A Matter of Size (kabbalah)

George has recently enrolled in a *Kabbalah* seminar, where a few cultured savants debate on the concealed wisdom of the Bible. In particular, their favorite hobby is searching the Bible for long sequences of equal letters.

Firstly, the seminar master selects an $N \times M$ rectangle of characters $C_{i,j}$ from the holy scriptures.

Afterwards, the adepts search in all directions (horizontal, vertical and diagonal) for contiguous sequences of equal letters and the longer the sequence found, the greater the glory for the finder.

Figure 1: A valuable sample of ancient Hebrew text.

Help George prove his wisdom by finding the longest sequence of equal letters!

> ☞ Among the attachments of this task you may find a template file `kabbalah.*` with a sample incomplete implementation.

Input

The first line contains two integers N and M.

Other N lines follow, each containing a string consisting of M lowercase letters from `'a'` to `'z'` (translated from the Hebrew alphabet).

Output

You need to write a single line with an integer: the longest sequence of equal letters present in the bible fragment.

Constraints

- $1 \leq N, M \leq 1000$.
- $C_{i,j}$ is a lowercase letter from `'a'` to `'z'` for each $i < N$, $j < M$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**25 points**]: $N = 1$.
- **Subtask 3** [**40 points**]: $N, M \leq 10$.
- **Subtask 4** [**30 points**]: No additional limitations.

Examples

input.txt	output.txt
4 6 uieiki zqnitz chquth keinoi	2
7 5 fjqhr aqnrb baqah zanqi oahqq rovna zgpao	4

Explanation

In the **first sample case**, the longest sequence consists of 2 vertical letters `'i'`.

In the **second sample case**, the longest sequence consists of 4 diagonal letters `'q'`.

The Floor is Lava (lava)

Whenever William steps on a floor made with tiles, he plays the "Hot Lava" game. In this game the floor is lava, that is, some of the tiles are to be absolutely avoided.

Figure 1: https://xkcd.com/735/

This particular floor is a $H \times W$ grid made of tiles. Some of those tiles are red, the remaining ones are white. William wants to get from tile $(0, 0)$ to tile $(H - 1, W - 1)$ using only white tiles, as quickly as possible. He can move *diagonally* from one tile to an adjacent one (by vertex). Moreover, he can move *vertically* or *horizontally* from one tile to one that is adjacent by edge, **or to the next tile in the same direction**, by doing a *long jump*.

Help William compute the quickest route, as the number of jumps (short or long jumps) needed.

> ☞ Among the attachments of this task you may find a template file `lava.*` with a sample incomplete implementation.

Input

The first line contains two integers H and W, the height and the width of the grid, respectively.

Other H lines follow, each containing a string of W characters (either `'.'` or `'#'`).

Output

You need to write a single line with an integer: the minimum number of jumps needed.

Constraints

- $1 \leq H, W \leq 1000$.
- The $(0, 0)$ and $(H - 1, W - 1)$ positions in the grid will always contain a `'.'` and never a `'#'`.
- There will always be some route from $(0, 0)$ to $(H - 1, W - 1)$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**10 points**]: $\min(H, W) = 1$. That is: the grid is a line.
- **Subtask 3** [**20 points**]: There are no `'#'` in the grid.
- **Subtask 4** [**25 points**]: $H, W \leq 10$.
- **Subtask 5** [**30 points**]: No additional limitations.

Examples

input.txt	output.txt
3 4 .### .... ###.	3
5 5 .##.. .##.. #.#.. #.##. .##..	5

Explanation

In the **first sample case** William could follow the path denoted by `'o'`s:

```
o###
.oo.
###o
```

Another possible solution:

```
o###
.o.o
###o
```

In the **second sample case**, there is a unique best solution:

```
o##..
o##..
#o#o.
#.##o
.##.o
```

Server Farm (plugs)

William wants to exploit his new independent house for setting up a brand-new server farm, and *start earning money immediately*! Since his budget is pretty limited, he started this ambitious project by collecting obsolete PCs from the recycling bin for electronic waste. He gathered this way N fully functional computers with diverse capabilities. In particular, the i-th computer has computing power of F_i nanoFLOPS and is powered via a plug of type T_i, which can be either:

- $T_i =$ `'L10'`, if the plug conforms to the *Type L/10amp* standard;
- $T_i =$ `'L16'`, if the plug conforms to the *Type L/16amp* standard.

