Problem A. Alice the Fan

Alice is a big fan of volleyball and especially of the very strong "Team A".

Volleyball match consists of up to five sets. During each set teams score one point for winning a ball. The first four sets are played until one of the teams scores at least 25 points and the fifth set is played until one of the teams scores at least 15 points. Moreover, if one of the teams scores 25 (or 15 in the fifth set) points while the other team scores 24 (or 14 in the fifth set), the set is played until the absolute difference between teams' points becomes two. The match ends when one of the teams wins three sets. The match score is the number of sets won by each team.

Alice found a book containing all the results of all matches played by "Team A". The book is old, and some parts of the book became unreadable. Alice can not read the information on how many sets each of the teams won, she can not read the information on how many points each of the teams scored in each set, she even does not know the number of sets played in a match. The only information she has is the total number of points scored by each of the teams in all the sets during a single match.

Alice wonders what is the best match score "Team A" could achieve in each of the matches. The bigger is the difference between the number of sets won by "Team A" and their opponent, the better is the match score. Find the best match score or conclude that no match could end like that. If there is a solution, then find any possible score for each set that results in the best match score.

Input

The first line contains a single integer m ($1 \le m \le 50\,000$) — the number of matches found by Alice in the book.

Each of the next m lines contains two integers a and b ($0 \le a, b \le 200$) — the number of points scored by "Team A" and the number of points scored by their opponents respectively.

Output

Output the solution for every match in the same order as they are given in the input. If the teams could not score a and b points respectively, output "`Impossible`".

Otherwise, output the match score formatted as "x:y", where x is the number of sets won by "Team A" and y is the number of sets won by their opponent.

The next line should contain the set scores in the order they were played. Each set score should be printed in the same format as the match score, with x being the number of points scored by "Team A" in this set, and y being the number of points scored by their opponent.

Example

standard input	standard output

6
75 0
90 90
20 0
0 75
78 50
80 100
	3:0
	25:0 25:0 25:0
	3:1
	25:22 25:22 15:25 25:21
	Impossible
	0:3
	0:25 0:25 0:25
	3:0
	25:11 28:26 25:13
	3:2
	25:17 0:25 25:22 15:25 15:11

Problem B. Bimatching

In the XIX Century dancing was one of the most important subjects learned by nobles at schools. Balls were the main or even the only opportunity to get to know new people, both for romantic and for business purposes. However, at that time many cavaliers died in wars. For these and other reasons, lack of cavaliers for dances was a rather common issue at balls. But there was an elegant solution to this problem. Some dances were modified for a cavalier to ask two ladies for a dance instead of one. New dances meant for a set of triples appeared as well as usual dances for a set of pairs.

Nowadays there are communities interested in a reconstruction of dances, and the number of cavaliers is still usually less than the number of ladies. So dances for lady-cavalier-lady triples are studied now, too.

Betty is a leader of a popular community of historical dances. Unfortunately, there are some pairs where a cavalier and a lady dislike each other and do not want to dance together in the same triple. Betty is rather observant and knows all such dislikes in the community. All she needs is a way to find the maximum possible number t of triples that can be selected for a dance. She does not even need to know such a *bimatching* between cavaliers and ladies, because she likes to puzzle dancers saying: "And now form t triples, please. I know that it is possible". Please help Betty to find the maximum lady-cavalier-lady bimatching capacity.

Input

The first line of the input contains the number of test cases t ($1 \le t \le 20$). Then t descriptions of test cases follow.

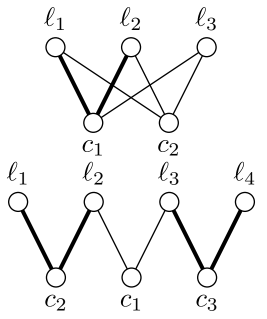
For each test case the first line contains the number n of cavaliers and the number m of ladies ($1 \le n, m$; $n + m \le 150$). Next n lines contain strings of m zeros and ones each. The character at position ℓ in row c is one if and only if lady ℓ can be in one triple with cavalier c. The sum of $n + m$ over all test cases in a single input does not exceed 150.

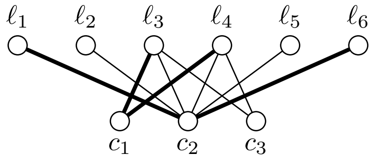
Output

For each test case output the maximum number t of triples in a single line.

