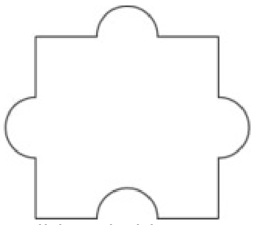
A. Puzzle Pieces

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a special jigsaw puzzle consisting of $n \cdot m$ identical pieces. Every piece has three tabs and one blank, as pictured below.



The jigsaw puzzle is considered solved if the following conditions hold:

1. The pieces are arranged into a grid with n rows and m columns.
2. For any two pieces that share an edge in the grid, a tab of one piece fits perfectly into a blank of the other piece.

Through rotation and translation of the pieces, determine if it is possible to solve the jigsaw puzzle.

Input

The test consists of multiple test cases. The first line contains a single integer t $(1 \le t \le 1000)$ — the number of test cases. Next t lines contain descriptions of test cases.

Each test case contains two integers n and m $(1 \le n, m \le 10^5)$.

Output

For each test case output a single line containing "YES" if it is possible to solve the jigsaw puzzle, or "NO" otherwise. You can print each letter in any case (upper or lower).

Example

input

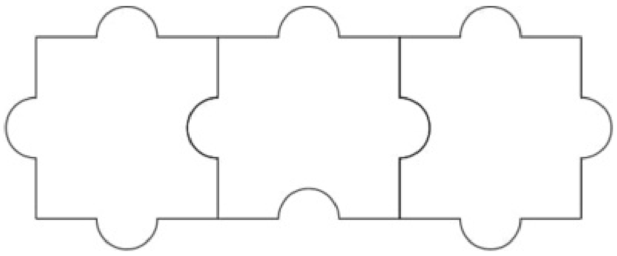
```
3
1 3
100000 100000
2 2
```

output

```
YES
NO
YES
```

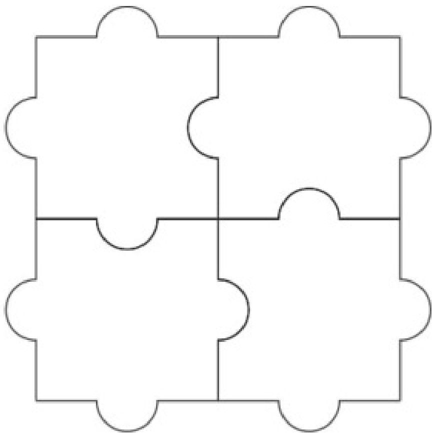
Note

For the first test case, this is an example solution:



For the second test case, we can show that no solution exists.

For the third test case, this is an example solution:



B. Card Constructions

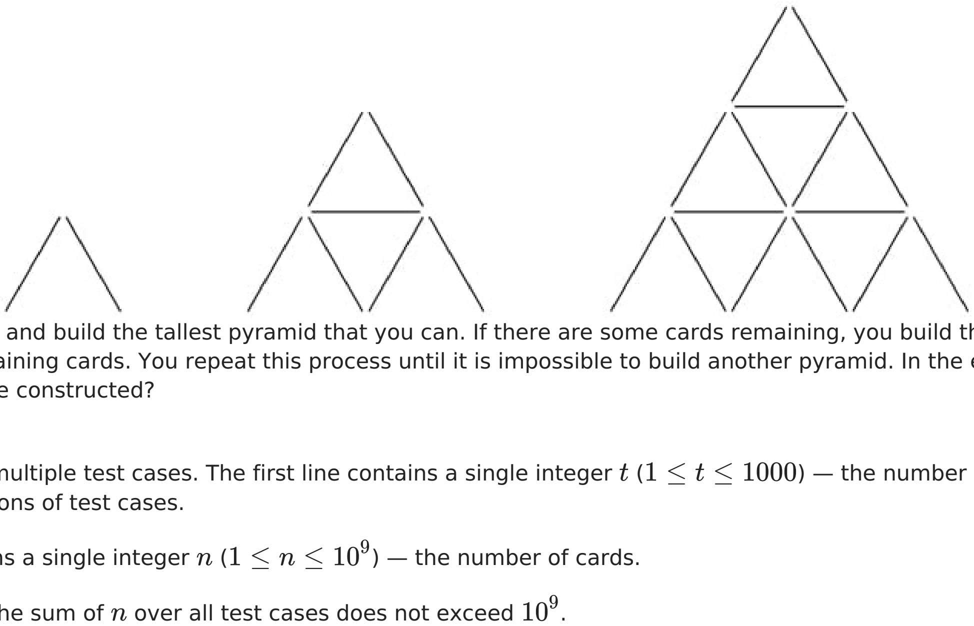
time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

