Educational Codeforces Round 34 (Rated for Div. 2)

A. Hungry Student Problem

1 second, 256 megabytes

Ivan's classes at the university have just finished, and now he wants to go to the local CFK cafe and eat some fried chicken.

CFK sells chicken chunks in small and large portions. A small portion contains 3 chunks; a large one — 7 chunks. Ivan wants to eat exactly x chunks. Now he wonders whether he can buy exactly this amount of chicken.

Formally, Ivan wants to know if he can choose two non-negative integers a and b in such a way that a small portions and b large ones contain exactly x chunks.

Help Ivan to answer this question for several values of x!

Input

The first line contains one integer n $(1 \le n \le 100)$ — the number of testcases.

The i-th of the following n lines contains one integer x_i $(1 \le x_i \le 100)$ — the number of chicken chunks Ivan wants to eat.

Output

Print n lines, in i-th line output YES if Ivan can buy exactly x_i chunks. Otherwise, print NO.

input

```
2
6
5
```

output

```
YES
NO
```

In the first example Ivan can buy two small portions.

In the second example Ivan cannot buy exactly 5 chunks, since one small portion is not enough, but two small portions or one large is too much.

B. The Modcrab

1 second, 256 megabytes

Vova is again playing some computer game, now an RPG. In the game Vova's character received a quest: to slay the fearsome monster called Modcrab.

After two hours of playing the game Vova has tracked the monster and analyzed its tactics. The Modcrab has h_2 health points and an attack power of a_2. Knowing that, Vova has decided to buy a lot of strong healing potions and to prepare for battle.

Vova's character has h_1 health points and an attack power of a_1. Also he has a large supply of healing potions, each of which increases his current amount of health points by c_1 when Vova drinks a potion. All potions are identical to each other. It is guaranteed that $c_1 > a_2$.

The battle consists of multiple phases. In the beginning of each phase, Vova can either attack the monster (thus reducing its health by a_1) or drink a healing potion (it increases Vova's health by c_1; **Vova's health can exceed h_1**). Then, **if the battle is not over yet**, the Modcrab attacks Vova, reducing his health by a_2. The battle ends when Vova's (or Modcrab's) health drops to 0 or lower. It is possible that the battle ends in a middle of a phase after Vova's attack.

Of course, Vova wants to win the fight. But also he wants to do it as fast as possible. So he wants to make up a strategy that will allow him to win the fight after the minimum possible number of phases.

Help Vova to make up a strategy! You may assume that Vova never runs out of healing potions, and that he can always win.

Input

The first line contains three integers h_1, a_1, c_1 $(1 \le h_1, a_1 \le 100, 2 \le c_1 \le 100)$ — Vova's health, Vova's attack power and the healing power of a potion.

The second line contains two integers h_2, a_2 $(1 \le h_2 \le 100, 1 \le a_2 < c_1)$ — the Modcrab's health and his attack power.

Output

In the first line print one integer n denoting the minimum number of phases required to win the battle.

Then print n lines. i-th line must be equal to HEAL if Vova drinks a potion in i-th phase, or STRIKE if he attacks the Modcrab.

The strategy must be valid: Vova's character must not be defeated before slaying the Modcrab, and the monster's health must be 0 or lower after Vova's last action.

If there are multiple optimal solutions, print any of them.

input

```
10 6 100
17 5
```

output

```
4
STRIKE
HEAL
STRIKE
STRIKE
```

input

```
11 6 100
12 5
```

output

```
2
STRIKE
STRIKE
```

In the first example Vova's character must heal before or after his first attack. Otherwise his health will drop to zero in 2 phases while he needs 3 strikes to win.

In the second example no healing needed, two strikes are enough to get monster to zero health and win with 6 health left.

C. Boxes Packing

1 second, 256 megabytes

Mishka has got n empty boxes. For every i ($1 \le i \le n$), i-th box is a cube with side length a_i.

