Problem E. Nice Patterns Strike Back

Input file:	`nice.in`
Output file:	`nice.out`
Time limit:	1 second

You might have noticed that there is the new fashion among rich people to have their yards tiled with black and white tiles, forming a pattern. The company *Broken Tiles* is well known as the best tiling company in our region. It provides the widest choices of nice patterns to tile your yard with. The pattern is *nice* if there is no square of size 2×2, such that all tiles in it have the same color. So patterns on the figure 1 are nice, while patterns on the figure 2 are not.

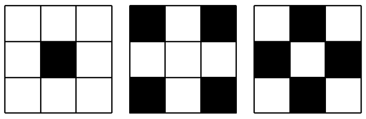


Figure 1.

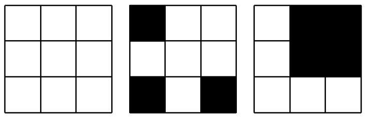


Figure 2.

The president of the company wonders whether the variety of nice patterns he can provide to the clients is large enough. Thus he asks you to find out the number of nice patterns that can be used to tile the yard of size $N \times M$. Now he is interested in the long term estimation, so he suggests $N \leq 10^{100}$. However, he does not like big numbers, so he asks you to find the answer modulo P.

Input

The input file contains three integer numbers: N ($1 \leq N \leq 10^{100}$), M ($1 \leq M \leq 5$) and P ($1 \leq P \leq 10000$).

Output

Write the number of nice patterns of size $N \times M$ modulo P to the output file.

Example

`nice.in`	`nice.out`
`2 2 5`	`4`
`3 3 23`	`0`

Problem G. Beautiful People

Input file:	`people.in`
Output file:	`people.out`
Time limit:	1 second

The most prestigious sports club in one city has exactly N members. Each of its members is strong and beautiful. More precisely, i-th member of this club (members being numbered by the time they entered the club) has strength S_i and beauty B_i. Since this is a very prestigious club, its members are very rich and therefore extraordinary people, so they often extremely hate each other. Strictly speaking, i-th member of the club Mr X hates j-th member of the club Mr Y if $S_i \le S_j$ and $B_i \ge B_j$ or if $S_i \ge S_j$ and $B_i \le B_j$ (if both properties of Mr X are greater then corresponding properties of Mr Y, he doesn't even notice him, on the other hand, if both of his properties are less, he respects Mr Y very much).

To celebrate a new 2003 year, the administration of the club is planning to organize a party. However they are afraid that if two people who hate each other would simultaneouly attend the party, after a drink or two they would start a fight. So no two people who hate each other should be invited. On the other hand, to keep the club prestige at the apropriate level, administration wants to invite as many people as possible.

Being the only one among administration who is not afraid of touching a computer, you are to write a program which would find out whom to invite to the party.

Input

The first line of the input file contains integer N — the number of members of the club. ($2 \le N \le 100\,000$). Next N lines contain two numbers each — S_i and B_i respectively ($1 \le S_i, B_i \le 10^9$).

Output

On the first line of the output file print the maximum number of the people that can be invited to the party. On the second line output N integers — numbers of members to be invited in arbitrary order. If several solutions exist, output any one.

Example

people.in	people.out

4
1 1
1 2
2 1
2 2
	2
	1 4

Problem C. Hyperhuffman

Input file:	`huffman.in`
Output file:	`huffman.out`
Time limit:	0.5 seconds

You might have heard about Huffman encoding — that is the coding system that minimizes the expected length of the text if the codes for characters are required to consist of an integral number of bits.

Let us recall codes assignment process in Huffman encoding. First the *Huffman tree* is constructed. Let the alphabet consist of N characters, i-th of which occurs P_i times in the input text. Initially all characters are considered to be active nodes of the future tree, i-th being marked with P_i. On each step take two active nodes with smallest marks, create the new node, mark it with the sum of the considered nodes and make them the children of the new node. Then remove the two nodes that now have parent from the set of active nodes and make the new node active. This process is repeated until only one active node exists, it is made the root of the tree.

Note that the characters of the alphabet are represented by the leaves of the tree. For each leaf node the length of its code in the Huffman encoding is the length of the path from the root to the node. The code itself can be constrcuted the following way: for each internal node consider two edges from it to its children. Assign 0 to one of them and 1 to another. The code of the character is then the sequence of 0s and 1s passed on the way from the root to the leaf node representing this character.

