Problem A. Crazy Bishops

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

When studying chess, many people are first taught how to checkmate the lone king. In this problem you have to check whether it is possible to checkmate the lone black king with two bishops and a king, starting from a given position.

Recall that a king can move one cell in each of the eight directions, a bishop can move any number of cells in diagonal direction. A king cannot move to the cell that is attacked by some piece. A bishop can move to any cell, but he must not move to the cell that is attacked by the black king, if it is not protected, in other case it can be taken by the black king and the game ends in draw.

Given the position on the board and the side that has the turn, you have to detect whether white can win, or black can force the game to end in draw.

Remember, that if the black king has no possible moves, but is not checkmated (i.e. it is stalemated), the game is considered to end in draw.

Input

The first line of the input file contains cells where white king and two bishops are located, respectively. The second line contains the cell where the black king is located. Standard chess notation is used, letters 'a' to 'h' stand for verticals and digits '1' to '8' for horizontals.

The last line contains "`white`" if it's white's turn and "`black`" if it's black's turn.

The position is guaranteed to be valid, i.e. no king can be taken on the current turn.

Output

Output "`white wins`" if white can win, and "`draw`" if it cannot.

Example

bishops.in	bishops.out
c2 a4 c1 a2 white	white wins
f6 f7 h8 h7 white	draw
c3 g8 h7 a1 black	draw

Problem B. Divide and Conquer

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

It is well known that n points can divide the line to at most $n+1$ parts, and n lines can divide the plane to at most $\frac{(n+1)n}{2}+1$ parts. This facts have always impressed Buddy. But recently he has learned about higher-dimensional spaces and this impressed him much more!

Now Buddy wonders how many parts can n hyperplanes divide m-dimensional space to. His elder brother Dubby told him that for each m this number can be represented as a polynomial for n with rational coefficients. Help him to find this polynomial.

Input

Input file contains m ($1 \le m \le 18$).

Output

On the first line of the output file print the degree of the polynomial. On the second line output coefficients of the polynomial, sepearted by spaces. Coefficicients must be listed starting from the leading one. All coefficients must be printed as irreducible fractions, if some coefficient is negative, minus sign must preceed its nominator, not denomimator.

Example

divide.in	divide.out
2	2 1/2 1/2 1/1
3	3 1/6 0/1 5/6 1/1

Problem C. Martians' DNA Analysis

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

After recent disciveries of remains of ancient Martian civilization, many biology specialists lay their hopes to DNA analysis of Martians. The task is complicated by the fact that unlike human DNA that only contains four nucleotides, Martians must have had more advanced organisms, so their DNA contains upto 36 nucleotides.

One step of analysis is detecting blocks that are repsonsible for some particular organism properties. But how can one identify complete blocks? One way to do so, is to search for so called *maximal repetitions*. Let us consider DNA as a string ω, where each character identifies some nucleotide. A non-empty substring of the string is called a maximal repetition if it is both *left-branching* and *right-branching*. The string α is called left-branching there are two different characters c_1 and c_2, such that both $c_1\alpha$ and $c_2\alpha$ are substrings of $\$\omega$ (here '$' is a character that does not occur in ω, it is introduced so that string that is prefix of ω and occurs somwhere else in it were left-branching). Similarly, α is right branching if both αc_1 and αc_2 are substrings of $\omega\$$ for some different c_1 and c_2. Long maximal repetitions are good candidates for complete blocks.

Given a string that represents a Martian's DNA, find the number of different (as strings) maximal repetitions in it, the length of the longest maximal repetition in it, and the number of its longest maximal repetitions.

Input

Input file contains non-empty string that contains only capital latin letters and digits and represents a Martian's DNA. The length of the string does not exceed 6000.

Output

Output the number of maximal repetitions in the DNA, the length of its longest maximal repetition, and the number of maximal repetitions that have this length. If there are no maximal repetitions, output three zeroes.

Example

dna.in	dna.out
ABCBABCA	3 3 1

In the example strings "`A`", "`AB`", "`ABC`", and "`B`" are left-branching, strings "`A`", "`ABC`", "`B`", and "`BC`" are right-branching, so strings "`A`", "`ABC`", and "`B`" are maximal repetitions. The longest of them is "`ABC`", its length is 3, there is only one maximal repetition of such length.