Mishka can put a box i into another box j if the following conditions are met:

- i-th box is not put into another box;
- j-th box doesn't contain any other boxes;
- box i is smaller than box j ($a_i < a_j$).

Mishka can put boxes into each other an arbitrary number of times. He wants to minimize the number of *visible* boxes. A box is called *visible* iff it is not put into some another box.

Help Mishka to determine the minimum possible number of *visible* boxes!

Input

The first line contains one integer n ($1 \le n \le 5000$) — the number of boxes Mishka has got.

The second line contains n integers a_1, a_2, ..., a_n ($1 \le a_i \le 10^9$), where a_i is the side length of i-th box.

Output

Print the minimum possible number of *visible* boxes.

input

```
3
1 2 3
```

output

```
1
```

input

```
4
4 2 4 3
```

output

```
2
```

In the first example it is possible to put box 1 into box 2, and 2 into 3.

In the second example Mishka can put box 2 into box 3, and box 4 into box 1.

D. Almost Difference

2 seconds, 256 megabytes

Let's denote a function

$$d(x, y) = \begin{cases} y - x, & \text{if } |x - y| > 1 \\ 0, & \text{if } |x - y| \le 1 \end{cases}$$

You are given an array a consisting of n integers. You have to calculate the sum of $d(a_i, a_j)$ over all pairs (i, j) such that $1 \le i \le j \le n$.

Input

The first line contains one integer n ($1 \le n \le 200000$) — the number of elements in a.

The second line contains n integers a_1, a_2, ..., a_n ($1 \le a_i \le 10^9$) — elements of the array.

Output

Print one integer — the sum of $d(a_i, a_j)$ over all pairs (i, j) such that $1 \le i \le j \le n$.

input

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

output

```
4
```

input

```
4
6 6 5 5
```

output

```
0
```

input

```
4
6 6 4 4
```

output

```
-8
```

In the first example:

1. $d(a_1, a_2) = 0$;
2. $d(a_1, a_3) = 2$;
3. $d(a_1, a_4) = 0$;
4. $d(a_1, a_5) = 2$;
5. $d(a_2, a_3) = 0$;
6. $d(a_2, a_4) = 0$;
7. $d(a_2, a_5) = 0$;

8. $d(a_3, a_4) = -2$;
9. $d(a_3, a_5) = 0$;
10. $d(a_4, a_5) = 2$.

E. Swapping Characters

1 second, 256 megabytes

We had a string s consisting of n lowercase Latin letters. We made k copies of this string, thus obtaining k identical strings $s_1, s_2, \ldots, s_k$. After that, in each of these strings we swapped exactly two characters (the characters we swapped could be identical, but they had different indices in the string).

You are given k strings $s_1, s_2, \ldots, s_k$, and you have to restore any string s so that it is possible to obtain these strings by performing aforementioned operations. Note that the total length of the strings you are given doesn't exceed 5000 (that is, $k \cdot n \le 5000$).

Input

The first line contains two integers k and n $(1 \le k \le 2500, 2 \le n \le 5000, k \cdot n \le 5000)$ — the number of strings we obtained, and the length of each of these strings.

Next k lines contain the strings $s_1, s_2, \ldots, s_k$, each consisting of exactly n lowercase Latin letters.

Output

Print **any** suitable string s, or `-1` if such string doesn't exist.

input
```
3 4
abac
caab
acba
```
output
```
acab
```

input
```
3 4
kbbu
kbub
ubkb
```
output
```
kbub
```

input
```
5 4
abcd
dcba
acbd
dbca
zzzz
```
output
```
-1
```

In the first example s_1 is obtained by swapping the second and the fourth character in `acab`, s_2 is obtained by swapping the first and the second character, and to get s_3, we swap the third and the fourth character.

In the second example s_1 is obtained by swapping the third and the fourth character in `kbub`, s_2 — by swapping the second and the fourth, and s_3 — by swapping the first and the third.

In the third example it's impossible to obtain given strings by aforementioned operations.

