Lab 8: Dynamic Programming: Part 4.
The "Let's Get it Right", Part 2 Lab

Due date: Friday, June 5.

Lab Assignment

Assignment Preparation

This is an individual lab. The goal of this lab to give you an opportunity to finish the design and development of the algorithm that you were asked to come up with for the second midterm exam.

You are welcome to work together with your classmates on the final solution, but you must submit your own version of it.

Free Group Multiplication Problem

Recall the specification from the midterm exam. Consider the following problem. We have a set of three elements $\{a, b, c\}$ an d a *multiplication* operation defined on them. The multiplication table (note: this operation is not commutative, nor is it associative) is shown below. The first argument is shown in the rows, the second — in the columns.

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

For example $ab = b$, but $ba = c$. We can multiply longer sequences of elements in different ways. E.g., abc can be computed as $(ab)c = bc = a$ or as $a(bc) = aa = b$.

Your goal is to construct an efficient algorithm which, given a string of symbols a, b, c, outputs **"yes"** if it is possible to parenthesize the string in such a way that the value of the resulting expression is a.

E.g., $aaabbc$ can be parenthesized $(((((aa)a)b)b)c) = a$. However, bab can only be parenthesized as either $(ba)b = cb = c$ or $b(ab) = bb = b$.

Note: each string s consisting of as, bs and cs can be parenthesized for the purpose of multiplication in a *number of different* ways. Each parenthization may lead to a different final result. **However**, there are only three possible results altogether: a, b and c.

This problem can also be solved using dynamic programming.

Task

You will develop and implement a dynamic programming algorithm for determining if a string s can be parenthesized to produce a sequence of multiplication that yields the value a.

To make grading straightforward, your solution shall satisfy the following conditions.

1. Create a Java class `Multiplier`. An instance of this class will have the multiplication matrix associated with it, and will use a special method (see below) to find if a given string can be parenthesized to yield a.

2. Implement `Multiplier.setMatrix(int[][] matrix, int n)` method which will acssociate with an instance of the `Multiplier` class an `n`×`n` multiplication matrix. The midterm problem has a multiplication matrix with `n`= 3, however, you shall design your solution to work with any matrix up to 26×26 in size.

   Without loss of generality, assume that the first row/column of the matrix represent character a, the second — b, the third — c, and so on until z.

3. Implement `Multiplier.parenthesize(String s)` method, which runs your dynamic programming algorithm to determine if the input string can be parenthesized using w.r.t. current multiplication matrix, to produce the value a as the result. The method shall return a boolean value.

4. Implement `TextChecker.recover(String s)` method, which (a) calls `parenthesize(s)` method first, and, if `true` is returned, finds and outputs the actual parenthization (sequence of multiplications) that lead to s becoming equal to a.

5. Implement a test environment for the `Multiplier` class, in which an instance of the class is created, gets filled with the multiplication matrix from the exam (see above) and then the `parenthesize()` and/or `recover()` methods are called on a bunch of strings obtained from an input file.

Sample Inputs

As a starting point, your test program can try determining whether an appropriate parenthization exists for the following strings

```
a
ab
ca
cab
abc
cba
aabb
ccab
aabbcc
ababab
bababa
abcabcabc
cabbaccabbac
acacacacaabcd
```

Deliverables

The lab only both electronic devliverables. Submit all your Java files using the following `handin` command:

Use `handin` to submit:

```
$ handin dekhtyar-grader lab08-349 <files>
```

In addition, submit a `README` file describing your implementation. Put any instructions on how to run your test program in it.

Good Luck!