Codeforces Global Round 6

A. Competitive Programmer

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

Bob is a competitive programmer. He wants to become red, and for that he needs a strict training regime. He went to the annual meeting of grandmasters and asked of them how much effort they needed to reach red.

"Oh, I just spent **hours** solving problems", said the -th of them.

Bob wants to train his math skills, so for each answer he wrote down the number of **minutes** ( ), thanked the grandmasters and went home. Bob could write numbers with leading zeroes — for example, if some grandmaster answered that he had spent hours, Bob could write instead of .

Alice wanted to tease Bob and so she took the numbers Bob wrote down, and for each of them she did one of the following independently:

- rearranged its digits, or
- wrote a random number.

This way, Alice generated numbers, denoted , ..., .

For each of the numbers, help Bob determine whether can be a permutation of a number divisible by (possibly with leading zeroes).

Input

The first line contains a single integer ( ) — the number of grandmasters Bob asked.

Then lines follow, the -th of which contains a single integer — the number that Alice wrote down.

Each of these numbers has between and digits '0' through '9'. They can contain leading zeroes.

Output

Output lines.

For each , output the following. If it is possible to rearrange the digits of such that the resulting number is divisible by , output "red" (quotes for clarity). Otherwise, output "cyan".

Example

input

```
6
603
006
205
228
1053
0000000000000000000000000000000000000000000000
```

output

```
red
red
cyan
cyan
cyan
red
```

Note

In the first example, there is one rearrangement that yields a number divisible by , and that is .

In the second example, there are two solutions. One is and the second is .

In the third example, there are possible rearrangments: , , , , , . None of these numbers is divisible by .

In the fourth example, there are rearrangements: , , .

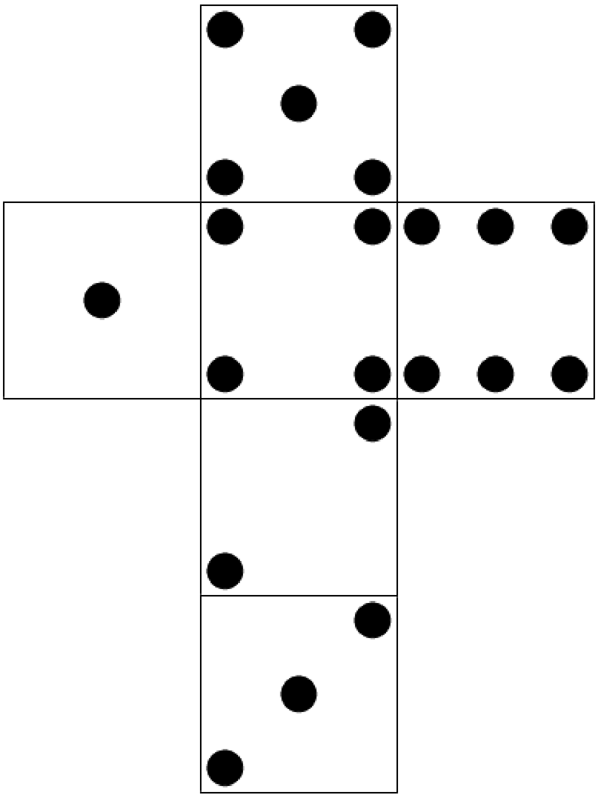
In the fifth example, none of the rearrangements result in a number divisible by .

In the sixth example, note that is a valid solution.

B. Dice Tower

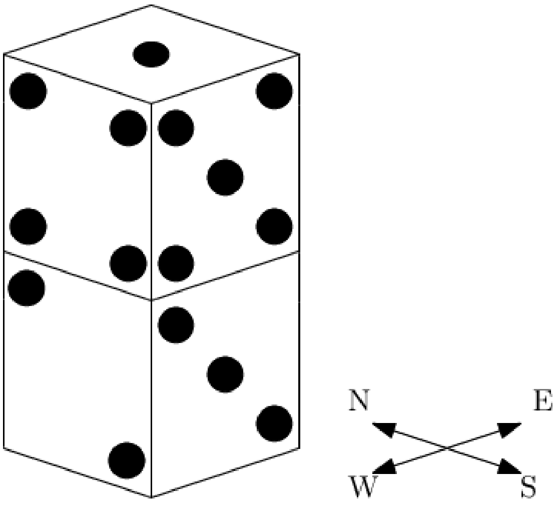
time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Bob is playing with -sided dice. A net of such standard cube is shown below.