In this problem you are asked to detect the length of the text after it being encoded with Huffman method. Since the length of the code for the character depends only on the number of occurences of this character, the text itself is not given — only the number of occurences of each character. Characters are given from most rare to most frequent.

Note that the alphabet used for the text is quite huge — it may contain up to 500 000 characters.

Input

The first line of the input file contains N — the number of different characters used in the text ($2 \le N \le 500\,000$). The second line contains N integer numbers P_i — the number of occurences of each character ($1 \le P_i \le 10^9$, $P_i \le P_{i+1}$ for all valid i).

Output

Output the length of the text after encoding it using Huffman method, in bits.

Example

huffman.in	huffman.out
3 1 1 4	8

Problem B. Beloved Sons

Input file:	`beloved.in`
Output file:	`beloved.out`
Time limit:	1 second

Once upon a time there lived a king and he had N sons. And the king wanted to marry his beloved sons on the girls that they did love. So one day the king asked his sons to come to his room and tell him whom do they love.

But the sons of the king were all young men so they could not tell exactly whom they did love. Instead of that they just told him the names of the girls that seemed beautiful to them, but since they were all different, their choices of beautiful girls also did not match exactly.

The king was wise. He did write down the information that the children have provided him with and called you, his main wizard.

"I want all my kids to be happy, you know," he told you, "but since it might be impossible, I want at least some of them to marry the girl they like. So please, prepare the marriage list."

Suddenly you recalled that not so long ago the king told you about each of his sons, so you knew how much he loves him. So you decided to please the king and make such a marriage list that the king would be most happy. You know that the happiness of the king will be proportional to the square root of the sum of the squares of his love to the sons that would marry the girls they like.

So, go on, make a list to maximize the king's happiness.

Input

The first line of the input file contains N — the number of king's sons ($1 \le N \le 400$). The second line contains N integer numbers A_i ranging from 1 to 1000 — the measures of king's love to each of his sons.

Next N lines contain lists of king's sons' preferences — first K_i — the number of the girls the i-th son of the king likes, and then K_i integer numbers — the girls he likes (all potentially beautiful girls in the kingdom were numbered from 1 to N, you know, beautiful girls were rare in those days).

Output

Output N numbers — for each son output the number of the beautiful girl he must marry or 0 if he must not marry the girl he likes.

Denote the set of sons that marry a girl they like by L, then you must maximize the value of

$$\sqrt{\sum_{i \in L} A_i^2}$$

Example

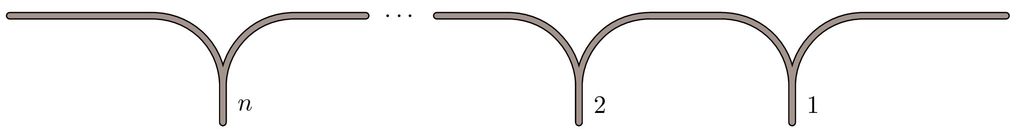
`beloved.in`	`beloved.out`

`4`
`1 3 2 4`
`4 1 2 3 4`
`2 1 4`
`2 1 4`
`2 1 4`
	`2 1 0 4`

Problem I. Railroad Sort

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

Consider the railroad station that has n dead-ends designed in a way shown on the picture. Dead-ends are numbered from right to left, starting from 1.



Let 2^n railroad cars get from the right. Each car is marked with some integer number ranging from 1 to 2^n, different cars are marked with different numbers.

You can move the cars through the dead-ends using the following two operations. If the car x is the first car on the path to the right of the dead-end i, you may move this car to this dead-end. If the car y is the topmost car in the dead-end j you can move it to the path on the left of the dead-end. Note, that cars cannot be moved to the dead-end from the path to its left and cannot be moved to the path on the right of the dead-end they are in.

Your task is to rearrange the cars so that the numbers on the cars listed from left to right were in the ascending order and all the cars are to the left of all the dead-ends.

One can prove that the required rearranging is always possible.

Input

The first line of the input file contains n — the number of dead-ends ($1 \le n \le 13$). The second line contains 2^n integer numbers — the numbers on the cars, listed from left to right.

