Algorithm Design and Analysis

Homework #2

Due: 14:20, Thursday, October 18, 2012

Homework submission instructions:

- Submit your programming assignment (problem 1) to the Judgegirl System (http://katrina.csie.ntu.edu.tw/judgegirl/). **Also, use SVN to commit your report (named "report.pdf") in the same folder of your code**.
- Submit the answer of the writing problems and the programming report by one of the following ways:
  1. Submit an electronic copy to the CEIBA system before the deadline; or
  2. Submit a hard copy right before the class on the day of the deadline.

  If you submit by both ways, the TA will randomly select only one of them for grading.
- Please make sure that you write down your name and school ID in the header of your documents. Otherwise, the TA will not grade them.
- If you choose to submit the answers of the writing problems via CEIBA, please combine the answers of all writing problems into only one file in the pdf format, with the file name in the format of "hw2 [ID].pdf" (e.g. "hw2 b99902010.pdf"). Otherwise, the TA will not grade them.
- What should be included in the report:
  1. Explain how your program works in detail.
  2. Derive the time complexity of your program and briefly explain why. No need for formal proof.
- If you are stuck with any problem, you are welcome to ask the TAs via e-mails or in person during the office hour. The TA may discuss with you and give you some useful hints. :D

Problem 1 – Open!! (programming assignment)

Description

Each XiGui (系櫃) in NTU CSIE is protected with a N-digit numerical combination lock. In each step of the attempt to open the lock, any number of **consecutive** digits can be selected to be rotated 36 degrees counter-clockwise, after which all selected digits will increase by one (with the exception of the digit '9', which will become '0' after the rotation, obviously). Note that you are not allowed to rotate the digits clockwise, since we want to make this problem a little bit easier ;). As a NTU CSIE student, you decide to open the lock with a minimum number of steps.

Input

The first line contains the number N. ($1 \leq N \leq 500$). The two following lines contain two N-digit strings in each line, representing the number on the lock initially and the number which can be used to open the lock, respectively.

Output

Please output an integer, representing the minimum number of steps required to open the lock.

Sample Input

```
4
1238
4330
```

Sample Output

```
5
```

Hint

1. 1238 → 1239 → 1230 → 2330 → 3330 → 4330
2. $O(100N^3)$ will get at most 9 points, while $O(10N^3)$ will get all the 10 points of the judgegirl score. (Note that there do exist some polynomially faster algorithms; however, $O(N^3)$ is good enough here.)

3. You may need 3 dimensions for your DP array. Think about how to split the digits into subproblems. Make sure that this dividing strategy can give you a correct solution.

Grading

Your grade for this problem will be given based on the following two parts:
The judge result (10%)
The report & the code (10%)

Problem 2 - Televisions

40 years ago, a television was very expensive, and as a result only rich people could afford to have their own television. Suppose there was a street called ADA Street, and there were *n* families living on this street. One day, a rich guy decided to donate *k* televisions to the families on this street so that everyone could enjoy this new technology. The rich guy wanted to choose *k* out of the *n* families to have the donated televisions installed, and after the installation the total distance that each family needs to walk to the nearest family with a television should be minimized.

For example, assume that there were 6 families on this street at coordinates 0, 3, 7, 8, 10, and 12, and the rich guy donated 3 televisions. If the rich guy installed the televisions in the families at coordinates 0, 8, and 10, then the total distance can be calculated as |0-0| + |3-0| + |7-8| + |8-8| + |10-10| + |12-10| = 6.

You are asked to derive a dynamic programming algorithm to derive the minimum total distance.

1. (10%) Prove that this problem exhibits optimal substructure.
2. (10%) Define your cost function. What do its value and each of its parameters represent? Derive the recurrence of the cost function.
3. (5%) Fill out the dynamic programming table which is used to solve the problem with the above given input data (6 families at coordinates 0, 3, 7, 8, 10, 12, and k=3).

Note: There could be multiple ways of installing the televisions with which the distance is minimized. What you are asked to do is just to find out the minimum total distance.

