## Codeforces Round #639 (Div. 1)

# A. 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 $0 \leq k \bmod n \leq n - 1$. 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 $t$ ($1 \leq t \leq 10^4$) — the number of test cases. Next $2t$ lines contain descriptions of test cases.

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

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

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<br>1<br>14<br>2<br>1 -1<br>4<br>5 5 5 1<br>3<br>3 2 1<br>2<br>0 1<br>5<br>-239 -2 -100 -3 -11 |
| **output** |
| YES<br>YES<br>YES<br>NO<br>NO<br>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 $2$.

# B. 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 $m$ ($1 \leq n, m \leq 1000$) — 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<br>.#.<br>###<br>##. |
| **output** |
| 1 |

| input |
| --- |
| 4 2<br>##<br>.#<br>.#<br>## |
| **output** |
| -1 |

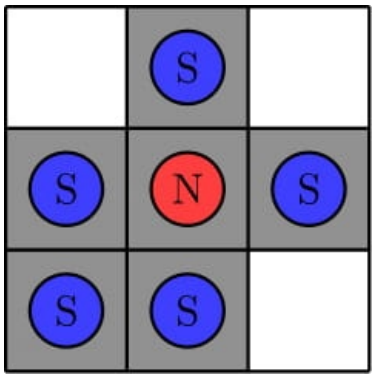
| input |
| --- |
| 4 5<br>....#<br>####.<br>.###.<br>.#... |
| **output** |
| 2 |

| input |
| --- |
| 2 1<br>.<br># |
| **output** |
| -1 |

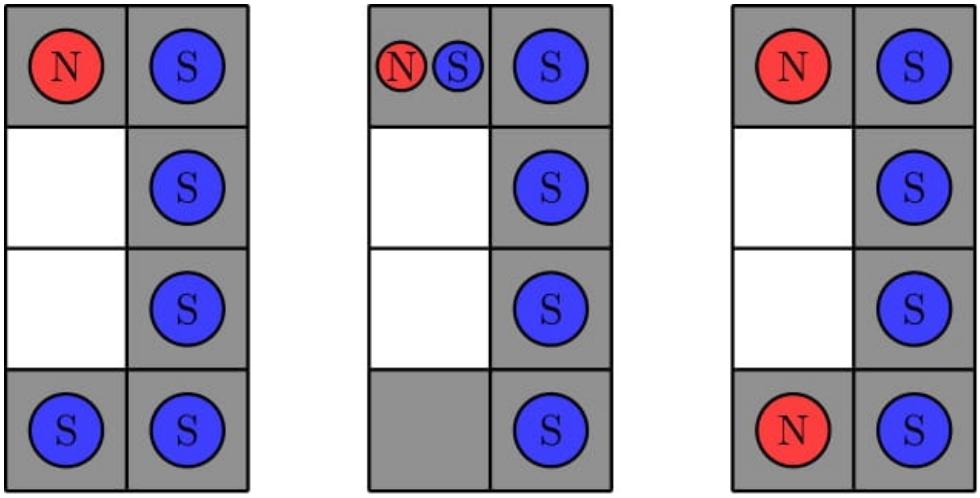
| input |
| --- |
| 3 5<br>.....<br>.....<br>..... |
| **output** |
| 0 |

### Note

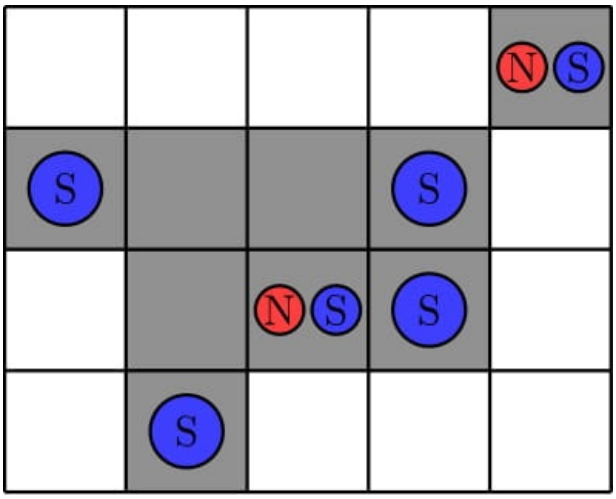
In the first test, here is an example placement of magnets:



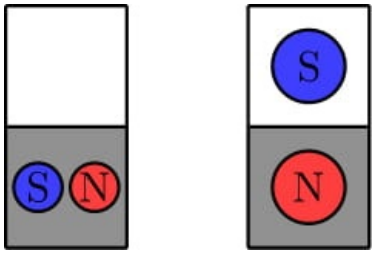
In the second test, we can show that no required placement of magnets exists. Here are three example placements that fail to meet the requirements. The first example violates rule $3$ since we can move the north magnet down onto a white square. The second example violates rule $2$ since we cannot move 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.

## C. 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<br>1 2 |
| **output** |
| 1<br>AE |

| |
|---|
| **input** |
| 4 3<br>1 2<br>2 3<br>3 1 |
| **output** |
| -1 |

| |
|---|
| **input** |
| 3 2<br>1 3<br>2 3 |
| **output** |
| 2<br>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 $x_3 = \max\{x_1, x_2\} + 1$.

## D. 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 $0 \leq b_i \leq a_i$ 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 $\sum_{i=1}^{n} b_i = k$.

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 $k$ ($1 \leq n \leq 10^5$, $1 \leq k \leq \sum_{i=1}^{n} a_i$) — the number of types of programming projects and the résumé size, respectively.

The next line contains $n$ integers $a_1, \ldots, a_n$ ($1 \leq a_i \leq 10^9$) — $a_i$ 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 $0 \leq b_i \leq a_i$ and $\sum_{i=1}^{n} b_i = k$. 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<br>1 2 3 4 5 5 5 5 5 |
| **output** |
| 1 2 3 3 3 4 4 4 4 4 |

| input |
| --- |
| 5 8<br>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 $\sum_{i=1}^{n} b_i = k$.

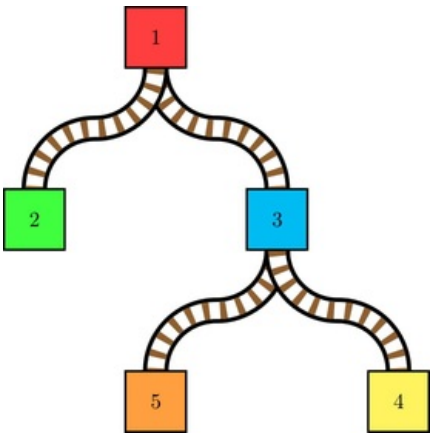
For the second test, the optimal answer is $f = 9$.

## E. Train Tracks

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

*That's right. I'm a Purdue student, and I shamelessly wrote a problem about trains.*

There are $n$ stations and $m$ trains. The stations are connected by $n - 1$ one-directional railroads that form a tree rooted at station $1$. All railroads are pointed in the direction from the root station $1$ to the leaves. A railroad connects a station $u$ to a station $v$, and has a distance $d$, meaning it takes $d$ time to travel from $u$ to $v$. Each station with at least one outgoing railroad has a switch that determines the child station an incoming train will be directed toward. For example, it might look like this:




Here, stations $1$ and $3$ have switches directed toward stations $2$ and $4$, respectively.

Initially, no trains are at any station. Train $i$ will enter station $1$ at time $t_i$. Every unit of time, starting at time $1$, the following two steps happen:

1. You can switch at most one station to point to a different child station. A switch change takes effect before step $2$.
2. For every train that is on a station $u$, it is directed toward the station $v$ indicated by $u$'s switch. So, if the railroad from $u$ to $v$ has distance $d$, the train will enter station $v$ in $d$ units of time from now.

Every train has a destination station $s_i$. When it enters $s_i$, it will stop there permanently. If at some point the train is going in the wrong direction, so that it will never be able to reach $s_i$ no matter where the switches point, it will immediately explode.

Find the **latest possible time of the first explosion** if you change switches optimally, or determine that you can direct every train to its destination so that no explosion occurs. Also, find the **minimum number of times you need to change a switch** to achieve this.

### Input

The first line contains two integers $n$ and $m$ ($1 \le n, m \le 10^5$) — the number of stations and trains, respectively.

The next $n - 1$ lines describe the railroads. The $i$-th line contains three integers $u, v, d$ ($1 \le u, v \le n$, $1 \le d \le 10^9$), denoting a railroad from station $u$ to station $v$ with distance $d$. It is guaranteed that the railroads form a tree rooted at station $1$. **The switch of a station $u$ is initially directed towards the last outgoing railroad from $u$ that appears in the input.**

The next $m$ lines describe the trains. The $i$-th line contains two integers $s_i, t_i$ ($1 \le s_i \le n$, $1 \le t_1 < t_2 < \cdots < t_m \le 10^9$) — the destination station and the time the $i$-th train enters station $1$, respectively.

### Output

Output two integers: the latest possible time of the first explosion (or $-1$ if it is possible to never have an explosion) and the minimum number of switch changes to achieve it.

**Examples**

| input |
| --- |
| 5 4<br>1 2 1<br>1 3 2<br>3 4 1<br>3 5 3<br>2 1<br>4 2<br>2 6<br>5 10 |
| **output** |
| -1 6 |

| input |
| --- |
| 5 4<br>1 2 1<br>1 3 2<br>3 4 1<br>3 5 3<br>5 1<br>4 2<br>4 3<br>2 4 |
| **output** |
| 4 0 |

| input |
| --- |
| 11 6<br>1 2 1<br>1 3 2<br>3 4 1<br>3 5 2<br>5 6 1<br>5 7 2<br>7 8 1<br>7 9 2<br>9 10 1<br>9 11 1<br>2 1<br>8 3<br>6 5<br>10 7<br>4 9<br>2 11 |
| **output** |
| 11 4 |

## Note

For the first test, here's an example timeline:

- At time $1$, train $1$ enters station $1$. We switch station $1$ to point to station $2$. Train $1$ is directed to station $2$.
- At time $2$, train $2$ enters station $1$, and train $1$ enters station $2$, where it stops permanently. We switch station $1$ to point to station $3$. Train $2$ is directed to station $3$.
- At time $4$, train $2$ enters station $3$. We switch station $3$ to point to station $4$. Train $2$ is directed to station $4$.
- At time $5$, train $2$ enters station $4$, where it stops permanently.
- At time $6$, train $3$ enters station $1$. We switch station $1$ to point to station $2$. Train $3$ is directed to station $2$.
- At time $7$, train $3$ enters station $2$, where it stops permanently. We switch station $3$ to point to station $5$.
- At time $10$, train $4$ enters station $1$. We switch station $1$ to point to station $3$. Train $4$ is directed to station $3$.
- At time $12$, train $4$ enters station $3$. Train $4$ is directed to station $5$.
- At time $15$, train $4$ enters station $5$, where it stops permanently.

For the second test, we switch nothing. At time $4$, train $2$ is directed to station $5$ and train $4$ is directed to station $3$. They both explode. It is impossible to prevent an explosion by time $4$.

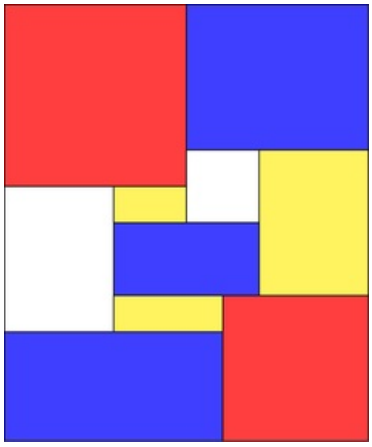
For the third test, denote a switch change by $(u \to v, t)$ if we make station $u$ point to station $v$ at time $t$. One solution is to make these $4$ switch changes: $(1 \to 2, 1)$,$(1 \to 3, 2)$,$(7 \to 8, 5)$,$(5 \to 6, 8)$. At time $11$, trains $4$,$5$, and $6$ explode. It is impossible to prevent an explosion by time $11$.

# F. Piet's Palette

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

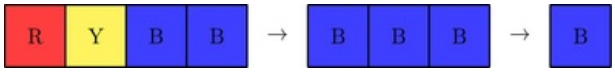
output: standard output

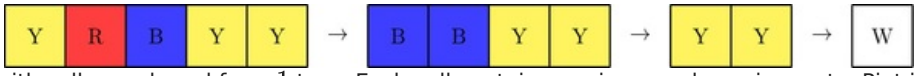
*Piet Mondrian is an artist most famous for his minimalist works, consisting only of the four colors red, yellow, blue, and white. Most people attribute this to his style, but the truth is that his paint behaves in a very strange way where mixing two primary colors only produces another primary color!*



*A lesser known piece, entitled "Pretentious rectangles"*

A sequence of primary colors (red, yellow, blue) is mixed as follows. While there are at least two colors, look at the first two. If they are distinct, replace them with the missing color. If they are the same, remove them from the sequence. In the end, if there is one color, that is the resulting color. Otherwise, if the sequence is empty, we say the resulting color is white. Here are two example mixings:




Piet has a color palette with cells numbered from $1$ to $n$. Each cell contains a primary color or is empty. Piet is very secretive, and will not share his palette with you, so you do not know what colors belong to each cell.

However, he did perform $k$ operations. There are four kinds of operations:

1. In a **mix** operation, Piet chooses a subset of cells and mixes their colors together in some order. The order is not necessarily by increasing indexes. He records the resulting color. Empty cells are skipped over, having no effect on the mixing process. The mixing does not change the color values stored in the cells.
2. In a **RY** operation, Piet chooses a subset of cells. Any red cells in this subset become yellow, and any yellow cells in this subset become red. Blue and empty cells remain unchanged.
3. In a **RB** operation, Piet chooses a subset of cells. Any red cells in this subset become blue, and any blue cells in this subset become red. Yellow and empty cells remain unchanged.
4. In a **YB** operation, Piet chooses a subset of cells. Any yellow cells in this subset become blue, and any blue cells in this subset become yellow. Red and empty cells remain unchanged.

Piet only tells you the list of operations he performs in chronological order, the indexes involved, and the resulting color of each mix operation. For each mix operation, you also know the order in which the cells are mixed. Given this information, determine the color of each cell in the initial palette. That is, you should find one possible state of the palette (before any operations were performed), or say that the described situation is impossible.

## Input

The first line contains two integers $n$ and $k$ ($1 \le n, k \le 1000$) — the number of cells in the palette and the number of operations, respectively.

The next $k$ lines describe the operations. The $i$-th line begins with the name of the $i$-th operation and an integer $m$ ($1 \le m \le n$) — the number of indexes involved. Then follow $m$ integers $j_1, \ldots, j_m$ ($1 \le j_i \le n$) — the indexes of the operation. It is guaranteed that all $j_i$ are distinct within an operation. If it is a mix operation, the indexes are listed in the order the colors are mixed, and the line also ends with a character representing the resulting color: "R" for red, "Y" for yellow, "B" for blue, and "W" for white.

## Output

Output "YES" if a solution exists, or "NO" otherwise. You can print each character in any case (upper or lower).

If the answer is "YES", on the next line output a string of length $n$, consisting of characters "R", "Y", "B", and "`.`", representing the paint colors in the $n$ cells of the initial palette (red, yellow, blue, and empty, respectively). If there are multiple solutions, print any. You can print each character in any case (upper or lower).

**Examples**

| input |
| --- |
| 3 2<br>mix 2 2 1 R<br>mix 2 1 3 Y |
| **output** |
| YES |

YB.

| input |
| --- |
| 2 3<br>mix 1 2 Y<br>RB 1 2<br>mix 1 2 W |
| **output** |
| NO |

| input |
| --- |
| 1 3<br>RY 1 1<br>YB 1 1<br>mix 1 1 B |
| **output** |
| YES<br>R |

| input |
| --- |
| 3 8<br>mix 2 1 2 R<br>mix 2 1 3 Y<br>RY 2 2 3<br>RB 3 1 2 3<br>YB 3 1 2 3<br>mix 1 1 W<br>mix 1 2 B<br>mix 1 3 Y |
| **output** |
| YES<br>.RY |

## Note

For the first test case, the answer "YB." is consistent with both mixings. The first mixing "BY" results in red, while the second mixing "Y" results in yellow (the empty cell is ignored). Other valid solutions include "BYR" and ".RY".

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

For the third test case, the answer "R" is consistent with all operations. In the first two operations it changes to "Y", then "B". In the final operation, the mixing "B" results in blue.

For the fourth test case, the answer ".RY" is consistent with all operations. The first two mixings are "R" and "Y", resulting in red and yellow, respectively. During the next three operations, the palette changes to ".YR", then ".YB", then ".BY". The final three mixings agree with this palette.