Problem D. Drawing Problem

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

Recently Andy has found an old book in the attic. The book from the end of the XIX-th century contains a lot of various puzzles. One puzzle impressed Andy very much — the complicated figure is drawn on the page and the task is to draw it in one stroke, never lifting the pen off the paper and never drawing the same non-zero part twice. To make the problem harder, the additional quest is to do it drawing the minimal number of segments.

Having spent several hours on the problem, Andy asked you to write the program that would do it for him.

Input

Input file contains the figure to be drawn as the set of segments. First the number of segments is specified. It is followed by segments descriptions. Each segment is described with coordinates of its endpoints. All segments are parallel to the sides of the page, that are aligned with coordinate axis.

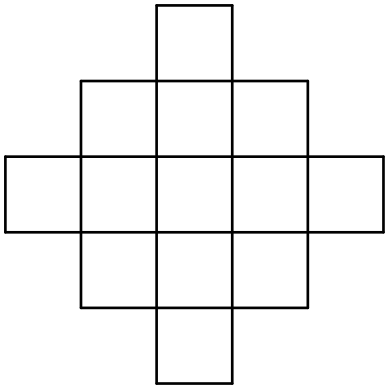
The number of segments does not exceed one hundred, all coordinates are integer and no not exceed one million by their absolute value, no two segments have a part of non-zero length in common.

Output

If it is impossible to fulfil the task, output zero on the first line of the output file. In the other case output the minimal number of segments to draw, followed by the segments themselves. Remember that it must be possible to draw segments continuously in order you list them, neither lifting the pen off the paper, nor drawing any non-zero segment twice. Endpoints of each segment must be listed in such an order, that it is drawn from the first endpoint to the second one.

Example

drawing.in	drawing.out
12 -2 2 -2 3 -1 1 -1 4 0 0 0 5 1 0 1 5 2 1 2 4 3 2 3 3 0 0 1 0 -1 1 2 1 -2 2 3 2 -2 3 3 3 -1 4 2 4 0 5 1 5	16 0 0 0 1 0 1 -1 1 -1 1 -1 4 -1 4 2 4 2 4 2 1 2 1 0 1 0 1 0 3 0 3 3 3 3 3 3 2 3 2 -2 2 -2 2 -2 3 -2 3 0 3 0 3 0 5 0 5 1 5 1 5 1 0 1 0 0 0
2 1 0 1 2 0 1 2 1	0



Problem E. Fibonacci Period

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

A sequence of integer numbers

$$\mathcal{F}_r = a_0, a_1, \ldots, a_n, \ldots,$$

is called the *Fibonacci sequence modulo* r if $a_0 = 0$, $a_1 = 1$, and $a_i = (a_{i-2} + a_{i-1}) \bmod r$ for all $i \geq 2$.

A number $p > 0$ is called the *period* of the sequence, if there is some i_0, such that for all $i \geq i_0$ the equation $a_i = a_{i+p}$ is satisfied. The sequence is called *periodic* if it has a period. Clearly, if the sequence is periodic, it has the smallest period.

Given r you have to find whether the sequence $\mathcal{F}_r$ is periodic, and if it is what is its smallest period.

Input

Input file contains an integer r ($2 \leq r \leq 2 \cdot 10^9$).

Output

If $\mathcal{F}_r$ is periodic, print its smallest period to the output file. If it is not, print 0.

Example

fibonacci.in	fibonacci.out
2	3
3	8
5	20

Problem F. Parallel Processes

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

When analyzing parallel processes that need synchronization, it is often important to know which types of tasks can be executed in parallel. Consider n tasks of m types. Let us call two types of tasks *independent*, if they can be executed in parallel. Tasks of one type are never independent and must be run sequentially.

Consider a sequence of tasks to be executed on k processors. Each second up to k next tasks can be executed if each pair of them can be run in parallel. Note, that it is not allowed to skip tasks or to save tasks for future execution. For example, if tasks of types A and B are independent, the sequence $AABABBAB$ can be executed on two processors in five seconds, one way to do so is $(A-), (AB), (AB), (B-), (AB)$.