He also found a very long power strip consisting of M sockets, where the i-th of them can be of three different types S_i:

- $S_i =$ `'L10'`, if the socket is designed for *Type L/10amp* plugs;
- $S_i =$ `'L16'`, if the socket is designed for *Type L/16amp* plugs;
- $S_i =$ `'bipasso'` if the socket is designed to accommodate both (otherwise incompatible) types.

Figure 1: L/16, L/10 and bipasso sockets (respectively).

Help William find the assignment of plugs into compatible sockets which allows for the highest total amount of nanoFLOPS!

> ☞ Among the attachments of this task you will find a template `plugs.*` with a sample incomplete implementation.

Input

The first line contains two integers N, M. The second line contains N integers F_i. The third line contains N space-separated plug types T_i. The fourth line contains M space-separated socket types S_i.

Output

You need to write a single line with an integer: the maximum amount of nanoFLOPS that William can produce by connecting their PCs into compatible plugs.

Constraints

- $1 \leq N, M \leq 20\,000$.
- $1 \leq F_i \leq 20\,000$ for each $i = 0 \ldots N-1$.
- T_i is either `'L10'` or `'L16'` for each $i = 0 \ldots N-1$.
- S_i is either `'L10'` or `'L16'` or `'bipasso'` for each $i = 0 \ldots M-1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**10 points**]: F_i is equal for all PCs.
- **Subtask 3** [**25 points**]: There are no bipasso sockets.
- **Subtask 4** [**30 points**]: $N, M \leq 50$.
- **Subtask 5** [**30 points**]: No additional limitations.

Examples

input.txt	output.txt
7 5 120 310 570 250 740 460 880 L10 L16 L10 L10 L16 L16 L16 bipasso L10 L16 L10 bipasso	2900
10 7 690 950 300 470 520 380 750 990 300 440 L10 L16 L10 L10 L16 L10 L10 L10 L16 L10 L16 L16 bipasso bipasso L16 L10 L16	4200

Explanation

In the **first sample case**, William can arrange in the power socket the PCs corresponding to the following amount of nanoFLOPS (in order):

$$460 + 570 + 880 + 250 + 740$$

In the **second sample case**, a possible arrangement is the following:

$$520 + (\text{empty}) + 990 + 690 + 300 + 750 + 950$$

Independent House (prefab)

William is exhausted of its annoying roommates and their never-ending parties. But after some months of money saving, he has finally accomplished to move into an independent house!

Since his budget is quite limited, he opted for a prefabricated building consisting of two blocks:

- a *ground block*, with the entrance and a living space, of size $A \times B$ centimeters;
- a *roof block* with a bedroom, to be put on top of the other.

The roof block is usually supposed to have the same size of the ground block. However, William has been cheated by an unprofessional ebay vendor which shipped a roof block of size $X \times Y$ centimeters to him. Now William has no choice but to make the best out of a bad situation, and use the provided blocks in some way. In particular, due to their peculiar joints[1] the blocks need to be arranged so that:

- the roof block is above the ground block;
- the roof block has a compatible orientation with the ground block, that is, you can rotate it only by multiples of 90 degrees with respect to the other;
- they are translated only by an integer amount of centimeters.

William is worried about the robustness of the building, which could be impaired because part of the roof block's floor could not lay on the ground block. In particular, for a certain disposition of the blocks, the *risk* can be calculated in this way:

- consider the region of the floor of the roof block *not laying on the ground block*;
- this region might consist of one or more connected parts: then, consider the *largest one*;
- the area of this connected part equals to the *risk* associated to this particular disposition.

Find the disposition giving the minimum risk!

> ☞ Among the attachments of this task you may find a template `prefab.*` with a sample incomplete implementation.

Input

The first and only line contains four integers A, B, X, Y.

Output

You need to write a single line with an integer: the minimum risk attainable with the given blocks.

[1] The joints are put at any point with integer coordinates (in centimeters) within the connecting face.