Examples

standard input	standard output	Illustration

2
2 3
111
111
3 4
0110
1100
0011
	1
	2
		 ℓ_1
		ℓ_2
		ℓ_3
		c_1
		c_2
		ℓ_1
		ℓ_2
		ℓ_3
		ℓ_4
		c_2
		c_1
		c_3

1
3 6
001100
111111
001100
	2	 ℓ_1
		ℓ_2
		ℓ_3
		ℓ_4
		ℓ_5
		ℓ_6
		c_1
		c_2
		c_3

Note

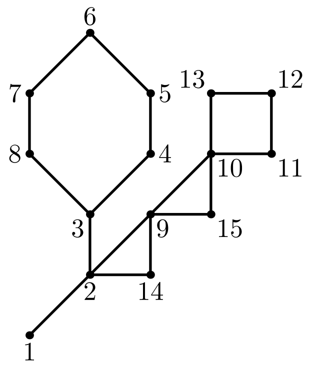
In the illustration each line means that a cavalier can dance with a lady. Bold line means that a cavalier asks a lady for a dance.

Problem C. Cactus Search

If you want to make an array problem harder — solve it on a tree.
If you want to make a tree problem harder — solve it on a cactus

Conventional wisdom

NEERC featured a number of problems in previous years about cactuses — connected undirected graphs in which every edge belongs to at most one simple cycle. Intuitively, a *cactus* is a generalization of a tree where some cycles are allowed. An example of a cactus from NEERC 2007 problem is given on the picture below.



You are playing a game on a cactus with Chloe. You are given a cactus. Chloe had secretly picked one vertex v from the cactus and your goal is to find it. You can make at most 10 guesses. If your guess is vertex v — you win. Otherwise, if your guess is another vertex u — Chloe helps you and tells you some vertex w which is adjacent to u and such that the distance from w to v is strictly less than the distance from u to v (here the *distance* is the number of edges in the shortest path between vertices).

Interaction Protocol

First, the testing system writes a line with two integers n and m ($1 \le n \le 500; 0 \le m \le 500$). Here n is the number of vertices in the graph. Vertices are numbered from 1 to n. Edges of the graph are represented by a set of edge-distinct paths, where m is the number of such paths. Each of the following m lines contains a path in the graph. A path starts with an integer k_i ($2 \le k_i \le 500$) followed by k_i integers from 1 to n. These k_i integers represent vertices of a path. Adjacent vertices in a path are distinct. The path can go to the same vertex multiple times, but every edge is traversed exactly once in the whole input. The graph in the input is a cactus.

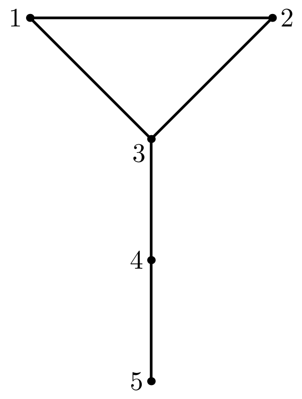
To prove that your program can find a secret vertex in at most 10 queries, you need to do that n times. Each time testing system picks some vertex before interacting with your program. Your program makes guesses by writing lines with a single number u ($1 \le u \le n$).

Testing system responds by writing lines with one of the two responses:

- "`FOUND`" — means that your guess is correct. After that, your program should proceed to the next test case or terminate if n vertices were already guessed.
- "`GO` w" — means that your guess is incorrect, but now you know that the distance from vertex w to the secret vertex is less than the distance from u. It is guaranteed that vertices u and w are connected by an edge.

Do not forget to flush the output after each guess!

Example

standard input	standard output	Illustration
5 2		
5 1 2 3 4 5		
2 1 3		
	3	
FOUND		
	3	
GO 4		
	4	
FOUND		
	3	
GO 2		
	2	
FOUND		
	3	
GO 1		
	1	
FOUND		
	3	
GO 4		
	4	
GO 5		
	5	
FOUND		

Note

Empty lines are added to the standard input and the standard output examples for clarity only. They are not present during the actual interaction.

Problem D. Distance Sum

You are given a connected undirected unweighted graph. The distance $d(u, v)$ between two vertices u and v is defined as the number of edges in the shortest path between them. Find the sum of $d(u, v)$ over all unordered pairs (u, v).

Input

The first line of the input contains two integers n and m ($2 \le n \le 10^5$; $n-1 \le m \le n+42$) — the number of vertices and the number of edges in the graph respectively. The vertices are numbered from 1 to n.

