South African Computer Olympiad
Training Camp 2, 2006
Day 1

Overview

Author	**Marco Gallotta**	**IOI 2002**	**Graham Poulter**
Problem	**booster**	**xor**	**align**
Source	booster.c booster.cpp booster.pas	N/A	align.c align.cpp align.pas
Input file	booster.in	xor.in	align.in
Output file	booster.out	xor.out	align.out
Time limit	1 second	N/A	2 seconds
Number of tests	10	10	10
Points per test	10	10	10
Total points	**100**	**100**	**100**

The maximum total score is 300 points.

South African Computer Olympiad
Training Camp 2, 2006
Day 1

Rocket Booster

Author

Marco Gallotta

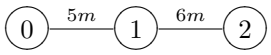
Introduction

Fred the manic storekeeper has upgraded his car with rocket boosters. However, he bought cheap boosters, and if he uses them for two consecutive sections of road they will overheat and explode (a section being a piece of road between two intersections).

Task

Fred wants you to determine the fastest time in which he can complete a race. Time is measured in "booster time units" — the time it takes to travel 1m with the boosters on. The boosters double the speed of the car, so it takes 2 booster time units to cover 1m without the boosters (assume that acceleration and deceleration are instantaneous). All roads can be travelled in either direction at the same speed; Fred only turns the boosters on or off at intersections.

Example



There are three intersections (marked 0, 1, 2 above) with roads from 0 to 1 (5m) and from 1 to 2 (6m). Fred needs to get from intersection 0 to intersection 2.

Fred could turn on his boosters at the beginning and get to the first intersection in 5 booster time units, but then he must turn off the boosters and he will then get to the end in a further 12 units. This would give a total time of 17 booster time units. He can, however, improve on this by only turning on the boosters after reaching the first intersection. He would get to the first intersection in 10 units and then get to the final intersection in 6 units, giving a total time of 16 booster time units.

Input (booster.in)

The first line will contain two values, N and R. There will be N intersections, numbered from 0 (the start) to $N-1$ (the destination). The next R lines will contain 3 values, s_i, e_i, d_i, representing the intersections at which a section of road starts (s_i) and ends (e_i) and its length (d_i) in metres.

Sample input

```
3 2
0 1 5
1 2 6
```

Output (booster.out)

The output must contain a single value, the minimum time required to get from intersection 0 to intersection $N-1$ in booster time units.

Sample output

```
16
```

Constraints

- $1 \leq N \leq 2000$
- $1 \leq R \leq 20000$
- $1 \leq d_i \leq 20000$

It is guaranteed that there will always be at least one route to the final intersection.

50% constraints

- $1 \leq N \leq 200$
- $1 \leq R \leq 1000$

Time limit

1 second.

Scoring

An optimal answer will score 100%, while a sub-optimal or invalid answer will score 0%.

XOR

Author

IOI 2002

Introduction

You are implementing an application for a mobile phone, which has a black-and-white screen. The x-coordinates of the screen start from the left and the y-coordinates from the top, as shown in the figures. For the application, you need various images, which are not all of the same size. Instead of storing the images, you want to create them using the phone's graphics library. You may assume that at the start of drawing an image, all pixels on the screen are white. The only graphics operation in the phone's library is XOR(L,R,T,B), which will reverse the pixel values in the rectangle with top-left coordinates (L,T) and bottom-right coordinates (R,B), where L stands for the left, T for the top, R for the right and B for the bottom coordinate.

Task

Given a set of black-and-white pictures, your task is to generate each picture from an initially white screen using as few XOR calls as you can. You are given the input files describing the images, and you are to submit files including the required XOR call parameters, not a program to create these files.

Example

As an example, consider the images in Figure 1. Applying XOR(2,4,2,6) to an all white image gives Figure 1(a). Applying XOR(3,6,4,7) to Figure 1(a) gives Figure 1(b), and applying XOR(1,3,3,5) to Figure 1(b) finally gives Figure 1(c).

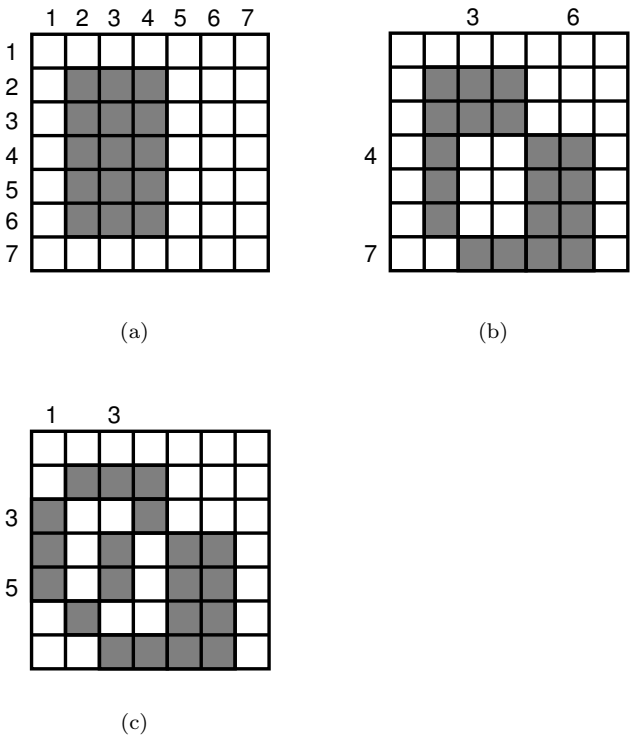


Figure 1: Some examples

Sample input

```
7
0 0 0 0 0 0 0
0 1 1 1 0 0 0
1 0 0 1 0 0 0
1 0 1 0 1 1 0
1 0 1 0 1 1 0
0 1 0 0 1 1 0
0 0 1 1 1 1 0
```

Input (xor.in)

You are given 10 problem instances in the text files named xor1.in to xor10.in. Each input file is organized as follows. The first line of an input file contains one integer N, meaning that there are N rows and N columns in the image. The remaining lines represent the rows of the image from top to bottom. Each line contains N integers: the pixel values in the row from left to right. Each of these integers is either a 0 or a 1, where 0 represents a white pixel and 1 represents a black pixel.

