# CS 161 Summer 1997

## Handout #17

## Problem Set 4 Due 3:05p, Tuesday, July 22, 1997

### Readings

1. Handout 15.
2. Chapters 15, 16.
3. We encourage you to read pages 266–268 of handout 16: they describe another dynamic programming algorithm, namely one solving a variant of the knapsack problem. The complete handout 16 will be required reading for problem set 5.
4. Read the project description, and start working on the project now! We have already covered all course material necessary to complete the project. Also, there will be no extensions.

### Exercises

Exercises are for you to work out, with the assistance of classmates and the course staff. Please don't hand in your solutions.

1. Exercise 15.1-7.
2. Exercise 15.3-6.
3. Exercise 15.3-7.
4. Exercise 16.2-2.
5. Show the optimal binary search tree for the following words and associated probabilities of occurrence:

| Word | Probability |
|---|---|
| a | 0.18 |
| and | 0.19 |
| I | 0.23 |
| it | 0.21 |
| or | 0.19 |

6. The country of Algorithmia uses a currency with coins (or banknotes) of $n$ different denominations; the $i$th coin has value $v_i > 0$ units. For example, in the US, the currency unit is the *cent*, and available "coins" include

the \$5 dollar bill ($v_1 = 500$), the \$1 dollar bill ($v_2 = 100$), the quarter ($v_3 = 25$), and the dime ($v_4 = 10$).

We own a store in Algorithmia. A customer walks in, buys the Algorithmia TV guide, and needs $N$ currency units worth of change for her payment. How can we give her the exact change using the smallest total number of coins? In other words, devise an algorithm that, given $N$, will

- return the minimum total number of coins we need to represent $N$ units of currency, and
- list, for each type of coin $i$, the number of such coins used in some optimal representation.

7. For the *longest increasing subsequence* problem, we are given a sequence of integers $a_1, a_2, \ldots, a_n$; our task is to find a sequence of indices $i_1 < i_2 < \cdots < i_k$ such that $k$ is maximal, and $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$. As an example, if the input is $3, 1, 4, 1, 5, 9, 2, 6, 5$, any longest increasing subsequence has length four; one of them is

$$a_2 = 1 < a_3 = 4 < a_5 = 5 < a_6 = 9.$$

Give an $O(n^2)$ algorithm to solve the longest increasing subsequence problem.

### Problems

Your problem set grade depends only on the solutions you hand in for the problems below.

1. **[5 points]** Exercise 15.1-5.
2. **[5 points]** Exercise 15.2-2.
3. **[20 points]** The Computer Division of SiliconBank wants you to design a data structure that handles the database of a single account in the bank. This data structure should support insertion of past and future transactions, deletion of existing transactions, and balance calculations. In particular, your data structure should support the following operations; you may assume that no two transactions occur on the same date:

   - INITIALIZE: Initializes the account. The initial balance of the account is \$0. This operation should take $O(1)$ time.
   - INSTRANS$(A, D)$: Inserts a transaction for the amount of $A$ dollars which took place on date $D$. $A$ can be either positive or negative, and should be added to the account balance starting from the following day. $D$ can be arbitrary; it need not be today's date. This operation should take $O(\log n)$ time, where $n$ is the number of transactions in the database.

- DelTrans($D$): Deletes the transaction that occured on date $D$, if there is any. When a transaction is deleted, the corresponding amount should be subtracted from the account balance starting from the following day. This operation should take $O(\log n)$ time.
- Balance($D$): Returns the account balance at the beginning of date $D$. This operation should take $O(\log n)$ time. No transaction need exist in the database for $D$.

Present a clear and concise description of each operation, but *do not write (pseudo-)code.*

4. **[5 points]** Exercise 16.1-3.

5. **[10 points]** Exercise 16.3-4.

6. **[25 points]** Problem 16-2.

7. **[10 points]** In this problem, you will modify the optimal binary search tree generation algorithm (handout 15) to allow for unsuccessful searches. Let

   - $q_j$, for $1 \leq j < N$, be the probability that a search is performed for any word $W$ satisfying $w_j < W < w_{j+1}$,
   - $q_0$ be the probability of performing a search for any word $W$ such that $W < w_1$, and
   - $q_N$ be the probability of performing a search for $W > w_N$.

   Notice that
   $$\sum_{i=1}^{N} p_i + \sum_{j=0}^{N} q_j = 1.$$

   (a) **[7 points]** Describe an efficient dynamic programming algorithm to build the optimal binary search tree for a given set of words $w_i$ and associated probabilities of occurrence $p_i$, and for given probabilities of unsuccessful searches $q_j$. Please give a clear and concise description of your algorithm, but *do not write (pseudo-)code.*

   (b) **[3 points]** Give a tight upper bound on the running time of your algorithm. Give a clear and intuitive argument justifying your answer; a formal and rigorous proof is not necessary.

8. **[20 points]** You bring an $\ell$–foot log of wood to your favorite sawmill. You want it cut in $k$ specific places: $\ell_1, \ell_2, \ldots, \ell_k$ feet from the left end. The sawmill charges $x$ dollars to cut an $x$–foot log any place you want.

   (a) **[15 points]** Describe an efficient dynamic programming algorithm to determine the order in which they should cut your log so as to minimize your cost. Please give a clear and concise description of your algorithm, but *do not write (pseudo-)code.*

(b) **[5 points]** The running time of your algorithm should be polynomial in $k$. What is this polynomial? Give a clear and intuitive argument justifying your answer; a formal and rigorous proof is not necessary.