He has an unlimited supply of these dice and wants to build a tower by stacking multiple dice on top of each other, while choosing the orientation of each dice. Then he counts the number of visible pips on the faces of the dice.

For example, the number of visible pips on the tower below is — the number visible on the top is , from the south and , from the west and , from the north and and from the east and .



The one at the bottom and the two sixes by which the dice are touching are not visible, so they are not counted towards total.

Bob also has favourite integers , and for every such integer his goal is to build such a tower that the number of visible pips is exactly . For each of Bob's favourite integers determine whether it is possible to build a tower that has exactly that many visible pips.

Input

The first line contains a single integer ( ) — the number of favourite integers of Bob.

The second line contains space-separated integers ( ) — Bob's favourite integers.

Output

For each of Bob's favourite integers, output "YES" if it is possible to build the tower, or "NO" otherwise (quotes for clarity).

Example

input

```
4
29 34 19 38
```

output

```
YES
YES
YES
NO
```

Note

The first example is mentioned in the problem statement.

In the second example, one can build the tower by flipping the top dice from the previous tower.

In the third example, one can use a single die that has on top.

The fourth example is impossible.

C. Diverse Matrix

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

Let be a matrix of size containing positive integers, not necessarily distinct. Rows of the matrix are numbered from to , columns are numbered from to . We can construct an array consisting of integers as follows: for each , let be the greatest common divisor of integers in the -th row, and for each let be the greatest common divisor of integers in the -th column.

We call the matrix **diverse** if all numbers ( ) are pairwise distinct.

The **magnitude** of a matrix equals to the maximum of .

For example, suppose we have the following matrix:

We construct the array :

1. is the greatest common divisor of , , and , that is ;
2. is the greatest common divisor of , , and , that is ;
3. is the greatest common divisor of and , that is ;
4. is the greatest common divisor of and , that is ;
5. is the greatest common divisor of and , that is .

So . All values in this array are distinct, so the matrix is diverse. The magnitude is equal to .

For a given and , find a diverse matrix that minimises the magnitude. If there are multiple solutions, you may output any of them. If there are no solutions, output a single integer .

Input

The only line in the input contains two space separated integers and ( ) — the number of rows and the number of columns of the matrix to be found.

Output

If there is no solution, output a single integer .

Otherwise, output rows. The -th of them should contain space-separated integers, the -th of which is — the positive integer in the -th row and -th column of a diverse matrix minimizing the magnitude.

Furthermore, it must hold that . It can be shown that if a solution exists, there is also a solution with this additional constraint (still having minimum possible magnitude).

Examples

input

```
2 2
```

output

```
4 12
2 9
```

input

```
1 1
```

output
0

Note

In the first example, the GCDs of rows are and , and the GCDs of columns are and . All GCDs are pairwise distinct and the maximum of them is . Since the GCDs have to be distinct and at least , it is clear that there are no diverse matrices of size with magnitude smaller than .

In the second example, no matter what is, will always hold, so there are no diverse matrices.

D. Decreasing Debts

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

There are people in this world, conveniently numbered through . They are using burles to buy goods and services. Occasionally, a person might not have enough currency to buy what he wants or needs, so he borrows money from someone else, with the idea that he will repay the loan later with interest. Let denote the debt of towards , or if there is no such debt.

Sometimes, this becomes very complex, as the person lending money can run into financial troubles before his debtor is able to repay his debt, and finds himself in the need of borrowing money.

When this process runs for a long enough time, it might happen that there are so many debts that they can be consolidated. There are two ways this can be done:

1. Let and such that or . We can decrease the and by and increase and by , where .
2. Let . We can set to .

The total debt is defined as the sum of all debts:

Your goal is to use the above rules in any order any number of times, to make the total debt as small as possible. Note that you don't have to minimise the **number** of non-zero debts, only the **total debt**.

Input

The first line contains two space separated integers ( ) and ( ), representing the number of people and the number of debts, respectively.

lines follow, each of which contains three space separated integers , ( ), ( ), meaning that the person borrowed burles from person .

Output

On the first line print an integer ( ), representing the number of debts after the consolidation. It can be shown that an answer always exists with this additional constraint.

After that print lines, -th of which contains three space separated integers , meaning that the person owes the person exactly burles. The output must satisfy , and .

For each pair , it should hold that or . In other words, each pair of people can be included at most once in the output.

Examples

input
3 2 1 2 10 2 3 5
output
2 1 2 5 1 3 5

input
3 3 1 2 10 2 3 15 3 1 10
output
1

```
2 3 5
```

input

```
4 2
1 2 12
3 4 8
```

output

```
2
1 2 12
3 4 8
```

input

```
3 4
2 3 1
2 3 2
2 3 4
2 3 8
```

output

```
1
2 3 15
```

Note

In the first example the optimal sequence of operations can be the following:

1. Perform an operation of the first type with , , , and . The resulting debts are: , , , all other debts are ;
2. Perform an operation of the second type with . The resulting debts are: , , all other debts are .

In the second example the optimal sequence of operations can be the following:

1. Perform an operation of the first type with , , , and . The resulting debts are: , , , all other debts are ;
2. Perform an operation of the first type with , , , and . The resulting debts are: , , , , all other debts are ;
3. Perform an operation of the second type with . The resulting debts are: , , , all other debts are ;
4. Perform an operation of the second type with . The resulting debts are: , , all other debts are ;
5. Perform an operation of the second type with . The resulting debts are: , all other debts are .

E. Spaceship Solitaire

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

Bob is playing a game of Spaceship Solitaire. The goal of this game is to build a spaceship. In order to do this, he first needs to accumulate enough resources for the construction. There are types of resources, numbered through . Bob needs at least pieces of the -th resource to build the spaceship. The number is called **the goal for resource** .

Each resource takes turn to produce and in each turn only one resource can be produced. However, there are certain milestones that speed up production. Every milestone is a triple , meaning that as soon as Bob has units of the resource , he receives one unit of the resource for free, without him needing to spend a turn. It is possible that getting this free resource allows Bob to claim reward for another milestone. This way, he can obtain a large number of resources in a single turn.

The game is constructed in such a way that there are never two milestones that have the same **and** , that is, the award for reaching units of resource is at most one additional resource.

A bonus is never awarded for of any resource, neither for reaching the goal nor for going past the goal — formally, for every milestone .

A bonus for reaching certain amount of a resource can be the resource itself, that is, .

Initially there are no milestones. You are to process updates, each of which adds, removes or modifies a milestone. After every update, output the minimum number of turns needed to finish the game, that is, to accumulate at least of -th resource for each .

Input

The first line contains a single integer ( ) — the number of types of resources.

The second line contains space separated integers ( ), the -th of which is the goal for the -th resource.

The third line contains a single integer ( ) — the number of updates to the game milestones.

Then lines follow, the -th of which contains three space separated integers , , (, ,). For each triple, perform the following actions:

- First, if there is already a milestone for obtaining units of resource , it is removed.
- If , no new milestone is added.
- If , add the following milestone: "For reaching units of resource , gain one free piece of ."
- Output the minimum number of turns needed to win the game.

Output

Output lines, each consisting of a single integer, the -th represents the answer after the -th update.

Example

input

```
2
2 3
5
2 1 1
2 2 1
1 1 1
2 1 2
2 2 0
```

output

```
4
3
3
2
3
```

Note

After the first update, the optimal strategy is as follows. First produce once, which gives a free resource . Then, produce twice and once, for a total of four turns.

After the second update, the optimal strategy is to produce three times — the first two times a single unit of resource is also granted.

After the third update, the game is won as follows.

- First produce once. This gives a free unit of . This gives additional bonus of resource . After the first turn, the number of resources is thus .
- Next, produce resource again, which gives another unit of .
- After this, produce one more unit of .

The final count of resources is , and three turns are needed to reach this situation. Notice that we have more of resource than its goal, which is of no use.

F. Almost Same Distance

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

Let be a simple graph. Let be a non-empty subset of vertices. Then is **almost- -uniform** if for each pair of distinct vertices the distance between and is either or .

You are given a tree on vertices. For each between and , find the maximum size of an almost- -uniform set.

Input

The first line contains a single integer ( ) – the number of vertices of the tree.

Then lines follows, the -th of which consisting of two space separated integers , ( ) meaning that there is an edge between vertices and .

It is guaranteed that the given graph is tree.

Output

Output a single line containing space separated integers , where is the maximum size of an almost- -uniform set.

Examples

input

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

output

```
4 3 2 1 1
```

input

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

output

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

Note

Consider the first example.

- The only maximum almost- -uniform set is .
- One of the maximum almost- -uniform sets is or , another one is .
- A maximum almost- -uniform set is any pair of vertices on distance .
- Any single vertex is an almost- -uniform set for .

In the second sample there is an almost- -uniform set of size , and that is .

G. Permutation Concatenation

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

Let be an integer. Consider all permutations on integers to in lexicographic order, and concatenate them into one big sequence . For example, if , then . The length of this sequence is .

Let be a pair of indices. We call the sequence a **subarray** of .

You are given . Find the number of distinct subarrays of . Since this number may be large, output it modulo (a prime number).

Input

The only line contains one integer ( ), as described in the problem statement.

Output

Output a single integer — the number of distinct subarrays, modulo .

Examples

input

```
2
```

output

```
8
```

input

```
10
```

output

```
19210869
```

Note

In the first example, the sequence . It has eight distinct subarrays: , , , , , , and .

H. Red-Blue Graph

time limit per test: 4 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

There is a directed graph on vertices numbered through where each vertex (except) has two outgoing arcs, red and blue. At any point in time, exactly one of the arcs is *active* for each vertex. Initially, all blue arcs are active and there is a token located at vertex . In one second, the vertex with token first switches its active arcs — the inactive arc becomes active and vice versa. Then, the token is moved along the active arc. When the token reaches the vertex , it stops. It is guaranteed that is reachable via arcs from every vertex.

You are given queries. Each query contains a state of the graph — a pair of the following form:

- is the vertex where the token is currently located;
- is a string consisting of characters. The -th character corresponds to the color of the active edge leading from the -th vertex (the character is 'R' if red arc is active, otherwise the character is 'B').

For each query, determine whether the given state is reachable from the initial state and the first time this configuration appears. Note that the two operations (change active arc and traverse it) are atomic — a state is not considered reached if it appears after changing the active arc but before traversing it.

Input

The first line contains a single integer ( ) — the number of vertices.

lines follow, -th of contains two space separated integers and ( ) representing a blue arc and red arc , respectively. It is guaranteed that vertex is reachable from every vertex.

The next line contains a single integer ( ) — the number of queries.

Then lines with queries follow. The -th of these lines contains an integer ( ) and a string of length consiting only of characters 'R' and 'B'. The -th of these characters is 'R' if the red arc going from is active and 'B' otherwise.

Output

Output lines, each containing answer to a single query.

If the state in the -th query is unreachable, output the integer . Otherwise, output — the first time when the state appears (measured in seconds, starting from the initial state of the graph which appears in time).

Example

input

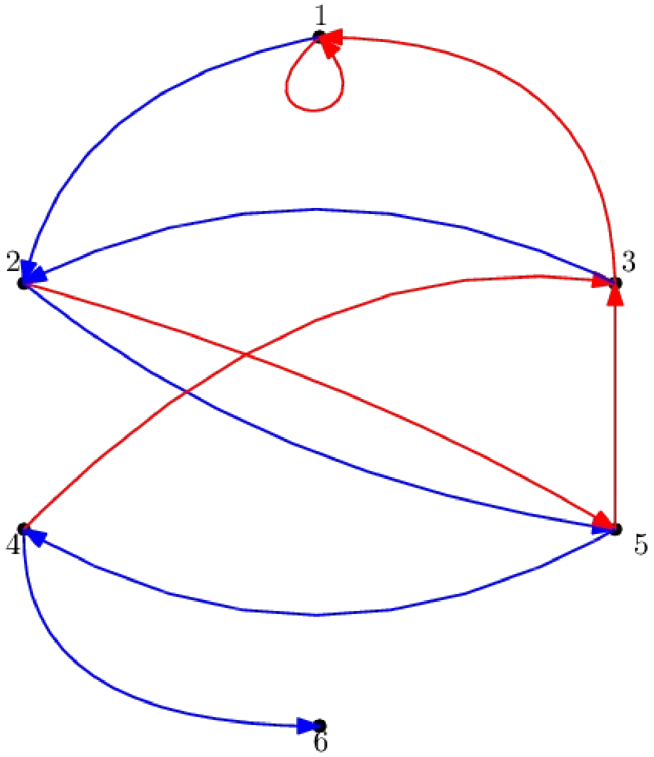
```
6
2 1
5 5
2 1
6 3
4 3
21
1 BBBBB
1 RBBBB
2 BBBBB
5 BRBBB
3 BRBBR
1 BRRBR
1 RRRBR
2 BRRBR
5 BBRBR
4 BBRBB
3 BBRRB
2 BBBRB
5 BRBRB
3 BRBRR
1 BRRRR
1 RRRRR
2 BRRRR
5 BBRRR
4 BBRRB
2 BRBBB
4 BRBBR
```

output

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
-1
-1
```

Note

The graph in the first example is depticed in the figure below.



The first queries denote the journey of the token. On the -th move the token would reach the vertex . The last two queries show states that are unreachable.