Problem A. Brackets

Input file:	`brackets.in`
Output file:	`brackets.out`
Time limit:	1 second
Memory limit:	64 megabytes

Given a string consisting of brackets of two types find its longest substring that is a regular brackets sequence.

Input

The input file contains a string containing only characters '`(`', '`)`', '`[`', and '`]`'. The length of the string does not exceed 100 000.

Output

Output the longest substring of the given string that is a regular brackets sequence.

Example

brackets.in	brackets.out
`([(][()]]()`	`[()]`
`([)]`	

Problem B. Dividing a Chocolate

Input file:	`choco.in`
Output file:	`choco.out`
Time limit:	1 second
Memory limit:	64 megabytes

The boy and Karlsson are dividing a chocolate that the boy's parents have presented him for his birthday. A chocolate is a rectangle that consists of $m \times n$ chocolate squares, separated from each other by cutlines.

Of course, Karlsson does the dividing. First Karlsson divides the chocolate into two rectangular pieces, breaking it along some cutline. Since Karlsson wants to divide chocolate fairly, he would not be satisfied if the two pieces have different sizes. In this case he cuts away the piece equal to the smaller piece out of the larger piece, and eats it. If the pieces are still not equal, he does so again, and so on. After the pieces are finally equal, Karlsson eats one of the pieces, and the boy eats another.

Karlsson wants to make an initial cut in such a way that he would eat as much chocolate as possible. However, the boy knows that Karlsson likes chocolate very much, and he watches for the process closely. Karlsson must be very careful not to offend the boy. So he must not successively cut away and eat chocolate from the same piece — this would seem too suspicious to the boy.

Help Karlsson to find out how much chocolate he can eat.

Input

Input file contains m and n ($1 \le m, n \le 10^9$).

Output

Output one integer number — how many chocolate squares Karlsson can eat. If Karlsson cannot divide chocolate using his method, output 0 instead.

Example

choco.in	choco.out
6 5	24

In the example Karlsson must initially break a chocolate into pieces of sizes 3×6 and 2×6. After that he can eat all chocolate but the boy's piece that would finally be 1×6.

Karlsson cannot, for example, cut a chocolate initially into pieces of sizes 5×5 and 1×5 — after doing so, he would have to successively cut a piece away from the first one several times, that would offend the boy.

Problem C. Thermal Death of the Universe

Input file:	`death.in`
Output file:	`death.out`
Time limit:	1 second
Memory limit:	64 megabytes

Johnie has recently learned about the thermal death concept. Given that the Global Entropy always increases, it will end in the thermal death of the Universe. The idea has impressed him extremely. Johnie does not want the universe to die this way.

So he decided to emulate the process to check how soon the thermal death can occur. He has created the mathematical model of the process in the following way. The universe is represented as an array of n integer numbers. The life of the universe is represented as the sequence of the following entropy operations: take elements from i-th to j-th and replace them with their average value. Since their average is not necessarily integer, it is rounded.

To keep balance, rounding is performed either up, or down depending on the current sum of all elements of the array. If their sum is less or equal to the sum of the original array, the rounding is performed up, in the other case — down.

Given the initial array and the sequence of the entropy operations, find the contents of the resulting array.

Input

The first line of the input file contains n and m — the size of the array and the number of operations respectively ($1 \le m, n \le 30\,000$). The second line contains n integer numbers — the initial contents of the array, they do not exceed 10^9 by their absolute value. The following m lines contain two integer numbers each and describe entropy operations.

Output

Output n integer numbers — the contents of the array after all operations.

Example

death.in	death.out

6 4
1 2 3 4 5 6
1 2
2 5
5 6
4 6
	2 3 3 5 5 5

Problem D. Equations System

Input file:	`eq.in`
Output file:	`eq.out`
Time limit:	1 second
Memory limit:	64 megabytes

In the eighth grade of the high school students learn how to solve systems of linear equations. This time you have to solve the system of two linear equations with two unknowns in polynomials over $\mathbb{F}_2$. That is, both coefficients and variables are polynomials with coefficients from $\mathbb{F}_2 = \{0, 1\}$, operations in this field are carried out modulo 2.

Given polynomials $a_1(t)$, $b_1(t)$, $c_1(t)$, $a_2(t)$, $b_2(t)$, and $c_2(t)$, find polynomials $x(t)$ and $y(t)$, such that:

$$\begin{cases} a_1(t)x(t) + b_1(t)y(t) = c_1(t) \\ a_2(t)x(t) + b_2(t)y(t) = c_2(t) \end{cases}$$

Input

Six lines of the input file describe polynomials $a_1(t)$, $b_1(t)$, $c_1(t)$, $a_2(t)$, $b_2(t)$, and $c_2(t)$ in this order.

