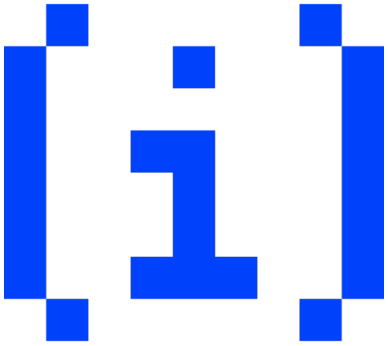


en[i]gma 0x02 Advanced

December 15, 2024

Problems

Problem	Time Limit	Memory Limit	Points
Now You Think with Portals	1 sec	64MB	100
The Good Tablecloth	1 sec	64MB	100
NIM	1 sec	512MB	100
Surprise Party	1 sec	64MB	100
Total			400

Now You Think with Portals

We are in a skyscraper and want to reach the T-th floor. We start from the S-th floor and can use 2 methods to change floors:

- via **ladders**: each ladder moves us from the current floor to either the next floor up or the next floor down.
- via **portals**: each floor provides (up to) 2 portals. Each portal can take us to a higher floor or a lower floor.
  But which floors? **We don't know!** The portals are pre-defined, but unknown to us in advance.
  The only thing we know is that:
  - if we use the `up` portal from the i-th floor, no subsequent portal will take us to the i-th floor or any floor below it, and
  - if we use the `down` portal, no subsequent portal will take us to the i-th floor or any floor above it. Therefore, a floor might have fewer than 2 portals.

  For example, if floor 1 has a portal to floor 2 (`up` portal), then floor 2 cannot have a `down` portal anywhere.

You need to find the best way to get from floor S to floor T.

The ideal solution avoids using ladders altogether!

Input (`STDIN`) - Output (`STDOUT`)

This problem is interactive, meaning you give instructions to the judge, and it responds with information for you to proceed.

1. The first line contains 3 integers F, S, and T: the number of floors, the starting floor, and the floor where the exit is. You are initially on floor S.
2. As long as you are not on floor T:
   - You print your move: `LADDER UP`, `LADDER DOWN`, `PORTAL UP`, `PORTAL DOWN`
   - The judge prints the floor number x you land on, in the form `OK x` and
   - If you requested a `PORTAL` move on a floor that doesn't have the corresponding portal, the judge will print `SAME`.
3. Once you reach floor T, the game ends and the judge computes your score.

Your score for this game is: $100\% - \frac{\text{number of ladder moves}}{\text{total moves}}$

Example

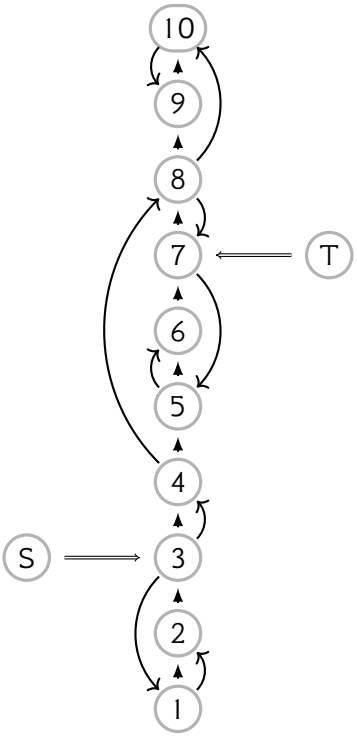
Input (STDIN)

```
10 3 7
```

One possible sequence of moves could be:

Program Output (STDOUT)	Judge Output (STDIN)
PORTAL DOWN	
	OK 1
LADDER UP	
	OK 2
LADDER UP	
	OK 3
PORTAL UP	
	OK 4
PORTAL UP	
	OK 8
LADDER DOWN	
	7

This solution would score $\frac{73}{7} = 57\%$.



Note: To ensure the interactive grader works properly, you must flush the lines you print. This is done as follows:

- In python: `print()` does this automatically

```
...
print("LADDER UP")
...
```

- In C/C++ (`cstdio`): use `printf` as usual

```
...
printf("LADDER UP");
...
```

- In C++ (`iostream`): use `endl`

```
...
cout << "LADDER UP" << endl;
...
```

The Good Tablecloth

Whenever guests come over, Mom always brings out the good dishes and the good tablecloth. But tomorrow, she has an unexpected visit, and the tablecloth is in the laundry! So, she asks Toto for help to color an old tablecloth, hoping to save the day.