Output

Output the sequence of operations. Each operation is identified with the number of the car moved in this operation. The type of the operation and the dead-end used are clearly determined uniquely.

Example

`railsort.in`	`railsort.out`

`2`
`3 2 1 4`
	`3 3 2 2 1 1 4 4 3 2 1 1 2 3 4 4`

The sequence of the operations in the example is the following: first we move all cars through dead-end 1 without changing their order, after that we put cars 3, 2 and 1 to the dead-end 2 and take them out of it, changing their order to the reverse. Finally we move the car 4 through the dead-end 2.

Problem B. Cryptography

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

Young cryptoanalyst Georgie is planning to break the new cipher invented by his friend Andie. To do this, he must make some linear transformations over the ring $\mathbb{Z}_r = \mathbb{Z}/r\mathbb{Z}$.

Each linear transformation is defined by 2×2 matrix. Georgie has a sequence of matrices $A_1, A_2, \ldots, A_n$. As a step of his algorithm he must take some segment $A_i, A_{i+1}, \ldots, A_j$ of the sequence and multiply some vector by a product $P_{i,j} = A_i \times A_{i+1} \times \cdots \times A_j$ of the segment. He must do it for m various segments.

Help Georgie to determine the products he needs.

Input

The first line of the input file contains r ($1 \le r \le 10\,000$), n ($1 \le n \le 30\,000$) and m ($1 \le m \le 30\,000$). Next n blocks of two lines, containing two integer numbers ranging from 0 to $r-1$ each, describe matrices. Blocks are separated with blank lines. They are followed by m pairs of integer numbers ranging from 1 to n each that describe segments, products for which are to be calculated.

Output

Print m blocks containing two lines each. Each line should contain two integer numbers ranging from 0 to $r-1$ and define the corresponding product matrix.

Separate blocks with an empty line.

Example

crypto.in	crypto.out
3 4 4	0 2
0 1	0 0
0 0	
	0 2
2 1	0 1
1 2	
	0 1
0 0	0 0
0 2	
	2 1
1 0	1 2
0 2	
1 4	
2 3	
1 3	
2 2	

Problem C. Fibonacci Subsequence

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

A sequence of integer numbers $a_1, a_2, \ldots, a_n$ is called a *Fibonacci sequence* if $a_i = a_{i-2} + a_{i-1}$ for all $i = 3, 4, \ldots, n$.

Given a sequence of integer numbers $c_1, c_2, \ldots, c_m$ you have to find its longest Fibonacci subsequence.

Input

The first line of the input file contains m $(1 \le m \le 3\,000)$. Next line contains m integer numbers not exceeding 10^9 by their absolute value.

Output

On the first line of the output file print the maximal length of the Fibonacci subsequence of the given sequence. On the second line print the subsequence itself.

Example

fibsubseq.in	fibsubseq.out
10 1 1 3 -1 2 0 5 -1 -1 8	5 1 -1 0 -1 -1

Problem E. Long Dominoes

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

Find the number of ways to tile an $m \times n$ rectangle with long dominoes — 3×1 rectangles.

Each domino must be completely within the rectangle, dominoes must not overlap (of course, they may touch each other), each point of the rectangle must be covered.

Input

The input file contains m and n ($1 \le m \le 9$, $1 \le n \le 30$).

Output

Output the number of ways to tile an $m \times n$ rectangle with long dominoes.

Example

`dominoes.in`	`dominoes.out`
`3 3`	`2`
`3 10`	`28`

Problem K. Unfair Contest

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

Chief Judge of the Galactic Programming Contest in 3141–3157 has recently published his memoirs in which he writes about sensational facts. The jury of the contest used to make problem sets specially designed for some teams to win. In the book the judge described the technics used in details. In this problem you are asked to implement the algorithm used by the judges.

Before the contest the jury prepares m problems, each characterized with its *intellectuality* I_i, *technics requirements* A_i, *code length* L_i, and *ordinariness* O_i. The goal of the jury is to select n problems for the contest.

Each team participating in the contest is in turn characterized by its *theoretical background* T_j, *programming technics* Z_j, *typing speed* V_j, and *contests practice* C_j.