Each polynomial is specified with its degree k followed by $k + 1$ coefficients, starting from the leading one. Each coefficient is either 0, or 1. Constant zero polynomial has the degree of -1 in this problem. Degrees of the polynomials do not exceed 100.

Output

Output two polynomials, one on a line — $x(t)$ and $y(t)$. Use the same format as in input. If there is no solution to the system of the equations, output "`No solution`" instead. If there are several solutions, output any one with the degree not exceeding 1000 (it can be proved that if there is some solution, there is such one).

Example

eq.in	eq.out

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

`1 1 0`
`1 1 1`
`0 1`
`1 1 0`
`1 1 1`
`0 1`
	`0 1`
	`0 1`

`1 1 0`
`1 1 1`
`0 1`
`1 1 0`
`1 1 1`
`-1`
	`No solution`

Problem E. Fool's Game

Input file:	`fool.in`
Output file:	`fool.out`
Time limit:	1 second
Memory limit:	64 megabytes

A card game, often called "Fool's Game", is quite popular in Russia. We will describe one round of a game for two players.

A standard deck of 36 cards is used. Sometimes the game is carried out using several decks. One suit is declared to be trump. Before the round each player has several cards, one of the player is *starting*, the other one is *covering*. For simplicity we will assume, that both players have the same number of cards before the round.

The starting player starts by laying one or several cards of the same rank down on the table. The covering player must in turn *cover* all the cards with some of her cards. A card can cover another if one of the following is true: it has the same suit and higher rank (ranks are ordered as usually: 6, 7, 8, 9, 10, jack, queen, king, ace), or it is a trump and the card to cover is not a trump (trump can only be covered with a higher trump). After the cards on the table are all covered, the starting player can *toss* some more cards to be covered. The rank of each card tossed must be among the ranks of the cards already on the table at the moment. Now the newly added cards must be covered by the covering player, after that the starting player can toss more cards, and so on.

The round ends when either the covering player cannot cover the cards on the table, or when the starting player does not want to toss more cards. In the first case the covering player loses the round and takes all the cards from the table, adding them to his cards. In the second case he wins the round and the cards on the table are removed from the game.

Given the cards of both players, and the initial cards layed down by the starting player, find out whether the covering player can always win the round, or the starting player can always force him to lose.

Input

The first line of the input file contains n — the number of cards that each of the players has in the beginning of the round ($1 \le n \le 36$), and the trump suit (suit is specified using one letter: '`S`' for spades, '`C`' for clubs, '`D`' for diamonds, '`H`' for hearts).

The second line contains n card descriptions — the cards of the starting player. Each card is specified by its rank ('`6`'…'`9`', '`T`' for 10, '`J`' for jack, '`Q`' for queen, '`K`' for king, '`A`' for ace) and its suit. The cards that are initially layed on the table are marked with asterisk '`*`' after the description. They always have the same rank. The third line contains n card descriptions — the cards of the covering player.

You may assume that the game is carried on using several decks, so cards may be duplicated.

Output

Output "`COVER`" if the covering player can always win the round, or "`TAKE`" if the starting player can always force him to lose.

Example

fool.in	fool.out

`4 H`
`6S* 7C 8D KH`
`7S KH KD AH`
	`COVER`

`4 S`
`6S* 7C 8D KH`
`7S KD KD KS`
	`TAKE`

Problem F. Lottery

Input file:	`lottery.in`
Output file:	`lottery.out`
Time limit:	1 second
Memory limit:	64 megabytes

Many lotteries now publish ads proclaiming that they have a super-prize in each ticket. The idea is the following. The ticket has n windows, each of which contains some letter. Initially all windows are covered by the opaque substance. A person who buys the ticket removes the covering substance from m windows and if the letters revealed can be combined to form the winning m-letter word, the ticket owner wins the super-prize. The advertised features are the following: each ticket can win, and the only letters that are hidden in the windows are the letters from the winning word.

Assuming that the person uncovers m randomly chosen windows, you have to find the probability of winning a super-prize. Of course, the winning probability depends on the exact multi-set of the letters in the windows. For example, if the winning word is "`WOW`" and there are four windows, the letters used may be:

- two '`W`'-s and two '`O`'-s, in this case the chance of winning is 1/2;
- three '`W`'-s and one '`O`', in this case the chance of winning is 3/4.

Thus you have to find the maximal and the minimal possible probability of winning.