Problem 3 – Breaking a String

A certain string-processing language allows a programmer to break a string into two pieces. Because this operation copies the string, it costs *n* time units to break a string of *n* characters into two pieces. Suppose a programmer wants to break a string into many pieces. The order in which the breaks occur can affect the total amount of time used. For example, suppose that the programmer wants to break a 20-character string after characters 2, 8, and 10 (numbering the characters in ascending order from the left-hand end, starting from 1). If she programs the breaks to occur in left-to-right order, then the first break costs 20 time units, the second break costs 18 time units (breaking the string from characters 3 to 20 at character 8), and the third break costs 12 time units, totaling 50 time units. If she programs the breaks to occur in right-to-left order, however, then the first break costs 20 time units, the second break

costs 10 time units, and the third break costs 8 time units, totaling 38 time units. In yet another order, she could break first at 8 (costing 20), then break the left piece at 2 (costing 8), and finally the right piece at 10 (costing 12), for a total cost of 40.

Design an algorithm that, given the numbers of characters after which to break, determines a least-cost way to sequence those breaks. More formally, given a string S with n characters and an array $\mathrm{L}[1..\mathrm{m}]$ containing the break points, compute the lowest cost for a sequence of breaks, along with a sequence of breaks that achieves this.

1. (10%) Show that this problem exhibits optimal substructure.
2. (10%) Define the cost function and derive the recurrence of the cost function. Explain what the value of the cost function represents and what each parameter represents.
3. (5%) Utilizing the recurrence, derive a dynamic programming algorithm to solve this problem.

Problem 4 - Diamond Pile

Two players play the "Diamond Pile" game with an initial pile of N diamonds. They take turn to take a few diamonds away (at least 1 diamond) from the pile. The player who goes first cannot take all n diamonds away in the first turn. If a player takes k diamonds in a turn, then his opponent can take at most $2k$ diamonds in the next turn. The player who takes the last diamond wins. In the special case of $N = 1$, the first player loses since he cannot make any move (based on the rule that he cannot take all diamonds away in the first turn).

For example, if $N = 7$, and the first player takes 1 diamond. Then the second can takes 1 or 2 diamonds. If he chooses to take 2 diamonds, then the first player can take 4 ($\leq 2 \times 2$) diamonds and win the game.

Assume both players are smart enough just like you. For a given N, we would like to determine which player could have a strategy that ensures himself can win the game.

1) (5%) If we use a Boolean array $c[1,2,\dots,N]$ to store the solutions of the sub-problems. For every $n = \{2,\dots,N\}$, if the first player would win when the game starts with n diamonds, then $c[n] = True$, otherwise $c[n] = False$. Obviously, $c[N]$ contains the solution to the problem. Can we use a dynamic programming algorithm to fill out the array $c[]$ to solve this problem? If yes, how? If not, why?
2) (5%) Suppose we use another 2D Boolean array $d[n][k], 1 \leq k \leq n \leq N$, as follows. Assume it is your turn now – there are n diamonds in the pile, and you

can only take at most k diamonds in this turn. If winning the game with the current condition can be guaranteed, then $d[n][k] = True$. Otherwise, $d[n][k] = False$. Obviously, $c[N] = d[N][N-1]$. How do you use a dynamic programming algorithm to fill out $d[n][k]$ and solve the problem? What is the complexity of your algorithm?

3) (5%) Prove that for every n, there is a threshold $f(n) \in \{1,2,\dots n\}$ such that $d[n][k] = True$ if and only if $k \geq f(n)$. (Note that you have to prove both "if" and "only if")
4) (10%) If we define another array $e[n] = f(n)$, then how do you use a dynamic programming algorithm to fill out $e[n]$ and solve the problem? What is the complexity of your algorithm?
5) (5%) Compute $c[1]$ to $c[15]$ and give an algorithm that run in $O(lg\, n)$ times to solve the problem (no need for formal proof).