The j-th team can solve i-th problem if and only if $T_j + C_j$ exceeds $I_i - O_i$. The team solves problems one after one, in the ascending order of expected solution time. Expected solution time for the problem is

$$e_{i,j} = \lceil I_i/O_i \rceil + \max(\lceil A_i/C_j \rceil, 5).$$

But the actual time required to solve the problem is

$$t_{i,j} = \max(I_i - T_j, 0) + \lceil A_i/(Z_j + C_j) \rceil + \lceil L_i/V_j \rceil.$$

If two problems have the same expected solution time, we consider the team lucky, and suggest that if first solves the problem with smaller actual required time.

If the team does not finish to solve the problem within the contest length l, it does not solve it. If the team finished to solve problem exactly when the contest is over, we consider that it has solved the problem.

The teams are finally arranged in the descending order by the number of problems solved, and if the number is equal, in the ascending order by the penalty time. The penalty time is the sum of penalties for each solved problem. The penalty time for the problem is the time from the beginning of the contest till the moment the problem is solved. Teams with equal both number of solved problems and penalty time have the same highest possible for them rank.

The unsuccesful runs are ignored in the model, because it is difficult to predict them.

Given the descriptions of m problems and t teams, select n such problems, that the team 1 gets the highest possible rank.

Input

The first line of the input file contains m, n, t and l ($1 \le m \le 16$, $1 \le n \le 12$, $n \le m$, $1 \le t \le 20$, $10 \le l \le 500$).

The following m lines describe problems, each line contains I_i, A_i, L_i and O_i (all numbers are integer, ranging from 1 to 1000, except L_i which is from 100 to 50 000).

The following t lines describe teams, each line contains T_j, Z_j, V_j and C_j (all numbers are integer, ranging from 1 to 1000).

Output

Output n integer numbers in the ascending order — the problems that must be selected.

Example

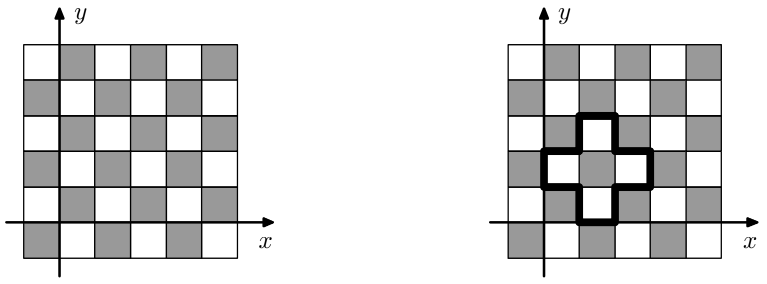
unfair.in	unfair.out

3 2 4 60
20 40 2200 10
60 10 600 10
10 20 1500 40
100 10 100 10
100 30 70 2
20 20 300 10
30 1 100 1
	2 3

Problem C. Black and White

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

Consider an infinite chessboard. Introduce a coordinate system on it in such a way that chessboard cells are unit squares with integer corner coordates. Let the cells be colored black and white like on the standard chessboard, let the cell with bottom left corner at $(0, 0)$ be colored black.



Somebody has drawn a closed polyline on the board. The vertices of the polyline are in the corners of the cells and its sides are parallel to the coordinate axes. It's interesting, what is the number of black and white cells inside the polyline. Find that out.

Input

The first line of the input file contains n — the number of vertices of the polyline ($1 \leq n \leq 50\,000$). The following n lines contain the coordinates of the vertices in counter-clockwise order. Coordinates are integer and do not exceed 10^9 by their absolute values. Polyline has no self-intersections and no self-touchings.

Output

Output two numbers: b and w — the number of black and white cells inside the polyline repectively.

Example

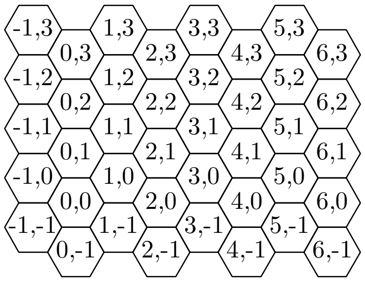
bw.in	bw.out

```
12
1 0
2 0
2 1
3 1
3 2
2 2
2 3
1 3
1 2
0 2
0 1
1 1
```
	`1 4`

Problem E. Islands

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

You are working in a team developing the new strategic game *Island Warriors*. The game proceeds on a hexagonal grid, the coordinate system on the map is introduced as shown on the picture below.



A hexagon can be either land, or sea. An *island* is a maximal connected set of land hexagons, an area of the island is the number of hexagons in it.

Your current task is writing random map generator. The map generated must satisfy some conditions, in particular no island area must exceed s.

The design of your module is the following. Initially all hexagons are sea. Special random generator provides you with the sequence of hexagons. Getting the next hexagon, you check whether it is already the land one. If it is, you just skip this hexagon. If it is sea, you check whether convering it to land results in an island with area exceeding s. If it does, you also skip this hexagon. In the other case you convert it to land.

So, the design step is completed, now its time to implement the module. Fortunately, your teammate has already implemented random generator, so you just have to implement the rest of the module. To check yourself you decided first to output only the number of islands in the resulting map and their areas.

Input

The first line of the input file contains n — the number of hexagons provided to you by random generator ($1 \le n \le 50\,000$), and s ($1 \le s \le n$).

The following n lines contain two integer numbers each — coordinates of the hexagons. Coordinates do not exceed 500 by their absolute values.

Output

Output the number of islands in the resulting map and their areas. List areas in the ascending order.

Example

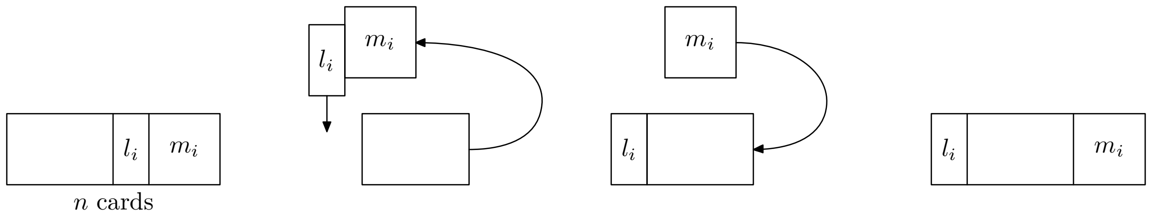
islands.in	islands.out
7 4	2
0 1	2 3
2 1	
3 0	
1 -1	
2 1	
1 0	
0 0	

Problem J. Cheater's Shuffle

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

Many card cheaters tricks are based on the shuffling of the deck. Collecting the cards from the table and shuffling the deck in a special way, the cheater may get the desired order of the cards. In this problem you have to model the process to see that the cheater can often reach his goal.

The deck contains n cards. The shuffling consists of exactly t *moves*. There are p ways to make each move. The i-th way to make some move is the following. The shuffler picks $m_i + l_i$ cards from the top of the deck and moves them to the bottom of the deck. There he drops the last l_i cards picked, letting them stay at the bottom of the deck, and returns the other m_i cards to the top of the deck. The order of the cards within the blocks of l_i and m_i cards, and the cards staying in the deck remains the same.



Given the initial order of the cards in the deck, the desired order, and the parameters of the shuffle, find out whether it is possible to reorder the cards in the specified way.

Input

The first line of the input file contains n, t, and p ($2 \le n \le 36$, $1 \le t \le 15$, $1 \le p \le 5$). The next two lines contain the initial and the desired order of the cards respectively. Cards are identified by integer numbers from 1 to n, all cards are different, the cards are listed from the bottom of the deck to its top.

The following p lines contain two numbers each — m_i and l_i ($1 \le l_i < n$, $0 \le m_i < n$, $l_i + m_i < n$).

Output

If it is impossible to reoreder the cards in the required way using exactly t shuffling moves, output "`Impossible`" on the first line of the output file. In the other case output t integer numbers — in which way to perform each move.

Example

shuffle.in	shuffle.out

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

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

Problem G. Inverse Range Minimum Query

Input file:	`rmq.in`
Output file:	`rmq.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Inverse problems represent a quickly growing area in computer science. Unlike traditional problems, where given some data D, the task is to solve some optimization or calculation problem P, in the inverse problem you are given a problem P and the result R of the optimization/calculation. You have to restore the original data D. In this problem you are asked to solve inverse range minimum query problem.

Consider an array $a[1..n]$. The answer to a range minimum query $Q(i, j)$ is the minimal value among $a[i], \ldots, a[j]$. You are given n and a series of range minimum queries with answers. Restore the original array a.