output: standard output

A card pyramid of height 1 is constructed by resting two cards against each other. For $h > 1$, a card pyramid of height h is constructed by placing a card pyramid of height $h-1$ onto a base. A base consists of h pyramids of height 1, and $h-1$ cards on top. For example, card pyramids of heights 1, 2, and 3 look as follows:



You start with n cards and build the tallest pyramid that you can. If there are some cards remaining, you build the tallest pyramid possible with the remaining cards. You repeat this process until it is impossible to build another pyramid. In the end, how many pyramids will you have constructed?

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Next t lines contain descriptions of test cases.

Each test case contains a single integer n ($1 \le n \le 10^9$) — the number of cards.

It is guaranteed that the sum of n over all test cases does not exceed 10^9.

Output

For each test case output a single integer — the number of pyramids you will have constructed in the end.

Example

input

```
5
3
14
15
24
1
```

output

```
1
2
1
3
0
```

Note

In the first test, you construct a pyramid of height 1 with 2 cards. There is 1 card remaining, which is not enough to build a pyramid.

In the second test, you build two pyramids, each of height 2, with no cards remaining.

In the third test, you build one pyramid of height 3, with no cards remaining.

In the fourth test, you build one pyramid of height 3 with 9 cards remaining. Then you build a pyramid of height 2 with 2 cards remaining. Then you build a final pyramid of height 1 with no cards remaining.

In the fifth test, one card is not enough to build any pyramids.

C. Hilbert's Hotel

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Hilbert's Hotel is a very unusual hotel since the number of rooms is infinite! In fact, there is exactly one room for every integer, **including zero and negative integers**. Even stranger, the hotel is currently at full capacity, meaning there is exactly one guest in every room. The hotel's manager, David Hilbert himself, decides he wants to shuffle the guests around because he thinks this will create a vacancy (a room without a guest).

For any integer k and positive integer n, let $k \bmod n$ denote the remainder when k is divided by n. More formally, $r = k \bmod n$ is the smallest non-negative integer such that $k - r$ is divisible by n. It always holds that $0 \le k \bmod n \le n - 1$. For example, $100 \bmod 12 = 4$ and $(-1337) \bmod 3 = 1$.

Then the shuffling works as follows. There is an array of n integers $a_0, a_1, \ldots, a_{n-1}$. Then for each integer k, the guest in room k is moved to room number $k + a_{k \bmod n}$.

After this shuffling process, determine if there is still exactly one guest assigned to each room. That is, there are no vacancies or rooms with multiple guests.

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. Next $2t$ lines contain descriptions of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the array.

The second line of each test case contains n integers $a_0, a_1, \ldots, a_{n-1}$ ($-10^9 \le a_i \le 10^9$).

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single line containing "YES" if there is exactly one guest assigned to each room after the shuffling process, or "NO" otherwise. You can print each letter in any case (upper or lower).

Example

input

```
6
1
14
2
1 -1
4
5 5 5 1
3
3 2 1
2
0 1
5
-239 -2 -100 -3 -11
```

output

```
YES
YES
YES
NO
NO
YES
```

Note

In the first test case, every guest is shifted by 14 rooms, so the assignment is still unique.

In the second test case, even guests move to the right by 1 room, and odd guests move to the left by 1 room. We can show that the assignment is still unique.

In the third test case, every fourth guest moves to the right by 1 room, and the other guests move to the right by 5 rooms. We can show that the assignment is still unique.

In the fourth test case, guests 0 and 1 are both assigned to room 3.

In the fifth test case, guests 1 and 2 are both assigned to room 2.

D. Monopole Magnets

time limit per test: 2 seconds
memory limit per test: 256 megabytes

input: standard input
output: standard output

A monopole magnet is a magnet that only has one pole, either north or south. They don't actually exist since real magnets have two poles, but this is a programming contest problem, so we don't care.

There is an $n \times m$ grid. Initially, you may place some north magnets and some south magnets into the cells. You are allowed to place as many magnets as you like, even multiple in the same cell.

An operation is performed as follows. Choose a north magnet and a south magnet to activate. If they are in the same row or the same column and they occupy different cells, then the north magnet moves one unit closer to the south magnet. Otherwise, if they occupy the same cell or do not share a row or column, then nothing changes. Note that the south magnets are immovable.

Each cell of the grid is colored black or white. Let's consider ways to place magnets in the cells so that the following conditions are met.

1. There is at least one south magnet in every row and every column.
2. If a cell is colored black, then it is possible for a north magnet to occupy this cell after some sequence of operations **from the initial placement**.
3. If a cell is colored white, then it is impossible for a north magnet to occupy this cell after some sequence of operations **from the initial placement**.

Determine if it is possible to place magnets such that these conditions are met. If it is possible, find the minimum number of north magnets required (there are no requirements on the number of south magnets).

Input

The first line contains two integers n and m $(1 \le n, m \le 1000)$ — the number of rows and the number of columns, respectively.

The next n lines describe the coloring. The i-th of these lines contains a string of length m, where the j-th character denotes the color of the cell in row i and column j. The characters "#" and "." represent black and white, respectively. It is guaranteed, that the string will not contain any other characters.

Output

Output a single integer, the minimum possible number of north magnets required.

If there is no placement of magnets that satisfies all conditions, print a single integer -1.

