CSC 284/484 Advanced Algorithms - applied homework 0
due: January 29th, 11:59pm EST

Grading:

- 284: 1 problem solved = A
- 484: 2 problems solved = A, 1 problem solved = B

This homework has different rules than the theoretical homework, most importantly, do not discuss any aspect of the applied problem with anybody else (except me), do not search for a solution online, do not use any written material when writing any part of the code (for example, no copy-paste, no open textbook when writing code, no reediting of an old source file from an old project, etc).

PROBLEM 1

We have a $n \times n$ chessboard filled with integers (each square contains one integer). We want select a subset of the squares so that 1) their total sum is maximized, and 2) there is no 2×2 square which has all of its squares selected.

For example, if $n = 2$ and the content of the chessboard is given by

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

then we will select the squares containing $2, 3, 4$.

For example, if $n = 3$ and the content of the chessboard is given by

$$\begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix}$$

then we will select all the squares except the middle one.

The problem can be solved using dynamic programming (the running time of the algorithm is going to be exponential in n, but it will be fast enough to run for $n \leq 23$). Solve it and implement the solution. Your program should read the input from standard input (that is, if you execute it on a unix system you can feed it a file using `./a.out < file`). Your program should output the results to the standard output. (`http://en.wikipedia.org/wiki/Standard_streams`).

INPUT FORMAT:

- the first line contains an integer $n \in \{1, \ldots, 23\}$,
- the next n lines each contain n integers separated by spaces (assume that each integer is between -1000 and $+1000$.

OUTPUT FORMAT:

- the first line contains the sum of squares in the optimal solution,
- the rest of the file contains the optimal solution in the following format: there are n lines, each line containing n numbers from $\{0, 1\}$ (zero means that the corresponding square is not selected; one means that the corresponding square is selected).

EXAMPLE INPUT:

```
3
2 2 2
2 2 2
2 2 2
```

EXAMPLE OUTPUT:

```
16
1 1 1
1 0 1
1 1 1
```

DEFINITIONS: Let $w \in \Sigma^*$ be word. The **length** of w is denoted by $|w|$, the i-th character of w is denoted $w[i]$ (we number the positions in w from 0 to $|w|-1$). A **subsequence** of w is a word $w[a_1]w[a_2]\dots w[a_k]$ where $0 \le a_1 < a_2 < \dots < a_k \le |w|-1$. The **reverse** of w is the word $w^R := w[|w|-1]w[|w|-2]\dots w[1]w[0]$. A word is called a **palindrome** if $w = w^R$. We let $w^1 := w$.

We are going to consider the following problem:

> Given a sequence of words $w_1, \dots, w_n \in \Sigma^*$ find $a_1, \dots, a_n \in \{1, R\}$ such that the word
>
> $$w_1^{a_1} w_2^{a_2} \dots w_n^{a_n}$$
>
> contains, as a subsequence, the longest possible palindrome.

For example if $n = 3$, $w_1 = abcd$, $w_2 = ocod$, and $w_3 = ab$, then $w_1^1 w_2 w_3^R = abcdocodba$ contains (as a subsequence) a palindrome of length 9 (abdocodba).

The problem can be solved using dynamic programming. Solve it and implement the solution. Your program must read the input from standard input (that is, if you execute it on a unix system you can feed it a file using `./a.out < file`). Your program must output the results to the standard output.

INPUT FORMAT:

- the first line contains an integer $n \in \{1, \dots, 1000\}$,
- the second line contains words $w_1, \dots, w_n \in \Sigma^*$ separated by spaces (we have $\Sigma = \{a, b, c, d, e, \dots, z\}$). Assume $|w_i| \le 5000$.

OUTPUT FORMAT:

- the first line contains the length of the longest palindrome,
- the second line contains the words $w_1^{a_1}, w_2^{a_2}, \dots, w_n^{a_n}$ separated by space (where $a_1, \dots, a_n \in \{1, R\}$ is the optimal solution),
- the third line contains the palindrome.

EXAMPLE INPUT:

```
3
abcd ocod ab
```

EXAMPLE OUTPUT:

```
9
abcd ocod ba
abdocodba
```

PROBLEM 3

A hopper is a virtual creature that visits Java programs and explores their arrays. Scientists observed a hopper and came to the following conclusions:

- a hopper only visits arrays with integer entries,
- a hopper always explores a sequence of array elements following the following rules:
  - a hopper cannot jump too far, that is, the next element is always at most D indices away (how far a hopper can jump depends on the length of its legs), and
  - a hopper doesn't like big changes in values—the next element differs from the current element by at most M, more precisely the absolute value of the difference is at most M (how big a change in values a hopper can handle depends on the length of its arms).
  - a hopper never visits the same element twice.
- a hopper will explore the array with the longest exploration sequence.

The scientists now need to prepare arrays for further study of hoppers and they need your help. They want a program that given an array and values D and M computes the length of the longest exploration sequence in the array.

INPUT FORMAT: The first line contains k, the number of problems. Then descriptions of the problems follow. Each problem is described on two lines. The first line of a problem description contains three numbers n, D, M, where n is the length of the array (as described above, D is the maximum length of a jump the hopper can make, and M is the maximum difference in values a hopper can handle). The next line contains n integers—the entries of the array. We have $1 \leq D \leq 7$, $1 \leq M \leq 10,000$, $1 \leq n \leq 10,000$ and the integers in the array are between $-1,000,000$ and $1,000,000$.

OUTPUT FORMAT: The output contains one line for each problem—the length of the longest exploration sequence.

EXAMPLE INPUT:

```
3 8 3 1 1 7 8 2 6 4 3 5 8 2 1 1 7 8 2 6 4 3 5 8 1 1 1 7 8 2 6 4 3
5
```

EXAMPLE OUTPUT:

```
8 3 2
```

PROBLEM 4

A recursive Pentago board of order 0 consists of a single square. For $k \geq 0$, a recursive Pentago board of order $k+1$ is obtained by taking four recursive Pentago boards of order k arranged in a 2×2 square and putting a rotation joint under the center of each recursive Pentagoboard of order k. Each recursive Pentago board of order k can now independently rotate by 0, 90, 180, or 270 degrees (of course, the rotation joins from the earlier levels of the recursive construction can still rotate). The board is completely filled with white and black stones, one in each square. We want to find out what is the largest rectangle of white pieces that can be achieved by rotating the joints. The objective is to maximize the area of the rectangle (or equivalently the number of stones in the rectangle).

INPUT FORMAT: The first line contains the number of test cases. Each test case is described on several lines. The first line contains k, the order of the recursive Pentago board. Then 2^k lines follow, each line containing 2^k numbers from $\{0, 1\}$. Zeros encode black pieces, ones encode white pieces. You can assume $k \leq 11$.

OUTPUT FORMAT: The output contains one line for each test case. The line contains a single integer, the area of the largest rectangle of white pieces that can be achieved by rotating the joints.

EXAMPLE INPUT:

```
2
2
1 0 0 0
0 0 1 1
1 0 0 1
1 0 0 1
3
0 1 0 0 1 1 0 0
0 1 1 1 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 1
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
```

EXAMPLE OUTPUT:

```
6
8
```