Constraints

- $1 \leq A, B, X, Y \leq 10\,000$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

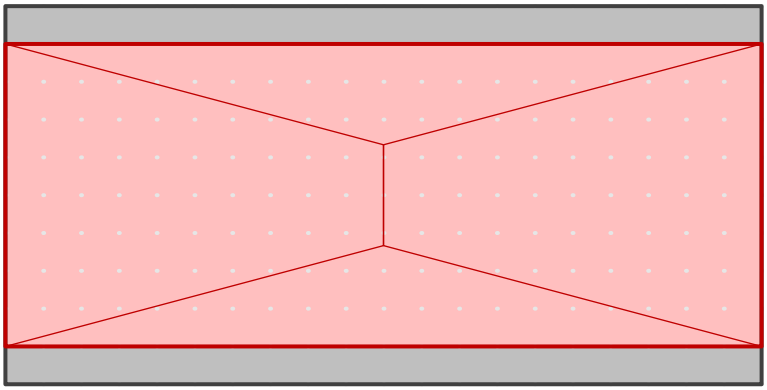
- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**25 points**]: $A = B$, $X = Y$.
- **Subtask 3** [**30 points**]: $A = B$.
- **Subtask 4** [**40 points**]: No additional limitations.

Examples

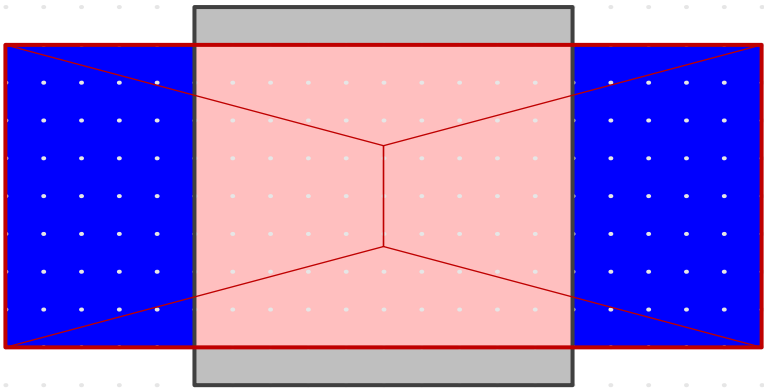
input.txt	output.txt
10 20 20 8	0
10 10 20 8	40
12 14 18 15	102

Explanation

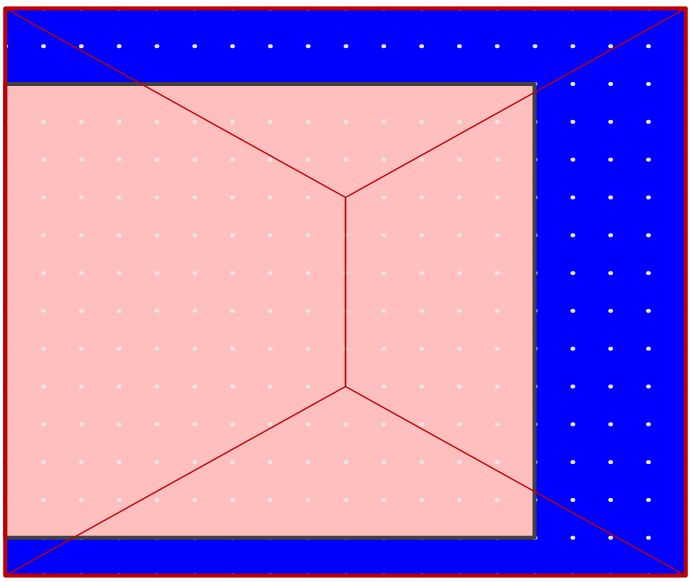
In the **first sample case**, we avoid any risk with the following configuration:



In the **second sample case**, we minimize the risk with the following configuration, where both risky regions (in blue) have area 40:



In the **third sample case**, a possible configuration minimizing the risk is the following, where the single risky region is highlighted in blue:



Reserved Seats (seats)

When commuting by train, Italian people tend to just sit in the first empty seat they find instead of spending time looking for their reserved seat. Specifically: when an Italian person boards a train, he/she will start looking for an empty seat (from the very first seat in the train) and they will stop only:

- When an empty seat is encountered: in this case, that person will occupy the seat. Or...
- When the actual seat, reserved to that person, is encountered: in this case, if the seat is occupied, the person will kindly ask the offending traveler to move. The offending traveler will apologize, get up, and go to their reserved seat (*not* just some empty seat, this time).

Note that, in the second case presented, a *chain reaction* of movings could fire up (A asks B to move, then B asks C to move, and so on).

Figure 1: A *Frecciarossa* train in the *Milano Centrale* station.

Train seats are numbered from 0 to $N-1$. Every person who boards the train will start looking for an empty seat from the 0-th seat (the "head" of the train), and then walk up to the $(N-1)$-th seat (the "tail" of the train).

