Introduction to Algorithms October 23, 2003
Massachusetts Institute of Technology 6.046J/18.410J
Professors Shafi Goldwasser and Silvio Micali Handout 17

Problem Set 5

This problem set is due **in recitation** on **Friday, October 31.**

Reading: Chapters §15.1-15.5, §16.1-16.3.

There are **four** problems. Each problem is to be done on a separate sheet (or sheets) of three-hole punched paper. Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

Problem 2-1. Spam and Customer Service.

You run a meat-processing plant in Cambridgeport. Unfortunately, due to a mixup with your suppliers in Peoria, you have accidentally produced a batch of spoiled sausages. When your customer service desk opens at 8am, there are n customers waiting in the lobby to return the rotten sausages. Say that customer i has s_i sausages, and that all the s_i's are distinct. Your underpaid customer service operator requires $k \cdot s_i$ minutes to handle customer i's complaint, for some constant k. You cannot "partially process" a complaint—all of customer i's s_i sausage returns must be handled together—and only one customer can be handled at a time.

Because you want to maintain the golden reputation of your company, you have decided to pay one dollar to each customer i for every minute before i's complaint is completely handled. For example, if customer 3 is handled first and customer 5 is handled second, you pay ks_3 dollars to customer 3 and $k(s_3 + s_5)$ dollars to customer 5. Give an efficient algorithm to find the order in which the customers' complaints should be processed so that the total cost to the company is minimized. Prove that your algorithm is optimal and analyze its running time.

Problem 2-2. Splitsville.

Let x be a string $x_1, x_2, \ldots, x_n$, where $x_i \in \Sigma$ for a finite *alphabet* Σ.

A *split* of x consists of two strings $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ and $x_{j_1}, x_{j_2}, \ldots, x_{j_\ell}$, where

(i) $i_1 < i_2 < \cdots < i_k$;

(ii) $j_1 < j_2 < \cdots < j_\ell$;

(iii) $\{i_1, \ldots, i_k\} \cap \{j_1, \ldots, j_\ell\} = \emptyset$; and

(iv) $\{i_1, \ldots, i_k\} \cup \{j_1, \ldots, j_\ell\} = \{1, \ldots, n\}$.

For example, the following are splits of `banana`:

```
bnn     aaa
ban     ana
banaa   n
```

Give an algorithm to determine if two strings y and z form a *split* of x. You algorithm should take as input three strings, $x, y,$ and z, where x is a string of n symbols from the alphabet Σ. If strings y and z form a split of x, it should output "yes". Otherwise, it should output "no". Prove that your algorithm is correct and analyze its running time. (Your algorithm should run in time $O(n^c)$ for some constant c.)

Problem 2-3. A Maki a Day ...

To maintain your health, you have decided that you must eat a sushi meal a day for the remaining n days, $d_1, d_2, \ldots, d_n$, of the fall semester. You have exactly two choices for acquiring a sushi meal:

1. On day d_i, you can buy a sushi meal (for consumption on day d_i) from the supermarket at price p_i. These prices are announced in advance, i.e. on day d_1.
2. On day d_i, you can place an order with `bulk-sushi.com` to have a sushi meal delivered to you every evening for days $d_i, d_{i+1}, d_{i+2}, \ldots, d_{i+11}$. The cost of this contract is Q. Note that Q is fixed and does not vary from day to day.

Since sushi is highly perishable, you cannot "stockpile" sushi meals; you must eat each sushi meal on the day that you get it. Also, because you hate to waste food and because you can only stomach exactly one sushi meal per day, you may not get two sushi meals on the same day. For example, you cannot order from `bulk-sushi.com` three days after placing a previous `bulk-sushi.com` order. Finally, you are not allowed to order from `bulk-sushi.com` on day d_i if there are less than 12 days remaining (including day d_i), i.e. you can not have leftover sushi meals at the end of the n day period.

The problem is to find the cost of the optimal sushi-ordering schedule.

(a) Consider the following greedy algorithm for this problem. This algorithm takes as input Q, the price of 12 sushi meals from `bulk-sushi.com`, and the prices $p_1, p_2, \ldots p_n$, the price of a sushi meal from the supermarket on each of the n days. At day d_i, it checks if the total cost of buying a sushi meal from the supermarket for that day and each of the next 11 days (i.e. days d_i through day d_{i+11}) is less than Q. If so, it buys a sushi meal for day d_i at price p_i and goes on to the next day. Otherwise, it orders 12 sushi meals (for day d_i through day d_{i+11}) from `bulk-sushi.com` at price Q and goes on to day d_{i+12}. Pseudocode for this algorithm is provided below.

BUYSUSHIGREEDILY$(Q, p_1, \ldots, p_n)$
1 Let $cost = 0$.
2 Go from day d_1 to day d_n.
3 On day d_i:
4 If Q is greater than the sum of p_i through p_{i+11} or if there are fewer than 12 days (meals) left,
5 Then buy a sushi meal from the supermarket, let $cost \leftarrow cost + p_i$ and go to day d_{i+1}.
6 Else
7 Place an order with `bulk-sushi.com`, let $cost \leftarrow cost + Q$ and go to day d_{i+12}.
8 Output $cost$.

Prove that BUYSUSHIGREEDILY does not produce an optimal solution to the problem.

(b) Give a dynamic programming algorithm to find the cost of an optimal sushi-ordering schedule. Prove your algorithm is correct and analyze its running time.

(c) Modify your algorithm from part **(b)** to output an actual optimal sushi-ordering schedule.

Problem 2-4. Printing Neatly.

You are given a sequence of words $w_1, w_2, \ldots, w_n$, where word w_i has length ℓ_i. You want to lay out these words neatly in lines of total length L each, by choosing "nice" line breaks. Each word contains its own whitespace.

A neat line is one that contains close to L characters (without going over). More precisely, the *badness*—this is actually a technical term in LaTeX—of a line $w_i, w_{i+1}, \ldots, w_j$ is given by $badness := L - \sum_{k=i}^{j} \ell_k$. The badness *must* be nonnegative; all lines must contain at most L characters.

For example with $L = 25$, one could lay out the following definition from Ambrose Bierce's *The Devil's Dictionary* in these ways:

```
HATRED, n. A sentiment<->| badness 3
appropriate to the<----->| badness 7
occasion of another's<-->| badness 4
superiority.<----------->| badness 13
```

or

```
HATRED, n. A<---------->| badness 13
sentiment appropriate to>| badness 1
the occasion of<-------->| badness 10
another's superiority.<->| badness 3
```

(a) The badness of a paragraph is the sum of the badnesses of each of the lines of the paragraph, except the last. Give the most efficient algorithm you can to lay out the given words $w_1, \ldots, w_n$ in a single paragraph in a way that minimizes the badness of the paragraph. Prove your algorithm is correct and analyze its running time.

(b) Consider the following alternative definition: the badness of a paragraph is the sum of the *cubes* of the badnesses of all lines by the paragraph, except the last. Is your algorithm from part **(a)** still optimal? Give a proof or a counterexample.

(c) Consider a third definition: the badness of a paragraph is the *maximum* badness of any line in the paragraph other than the last. Give a dynamic programming algorithm to minimize the badness of a paragraph. Prove your algorithm is correct and analyze its running time.