She is very strict about what makes a good tablecloth:

- It has only 2 colors: red and green.
- It has horizontal and vertical lines dividing it into rectangular sections (parts).
- To make the lines visible to the eye, the parts that share an edge must be colored differently.

She hands the tablecloth to Toto and specifies the lines she wants to see. Totos needs to color it appropriately.

Input Data (`STDIN`)

The first line of input provides 2 positive integer numbers:

- N, the length, and
- , the width of the tablecloth, separated by a space (`' '`).

The next line starts with a positive integer h, the number of horizontal lines, followed by a new line containing h positive integers h_i, separated by spaces (`' '`), where h_i is the distance of the i–th line from the top edge of the tablecloth.

This is followed by a new line that starts with a positive integer v, the number of vertical lines, followed by a new line containing v positive integers v_i, separated by spaces (`' '`), where v_i is the distance of the i–th line from the leftmost edge of the tablecloth.

It holds that $K = h + v$.

Output Data (`STDOUT`)

The program will output N lines, each containing M characters, which are either `G` or `R`, such that the resulting output forms the tablecloth after Toto's intervention.

Note: If there are more than 1 possible answers, print any you prefer.

Examples

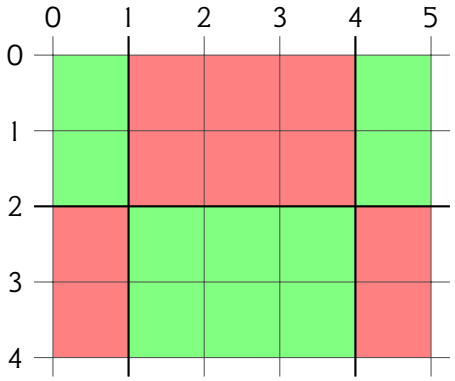
1st

Input (STDIN)

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

Output (STDOUT)

```
GRRRG
GRRRG
RGGGR
RGGGR
```



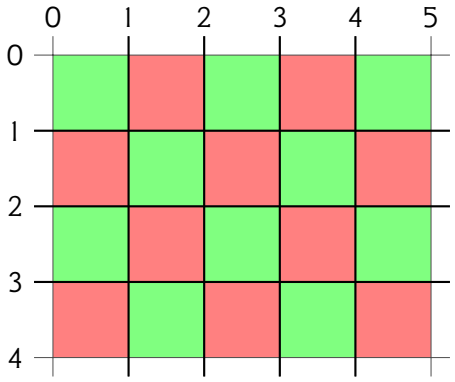
2nd

Input (STDIN)

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

Output (STDOUT)

```
GRGRG
RGRGR
GRGRG
RGRGR
```



Subtasks

- 70% of the points: $1 \leq N, M, K \leq 500$

- 100% of the points: $1 \leq N, M \leq 5,000, \quad 1 \leq K \leq 10^6$

  The same line-column may appear multiple times: if it appears an odd number of times, it should appear on the tablecloth; otherwise, it should not.

NIM

Annoula and Toto are playing the classic game of Nim. The game is played as follows:

> *«Starting with a certain number of marbles on the table, each player must remove 1, 2, or 3 marbles on their turn.*
> *The winner is the one who takes the last marble.*
> *Annoula always plays first.»*

Since both players play optimally, they already know that the outcome of the game is predetermined based on the initial number of marbles. As a result, they got bored with these rules and are now experimenting with different choices for the number of marbles that can be removed.

Help Annoula figure out which starting numbers of marbles guarantee that, if she plays optimally, she will win!

Important! If the remaining marbles are fewer than one of the available choices, Annoula and Totos cannot make that move.

Input Data (STDIN)

The 1st line contains a positive integer M $(M \leq 10)$, the number of choices the players have for removing marbles.
The 2nd line line contains M integers, the choices $m_1, m_2, \ldots, m_M$ $(m_i \leq 100)$ in ascending order.
The 3d line contains a positive integer Q, the number of queries for the initial number of marbles.
This is followed by Q $(Q \leq 20)$ lines, each containing a number q_i, the initial number of marbles.