Examples

input

```
3 3
.#.
###
##.
```

output

```
1
```

input

```
4 2
##
.#
.#
##
```

output

```
-1
```

input

```
4 5
....#
####.
.###.
.#...
```

output

```
2
```

input

```
2 1
.
#
```

output

```
-1
```

input

```
3 5
```

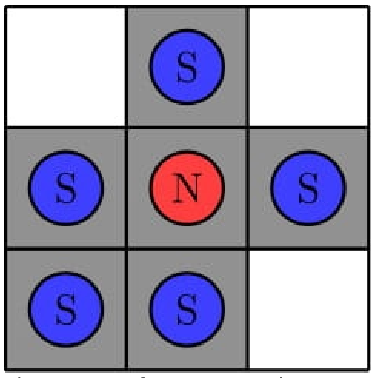
```
.....
.....
.....
```

output

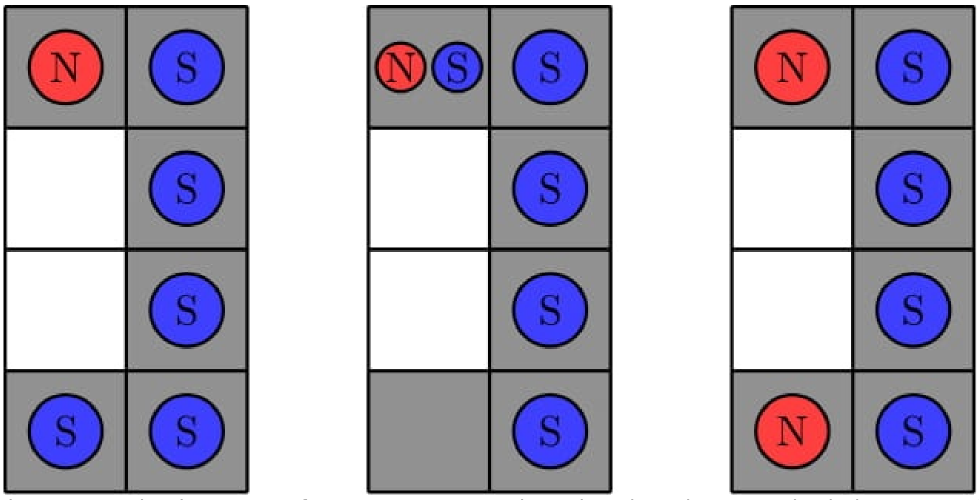
```
0
```

Note

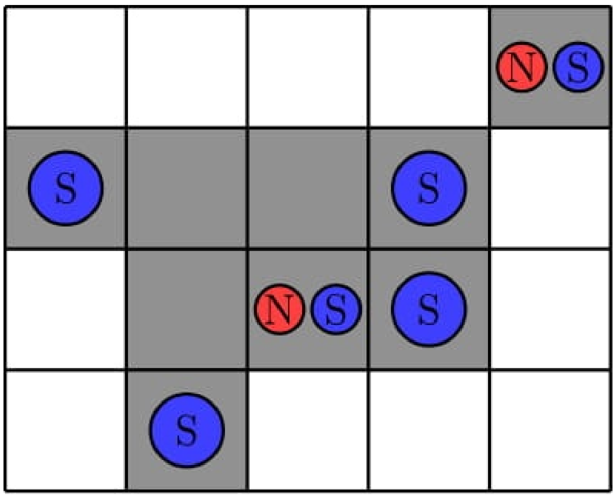
In the first test, here is an example placement of magnets:



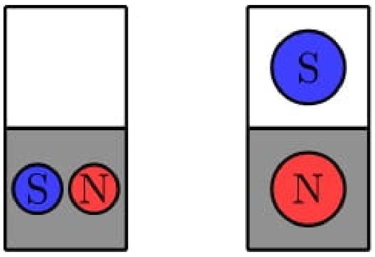
In the second test, we can show that no required placement of magnets exists. Here are three example placements that fail to meet the requirements. The first example violates rule 3 since we can move the north magnet down onto a white square. The second example violates rule 2 since we cannot move the north magnet to the bottom-left black square by any sequence of operations. The third example violates rule 1 since there is no south magnet in the first column.



In the third test, here is an example placement of magnets. We can show that there is no required placement of magnets with fewer north magnets.



In the fourth test, we can show that no required placement of magnets exists. Here are two example placements that fail to meet the requirements. The first example violates rule 1 since there is no south magnet in the first row. The second example violates rules 1 and 3 since there is no south magnet in the second row and we can move the north magnet up one unit onto a white square.



In the fifth test, we can put the south magnet in each cell and no north magnets. Because there are no black cells, it will be a correct placement.

E. Quantifier Question

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Logical quantifiers are very useful tools for expressing claims about a set. For this problem, let's focus on the set of real numbers specifically. **The set of real numbers includes zero and negatives.** There are two kinds of quantifiers: universal ($\forall$) and existential ($\exists$). You can read more about them here.

The universal quantifier is used to make a claim that a statement holds *for all real numbers*. For example:

- $\forall x, x < 100$ is read as: for all real numbers x, x is less than 100. This statement is false.
- $\forall x, x > x - 1$ is read as: for all real numbers x, x is greater than $x - 1$. This statement is true.

The existential quantifier is used to make a claim that *there exists some real number* for which the statement holds. For example:

- $\exists x, x < 100$ is read as: there exists a real number x such that x is less than 100. This statement is true.
- $\exists x, x > x - 1$ is read as: there exists a real number x such that x is greater than $x - 1$. This statement is true.

Moreover, these quantifiers can be nested. For example:

- $\forall x, \exists y, x < y$ is read as: for all real numbers x, there exists a real number y such that x is less than y. This statement is true since for every x, there exists $y = x + 1$.
- $\exists y, \forall x, x < y$ is read as: there exists a real number y such that for all real numbers x, x is less than y. This statement is false because it claims that there is a maximum real number: a number y larger than every x.

Note that the order of variables and quantifiers is important for the meaning and veracity of a statement.

There are n variables $x_1, x_2, \ldots, x_n$, and you are given some formula of the form

$$f(x_1, \ldots, x_n) := (x_{j_1} < x_{k_1}) \wedge (x_{j_2} < x_{k_2}) \wedge \cdots \wedge (x_{j_m} < x_{k_m}),$$

where $\wedge$ denotes logical AND. That is, $f(x_1, \ldots, x_n)$ is true if every inequality $x_{j_i} < x_{k_i}$ holds. Otherwise, if at least one inequality does not hold, then $f(x_1, \ldots, x_n)$ is false.

Your task is to assign quantifiers $Q_1, \ldots, Q_n$ to either universal ($\forall$) or existential ($\exists$) so that the statement

$$Q_1 x_1, Q_2 x_2, \ldots, Q_n x_n, f(x_1, \ldots, x_n)$$

is true, and **the number of universal quantifiers is maximized**, or determine that the statement is false for every possible assignment of quantifiers.

