CS140: Algorithms
Fall 2001

Solutions to Assignment 9

1. You win the candy store jackpot! Your prize is a bag full of the candy of your choice. The bag they let you fill is huge but, unfortunately, it is plastic and will break if you try to put in more than K pounds. Alas, you don't like candy. But you figure you can sell the candy at Platt and make big bucks. So your objective, in choosing the candy, is to maximize its value. The candy store has n different kinds of candy. Candy i costs p_i per pound and the store currently has x_i pounds of it. (Typically $x_i < K$.) Design an algorithm to determine an optimal selection of candy.

   Sort the candies so that $p_1 \geq p_2 \geq \ldots p_n$. Choose as much of candy 1 as will fit in your bag. If you exhaust the supply of cand 1 and there is room left over in the bag, recursively find the optimal selctions for candies $i = 2, \ldots, n$ with a bag that can hold $K - x_1$.

2. You are writing the HELP section of a software tool and decide to organize the help topics in a balanced binary search tree in order to minimize the search time. The help topics are {AVOCADO, BANANA, CARROT, DANDELION, ETCETERABERRY, FIZZLEWACKY, GRAPE, HUPPALA, ICKYPOO, JELLO, and KRUD}. Give a blanced binary search tree for this input.

   After many months of use you discover users rarely search for FIZZLEWACKY but often look for KRUD. And unfortunately, FIZZLEWACKY has a shorter search path than KRUD in your tree. Give a (not-necessarily-balanced) binary search tree for the help topics in which the search path for KRUD has length 1. (We'll define the length of the search path for a key as the number of queries needed to find that key.)

   You realize that your original tree minimizes the worst case search time but it might be better to minimize the expected search time. The expected search time is $\sum_i p_i \ell(w_i)$, where $w_1, \ldots w_n$ are the help topics, $\ell(w_i)$ is the length of the search path for w_i, and p_i is the probability that a user searches on the i-the topic.

   (a) Give a recursive definition for the minimum expected search time $E_{OPT}(w_1, \ldots, w_n, p_1, \ldots, p_n)$.

$$E_{OPT}(w_1,\ldots,w_n,p_1,\ldots,p_n) = \sum_{i=1}^{n} p_i + \min_{i=1,\ldots,n-1} E_{OPT}(w_1,\ldots,w_i,p_1,\ldots,p_i) + E_{OPT}(w_{i+1},\ldots,w_n,p_{i+1},\ldots,p_n)$$

(b) Give a dynamic programming algorithm that computes E_{OPT}.

Construct an $n \times n$ table T. For $i \leq j$, entry $T(i,j)$ will be $E_{OPT}(w_i,\ldots,w_j,p_i,\ldots,p_j)$. We begin by computing the diagonal elements, $T(i,i) = p_i$. We continue computing successive diagonals using the following definition.

$$T(i,j) = \sum_{k=i}^{j} p_k + \min_{k=i,\ldots,j-1} T(i,k) + T(k+1,j)$$

When we conclude the solution is $T(1,n)$.

(c) Give a dynamic programming algorithm that produces a search tree with expected search time E_{OPT}.

Amend the previous algorithm to store, for each $T(i,j)$, an index k that minimizes $\sum_{k=i}^{j} p_k + \min_{k=i,\ldots,j-1} T(i,k) + T(k+1,j)$. The index stored at the $T(1,n)$ will be the root of the search tree. Recursively build the left and right subtrees, which are rooted at, respectively, the index stored $T(1,k)$ and at $T(k+1,n)$.

(d) Analyze the running times of your algorithms.

Each table entry takes $O(n)$ time to compute. There are $O(n^2)$ table entries. The running time is $O(n^3)$.

3. M is an $n \times m$ matrix of distinct integers in which the rows are sorted in ascending order (smallest at left) as well as the columns (smallest at top). The numbering convention we'll follow is that $M(1,1)$ is the upper, left element of the matrix and $M(n,m)$ is the lower right element.

(a) Construct a sample 3×4 matrix that is sorted as described.

Left to the student.

(b) Give the indices of the elements of M that are guaranteed to be strictly smaller than the element $M(k, \ell)$.

Indices i, j where $i < k$ and $j < \ell$.

(c) Give the indices of the elements of M that are guaranteed to be strictly larger than the element $M(k, \ell)$.

Indices i, j where $i > k$ and $j > \ell$.

(d) Design an algorithm to search for an integer x in M. The algorithm can access M through queries of the form "How does $M(k, \ell)$ compare to x?" which is answered (a) "$M(k, \ell) = x$," (b) "$M(k, \ell) > x$," or (c) "$M(k, \ell) < x$." Your algorithm should return the location of x in M (the indices) or report that x is not in M.

To receive full credit your algorithm should make at most $m + n - 1$ queries in worst case. To receive half-credit give an $O(n + m)$ algorithm.

Query position $M(1, m)$. If $M(1, m) = x$ we are done. If $M(1, m) < x$ we can eliminate the first row of M and recurse. If $M(1, m) > x$ we can eliminate the last column of M and recurse. In each step we eliminate a row or column so $m + n - 1$ queries are needed in worst case.

4. Given a parenthesis-free expression over the integers and $\{+, -, *\}$ we want to parenthesize the expression so as to maximize its value. For example, the expression $-2 + 5 * 3 - 10$ can be parenthesized as $-2 + ((5 * 3) - 10)$, which evaluates to 3. It can also be parenthesized as $((-2 + 5) * 3) - 10$, which evaluates to -1.

   (a) Parenthesize the following expressions so as to maximize their values.

      i. $10 \;*\; -3 \;+ 5$

      ii. $3 \;-\; 4 \;-\; 2 \;*\; 5 \;+\; -1$

   Left to the student.

   (b) Design an algorithm that takes as input an expression $v_1 o_1 v_2 o_2 \ldots o_{n-1} v_n$, where each v_i is an integer and each o_i is an operator from $\{+, -, *\}$, and parenthesizes it so as to maximize its values.

   First we give an algorithm to compute the maximum value of the input expression. We'll construct an $n \times n$ table T. At $T(i, j)$, where $i \leq j$, we'll store the maximum and minimum values of $v_i o_i v_{i+1} \ldots o_{j-1} v_j$. We initialize the elements on the main diagonal as $T(i, i) = v_i$. We compute successive diagnonals using the recursive definitions

$$T(i,j).max = max_{k=i,\ldots j-1}(T(i,k).max o_k T(k+1,j).max, T(i,k).max o_k T(k+1,j).min, T(i,k).min o_k T(k+1,j).max, T(i,k).min o_k T(k+1,j).min)$$

   and

$$T(i,j).min = min_{k=i,\ldots j-1}(T(i,k).max o_k T(k+1,j).max, T(i,k).max o_k T(k+1,j).min, T(i,k).min o_k T(k+1,j).max, T(i,k).min o_k T(k+1,j).min).$$

   When we conclude, the maximum value of the input expression is at $T(1, n)$.

   The algorithm can be modified in as described for the search tree problem above to construct the actual parenthesization.