

# CSE 548 Homework Assignment 2

Due in Class on Tuesday, October 20

October 14, 2009

There are four problems. Generic format of the solution is as follows:

1. Discuss your idea briefly.
2. Give the recursion and justify it.
3. Explain how you memoize.
4. Argue the time and space complexity of your algorithm.

## Problems

1. A stock market analyst has a large data set of prices recorded daily for a particular stock of interest. To study a trend for the stock, he would like to find a period during which the price of the stock increased the most. Specifically, he has an array of daily closing prices  $p = [p_1, p_2, \dots, p_n]$ , where  $p_i$  is the closing price on the  $i$ th day on record. He would like to find a pair of days  $(i, j)$  for which  $i < j$  and  $p_j - p_i$  is the largest among all such pairs. Help him design an  $O(n)$ -time algorithm to find such a pair.
2. As a graduate student, renting cheaply is always a plus. Two landlords Alice and Bob run a pretty weird business model. For each of the next  $n$  months, they pre-set their monthly rent prices. In the  $i^{\text{th}}$  month, Alice's apartments are set at  $a_i$  dollars each, while Bob's apartments are set at  $b_i$  dollars each. Each of them also allows tenants to specify which months in the next  $n$  months they want to sign the lease for. For example, if you want, you can arrange your leases so that you will stay in an Alice's apartment for 3 months, then switch to Bob's for 2 months, and then switch back to Alice's for 1 more, and so on. The drawback, of course, is the moving cost, which is fixed at  $c$  dollars. For instance, if you arrange your leases in the order,

<Alice, Alice, Bob, Alice, Bob, Bob>

Then, your total cost for the next 6 months is  $a_1 + a_2 + c + b_3 + c + a_4 + c + b_5 + b_6$ . (We ignore the initial moving-in cost, because that has to be paid no matter where we stay.)

## PART 1

Consider the following strategy:

CRAZY APARTMENT HUNTING

```
1: for  $i \leftarrow 1, n$  do
2:   if  $a_i < b_i$  then
3:      $D[i] \leftarrow Alice$ 
4:   else
5:      $D[i] \leftarrow Bob$ 
6:
```

By giving a counter example, show why the strategy is sub-optimal.

## PART 2

Given the rental prices  $a_1, \dots, a_n$  and  $b_1, \dots, b_n$ , along with the moving cost  $c$ , determine the cheapest way to sign leases with Bob and Alice. The output is the cost and an array  $D$  of  $n$  elements, in which each element is either Alice or Bob.

3. Computational biology is a big driver for dynamic programming. Consider a utility that processes DNA strands. A primitive operation in the utility splits a strand into two pieces. Since this operation involves copying the original strand, it takes  $n$  units of time to split a strand of length  $n$ , regardless of the location of the cut. Suppose, now, that you want to break a strand into many pieces. The order in which the breaks are made can affect the total running time. For example, if you want to cut a 20-units long strand at positions 3 and 10, then making the first cut at position 3 incurs a total cost of  $20 + 17 = 37$ , while doing position 10 first has a better cost of  $20 + 10 = 30$ .

Give a dynamic programming algorithm that, given the locations of  $m$  cuts in a strand of length  $n$ , finds the minimum cost of breaking the strand into  $m + 1$  pieces. The locations of  $m$  cuts are given in an array  $c_1, \dots, c_m$ , where  $1 \leq c_1 < \dots < c_m < n$ .

4. Interleaving is an efficient coding strategy to protect data from corruption and burst errors, that blow away chunks of bits at a time. It is employed in many areas like storage, wireless transmissions etc. Now consider a communication interface that waits on two application buffers which are filled bit-wise. Both the applications repeatedly pump the same data into their own respective buffers for some finite time. (broadcast-type applications might need this setup). Of course the two applications write different data, but repeatedly do so using into own buffers. Think of them as two messages being transmitted repeatedly using the same interface.

The interface is implemented to empty the buffers in a hasty fashion. This might be useful for urgent communication. So the interface transmits an unpredictable interleaving of the two messages. Also as the applications may not be synchronized amongst themselves, there may not be a *neat and simple* interleaving. What the receiver knows is that the received message is an interleaving of two sequences that are repetitions of two different canonical messages.

Assuming that left to right order of the individual messages is preserved in the received interleaved message, give a dynamic programming solution to recover the two sent messages.

**\*\*UPDATE\*\*** : To make things simple, assume we have a dictionary of all possible messages. So now the problem is just to see if the received message is an interleaving of some pair of messages.

### Breaking this down

Say the two applications are  $A_1$  and  $A_2$  and their messages are  $m_1 = 101$  and  $m_2 = 01$ .

$A_1$  sends a repetition of  $m_1$ ,  $r(m_1) = 101101101101$ .  $A_2$  sends a repetition of  $m_2$ ,  $r(m_2) = 010101$ .

The receiver receives an interleaving of  $r(m_1)$  and  $r(m_2)$ ,  $I[r(m_1), r(m_2)]$ . This interleaving is quite random. The only property that it needs to satisfy is that while interleaving parts of individual messages, left to right associativity of messages is maintained. So the rightmost part of a message wont appear before leftmost part in the interleaved message.

Following are some examples of valid interleavings:

1.  $I_1[r(m_1), r(m_2)] = 101100111001110101$
2.  $I_2[r(m_1), r(m_2)] = 101010101101110101$
3.  $I_3[r(m_1), r(m_2)] = 101101001101101101$

The algorithm you design must be able to *unravel* the interleaving  $I$ , into simple repetitions. But now that we have a dictionary, the algorithm must test if the received message is an interleaving of the pair of messages in question.