Each of the following m lines contains two integers x_i and y_i ($1 \le x_i, y_i \le n$; $x_i \ne y_i$) — the endpoints of the i-th edge.

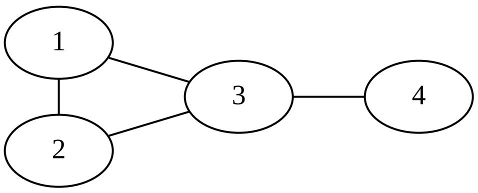
There is at most one edge between every pair of vertices.

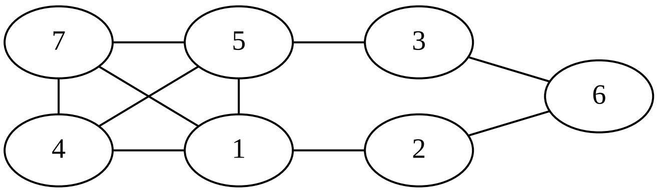
Output

Output a single integer — the sum of the distances between all unordered pairs of vertices in the graph.

Examples

standard input	standard output	Illustration

4 4
1 2
2 3
3 1
3 4
	8	

7 10
1 2
2 6
5 3
5 4
5 7
3 6
1 7
5 1
7 4
4 1
	34	

Note

In the first example the distance between four pairs of vertices connected by an edge is equal to 1 and $d(1, 4) = d(2, 4) = 2$.

Problem E. Easy Chess

Elma is learning chess figures.

She learned that a rook can move either horizontally or vertically. To enhance her understanding of rook movement Elma's grandmother gave Elma an 8×8 chess board and asked her to find a way to move the rook from `a1` to `h8` making exactly n moves, so that all visited cells are different.

A visited cell is the initial cell `a1` and each cell on which the rook lands after a move.

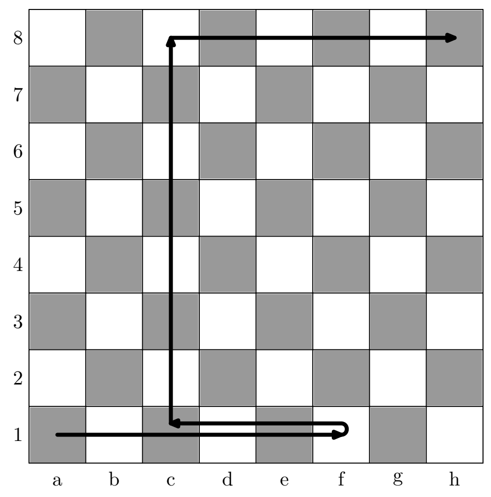
Input

The input contains a single integer n ($2 \le n \le 63$) — the desired number of moves.

Output

Output a space-separated list of $n+1$ visited cells in the order they are visited by the rook. All cells must be different. The list should start with `a1` and end with `h8`. A solution always exists.

Example

standard input	standard output	Illustration
4	a1 f1 c1 c8 h8	

Problem F. Fractions

You are given a positive integer n.

Find a sequence of fractions $\frac{a_i}{b_i}$, $i = 1 \ldots k$ (where a_i and b_i are positive integers) for some k such that:

$$\begin{cases} b_i \text{ divides } n,\ 1 < b_i < n \text{ for } i = 1 \ldots k \\ 1 \le a_i < b_i \text{ for } i = 1 \ldots k \\ \sum\limits_{i=1}^{k} \frac{a_i}{b_i} = 1 - \frac{1}{n} \end{cases}$$

Input

The input consists of a single integer n ($2 \le n \le 10^9$).

Output

In the first line print "`YES`" if there exists such a sequence of fractions or "`NO`" otherwise.

If there exists such a sequence, next lines should contain a description of the sequence in the following format.

The second line should contain integer k ($1 \le k \le 100\,000$) — the number of elements in the sequence. It is guaranteed that if such a sequence exists, then there exists a sequence of length at most $100\,000$.

Next k lines should contain fractions of the sequence with two integers a_i and b_i on each line.

Examples

standard input	standard output
2	NO
6	YES 2 1 2 1 3

Note

In the second example there is a sequence $\frac{1}{2}, \frac{1}{3}$ such that $\frac{1}{2} + \frac{1}{3} = 1 - \frac{1}{6}$.

Problem G. Guest Student

Berland State University invites people from all over the world as guest students. You can come to the capital of Berland and study with the best teachers in the country.