Output (xor.out)

You are to submit 10 output files corresponding to the given input files.

The first line of each output file must contain a single integer K, the number of XOR calls required to generate the image given in the input. The following K lines represent these calls from the first call to the last call to be executed. Each of these K lines contains four integers: the XOR call parameters L, R, T, B in that order.

Sample output

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

Constraints

- $5 \leq N \leq 2000$

Scoring

If

- the XOR calls specified in your output file do not generate the required image, or
- the number of XOR calls specified in your output file is not K, or
- in your output file, $K > 40000$, or
- your output file contains a XOR call with $L > R$ or $T > B$, or
- your output file contains a XOR call which does not have positive coordinates, or
- your output file contains a XOR call with a coordinate value exceeding N

then your score is 0%. Suppose that the smallest number of XOR calls used by any contestant is L. If $K > 2 \times L$ then you will score 0%. Otherwise, your score is:

$$100\frac{2L - K}{L}\%$$

South African Computer Olympiad
Training Camp 2, 2006
Day 1

Sequence Alignment

Author

Graham Poulter

Introduction

Alignment of pairs of DNA and protein sequences was one of the first bioinformatics algorithms and is to this day the most heavily used. One variant, the Basic Local Alignment Search Tool (BLAST), has been called "the Google of molecular biology" and the NIH BLAST page is used tens of thousands of times each day.

Alignment of sequences is complex because there may be gaps in either or both of the sequences. For example, given the sequences `AATGC` and `AGGC`, the following are three valid alignments, where `_`'s indicate gaps:

```
AATGC      A_ATGC      AATGC____
_AGGC      AG__GC      _____AGGC
```

However, it is illegal for a gap in one sequence to be aligned with a gap in the other.

One wants the alignment that is "closest" in some sense, in order to determine how closely related the sequences are. Each alignment is given a score S, which is determined by two terms:

1. Wherever a letter from one sequence is aligned with a letter from the other, a "match" score is taken from a lookup table, depending on the letters involved (see the example). The lookup table is symmetric i.e., the match score for (A, B) is the same as for (B, A). These letter match scores are all added to S.

2. For each gap of length L (i.e., L underscores in a row), the score S is reduced by a "gap penalty" $G + QL$, where G is the "gap opening penalty" and Q is the "gap extension penalty" (parameters given in the input). The exception is gaps at the start or end of the sequence, which are ignored.

Task

You are to write a program that takes two sequences, an alphabet, a match lookup table, and the gap open and gap extension penalties, and finds an alignment of the sequences which maximises S.

Example

Again, consider the strings `AATGC` and `AGGC`, and let $G = 2$ and $Q = 1$. The match lookup table is shown below:

	A	C	G	T
A	5	0	0	0
C	0	5	0	0
G	0	0	5	−1
T	0	0	−1	5

Now consider the three alignments shown above. In the first, the match score is $5 - 1 + 5 + 5 = 14$ and the gap is not penalised (because it is at the start), so $S = 14$. In the second, the match score is $5+5+5 = 15$ but there are gap penalties of $2+1\cdot 1 = 3$ and $2+1\cdot 2 = 4$, so $S = 8$. In the third alignment there are no matches and no proper gaps, so $S = 0$.

Input (align.in)

The first line contains the string for sequence A, and the second line contains the string for sequence B. The third line contains the gap open penalty G, and the fourth line contains the gap extension penalty Q. The fifth line contains a string of N unique uppercase letters (in an arbitrary order) which form the alphabet for genetic sequences. A and B will consist entirely of letters from this alphabet.

Lines 6 to $N + 5$ each contain N numbers representing the match-scores for each letter of the alphabet against each other letter. The table will be symmetrical about its diagonal from top left to bottom right. The ith letter in the alphabet string corresponds to row and column i of the table.

Sample input

```
AATGC
AGGC
2
1
ACGT
5 0 0 0
0 5 0 0
0 0 5 -1
0 0 -1 5
```

Output (align.out)

The output is two lines, consisting of an alignment of the input sequences with the highest possible alignment score. Use underscores (`_`) to represent gaps in a sequence.

Sample output

```
AATGC
_AGGC
```

South African Computer Olympiad
Training Camp 2, 2006
Day 1

Constraints

The lengths of A and B be between 1 and 1000 letters. The gap open and extension penalties G and Q will be integers between 0 and 1000. The elements of M will be integers between -1000 and $+1000$.

50% constraints

Half of the test cases will have $G = 0$.

Time limit

2 seconds.

Scoring

For each test case, 10 points are given for any alignment achieving the highest possible alignment score, and 0 points otherwise. Alignments which align a gap in A with a gap in B (one underscore on top of another) will score 0.