

## Homework 8, CS420, Fall 2011

1. Let the *cost* of a match between two strings be the minimum number of insertions and deletions you have to perform to one of the strings to turn it into the other.
  - (a) Fill out a dynamic programming table similar to the one in Figure 15.8, except where the entries give the costs of the minimum-cost matches of prefixes, rather than lengths of the maximum common subsequences.
  - (b) Suppose you have a small pattern text  $P$  and a large text  $T$ , such as a genome. You want to find a substring of  $T$  that has a minimum-cost match to  $P$ . Describe an  $O(|P||T|)$  algorithm.
2. You have a ladder with  $n$  rungs and a bunch of identical jars. You want to test how high a rung a jar can be dropped from without breaking. (Assume there is a highest rung that you can drop a jar without breaking, and that if you drop one from the next higher rung, it always breaks.)
  - (a) Give the best big- $\Theta$  bound on the number of drops you need to perform to find this rung.
  - (b) Suppose you only have two jars. Give a big- $\Theta$  bound you can get on the number of drops to determine the rung. It should be  $o(n)$ .
  - (c) Suppose you only have three jars. Give a big- $\Theta$  bound on the number of jars to determine the rung.
3. Now let's consider the problem of finding the absolute minimum number of jars you need, not just an asymptotic bound.

After your first drop, if the jar doesn't break, you have removed the bottom part of the ladder from consideration. What remains is a smaller ladder and  $k$  jars. This is a recursive problem. On the other hand, if the jar does break, you have removed the top part of the ladder. What remains is a recursive problem on  $k - 1$  jars and a smaller ladder.

- (a) Using this insight, come up with a recurrence for  $T(k, j)$ , the absolute maximum height of a ladder you can handle with  $k$  jars and  $j$  drops.
  - (b) Fill in a dynamic-programming table to find the tallest ladder you can handle with eight drops and four jars.
4. A plane divides a space into two regions. Two planes can be used to divide a space into four regions. Three planes can be used to divide a space into eight regions. We might be tempted to think that four planes can divide it into sixteen regions, but this is not true; you they can only divide it into fifteen regions, because the last plane inserted must miss one of the eight regions formed by the first three.

Question: How many regions can five planes divide a space into?

Here are some hints. You could spend all morning trying to picture this complicated arrangement in your mind, or you could solve it quite easily using dynamic programming.

Come up with a recurrence  $R$  for the number of regions of a  $k$ -dimensional space can be formed by cutting it up with  $n$   $k - 1$ -dimensional spaces. For  $k = 1$ , this is just the question of how many intervals of a line  $n$  points can divide it into.  $R(1, 0) = 1$  and  $R(1, n) = R(1, n - 1) + 1$  because each new point added causes the number of regions to increase by 1.

Next, consider instances of the form  $R(2, n)$ . This is the question of how many regions of the plane  $n$  lines can divide it into. When you add the  $n^{\text{th}}$  line to the plane, what does its intersection with the previous  $n - 1$  lines look like? What does this have to do with the case of  $R(1, n - 1)$ ? What does  $R(1, n - 1)$  tell you the increase in the number of regions of the plane caused by adding the  $n^{\text{th}}$  line? How does this give you  $R(2, n)$  as a function of  $R(1, n - 1)$  and  $R(2, n - 1)$ ?

See if you can generalize the same trick to three dimensions. What does the intersection of the  $n^{\text{th}}$  plane with the previous  $n - 1$  planes look like?

5. In class, I talked about the following recursive function:

$$\begin{aligned} A(0, 0) &= 2; \\ A(0, n) &= A(0, n-1) + 1 \text{ for } n > 0; \\ A(m, 1) &= 2 \text{ for } m > 0; \\ A(m, n) &= A(m-1, A(m, n-1)) \text{ for } m > 0, n > 1; \end{aligned}$$

Make a table with columns 0 through 16, corresponding to values of  $n$ . Give it rows 1 through 4, corresponding to values of  $m$ . Fill in the first three rows, so that the entry at each location  $(m, n)$  gives the value of  $A(m, n)$ . If you fill it in in row order, you can use dynamic programming.

Notice that the rows can be viewed as “addition-base-2”, “multiplication-base-2”, “exponentiation-base-2”.

When you are done, fill in columns 1-4 of row 4. See if you can express the element in column 5 as a power of two, and see if you can express the element in column 6 as a power of a power of two.