Berland State University works every day of the week, but classes for guest students are held on the following schedule. You know the sequence of seven integers $a_1, a_2, \dots, a_7$ ($a_i = 0$ or $a_i = 1$):

- $a_1 = 1$ if and only if there are classes for guest students on Sundays;
- $a_2 = 1$ if and only if there are classes for guest students on Mondays;
- ...
- $a_7 = 1$ if and only if there are classes for guest students on Saturdays.

The classes for guest students are held in at least one day of a week.

You want to visit the capital of Berland and spend the minimum number of days in it to study k days as a guest student in Berland State University. Write a program to find the length of the shortest continuous period of days to stay in the capital to study exactly k days as a guest student.

Input

The first line of the input contains integer t ($1 \le t \le 10\,000$) — the number of test cases to process. For each test case independently solve the problem and print the answer.

Each test case consists of two lines. The first of them contains integer k ($1 \le k \le 10^8$) — the required number of days to study as a guest student. The second line contains exactly seven integers $a_1, a_2, \dots, a_7$ ($a_i = 0$ or $a_i = 1$) where $a_i = 1$ if and only if classes for guest students are held on the i-th day of a week.

Output

Print t lines, the i-th line should contain the answer for the i-th test case — the length of the shortest continuous period of days you need to stay to study exactly k days as a guest student.

Example

standard input	standard output

3
2
0 1 0 0 0 0 0
100000000
1 0 0 0 1 0 1
1
1 0 0 0 0 0 0
	8
	233333332
	1

Note

In the first test case you must arrive to the capital of Berland on Monday, have classes on this day, spend a week until next Monday and have classes on the next Monday. In total you need to spend 8 days in the capital of Berland.

Problem H. Harder Satisfiability

A *fully quantified* boolean 2-CNF formula is a formula in the following form: $Q_1 x_1 \dots Q_n x_n\, F(x_1, \dots, x_n)$. Each Q_i is one of two quantifiers: a *universal* quantifier $\forall$ ("for all"), or an *existential* quantifier $\exists$ ("exists"); and F is a conjunction (boolean `AND`) of m clauses $s \vee t$ (boolean `OR`), where s and t are some variables (not necessarily different) with or without negation. This formula has no free variables, so it evaluates to either `true` or `false`. We can evaluate a given fully quantified formula with a simple recursive algorithm:

1. If there are no quantifiers, return the remaining expression's value of `true` or `false`.
2. Otherwise, recursively evaluate formulas: $F_z = Q_2 x_2 \dots Q_n x_n\, F(z, x_2, \dots, x_n)$ for $z = 0, 1$.
3. If $Q_1 = \exists$ return $F_0 \vee F_1$; otherwise if $Q_1 = \forall$ return $F_0 \wedge F_1$.

You are given some fully quantified boolean 2-CNF formulas. Find out if they are true or not.

Input

The first line of the input contains a single integer t ($1 \le t \le 10^5$) — the number of test cases.

The first line of a test case contains two integers n and m ($1 \le n, m \le 10^5$) — the number of variables and the number of clauses in F. The next line contains a string s with n characters describing the quantifiers. If s_i = '`A`' then Q_i is a universal quantifier $\forall$, otherwise if s_i = '`E`' then s_i is an existential quantifier $\exists$.

Next m lines describe clauses in F. Each line contains two integers u_i and v_i ($-n \le u_i, v_i \le n$; $u_i, v_i \ne 0$). If $u_i \ge 1$ then the first variable in the i-th clause is x_{u_i}. Otherwise, if $u_i \le -1$ then the first variable is $\overline{x_{-u_i}}$ (negation of x_{-u_i}). The second variable in the i-th clause is similarly described by v_i.

The sum of values of n for all test cases does not exceed 10^5; the sum of values of m does not exceed 10^5.

Output

For each test case output "`TRUE`" if the given formula is true or "`FALSE`" otherwise.

Example

standard input	standard output
3 2 2 AE 1 -2 -1 2 2 2 EA 1 -2 -1 2 3 2 AEA 1 -2 -1 -3	TRUE FALSE FALSE

Note

The first sample corresponds to a formula $\forall x_1\, \exists x_2\, (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) = \forall x_1\, \exists x_2\; x_1 \oplus x_2$. For any x_1 we can choose $x_2 = \overline{x_1}$ making it true, hence the formula is true.