F. Clear The Matrix

1 second, 256 megabytes

You are given a matrix f with 4 rows and n columns. Each element of the matrix is either an asterisk (`*`) or a dot (`.`).

You may perform the following operation arbitrary number of times: choose a square submatrix of f with size $k \times k$ (where $1 \le k \le 4$) and replace each element of the chosen submatrix with a dot. Choosing a submatrix of size $k \times k$ costs a_k coins.

What is the minimum number of coins you have to pay to replace all asterisks with dots?

Input

The first line contains one integer n $(4 \le n \le 1000)$ — the number of columns in f.

The second line contains 4 integers a_1, a_2, a_3, a_4 $(1 \le a_i \le 1000)$ — the cost to replace the square submatrix of size 1×1, 2×2, 3×3 or 4×4, respectively.

Then four lines follow, each containing n characters and denoting a row of matrix f. Each character is either a dot or an asterisk.

Output

Print one integer — the minimum number of coins to replace all asterisks with dots.

input
```
4
1 10 8 20
***.
***.
***.
...*
```
output
```
9
```

input
```
7
2 1 8 2
.***...
.***..*
.***...
....*..
```

output

```
3
```

input

```
4
10 10 1 10
***.
*..*
*..*
.***
```

output

```
2
```

In the first example you can spend 8 coins to replace the submatrix 3×3 in the top-left corner, and 1 coin to replace the 1×1 submatrix in the bottom-right corner.

In the second example the best option is to replace the 4×4 submatrix containing columns $2-5$, and the 2×2 submatrix consisting of rows $2-3$ and columns $6-7$.

In the third example you can select submatrix 3×3 in the top-left corner and then submatrix 3×3 consisting of rows $2-4$ and columns $2-4$.

G. Yet Another Maxflow Problem

4 seconds, 256 megabytes

In this problem you will have to deal with a very special network.

The network consists of two parts: part A and part B. Each part consists of n vertices; i-th vertex of part A is denoted as A_i, and i-th vertex of part B is denoted as B_i.

For each index i ($1 \le i < n$) there is a directed edge from vertex A_i to vertex A_{i+1}, and from B_i to B_{i+1}, respectively. Capacities of these edges are given in the input. Also there might be several directed edges going from part A to part B (but never from B to A).

You have to calculate the maximum flow value from A_1 to B_n in this network. Capacities of edges connecting A_i to A_{i+1} might sometimes change, and you also have to maintain the maximum flow value after these changes. Apart from that, the network is fixed (there are no changes in part B, no changes of edges going from A to B, and no edge insertions or deletions).

Take a look at the example and the notes to understand the structure of the network better.

Input

The first line contains three integer numbers n, m and q ($2 \le n, m \le 2 \cdot 10^5$, $0 \le q \le 2 \cdot 10^5$) — the number of vertices in each part, the number of edges going from A to B and the number of changes, respectively.

Then $n - 1$ lines follow, i-th line contains two integers x_i and y_i denoting that the edge from A_i to A_{i+1} has capacity x_i and the edge from B_i to B_{i+1} has capacity y_i ($1 \le x_i, y_i \le 10^9$).

Then m lines follow, describing the edges from A to B. Each line contains three integers x, y and z denoting an edge from A_x to B_y with capacity z ($1 \le x, y \le n$, $1 \le z \le 10^9$). There might be multiple edges from A_x to B_y.

And then q lines follow, describing a sequence of changes to the network. i-th line contains two integers v_i and w_i, denoting that the capacity of the edge from A_{v_i} to A_{v_i+1} is set to w_i ($1 \le v_i < n$, $1 \le w_i \le 10^9$).

Output

Firstly, print the maximum flow value in the original network. Then print q integers, i-th of them must be equal to the maximum flow value after i-th change.

input

```
4 3 2
1 2
3 4
5 6
2 2 7
1 4 8
4 3 9
1 100
2 100
```

output

```
9
14
14
```

This is the original network in the example: