CS 577: Introduction to Algorithms Fall 2019

Homework 4

Instructor: Dieter van Melkebeek TA: Andrew Morgan

This homework covers dynamic programming. In order to get feedback on problem 3, you need to submit it via Canvas by **11:59pm on 10/8**. Please refer to the homework guidelines on Canvas for detailed instructions.

Warm-up problems

1. Design an algorithm that takes as input two binary sequences a and b of lengths m and n respectively, and outputs the smallest length of a sequence c such that both a and b are subsequences of c. For example, if $a = 011$ and $b = 0101$, then the answer is 4, as witnessed by $c = 0101$. Your algorithm should run in time $O(m \cdot n)$.

2. Consider the multiplication defined in Table 1. For example, $ab = b$ and $ba = c$. Note that this multiplication is neither associative nor commutative.

   Table 1: Multiplication Table

	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

   You want to know, given a string of n symbols a, b, c, with $n \geq 1$, whether or not it is possible to parenthesize the string in such a way that the value of the resulting expression is a. For example, the string $bbbbac$ can be parenthesized as $((b(bb))(ba))c$, and that evaluates to a.

   Design an algorithm to solve this problem in time polynomial in n.

Feedback problem

3. A lab in the Chemistry department was running some experiments with some highly reactive liquid chemicals. Now that the experiments have ended (successfully), the lab assistant needs to place the n chemical substances into k bottles, which will then be shipped to a facility for safe destruction. The chemicals are numbered 1 through n, and need to be place in the k bottles in this order. We know that $k < n$ so we have to mix some of the chemicals together. The problem is that when some substances are mixed, chemical reactions between them produce energy. Specifically, when substances i and j are put in the same bottle, they produce e_{ij} units of energy. In order to reduce the risk of explosion during transportation, we want to minimize the total amount of energy produced.

   More formally, you are given a number k of bottles, a number n of substances, and nonnegative numbers e_{ij} for every pair of substances. We need to determine integers $0 \leq t_1 \leq t_2 \leq \ldots \leq$

$t_{k-1} \leq n$ (indicating the last substances put in bottles 1 through $k-1$) such that

$$\sum_{i=1}^{k} \sum_{t_{i-1} < \ell < m \leq t_i} e_{\ell m}$$

is minimized, where $t_0 = 0$ and $t_k = n$. The inner sum in this expression represents the energy produced in bottle i.

For example, consider the instance with $k = 2$, $n = 3$, and energies $e_{12} = 10$, $e_{13} = 5$, and $e_{32} = 42$. Observe that there are 4 possible ways to split the 3 substances into the 2 bottles:

- Split 1: $\{1, 2, 3\}\{\}$
- Split 2: $\{1\}\{2, 3\}$
- Split 3: $\{1, 2\}\{3\}$
- Split 3: $\{\}\{1, 2, 3\}$

The first and last ones are equivalent and yield a total energy of $42 + 5 + 10 = 57$. The second has energy 42, and the third has energy 10, so the answer is 10. Note that if we could change the order of the substances, the energy could be reduced even further (bottle 1 contains $\{1, 3\}$ and bottle 2 contains $\{2\}$, for a total energy of 5), but this is not allowed.

(a) Design an $O(n^2)$ algorithm that outputs the (minimum) energies for the subinstances consisting of substances i through j for *all* $1 \leq i \leq j \leq n$ and $k = 1$.

(b) Design an $O(kn^2)$ algorithm to solve the problem for a given instance with n substances and a given k.

Additional problems

4. Gerrymandering is the practice of carving up electoral districts in very careful ways so as to lead to outcomes that faver a particular political party. Recent court challenges to the practice have argued that through this calculated redistricting, large numbers of voters are being effectively (and intentionally) disenfranchised.

   Computers, as it turns out, have been implicated as some of the main "villains" in much of the news coverage on this topic: it is only thanks to powerful software that gerrymandering grew from an activity carried out by a bunch of people with maps, pencil, and paper into the industrial-strength process it is today. Why is gerrymandering a computational problem? Partly it is the database issues involved in tracking voter demographics down to the level of individual streets and houses; and partly it is the algorithmic issues involved in grouping voters into districts. Let's think a bit about what these latter issues look like.

   Suppose we have a set of precincts $P_1, P_2, \ldots, P_n$, each containing m registered voters. We're supposed to group these precincts into two districts, each consisting of $n/2$ of the precincts. Now, for each precinct, we have information on how many voters are registered to each of two political parties. (Suppose for simplicity that every voter is registered to one of these two.) We say that the set of precincts is susceptible to gerrymandering if it is possible to perform the division in such a way that the same party holds a majority in both districts.

   Design an algorithm to determine whether a given set of precincts is susceptible to gerrymandering. The running time of your algorithm should be polynomial in n and m.

Example Suppose we have $n = 4$ precincts, and the following information on registered voters. Party A has 55, 43, 60, and 47 voters in districts P_1, P_2, P_3, P_4 respectively, and party B has 45, 57, 40, and 53. This set of precincts is susceptible, since if we grouped precincts P_1 and P_4 into one district, and precincts P_2 and P_3 into the other, then party A would have a majority in both districts. (Presumably, the "we" who are doing the grouping here are members of party A.) This example is a quick illustration of the basic unfairness in gerrymandering: although party A holds only a slim majority in the overall population (205 to 195), it ends up with a majority in not one but both districts.

5. In modern origami (the Japanese art of paper folding), one typically starts with a square sheet of paper and attempts to transform this square into a three-dimensional animal, geometric object, or any other sculpture one can think of, using nothing but a sequenc of folds. In traditional 17th–18th century origami, however, the starting shape of the paper was less strictly prescribed.

   Hiro has stumbled across a book containing instructions for n origamc sculptures from this early period, each of which starts from rectangulap paper of size $a_i \times b_i$ where a_i and b_i are positive integers. He would like to make a diorama containing as many of these (not necessarily distinct) sculptures as possible, but he only has access to a single sheet of paper of size $A \times B$ (where A and B are also positive integers) and no scissors. By folding the paper carefully and tearing along the crease, Hiro is confident that he can make perfect horizontal and vertical cuts across an entire sheet of paper, splitting the sheet into two. However, the two new edges created by each of these cuts are *frayed* and look worse than the original edges of the $A \times B$ sheet, so Hiro would like to minimize the number of these edges as well.

   Design an algorithm that on input $A, B, a_1, \ldots, a_n, b_1, \ldots, b_n$, computes the maximum number of sculptures that can be made from the starting sheet of paper, as well as the minimum number of frayed edges that must be visible when Hiro creates this maximum number of sculptures. Your algorithm should run in time polynomial in A, B, and n.

Challenge problem

6. Design an algorithm for problem 3(b) that runs in time $O(kn \log n)$ when given access to the one-bottle energies computed in part (a).

Programming problem

7. SPOJ problem Square Brackets (problem code SQRBR).