The second sample changes the order of quantifiers. Now the answer is "`FALSE`", because for any value of x_1 we can choose $x_2 = x_1$ and the formula becomes false.

The third formula is $\forall x_1\, \exists x_2\, \forall x_3\, (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3})$. If we substitute $x_1 = 1$, $x_3 = 1$ then no assignment of x_2 can make the second clause true, so the formula is false.

Problem I. Interval-Free Permutations

Consider a permutation $p_1, p_2, \ldots p_n$ of integers from 1 to n. We call a sub-segment $p_l, p_{l+1}, \ldots, p_{r-1}, p_r$ of the permutation an *interval* if it is a reordering of some set of consecutive integers. For example, the permutation $(6, 7, 1, 8, 5, 3, 2, 4)$ has the intervals $(6, 7)$, $(5, 3, 2, 4)$, $(3, 2)$, and others.

Each permutation has some trivial intervals — the full permutation itself and every single element. We call a permutation *interval-free* if it does not have non-trivial intervals. In other words, interval-free permutation does not have intervals of length between 2 and $n - 1$ inclusive.

Your task is to count the number of interval-free permutations of length n modulo prime number p.

Input

In the first line of the input there are two integers t $(1 \le t \le 400)$ and p $(10^8 \le p \le 10^9)$ — the number of test cases to solve and the prime modulo. In each of the next t lines there is one integer n $(1 \le n \le 400)$ — the length of the permutation.

Output

For each of t test cases print a single integer — the number of interval-free permutations modulo p.

Examples

standard input	standard output

4 998244353
1
4
5
9
	1
	2
	6
	28146

1 437122297 20	67777575

Note

For $n = 1$ the only permutation is interval-free. For $n = 4$ two interval-free permutations are $(2, 4, 1, 3)$ and $(3, 1, 4, 2)$. For $n = 5$ — $(2, 4, 1, 5, 3)$, $(2, 5, 3, 1, 4)$, $(3, 1, 5, 2, 4)$, $(3, 5, 1, 4, 2)$, $(4, 1, 3, 5, 2)$, and $(4, 2, 5, 1, 3)$. We will not list all 28146 for $n = 9$, but for example $(4, 7, 9, 5, 1, 8, 2, 6, 3)$, $(2, 4, 6, 1, 9, 7, 3, 8, 5)$, $(3, 6, 9, 4, 1, 5, 8, 2, 7)$, and $(8, 4, 9, 1, 3, 6, 2, 7, 5)$ are interval-free.

The exact value for $n = 20$ is 264111424634864638.

Problem J. JS Minification

International Coding Procedures Company (ICPC) writes all its code in Jedi Script (JS) programming language. JS does not get compiled, but is delivered for execution in its source form. Sources contain comments, extra whitespace (including trailing and leading spaces), and other non-essential features that make them quite large but do not contribute to the semantics of the code, so the process of *minification* is performed on source files before their delivery to execution to compress sources while preserving their semantics.

You are hired by ICPC to write JS minifier for ICPC. Fortunately, ICPC adheres to very strict programming practices and their JS sources are quite restricted in grammar. They work only on integer algorithms and do not use floating point numbers and strings.

Every JS source contains a sequence of lines. Each line contains zero or more tokens that can be separated by spaces. On each line, a part of the line that starts with a hash character (`#` code 35), including the hash character itself, is treated as a comment and is ignored up to the end of the line.

Each line is parsed into a sequence of tokens from left to right by repeatedly skipping spaces and finding the longest possible token starting at the current parsing position, thus transforming the source code into a sequence of tokens. All the possible tokens are listed below:

- A *reserved* token is any kind of operator, separator, literal, reserved word, or a name of a library function that should be preserved during the minification process. Reserved tokens are fixed strings of non-space ASCII characters that do not contain the hash character (`#` code 35). All reserved tokens are given as an input to the minification process.
- A *number* token consists of a sequence of digits, where a *digit* is a character from zero (`0`) to nine (`9`) inclusive.
- A *word* token consists of a sequence of characters from the following set: lowercase letters, uppercase letters, digits, underscore (`_` code 95), and dollar sign (`$` code 36). A word does not start with a digit.

Note, that during parsing the longest sequence of characters that satisfies either a number or a word definition, but that appears in the list of reserved tokens, is considered to be a reserved token instead.

During the minification process words are renamed in a systematic fashion using the following algorithm:

1. Take a list of words that consist only of lowercase letters ordered first by their length, then lexicographically: "`a`", "`b`", ..., "`z`", "`aa`", "`ab`", ..., excluding reserved tokens, since they are not considered to be words. This is the *target word list.*
2. Rename the first word encountered in the input token sequence to the first word in the target word list and all further occurrences of the same word in the input token sequence, too. Rename the second new word encountered in the input token sequence to the second word in the target word list, and so on.

The goal of the minification process is to convert the given source to the shortest possible line (counting spaces) that still parses to the same sequence of tokens with the correspondingly renamed words using these JS parsing rules.

Input

The first line of the input contains a single integer n ($0 \leq n \leq 40$) — the number of reserved tokens.

The second line of the input contains the list of reserved tokens separated by spaces without repetitions in the list. Each reserved token is at least one and at most 20 characters long and contains only characters with ASCII codes from 33 (exclamation mark) to 126 (tilde) inclusive, with exception of a hash character ('#' code 35).

The third line of the input contains a single integer m ($1 \le m \le 40$) — the number of lines in the input source code.

Next m lines contain the input source, each source line is at most 80 characters long (counting leading and trailing spaces). Each line contains only characters with ASCII codes from 32 (space) to 126 (tilde) inclusive. The source code is valid and fully parses into a sequence of tokens.

Output

Write to the output a single line that is the result of the minification process on the input source code. The output source line shall parse to the same sequence of tokens as the input source with the correspondingly renamed words and shall contain the minimum possible number of spaces needed for that. If there are multiple ways to insert the minimum possible number of spaces into the output, use any way.

Examples

standard input

```
16
fun while return var { } ( ) , ; > = + ++ - --
9
fun fib(num) { # compute fibs
  var return_value = 1, prev = 0, temp;
  while (num > 0) {
    temp = return_value; return_value = return_value + prev;
    prev = temp;
    num--;
  }
  return return_value;
}
```

standard output

```
fun a(b){var c=1,d=0,e;while(b>0){e=c;c=c+d;d=e;b--;}return c;}
```

standard input

```
10
( ) + ++ : -> >> >>: b c)
2
($val1++ + +4 kb) >> :out
b-> + 10 >>: t # using >>:
```

standard output

```
(a+++ +4c )>> :d b->+10>>:e
```

Problem K. King Kog's Reception

King Kog got annoyed of the usual laxity of his knights — they can break into his hall without prior notice! Thus, the King decided to build a reception with a queue where each knight chooses in advance the time when he will come and how long the visit will take. The knights are served in the order of the recorded time, but each knight has to wait until the visits of all the knights before him are finished.

Princess Keabeanie wants to see her father. However, she does not want to interrupt the knights so she joins the queue. Unfortunately, the knights change their minds very often — they can join the queue or cancel their visits. Please help the princess to understand how long she will have to wait until she sees her father if she enters the queue at the specified moments of time given the records at the reception.

Input

The first line of the input contains a single integer q ($1 \le q \le 3 \cdot 10^5$) — the number of events. An event can be of three types: *join*, *cancel*, or *query*.

- Join "`+` t d" ($1 \le t, d \le 10^6$) — a new knight joins the queue, where t is the time when the knight will come and d is the duration of the visit.
- Cancel "`-` i" ($1 \le i \le q$) — the knight cancels the visit, where i is the number (counted starting from one) of the corresponding join event in the list of all events.
- Query "`?` t" ($1 \le t \le 10^6$) — Keabeanie asks how long she will wait if she comes at the time t.

It is guaranteed that after each event there are no two knights with the same entrance time in the queue. Cancel events refer to the previous joins that were not cancelled yet.

Keabeanie can come at the same time as some knight, but Keabeanie is very polite and she will wait for the knight to pass.

Output

For each query write a separate line with the amount of time Keabeanie will have to wait.

Example

standard input	standard output
19 ? 3 + 2 2 ? 3 ? 4 + 5 2 ? 5 ? 6 + 1 2 ? 2 ? 3 ? 4 ? 5 ? 6 ? 7 ? 9 - 8 ? 2 ? 3 ? 6	0 1 0 2 1 3 2 1 2 1 0 0 2 1 1

Problem L. Lazyland

The kingdom of Lazyland is the home to n idlers. These idlers are incredibly lazy and create many problems to their ruler, the mighty King of Lazyland.

Today k important jobs for the kingdom ($k \le n$) should be performed. Every job should be done by one person and every person can do at most one job. The King allowed every idler to choose one job they wanted to do and the i-th idler has chosen the job a_i.

Unfortunately, some jobs may not be chosen by anyone, so the King has to persuade some idlers to choose another job. The King knows that it takes b_i minutes to persuade the i-th idler. He asked his minister of labour to calculate the minimum total time he needs to spend persuading the idlers to get all the jobs done. Can you help him?

Input

The first line of the input contains two integers n and k ($1 \le k \le n \le 10^5$) — the number of idlers and the number of jobs.

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le k$) — the jobs chosen by each idler.

The third line of the input contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le 10^9$) — the time the King needs to spend to persuade the i-th idler.

Output

The only line of the output should contain one number — the minimum total time the King needs to spend persuading the idlers to get all the jobs done.

Examples

standard input	standard output

8 7
1 1 3 1 5 3 7 1
5 7 4 8 1 3 5 2
	10

3 3
3 1 2
5 3 4
	0

Note

In the first example the optimal plan is to persuade idlers 1, 6, and 8 to do jobs 2, 4, and 6.

In the second example each job was chosen by some idler, so there is no need to persuade anyone.

Problem M. Minegraphed

Marika is developing a game called "Minegraphed" that involves moving around a 3D rectangular world.

Each cell of a parallelepiped game field is either an empty cell or an obstacle cell. You are always standing inside an empty cell, located either in the bottommost layer or on top of an obstacle cell. Each move can be made in one of four directions: north, east, south, or west. You make a move according to these rules:

- If you try to move outside of the parallelepiped, then you can't make this move.
- Otherwise, if the cell in front of you is empty, then you move one cell forward and then fall towards the bottom until you reach the bottommost layer or an obstacle cell.
- Otherwise, if you are in non-topmost layer, the cell in front of you is an obstacle, the cell above you and the cell above that obstacle are both empty, then you move (climb up) on top of that obstacle.
- Otherwise, you can't make this move.

Marika prepared a directed graph with n vertices. Now she wants to lay out the field and label n different possible standing positions with numbers from 1 to n, so that it is possible to get from the cell labeled i to the cell labeled j by making valid moves if and only if in Marika's graph it is possible to get from vertex i to vertex j along the edges of the graph. Help Marika design a field satisfying this property.

Input

The first line contains a single integer n ($1 \le n \le 9$) — the number of vertices. Each of the next n lines contains n integers equal to 0 or 1. The j-th number in the i-th line is 1 if there is an edge from vertex i to vertex j and 0 otherwise. The i-th number in the i-th line is always zero.

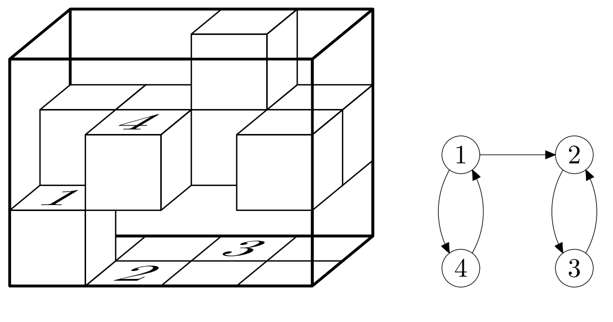
Output

Output a layer-by-layer description of a field that has the same reachability as the given graph.

The first line should contain three positive integers x, y, and z, the west-east, north-south, and top-bottom sizes of the designed parallelepiped. Blocks describing z layers should follow, from the topmost layer to the bottommost layer, separated by single empty lines. Each block should contain y lines of length x, consisting of dots ('`.`'), hashes ('`#`'), and digits from 1 to n. A hash denotes an obstacle cell. A dot denotes an unlabeled empty cell. A digit denotes a labeled empty cell. Each digit from 1 to n should appear exactly once. Each digit should be located either in the bottommost layer or on top of an obstacle cell. It is okay for obstacles to be "hanging in the air" (there is no gravity for them).

The volume of the field $x \times y \times z$ should not exceed 10^6. It is guaranteed that there is a correct field fitting into this limit for any possible graph in the input.

Example

standard input	standard output	Illustration
4	4 2 3	
0 1 0 1	..#.	
0 0 1 0	.4..	
0 1 0 0		
1 0 0 0	####	
	1#.#	
	..3.	
	#2..	

Note

Note that you can climb up from cell 1 to cell 4, but you cannot climb up from cell 2 to cell 1 because of a "ceiling" obstacle.