we have seen that a brute force solution of the LCS problem would be to list all subsequences of X of size $\ell = 1,\ldots,m-1$ and then check if they are present in Y. This is an exponential solution because there are 2^m possible subsequences of X. In the next few parts, you will prove that $\lambda(m,n)$ can be truly exponential (and consequently the running time to compute L becomes exponential).

Let $X_{3n} = [0,1,2,\ldots,0,1,2] = [(012)^n]$, i.e.,$X[i] = \begin{cases} 0 & i \mod 3 = 1 \\ 1 & i \mod 3 = 2 \\ 2 & i \mod 3 = 0 \end{cases}$

and $Y_{3n} = [1,0,2,\ldots,1,0,2] = [(102)^n]$, i.e., $Y[i] = \begin{cases} 1 & i \mod 3 = 1 \\ 0 & i \mod 3 = 2 \\ 2 & i \mod 3 = 0 \end{cases}$

(c) (3 points) Prove by induction that $\lambda[3n,3n] = \Omega(2^n)$ [2]. Therefore, it follows that there are instances of X, Y where there are exponential number of distinct LCS substrings which would cause the running time of an algorithm to compute L to be exponential in time.

Count Version. In the set version we looked at the set of the LCS strings itself. For the count version, we will look at the actual subsequences defined by the index. Then, we say that two LCS defined by $\langle i_1,\ldots,i_k; j_1,\ldots,j_k\rangle$ and $\langle i'_1,\ldots,i'_k; j'_1,\ldots,j'_k\rangle$ are distinct subsequence if there exists at least one index different in the two subsequences, i.e., $\exists\ \ell : 1 \leq \ell \leq k$ such that either $i_\ell \neq i'_\ell$ or $j_\ell \neq j'_\ell$. For example, $X = [AC], Y = [ACC]$ we have that the length is 2 and the set contains just the string $\{AC\}$. However, the count is actually 2 as we can form AC by using either of the two C's in Y. Let $Count[i,j]$ be the number of distinct pairs of LCS for $X[1,\ldots,i]$ and $Y[1,\ldots,j]$ and these have length $c[i,j]$. You may assume that $Count[i,j] = 1$ when $ij = 0$, which corresponds to the empty string being the LCS.

(e) (7 points) These are the asks for the question:

[2] You can actually prove a tighter bound that $\lambda[3n,3n] = \Theta(4^n/\sqrt{n})$ but this requires a little more math. However, you can begin by proving that it is actually $\Omega(\sqrt{6}^n)$ as a first step. This is just an aside.

- Give and justify a recursive formulation for $Count[i, j]$ in terms of the subproblems .

  (2 points) When $X[i] = Y[j]$

  (3 points) When $X[i] \neq Y[j]$
- (2 points) State and justify the running time of an efficient implementation of your algorithm to compute $Count$. As always, assume that the arithmetic operations and comparisons can be made in $\Theta(1)$ time.

(**Hint**: As always, you will look at two cases $X[i] = Y[j]$ and $X[i] \neq Y[j]$. However, you will have to update $Count[i, j]$ by looking at length values of the other values too. For example, when $X[i] = Y[j]$, we know that $c[i, j] = 1 + c[i - 1, j - 1]$. However, how do you update $Count[i, j]$? There are three sub-cases. If $c[i, j - 1] = c[i - 1, j] = c[i, j]$, $c[i - 1, j] < c[i, j - 1] = c[i, j]$, and $c[i, j - 1] < c[i - 1, j] = c[i, j]$. In addition, we have the naive case of $c[i, j] > \max(c[i-1, j], c[i, j-1]$. Now think about coming up with corresponding cases for $X[i] \neq Y[j]$.

Follow the hint to get full credit.)

(f) (2 points) Assume wlog that $n = \min(m, n)$. In part (d), you proved that λ can be exponential. With this, revise the running time of your algorithm from part (e) where you will assume that comparisons and additions (and subtractions) of two integers that are i bits long takes time $\Theta(i)$.