Output Data (STDOUT)

The output consists of Q lines, each with a single number, `0` or `1`, depending on whether Annoula can **definitely** win with the corresponding initial number of marbles.

Examples

1st

Input (STDIN)

```
3
1 2 3
5
2
4
5
8
10
```

Output (STDOUT)

```
1
0
1
0
1
```

Example Explanation:

The players have 3 choices:
They can remove 1, 2, or 3 marbles per turn.
Additionally, there are 5 queries:

- $q_1 = 2$: Annoula can play the second move and win the game directly.
- $q_2 = 4$: No matter what move Annoula makes, Totos can win.
  - If Annoula removes 1 marble, Totos can take the remaining 3.
  - If she removes 2, Totos can remove the other 2.
  - If she removes 3, Totos takes the last 1.
- $q_3 = 5$: Annoula can only remove 1 marble and leave Toto with 4, a situation in which Totos cannot win (as explained above). Hence, Annoula wins.
- $q_4 = 8$: Similarly, Annoula cannot win starting from 8 marbles.
- , $q = 10$: Following the same logic, Annoula can win starting from 10 marbles.

2nd

Input (STDIN)

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

Output (STDOUT)

```
0
1
0
```

Subtasks

- 20% of the points: the m_i values are the numbers $\{1, 2, \ldots, M\}$, $q_i \leq 100000$
- 30% of the points: $q_i \leq 25$
- 50% of the points: $q_i \leq 10000$

Surprise Party

Totos and Annoula's friends are impressed by their hard work and decide to relax them by organizing a surprise party. They start writing a list of items needed to make the party as fun as possible, along with a value indicating the importance of each item for the party, i.e., the cost incurred if that item is missing from the party.

The K friends decide to divide the N items from the list among themselves in sequence, so that each friend is responsible for bringing a continuous segment of items. However, recognizing that there's always a chance someone might forget to bring an item-and the more items someone is responsible for, the more likely they are to forget something-the friends want to ensure that the total estimated cost of potential missing items is minimized.
More formally: You are given positive integers:

- N, the number of items.
- K, the number of friends.

You are also given a sequence A consisting of N positive integers $A_1, A_2, \ldots, A_N$, which represent the cost of missing each item.

We need to divide the sequence A into K segments such that the sum of the estimated costs of each segment is minimized. The estimated cost of a segment of A, consisting of the (consecutive) positions $\{i, i+1, \ldots, j\}$ where $i \leq j$, is defined as:

$$cost(i, j) = \sum_{w=i}^{j} A_w \cdot (j - i + 1)$$

That is, we aim to find the minimum value of the expression:

$$\text{cost}(1, i_1 - 1) + \text{cost}(i_1, i_2 - 1) + \cdots + \text{cost}(i_{K-1}, N)$$

where $1 < i_1 < i_2 < \cdots < i_{K-1} \leq N$.

Input Data (STDIN)

The 1st line contains two integers N and K, separated by a space (`' '`): he number of items and the number of friends, respectively.
The 2nd line contains N integers $A_1, \ldots, A_N$: the cost of missing each item.

Output Data (STDOUT)

The output consists of a single line with one integer: the minimum total estimated cost.

Example

Input (STDIN)

```
6 3
11 11 11 24 26 100
```

Output (STDOUT)

```
299
```

Example Explanation:

The first friend is responsible for the segment with the first three items, with an estimated cost of $11 \cdot 3 + 11 \cdot 3 + 11 \cdot 3 = 99$,The second friend is responsible for the next two items, with an estimated cost of $24 \cdot 2 + 26 \cdot 2 = 100$. The third friend is responsible for the last item, with an estimated cost of $100 \cdot 1 = 100$. Thus, the total estimated cost is $99 + 100 + 100 = 299$, which is the minimum.

Constraints:

- $1 \leq K \leq N \leq 2000$
- $1 \leq A_1 \leq 10^9$

Subtasks

- 20% of the points: $K = 2$.
- 40% of the points: $K \leq N \leq 250$
- 20% of the points: $A_1 \leq A_2 \leq \cdots \leq A_N$, i.e., the elements of A are sorted in ascending order.
- 20% of the points: no additional constraints.