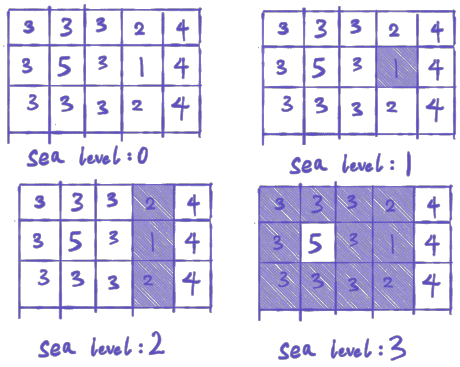
Polar Bear

Deep in the sea of *Floating Islands* Polar Bear lives on the *Grid Island*. The *Grid Island* has a rectangular shape and is divided into an n x m grid. Area of each grid area has a certain height associated with it. As the consequence of ocean warming, the water level in the sea of *Floating Islands* started increasing. Polar Bear did not think much of it since, after all, his island is floating, but it turned out that due to relative density of his island to the density of water, some of the grid areas started flooding. When the sea level reaches level i, all the grid areas with height of i or less are flooded. Adjacent unflooded grid areas (sharing a common edge) form islands within the *Grid Island*.

The depiction of the *Grid Islands* as the sea levels change is shown on the right.

Numbers in each grid area denote its height. Flooded fields are shaded. There will be

- 1 island when the sea level is 0,
- 1 island when the sea level is 1,
- 2 islands when the sea level is 2,
- 2 islands when the sea level is 3,
- 1 island when the sea level is 4 (not shown in the image),
- and no islands when the sea level is 5 (not shown in the image).



Your task is to find the number of islands for each sea level so that Polar Bear knows how the *Grid Island* is changing.

Input

The first line contains two numbers n and m separated by a single space, the dimensions of the *Grid Island*, where $1 \leq n, m \leq 1000$.

Next n lines contain m integers from the range $[1, 10^9]$ separated by single spaces, denoting the elevations of the respective grid areas.

Next line contains an integer T, $1 \leq T \leq 10^5$, the number of sea levels for which you need to count the number of islands.

The last line contains T integers t_i, separated by single spaces, such that $0 \leq t_1 \leq t_2 \leq ... \leq t_{T-1} \leq t_T \leq 10^9$. The i-th integer denotes the sea level of the i-th query.

Output

Output a single line consisting of T numbers c_i separated by single spaces, where c_i is the number of islands when the sea level is equal to t_i. The output ends with a newline.

Example

```
Input:
3 5
3 3 3 2 4
3 5 3 1 4
3 3 3 2 4
5
1 2 3 4 5

Output:
1 2 2 1 0
```

Giant Squad

Conflict Empire is fighting a war against Light Kingdom. The commander of Conflict Army is building a squad of giants. Giants are huge creatures, typically a dozen feet tall, inhabiting the territory of Conflict Empire. Giants are so tall that their height differences are also very large, preventing orders from being effectively transmitted. To address this issue, the commander decides to form squads with only odd number of giants with different heights and select the captain in such a way so that the number of giants in the squad who are shorter than that captain is equal to the number of giants who are taller than that captain. You are given heights of all giants in that squad and you are asked to determine the height of the captain.

Input

The input consists of a single line, starting with an integer N ($1 < N < 11$, N is odd), the number of giants in that squad, followed by N integers representing heights of the giants of the squad. Those N integers will be between 11 and 20 (inclusive). Additionally, heights will be given in strictly increasing or decreasing order.

Output

You should print a single integer x, where x is the height of the captain.

Example 1

```
Input:
5 19 17 16 14 12

Output:
16
```

Example 2

```
Input:
5 12 14 16 17 18

Output:
16
```

Line Up

Queuing, it's what the British are renowned for doing - and doing very well. Better than anyone else in the world, if reputation is to be believed. Today, queues can be found just about anywhere (even in the virtual world).

Standard *Queues* are well known to most computer science students. It's a first-in-first-out data structure. Kobayashi is already familiar with it. But she is assigned to write some code to maintain a variant of Standard Queues: **Group Queues**.

In a group queue each element belongs to a group. If an element is pushed to the queue and there is no element that belongs to the same group in the queue, it will be placed behind the last element in the queue. Otherwise, if there are already some elements that belong to the same group, it should be placed immediately behind them.

Dequeuing works the same as int the standard queue: We will remove the first element according to the current order from head to tail in the queue.

Kanna is preparing for her school's Sports Festival, and she wants Kobayashi to come to this important event. But Kobayashi will not be able to attend unless she can finish her program early. As Kobayashi's colleague and close friend, you decide to help her. You need to write a program that simulates such a group queue.