The other way to run the given sequence of tasks on two processors is $(A-), (AB), (AB), (BA), (B-)$. We say that two ways are *essentially different* if there is some second when the sets of tasks that are being executed are different. One can see that there are two essentially different ways to run the sequence $AABABBAB$ of tasks on two processors in minimal time. If we also take into account which processor each task is executed on, there are 64 ways to execute the given set of tasks (in each of the two essentially different ways we can exchange the tasks between processors each second). So we say that there are 64 *exact* ways to execute the set of tasks.

Given types of tasks that are independent, the number of processors, and the sequence of tasks, you have to find out the minimal time required to execute the tasks, the number of essentially different ways to do so, and the number of different exact ways of the execution.

Input

The first line of the input file contains n — the number of tasks ($1 \le n \le 100$), m — the number of types of tasks ($1 \le m \le 26$, types of tasks are identified with first m capital letters of English alphabet), k — the number of processors ($1 \le k \le 10$) and p — the number of pairs of tasks that are independent. Next line contains p pairs of letters, specifying independent tasks. Last line contains n letters — the sequence of tasks to be executed.

Output

On the first line of the output file print the minimal number of seconds needed to execute given tasks. On the second line print the number of essentially different ways to do so. On the third line print the number of exact ways.

Example

parallel.in	parallel.out
8 2 2 1 AB AABABBAB	5 2 64
10 1 3 0 AAAAAAAAAA	10 1 59049

Problem G. Beautiful Permutation

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

Consider a permutation of integer numbers from 1 to n. Let us call length of the longest monotonic subsequence of the permutation its *ugliness*.

For example, the ugliness of the permutation $\langle 1, 2, 5, 3, 4\rangle$ is 4 because it has a monotonic subsequence $(1, 2, 3, 4)$ of length 4, but has none of length 5. The ugliness of the permutation $\langle 5, 6, 3, 4, 1, 2\rangle$ is 3 because it has a monotonic subsequence $(5, 3, 1)$ of length 3.

Let us call *beautiful* those permutations that have a smallest possible ugliness for given n. Given n, you must find the first in lexicographic order beautiful permutation of size n.

Input

Input file contains n ($1 \le n \le 10\,000$).

Output

Output the first in lexicographic order beautiful permutation of size n.

Example

perm.in	perm.out
2	1 2
4	2 1 4 3

Problem H. Build More Roads!

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

The king of Talkland is planning to build new roads in the kingdom. Indeed, the old roads are all muddy and full of ruts. The new roads will be paved with yellow bricks and shine on the sun. Of course, all roads will be two-way.

There are n cities in Talkland and the king wants to build as many roads as possible. However, he knows that if there are r cities, such that each pair of them is connected by a road, citizens of those communicate too much, so the risk of the revolution becomes too strong. Therefore he wants to build as many roads as possible, but in such way that no r cities are all pairwise connected.

You, his transportation and educaction wizard, must help!

Input

Input file contains n and r ($3 \le n \le 100$, $3 \le r \le n$).

Output

On the first line of the output file output m — the number of roads to build. Next m lines must contain the descriptions of the roads, for each road output the cities it must connect.

Example

roads.in	roads.out
4 3	4 1 2 2 3 3 4 4 1

Problem I. Secret Lab

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

The famous agent 003 James Summer is planning to steal the new superbomb from the secret lab located in the mountains. The lab has several rooms that are connected by the doors. Each door can be opened by some researchers working in the lab with their electronic keys.

James can kill some researchers and take their keys. After that he can open all the doors that could be opened by the researchers. Also he can simply wait at the door until somebody passes by it and walk through it together with him. Agents have investigated the daily routine of the lab, so it is known about each researcher when he will walk through the doors.

When killing somebody, James increases the risk of failing his mission. Also the risk increases every second he stays on the territory of the lab. So James needs to choose which researchers' keys he should get, and how to get to the room where the superbomb is located and out, minimizing the risk of mission failure.

Input

The first line of the input file contains n — the number of rooms in the lab, m — the number of doors, and k — the number of researchers ($2 \le n \le 20$, $1 \le m \le 100$, $1 \le k \le 10$). Next m lines contain descriptions of doors — each door is described with numbers of rooms it connects. All doors are two way and can be opened from either side. Two rooms may be connected with more than one door.