Input

The first line of the input file contains n ($1 \le n \le 60$). The second line contains the winning word. It contains only capital letters of the English alphabet, its length does not exceed n.

Output

On the first line of the output file print the maximal possible probability of winning, as an irreducible fraction. On the second line print the minimal possible probability of winning in the same form.

Example

lottery.in	lottery.out
4 WOW	3/4 1/2
7 VICTORY	1/1 1/1
60 ABRACADABRA	635250/29290609 1/6854002506

Problem G. Two Pipelines

Input file:	`pipe.in`
Output file:	`pipe.out`
Time limit:	1 second
Memory limit:	64 megabytes

There are two oil pipelines going through Flatland. Each pipeline can be considered as a line on the plane.

Recently the president of Flatland has decided to bring oil to all n cities of Flatland. Therefore he would like to build pipes from each city to one of the pipelines. The minister of industry and education has provided him with data about demand for oil in each city. Transporting a barrel of oil for one kilometer costs 1 flatlar.

The president ordered to connect each city to the closest pipleine, but it turned out that there is one problem. One pipeline belongs to a strong Hrundi Empire, while the other belongs to Bordland Republic. Since president wants to keep good relationships with both countries, he must not give preference to one of the pipelines. That is, if there would be x cities connected to one pipeline and y cities connected to the other, $|x - y|$ must not exceed c.

Help president to decide which city should be connected to which pipeline, so that he keeps his international reputation clean, and spends as few money as possible.

Input

The first line of the input file contains n and c ($1 \le c \le n \le 200$). The second line contains the description of the first pipeline — coordinates of two different points that are lying on it: x_1, y_1, x_2, y_2. The third line describes the second pipeline in the same format.

The following n lines describe cities. Each city is described with three integer numbers: its coordinates and the demand for the oil in thousands of barrels per day (it does not exceed 1000).

All coordinates are integer, given in kilometers and do not exceed 10^4 by absolute values. No two cities coincide. Pipelines do not coincide either.

Output

For each city output 1 if it must be connected to the first pipeline, or 2 if it must be connected to the second pipeline.

Example

pipe.in	pipe.out

3 1
0 0 10 0
0 5 7 5
-2 1 1
1 1 1
4 2 10
	1 2 1

Problem H. Manhattan Police

Input file:	`police.in`
Output file:	`police.out`
Time limit:	1 second
Memory limit:	64 megabytes

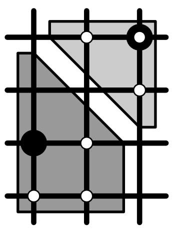
In year 3141 Manhattan turned to the region of criminals and gangsters, so it was very dangerous even to drive there by car.

Remember, Manhattan has m avenues running from east to west and n streets running from north to south. Each avenue-street pair meets at some crossing. Driving on a high speed along some avenue or some street is rather safe, but turning at any crossing is extremely dangerous because you have to slow down and immediately the band of gangsters attacks you.

The police department of Manhattan has decided to repair the situation a bit. It was decided to put policemen to some crossings to allow drivers to turn at it safely. Of course, policemen are equipped with modern arms, so they usually can defend themselves and people around from gangsters.

However, the funding of the police department is limited, and the new policemen must be paid a salary. Additionally, the chief of the department would like to spend as few money as possible for such a horrible district as Manhattan. So now the wants to find out what minimal amount of money is needed to allow people of Manhattan to get from any location to any other location turning only on crossings guarded by policemen.

The additional problem is that some regions of Manhattan are more dangerous, and some are less. A policeman guarding a crossing in more dangerous region must be paid a higher salary. Every region has its center at some crossing. Every other crossing belongs to the region the center of which is closest to it. The distance is measured using so called *Manhattan distance* (no surprise it is called so) — the distance between two crossings is the number of street segements one needs to pass to get from one crossing to another. If there are several region centers on the same and smallest distance from some crossing, then it is the area of the fight between the bands controlling the regions, so no policeman would ever agree to guard it.



Input

The first line of the input file contains m, n and r — the number of avenues, streets and regions in Manhattan respectively ($2 \le m, n \le 500$, $1 \le r \le 1000$). The following r lines contain three integer numbers each — the avenue and the street, at the crossing of which the center of the corresponding region is located, and the salary of the policeman that would guard the crossing in this region. Avenues and streets are numbered consequently, starting from 1. Required salaries are positive and do not exceed 1000. No region centers coincide.

Output

On the first line of the output file print two numbers: k — the number of crossings to guard and s — the smallest possible total salary of the guarding policemen.

After that print k pairs of numbers — the avenue and the street, at the crossing of which the correspoding policeman must be placed.

If it is impossible to establish the police guards to let people travel from any location to any other in Manhattan, output -1 instead of k, and do not output anything else.

Example

police.in	police.out

4 3 2
3 1 200
1 3 150
	6 1050
	1 2
	1 3
	2 3
	3 2
	4 1
	4 2

3 3 2
1 2 200
3 2 150
	-1

Problem I. Regular Words

Input file:	`regular.in`
Output file:	`regular.out`
Time limit:	1 second
Memory limit:	64 megabytes

Consider words of length $3n$ over alphabet $\{A, B, C\}$. Denote the number of occurences of A in a word α as $A(\alpha)$, analogously let the number of occurences of B be denoted as $B(\alpha)$, and the number of occurenced of C as $C(\alpha)$.

Let us call the word ω *regular* if the following conditions are satisfied:

- $A(\omega) = B(\omega) = C(\omega)$;
- if γ is a prefix of ω, then $A(\gamma) \geq B(\gamma) \geq C(\gamma)$.

For example, if $n = 2$ there are 5 regular words: $AABBCC$, $AABCBC$, $ABABCC$, $ABACBC$ and $ABCABC$.

Regular words in some sense generalize regular brackets sequences (if we consider two-letter alphabet and put similar conditions on regular words, they represent regular brackets sequences).

Given n, find the number of regular words.

Input

The input file contains n ($0 \leq n \leq 60$).

Output

Output the number of regular words of length $3n$.

Example

regular.in	regular.out
2	5
3	42

Problem J. Greate Siberian Wall

Input file:	`wall.in`
Output file:	`wall.out`
Time limit:	1 second
Memory limit:	64 megabytes

Inspired by the example of the Greate Chinese Wall, the inhabitants of one Siberia region have decided to build their own defensive wall.

The wall must have the form of convex polygon and surround all tribe villages. The area surrounded by the wall must be as small as possible.

The wizard of the tribe has created the plan of the wall, but it has turned out that the leader of the tribe can count only to three. Therefore he cannot understand the plan, if the number of vertices of the polygon is greater than three. Now the wizard must promptly create the new plan, in which the wall is the triangle. Help him!

Input

The input file contains n — the number of tribe villages ($3 \le n \le 80$), followed by n pairs of integer numbers — coordinates of the villages. All coordinates do not exceed 10^4 by their absolute value. It is guaranteed that there are three villages not lying on the same line.

Output

Output three pairs of real numbers — the coordinates of the vertices of the wall triangle. Its must surround all villages, and the surrounded area must be as small as possible.

Example

wall.in	wall.out

4
0 0
0 1
1 0
1 1
	0.0 0.0
	2.0 0.0
	0.0 2.0

Problem K. Words Game

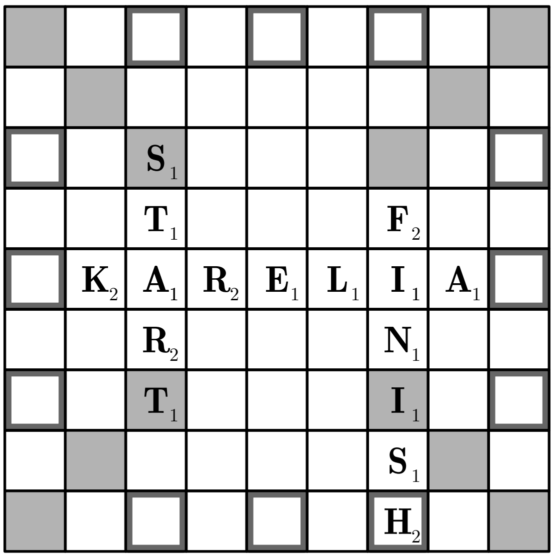
Input file:	`words.in`
Output file:	`words.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

A words game called *Scrabble* was once quite popular. We will consider a simplified version of the game rules.

The game is played on a board that consists of $m \times n$ cells. Each cell may be occupied with some letter, letters on the board form words that can be read from left to right, or from top to bottom.

Each letter of the alphabet has some value. Some cells of the board are so called *bonus cells*. There are two types of bonuses: a *letter bonus*, and a *word bonus*, each bonus has some coefficient. The value of the word is calculated in the following way. First the values of all letters of the word that are placed on a letter bonus cell are multiplied by its coefficient. The value of the word is the sum of the values of its letters, multiplied by the coefficients of all word bonus cells.