Input

The first line of the input file contains n — the size of the array, and m — the number of queries ($1 \le n, m \le 100\,000$). The following m lines contain three integer numbers each: numbers i, j and q mean that $Q(i, j) = q$ ($1 \le i \le j \le n$, $-2^{31} \le q \le 2^{31} - 1$).

Output

If data in the input file is inconsistent, i.e. no such array a exists, output "`inconsistent`" at the first line of the output file.

In the other case output "`consistent`" at the first line of the output file. The second line must contain the array. The elements of the array must be integers between -2^{31} and $2^{31} - 1$. If there are several solutions, output any one.

Examples

rmq.in	rmq.out

3 2
1 2 1
2 3 2
	consistent
	1 2 3

3 3
1 2 1
1 1 2
2 3 2
	inconsistent

Problem A. Finite Automata

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

The *Broken Tiles* company has designed a robot for tiling the roads. The robot has an infinite supply of pavement tiles, each of which has the size of 2×1 feet. The roads to tile have a size of $m \times n$ feet. Here m is the width of the road, and n is the length of the road.

The program for the robot is the sequence of the commands. There are three commands: '`H`', '`V`' and '`S`'. The robot considers the road to tile consisting of $m \times n$ unit squares. The road runs from west to east.

The robot begins the execution of the program standing in the north-western square of the road. Each step it considers the current command. If it is '`H`', the robot puts the tile horizontally — the longer side from east to west, the western square of the tile occupies the current square. If the command is '`V`', the robot puts the tile vertically — the longer side from north to south, the northern square of the tile occupies the current square. Finally, if the command is '`S`', the robot does nothing on the current square. After executing the command, the robot moves one square southwards. If it crosses the border of the road, it moves one square eastwards, and moves to the northern square of the new column.

The road is said to be tiled correctly, if all of its squares are covered by the tiles, and no tiles overlap each other or cross the border of the road. The program is said to be correct for m, if it causes the robot to tile the $m \times n$ road correctly for some n, and after finishing the execution of the program the robot steps out of the southeastern corner of the road. For example, the program "`HHSS`" is correct for 2 (it correctly tiles the road for $n = 2$). The program "`HHV`", in turn, is incorrect for 2 for two reasons: first two tiles overlap with the third tile, and the robot does not leave the road in the end of the program for any n.

The designers of the company asked the main programmer of the company to write the program that would verify the correctness of the program for the robot.

But the main programmer of the company has recently learned the theory of finite automata and decided that everything should be programmed using finite automata only. Fortunately, it turned out, that for each m there indeed exists a finite automaton that accepts those and only those strings that form a correct for m program for the robot.

But the main programmer has gone to the Open Automata Documentation summit, leaving you alone with his ideas. The designers (and, more important, managers) are waiting for the verification automaton. So given m, you have to create a finite automaton for recognizing the correct programs. Since this is a business project, the automaton must be deterministic.

Recall, that the deterministic finite automaton (DFA) is an ordered set $\langle \Sigma, U, s, T, \varphi \rangle$ where Σ is the finite set called *input alphabet* ($\Sigma = \{$`H`, `V`, `S`$\}$ in our case), U is the finite set of *states*, $s \in U$ is the *initial state*, $T \subset U$ is the set of *terminal states* and $\varphi : U \times \Sigma \to U$ is the *transition function*.

The input of the automaton is the string α over Σ. Initially the automaton is in state s. Each step it reads the first character c of the input string and changes its state to $\varphi(u, c)$ where u is the current state. After that the first character of the input string is removed and the step repeats. If the automaton is in the terminal state after its input string is empty, it accepts the initial string α, in the other case it rejects it.

Input

The input file contains m ($1 \le m \le 10$).

Output

The first line of the output file must contain u — the number of states of the automaton, and s — the

Problem G. Palindromes

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

A non-empty string is called a *palindrome* if it coincides with its reverse, i.e. it the same if read from left to right, and from right to left. The examples of the palindromes are: “`ABBA`”, “`MADAMIMADAM`”, “`ABACABADABACABA`”.

Given a string s, find the number of different palindromes that are substrings of s.

Input

The input file contains a string s that consists of 1 to 5 000 of capital English letters.

Output

Output the number of different palindromes that are substrings of s.

Example

palindromes.in	palindromes.out
`ABACABADABACABA`	`15`