Note that the order the variables appear in the statement is fixed. For example, if $f(x_1, x_2) := (x_1 < x_2)$ then you are not allowed to make x_2 appear first and use the statement $\forall x_2, \exists x_1, x_1 < x_2$. If you assign $Q_1 = \exists$ and $Q_2 = \forall$, it will **only** be interpreted as $\exists x_1, \forall x_2, x_1 < x_2$.

Input

The first line contains two integers n and m ($2 \le n \le 2 \cdot 10^5$; $1 \le m \le 2 \cdot 10^5$) — the number of variables and the number of inequalities in the formula, respectively.

The next m lines describe the formula. The i-th of these lines contains two integers j_i,k_i ($1 \le j_i, k_i \le n$, $j_i \ne k_i$).

Output

If there is no assignment of quantifiers for which the statement is true, output a single integer -1.

Otherwise, on the first line output an integer, the maximum possible number of universal quantifiers.

On the next line, output a string of length n, where the i-th character is "A" if Q_i should be a universal quantifier ($\forall$), or "E" if Q_i should be an existential quantifier ($\exists$). All letters should be upper-case. If there are multiple solutions where the number of universal quantifiers is maximum, print any.

Examples

input

```
2 1
1 2
```

output

```
1
AE
```

input

```
4 3
```

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

output

```
-1
```

input

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

output

```
2
AAE
```

Note

For the first test, the statement $\forall x_1, \exists x_2, x_1 < x_2$ is true. Answers of "EA" and "AA" give false statements. The answer "EE" gives a true statement, but the number of universal quantifiers in this string is less than in our answer.

For the second test, we can show that no assignment of quantifiers, for which the statement is true exists.

For the third test, the statement $\forall x_1, \forall x_2, \exists x_3, (x_1 < x_3) \wedge (x_2 < x_3)$ is true: We can set $x_3 = \max\{x_1, x_2\} + 1$.

F. Résumé Review

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Uh oh! Applications to tech companies are due soon, and you've been procrastinating by doing contests instead! (Let's pretend for now that it is actually possible to get a job in these uncertain times.)

You have completed many programming projects. In fact, there are exactly n types of programming projects, and you have completed a_i projects of type i. Your résumé has limited space, but you want to carefully choose them in such a way that maximizes your chances of getting hired.

You want to include several projects of the same type to emphasize your expertise, but you also don't want to include so many that the low-quality projects start slipping in. Specifically, you determine the following quantity to be a good indicator of your chances of getting hired:

$$f(b_1, \ldots, b_n) = \sum_{i=1}^{n} b_i(a_i - b_i^2).$$

Here, b_i denotes the number of projects of type i you include in your résumé. Of course, you cannot include more projects than you have completed, so you require $0 \le b_i \le a_i$ for all i.

Your résumé only has enough room for k projects, and you will absolutely not be hired if your résumé has empty space, so you require $\sum\limits_{i=1}^{n} b_i = k$.

Find values for $b_1, \ldots, b_n$ that maximize the value of $f(b_1, \ldots, b_n)$ while satisfying the above two constraints.

Input

The first line contains two integers n and k ($1 \le n \le 10^5$, $1 \le k \le \sum\limits_{i=1}^{n} a_i$) — the number of types of programming projects and the résumé size, respectively.

The next line contains n integers $a_1, \ldots, a_n$ ($1 \le a_i \le 10^9$) — a_i is equal to the number of completed projects of type i.

Output

In a single line, output n integers $b_1, \ldots, b_n$ that achieve the maximum value of $f(b_1, \ldots, b_n)$, while satisfying the requirements $0 \le b_i \le a_i$ and $\sum\limits_{i=1}^{n} b_i = k$. If there are multiple solutions, output any.

Note that you do not have to output the value $f(b_1, \ldots, b_n)$.

Examples

input

```
10 32
1 2 3 4 5 5 5 5 5 5
```

output

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

input

```
5 8
4 4 8 2 1
```

output

```
2 2 2 1 1
```

Note

For the first test, the optimal answer is $f = -269$. Note that a larger f value is possible if we ignored the constraint $\sum_{i=1}^{n} b_i = k$.

For the second test, the optimal answer is $f = 9$.