A. Clear Symmetry

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Consider some square matrix A with side n consisting of zeros and ones. There are n rows numbered from 1 to n from top to bottom and n columns numbered from 1 to n from left to right in this matrix. We'll denote the element of the matrix which is located at the intersection of the i-row and the j-th column as $A_{i,j}$.

Let's call matrix A *clear* if no two cells containing ones have a common side.

Let's call matrix A *symmetrical* if it matches the matrices formed from it by a horizontal and/or a vertical reflection. Formally, for each pair (i,j) $(1 \le i,j \le n)$ both of the following conditions must be met: $A_{i,j} = A_{n-i+1,j}$ and $A_{i,j} = A_{i,n-j+1}$.

Let's define the *sharpness* of matrix A as the number of ones in it.

Given integer x, your task is to find the smallest positive integer n such that there exists a clear symmetrical matrix A with side n and sharpness x.

Input

The only line contains a single integer x $(1 \le x \le 100)$ — the required sharpness of the matrix.

Output

Print a single number — the sought value of n.

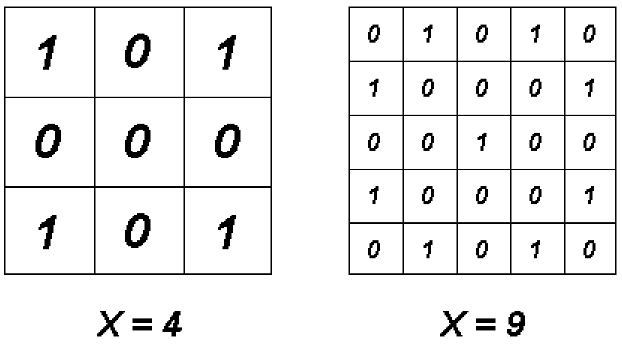
Sample test(s)

```
input
4
output
3
```

```
input
9
output
5
```

Note

The figure below shows the matrices that correspond to the samples:



B. Guess That Car!

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A widely known among some people Belarusian sport programmer Yura possesses lots of information about cars. That is why he has been invited to participate in a game show called "Guess That Car!".

The game show takes place on a giant parking lot, which is $4n$ meters long from north to south and $4m$ meters wide from west to east. The lot has $n+1$ dividing lines drawn from west to east and $m+1$ dividing lines drawn from north to south, which divide the parking lot into $n \cdot m$ 4 by 4 meter squares. There is a car parked strictly inside each square. The dividing lines are numbered from 0 to n from north to south and from 0 to m from west to east. Each square has coordinates (i, j) so that the square in the north-west corner has coordinates $(1, 1)$ and the square in the south-east corner has coordinates (n, m). See the picture in the notes for clarifications.

Before the game show the organizers offer Yura to occupy any of the $(n+1) \cdot (m+1)$ intersection points of the dividing lines. After that he can start guessing the cars. After Yura chooses a point, he will be prohibited to move along the parking lot before the end of the game show. As Yura is a car expert, he will always guess all cars he is offered, it's just a matter of time. Yura knows that to guess each car he needs to spend time equal to the square of the euclidean distance between his point and the center of the square with this car, multiplied by some coefficient characterizing the machine's "rarity" (the rarer the car is, the harder it is to guess it). More formally, guessing a car with "rarity" c placed in a square whose center is at distance d from Yura takes $c \cdot d^2$ seconds. The time Yura spends on turning his head can be neglected.

It just so happened that Yura knows the "rarity" of each car on the parking lot in advance. Help him choose his point so that the total time of guessing all cars is the smallest possible.

Input

The first line contains two integers n and m $(1 \le n, m \le 1000)$ — the sizes of the parking lot. Each of the next n lines contains m integers: the j-th number in the i-th line describes the "rarity" c_{ij} ($0 \le c_{ij} \le 100000$) of the car that is located in the square with coordinates (i, j).

Output

In the first line print the minimum total time Yura needs to guess all offered cars. In the second line print two numbers l_i and l_j ($0 \le l_i \le n$, $0 \le l_j \le m$) — the numbers of dividing lines that form a junction that Yura should choose to stand on at the beginning of the game show. If there are multiple optimal starting points, print the point with smaller l_i. If there are still multiple such points, print the point with smaller l_j.

Please do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams or the `%I64d` specifier.