Input

The 1st line contains an integer `n` (`1 <= n <= 1,000`) equal to the number of groups, as described above.

Then `n` group descriptions follow. Each group is described on one line by an integer `k` (the number of elements belonging to the group) followed by `k` integers (the indexes of the elements in this group).

Elements are indexed with integer in the range `[ 0 , 999,999 ]`. And a group may consist of up to `1,000` elements.

Finally, a list of commands follows. There are 3 different kinds of commands:

- `Push ind` : Push the element with index `ind` into the group queue.
- `Pop` : Output the first element and remove it from the queue.
- `Shutdown` : Stop your program (end of input).

There may be up to `200,000` commands in the input file, so the implementation of the group queue should be efficient: both enqueuing and dequeuing of an element should only take a constant time.

Output

For each `Pop` command, print an integer, the index of the element which is dequeued, followed by a newline.

Example 1

```
Input:
2
3 1 2 3
3 4 5 6
Push 1
Push 4
Push 2
Push 5
Push 3
Push 6
Pop
Pop
Pop
Pop
Pop
```

```
Pop
Shutdown

Output:
1
2
3
4
5
6
```

Example 2

```
Input:
2
3 1 2 3
3 4 5 6
Push 1
Push 2
Push 4
Pop
Pop
Pop
Push 3
Push 5
Pop
Pop
Push 6
Pop
Pop
Shutdown

Output:
1
2
4
5
3
6
```

APS Homework

Daru is worried about a very time-consuming APS homework. As a super hacker, he develops an AI called Amadeus to help him solve his homework problems automatically.

There are N problems for this week. It takes a_i seconds for Amadeus to solve the i-th problem. Since Daru is a super hacker, he has broken into Gradescope to figure out how long it takes Gradescope to evaluate each submission. He knows that it takes b_i seconds for Gradescope autograder to evaluate and accept the correct solution for the i-th problem. (Daru is not interested in cheating by also downloading all the test cases from Gradescope since it's not cool and he knows that this would violate the academic integrity policies).

Amadeus can not work on multiple problems simultaneously. It will follow an order given by Daru to solve problems. Once Amadeus solves a problem, it submits the solution to Gradescope and continues to the next problem.

It takes no time to submit a solution and Gradescope is able to evaluate multiple solutions at the same time. The solution for the i-th problem will be accepted in exactly b_i seconds after it is submitted to Gradescope. Amadeus always produces a correct solution, so Gradescope always accepts after the first submission.

Daru wants to find an order of problems to minimize the time necessary for all problems to be accepted on Gradescope. Can you help him to figure out the optimal order?

Input

The first line contains one integer N. N is the number of problems in homework, 1 <= N <= 1000.

The i-th line of the following N lines contains a pair of integers a_i and b_i, 1 <= a_i, b_i <= 10000, as described above.

Output

Output one number followed by a newline. The total number of seconds that it takes Amadeus to solve, submit and get accept verdict for all the problems when the order of the problems is optimal. (Note, that there may be multiple orders that result in the same time. Since you only report on the time, the actual order is irrelevant.) The output ends with a newline.

Example 1

```
Input:
3
2 1
2 5
3 2

Output:
8
```

The optimal order is to do the 2nd problem first, then solve the 3rd problem and leave the 1st problem to solve at the end.

The solution for the 1st problem will be accepted at time 5 + 2 + 1 = 8 (Amadeus starts working on this problem at time 5). The solution for the 2nd problem will be accepted at time 0 + 2 + 5 = 7. The evaluation for the 3rd problem will be accepted at time 2 + 3 + 2 = 7,

Hence the total number of seconds consumed from start to end is 8.

Game Enthusiast

Kou loves playing video games and wants to share her games with her classmates. She has a USB stick but it is too small to hold all of her games. Kou decided to copy only some of her games to that USB stick but struggled to determine which games to select and turned to you for help.

You are given the size of N games and the capacity of the USB stick. You need to find subset of games to store on the memory stick such that the leftover unused space is minimum.

Input

The first line of the input contains two integers V (1 <= V <= 10,000) and N (1 <= N <= 20), representing the capacity of the USB stick and the number of games.

The second line contains N integers, indicating the size of Kou's games. All of these integers are between 1 and 10,000.

Output

Print one line containing a single integer W, the total space occupied by the games copied onto the USB stick such that the USB stick has the least space unused.

Example 1

```
Input:
5 3
1 3 4

Output:
5
```

Example 2

```
Input:
10 4
9 8 4 2

Output:
10
```

Example 3

```
Input:
20 4
10 5 7 4

Output:
19
```