After that k blocks describing researchers are provided. Each block starts with integer number r_i — the risk James takes if killing this researcher ($1 \le r_i \le 32\,000$). Then d_i is specified — the number of doors that this researcher can open, followed by the list of the doors. Doors are numbered starting from 1, as they are given in the file. The daily routine of the researcher follows. It starts with a_i — the number of doors that he opens daily, followed by a_i pairs of integer numbers — the number of the door that is opened by the researcher and the time when he does it, measured in seconds from the beginning of the workday (this time is positive and does not exceed the length of the workday, that is 8 hours or 28 800 seconds). The events are listed in chronological order, $a_i \le 10$. It is guaranteed that each researcher walks only through the doors that he can open with his electronic key.

James can start his mission any time during workday, and his mission must be over before the end of the workday. Every second he stays in a lab his risk increases by 1. James enters the lab in the hall, which is room number 1. The superbomb is located in the room number n. James must get to the room where superbomb is and return to lab hall. You should take into account that James needs at least one second before he can walk through two doors, in particular at least one second before he can walk through the first door. His mission is over one second after he enters the lab hall with the superbomb.

Output

If it is impossible to fulfil the mission, print "`mission impossible`" on the first line of the output file. In the other case print the minimal risk James must take to fulfil the mission.

After that print the plan James must follow. First print the number of researchers that must be killed followed by their numbers. After that print the time from the beginning of the workday James must enter the lab. It must be followed by the list of doors James passes, each with the time James passes it, measured in seconds from the beginning of the workday he enters the lab. The last number must be the time James's mission is over.

Example

secret.in	secret.out
3 3 2 1 2 2 3 2 3 3000 2 2 3 3 2 3600 3 7200 2 14400 7000 2 1 2 3 1 600 1 3601 1 3700	3101 1 1 3600 1 3601 2 3602 2 3603 1 3700 3701
2 1 1 1 2 1000 1 1 2 1 28799 1 28800	1003 1 1 0 1 1 1 2 3
4 3 2 1 2 2 3 3 4 1000 1 1 0 1000 1 3 0	mission impossible

Problem J. Snooker

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

Snooker is a game played on a billiard-table. The game was invented in the nineteenth century in India. The game is played with balls of equal radius, that we will consider equal to 1.

Snooker is played with the following set of balls: fifteen object balls that are not numbered and are solid red (called *reds*), six object balls of other colors that are not numbered (called *colors*) and a cue ball (called the *white ball*).

Due to the rules of the game the player make strikes until he makes a foul. His strikes in turn have *object* on red balls or on color balls. If the strike has object on red balls, the player must force the white ball to contact some red ball before any color one, if the player has object on color balls, he must force the white ball contact some color ball before any red one. If the player fails to do so, it is considered a foul (there are also other types of fouls which are of no interest to us in this problem).

A special situation called *snooker* arises when there is no possibility to fulfil the requirement to hit the object ball with a *direct strike*. That means that if the center of the white ball moves along the straight line and the white ball does not make contact with borders of the table, it cannot hit any object ball without first hitting some forbidden one. Experienced snooker players can get out of snooker by making a strike with the help of the border or a spin strike, but snooker is indeed a big problem for beginners.

Given positions of all balls on the table, you have to determine whether the position on the table is a snooker. For the purpose of this problem we will ignore the limitation that there can be at most fifteen reds and six colors and assume that there can be at most one hundred balls on the table. We will also ignore the size of the table and the existance of pockets, considering the game taking place on an infinite plane.

Input

The first line of the input file contains r — the number of reds, c — the number of colors respectively, and the letter '`R`' if the object is on red balls and '`C`' if it is on color ones. Next line contains coordinates of the white ball. The following r lines contain coordinates of red balls, then c lines contain coordinates of color balls. All coordinates are real and do not exceed 1000.0 by their absolute value.

No balls intersect, although some may touch each other. No ball touches the white ball.

Output

Output "`YES`" if the position on the table is a snooker and "`NO`" if it is not.

Example

snooker.in	snooker.out
4 2 C 0.0 0.0 1.8 1.8 1.8 -1.8 -1.8 1.8 -1.8 -1.8 3.0 4.0 -6.0 -2.0	YES
1 0 R 0.0 0.0 600.0 600.0	NO