Sample test(s)

input

```
2 3
3 4 5
3 9 1
```

output

```
392
1 1
```

input

```
3 4
1 0 0 0
0 0 3 0
0 0 5 5
```

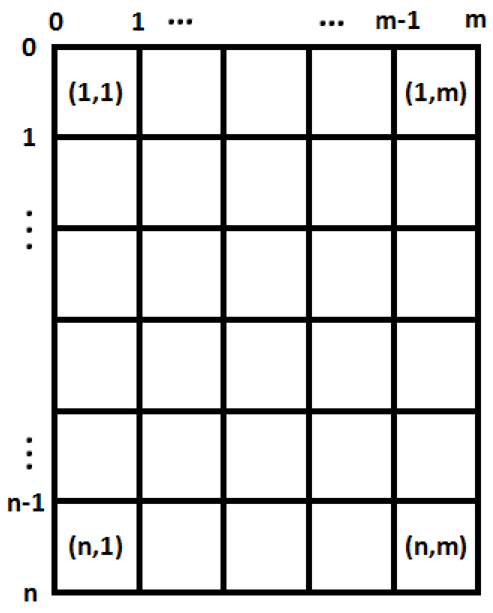
output

```
240
2 3
```

Note

In the first test case the total time of guessing all cars is equal to 3·8+3·8+4·8+9·8+5·40+1·40=392.

The coordinate system of the field:



0
1
…
…
m-1
m
0
1
⋮
⋮
n-1
n
(1,1)
(1,m)
(n,1)
(n,m)

C. Fragile Bridges

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are playing a video game and you have just reached the bonus level, where the only possible goal is to score as many points as possible. Being a perfectionist, you've decided that you won't leave this level until you've gained the maximum possible number of points there.

The bonus level consists of n small platforms placed in a line and numbered from 1 to n from left to right and $(n - 1)$ bridges connecting adjacent platforms. The bridges between the platforms are very fragile, and for each bridge the number of times one can pass this bridge from one of its ends to the other before it collapses forever is known in advance.

The player's actions are as follows. First, he selects one of the platforms to be the starting position for his hero. After that the player can freely move the hero across the platforms moving by the undestroyed bridges. As soon as the hero finds himself on a platform with no undestroyed bridge attached to it, the level is automatically ended. The number of points scored by the player at the end of the level is calculated as the number of transitions made by the hero between the platforms. Note that if the hero started moving by a certain bridge, he has to continue moving in the same direction until he is on a platform.

Find how many points you need to score to be sure that nobody will beat your record, and move to the next level with a quiet heart.

Input

The first line contains a single integer n $(2 \le n \le 10^5)$ — the number of platforms on the bonus level. The second line contains $(n - 1)$ integers a_i $(1 \le a_i \le 10^9, 1 \le i < n)$ — the number of transitions from one end to the other that the bridge between platforms i and $i + 1$ can bear.

Output

Print a single integer — the maximum number of points a player can get on the bonus level.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams or the `%I64d` specifier.

Sample test(s)

input

```
5
2 1 2 1
```

output

```
5
```

Note

One possibility of getting 5 points in the sample is starting from platform 3 and consequently moving to platforms 4, 3, 2, 1 and 2. After that the only undestroyed bridge is the bridge between platforms 4 and 5, but this bridge is too far from platform 2 where the hero is located now.

D. Brand New Problem

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A widely known among some people Belarusian sport programmer Lesha decided to make some money to buy a one square meter larger flat. To do this, he wants to make and carry out a Super Rated Match (SRM) on the site Torcoder.com. But there's a problem — a severe torcoder coordinator Ivan does not accept any Lesha's problem, calling each of them an offensive word "duped" (that is, duplicated). And one day they nearly quarrelled over yet another problem Ivan wouldn't accept.

You are invited to act as a fair judge and determine whether the problem is indeed *brand new*, or Ivan is right and the problem bears some resemblance to those used in the previous SRMs.

You are given the descriptions of Lesha's problem and each of Torcoder.com archive problems. The description of each problem is a sequence of words. Besides, it is guaranteed that Lesha's problem has no repeated words, while the description of an archive problem may contain any number of repeated words.

The "similarity" between Lesha's problem and some archive problem can be found as follows. Among all permutations of words in Lesha's problem we choose the one that occurs in the archive problem as a subsequence. If there are multiple such permutations, we choose the one with the smallest number of inversions. Then the "similarity" of a problem can be written as $p = \frac{n \cdot (n-1)}{2} - x + 1$, where n is the number of words in Lesha's problem and x is the number of inversions in the chosen permutation. Note that the "similarity" p is always a positive integer.

The problem is called *brand new* if there is not a single problem in Ivan's archive which contains a permutation of words from Lesha's problem as a subsequence.

Help the boys and determine whether the proposed problem is new, or specify the problem from the archive which resembles Lesha's problem the most, otherwise.