The picture below shows some position in the game. Shaded cells are letter bonus cells with coefficient 2. Cells with a bold border are word bonus cells with coefficient 2. Numbers near the letters are their values. The word "`KARELIA`" has the value of 9, the word "`START`" — 8, the word "`FINISH`" — 18.

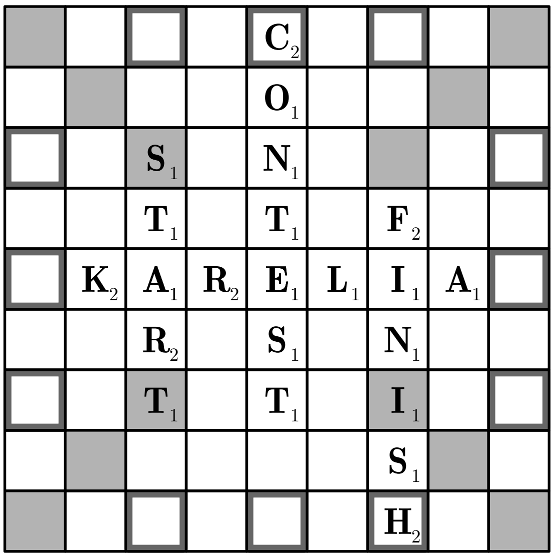


Initially some word is laid out on the board, and after that players in turn make their moves. Before the turn the player has t letters, that he can put to the board. He must create a new word on the board, and the following conditions must be satisfied:

- at least one letter must be put to the board;
- all letters that are put to the board must belong to the new word;
- at least one letter of the new word must be a letter that is already on the board before the turn;
- it is not allowed to put a *corrupt* letter to the board: if some letter is adjacent to the letter put to the board, it must be the letter from the new word (two letter are adjacent if their cells have a common side);
- it is not allowed to make a *chain* — that is, the new word must be a maximal continuous sequence of letters, there must be no letter immediately before or immediately after the word.

We make no other restrictions that are usually used: we do not require the new word to be unique (not to exist on the board already), we allow word extention, that is — the new word may contain some word already on the board as a substring (as long as the conditions above are met).

The picture below shows the example of the game board above after adding the word "`CONTEST`", its value is 16.



Given the description of the game board, the position on the board, and the letters that the player owns before the turn, find out what is the maximal value of the word he can create on the board. You are provided a small dictionary, so you know what words are allowed to be created.

Input

The first line of the input file contains m and n ($5 \le m, n \le 15$). The next line contains b — the number of bonus cell types ($0 \le b \le 18$), b blocks follow. Each block is started with the line that describes the bonus: it contains bonus type ('`L`' for letter bonus and '`W`' for word bonus), and the coefficient, separated by a space. The following line contains the number of cells that have such bonus. One or more lines with cells coordinates follow, each cell is specified by its row and column. Each cell is described as bonus cell at most once. Rows are numbered from top to bottom, columns — from left to right. Coefficient is an integer number from 2 to 10. Product of all word multipliers in any row and any column does not exceed 1000.

The next line contains t — the number of different letter values ($1 \le t \le 26$). Only capital letters of the English alphabet are used. The following t lines contain values descriptions: one or several letters are followed by a space and a number — the value of these letters (it is a positive integer number that does not exceed 100). Each letter of the English alphabet is guaranteed to have exactly one value.

The following m lines contain n characters each and describe the board before the turn: empty cell is denoted by '`.`' (dot), occupied cell is denoted by the letter it contains. It is guaranteed that there is at least one letter on the board. You must not assume that the position can indeed be obtained in the progress of the real game.

The next line contains k — the number of letters that the player owns before his turn ($1 \le k \le 6$). The following line contains these k letters.

The next line contains w — the number of words in a dictionary ($1 \le w \le 10\,000$). The following w lines contain one word each — the dictionary itself. No word is longer than 15 characters. The dictionary contains no duplicates.

Output

Output the maximal value of the word that can be created. If it is impossible to create a new word according to the rules above, output 0.

Example

words.in	words.out
9 9 2 L 2 12 1 1 1 9 2 2 2 8 3 3 3 7 7 3 7 7 8 2 8 8 9 1 9 9 W 2 12 1 3 1 5 1 7 3 1 3 9 5 1 5 9 7 1 7 9 9 3 9 5 9 7 3 ADEILMNOPST 1 BCFGHJKRU 2 QVWXYZ 5 ....S...... ..T...F.. .KARELIA. ..R...N.. ..T...I.. ......S.. ......H.. 6 CNOSTT 10 COMPUTER CONTEST COST GAME JAVA KEYBOARD OS PROGRAM PASCAL WINNER	16