Input

The first line contains a single integer n $(1 \le n \le 15)$ — the number of words in Lesha's problem. The second line contains n space-separated words — the short description of the problem.

The third line contains a single integer m $(1 \le m \le 10)$ — the number of problems in the Torcoder.com archive. Next m lines contain the descriptions of the problems as "$k\ s_1\ s_2\ \dots\ s_k$", where k $(1 \le k \le 500000)$ is the number of words in the problem and s_i is a word of the problem description.

All words from all problem descriptions contain no more than 10 lowercase English letters. It is guaranteed that the total length of words in all problem descriptions does not exceed 500015.

Output

If Lesha's problem is *brand new*, print string "`Brand new problem!`" (without quotes).

Otherwise, on the first line print the index of the archive problem which resembles Lesha's problem most. If there are multiple such problems, print the one with the smallest index. On the second line print a string consisting of characters `[:`, character `|` repeated p times, and characters `:]`, where p is the "similarity" between this problem and Lesha's one. The archive problems are numbered starting from one in the order in which they are given in the input.

Sample test(s)

input
```
4
find the next palindrome
1
10 find the previous palindrome or print better luck next time
```
output
```
1
[:||||||:]
```

input
```
3
add two numbers
3
1 add
2 two two
3 numbers numbers numbers
```
output
```
Brand new problem!
```

input
```
4
these papers are formulas
3
6 what are these formulas and papers
5 papers are driving me crazy
4 crazy into the night
```

output

```
1
[:||||:]
```

input

```
3
add two decimals
5
4 please two decimals add
5 decimals want to be added
4 two add decimals add
4 add one two three
7 one plus two plus three equals six
```

output

```
3
[:|||:]
```

Note

Let us remind you that the number of inversions is the number of pairs of words that follow in the permutation not in their original order. Thus, for example, if the original problem is "add two numbers", then permutation "numbers add two" contains two inversions — pairs of words "numbers" and "add", "numbers" and "two".

Sequence $b_1, b_2, ..., b_k$ is a subsequence of sequence $a_1, a_2, ..., a_n$ if there exists such a set of indices $1 \le i_1 < i_2 < ... < i_k \le n$ that $a_{i_j} = b_j$ (in other words, if sequence b can be obtained from a by deleting some of its elements).

In the first test case the first problem contains the "find the palindrome next" permutation as a subsequence, in which the number of inversions equals 1 (words "palindrome" and "next").

In the second test case there is no problem that contains a permutation of words from Lesha's problem as a subsequence.

E. Thoroughly Bureaucratic Organization

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Once n people simultaneously signed in to the reception at the recently opened, but already thoroughly bureaucratic organization (abbreviated TBO). As the organization is thoroughly bureaucratic, it can accept and cater for exactly one person per day. As a consequence, each of n people made an appointment on one of the next n days, and no two persons have an appointment on the same day.

However, the organization workers are very irresponsible about their job, so none of the signed in people was told the exact date of the appointment. The only way to know when people should come is to write some requests to TBO.

The request form consists of m empty lines. Into each of these lines the name of a signed in person can be written (it can be left blank as well). Writing a person's name in the same form twice is forbidden, such requests are ignored. TBO responds very quickly to written requests, but the reply format is of very poor quality — that is, the response contains the correct appointment dates for all people from the request form, but the dates are in completely random order. Responds to all requests arrive simultaneously at the end of the day (each response specifies the request that it answers).

Fortunately, you aren't among these n lucky guys. As an observer, you have the following task — given n and m, determine the minimum number of requests to submit to TBO to clearly determine the appointment date for each person.

Input

The first line contains a single integer t $(1 \le t \le 1000)$ — the number of test cases. Each of the following t lines contains two integers n and m $(1 \le n, m \le 10^9)$ — the number of people who have got an appointment at TBO and the number of empty lines in the request form, correspondingly.

Output

Print t lines, each containing an answer for the corresponding test case (in the order they are given in the input) — the minimum number of requests to submit to TBO.

Sample test(s)

input

```
5
4 1
4 2
7 3
1 1
42 7
```

output

```
3
2
3
0
11
```

Note

In the first sample, you need to submit three requests to TBO with three different names. When you learn the appointment dates of three people out of four, you can find out the fourth person's date by elimination, so you do not need a fourth request.

In the second sample you need only two requests. Let's number the persons from 1 to 4 and mention persons 1 and 2 in the first request and persons 1 and 3 in the second request. It is easy to see that after that we can clearly determine each person's appointment date regardless of the answers obtained from TBO.

In the fourth sample only one person signed up for an appointment. He doesn't need to submit any requests — his appointment date is tomorrow.