Giorgio loves travelling by train. Today, sitting in his reserved seat, he has seen a lot of travelers boarding/leaving, moving around the train, asking other travelers to get up. Since Giorgio hates when his reserved place is occupied, he wanted to measure how bad the Italian situation really is and started wondering: how many people, in total, had to get up and apologize to other travelers?

You have the list of "events" that Giorgio witnessed (from the train departure to its arrival), which can be either "a person, for which the k-th seat was reserved, *boards* the train" or "a person, for which the k-th seat was reserved, *leaves* the train". Compute how many times a person had to get up and give their seat to another passenger.

☞ Among the attachments of this task you may find a template file `seats.*` with a sample incomplete implementation.

Input

The first line contains two integers N and Q, the number of seats in the train and the number of events, respectively.

Other Q lines follow, each containing a character T_i representing the type of event (`'b'` for "boards the train", `'l'` for "leaves the train"), and an integer K_i representing the seat *reserved to the person* in this event (not necessarily the seat they actually sat on).

Output

You need to write a single line with an integer: how many times someone had to apologize and get up.

Constraints

- $1 \leq N, Q \leq 100\,000$.
- T_i is either '`b`' or '`l`' for each $i = 0 \ldots Q-1$.
- $0 \leq K_i \leq N-1$ for each $i = 0 \ldots Q-1$.
- The events are consistent (e.g. the same person won't board the train twice, and it won't happen that a non-existent person leaves the train).
- After a person leaves the train, a new person with the same reserved seat could board the train.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**25 points**]: $N, Q \leq 10$.
- **Subtask 3** [**40 points**]: $N, Q \leq 1000$.
- **Subtask 4** [**30 points**]: No additional limitations.

Examples

input.txt	output.txt
3 3 b 1 b 2 b 0	2
4 6 b 1 b 2 b 0 l 0 b 3 b 0	3

Explanation

In the **first sample case** three people board the train in the following manner:

```
A >>   ___ ==> A__
B >>   A__ ==> AB_
C >>   AB_ ==> CB_  (A)
A ->   CB_ ==> CA_  (B)
B ->   CA_ ==> CAB
```

In the **second sample case**, five people board the the train and one leaves it:

```
A >>   ____ ==> A___
B >>   A___ ==> AB__
C ->   AB__ ==> CB__  (A)
A ->   CB__ ==> CA__  (B)
B ->   CA__ ==> CAB_
       CAB_ ==> _AB_   ~> C
D >>   _AB_ ==> DAB_
E ->   DAB_ ==> EAB_  (D)
D ->   EAB_ ==> EABD
```

Antique Street Lights (`streetlights`)

George is very passionate about the *good old times*, so much that he switched the illumination in his street to old-fashioned oil lanterns.

The street is now nicely illuminated by a sequence of N equally spaced lanterns, providing joy and delight to the whole neighbourhood.

However, he got the permission for doing so only by accepting to personally maintain the lanterns lit up: in fact, he has to ensure that given any subsequence of consecutive M lanterns, *at least* K *of them* are lit up at any given time.

Tonight an heavy storm is coming down on George's neighbourhood, and some particularly strong wind gusts just extinguished part of the lanterns. George can see for each lantern i whether it is lit ($L_i = 1$) or not ($L_i = 0$).

Since he doesn't want to risk his concession to be revoked, it is time to get out in the storm and turn on some lights!

Figure 1: An artistic view of George's street.

Help George not to freeze in the storm, and find the minimum amount of lanterns which need to be lit up in order to comply with the city regulations!

☞ Among the attachments of this task you may find a template file `streetlights.*` with a sample incomplete implementation.

Input

The first line contains three integers N, M, K.

The second line contains N integers L_i, such that $L_i = 0$ if the corresponding lantern is off and $L_i = 1$ if it is on.

Output

You need to write a single line with an integer: the minimum number of lanterns which need to be lit up.

Constraints

- $1 \leq K \leq M \leq N \leq 100\,000$.
- $0 \leq L_i \leq 1$ for each $i = 0 \ldots N - 1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**10 points**]: $K = M$.
- **Subtask 3** [**10 points**]: $M = N$.
- **Subtask 4** [**20 points**]: $N \leq 20$.
- **Subtask 5** [**25 points**]: $N \leq 1000$.
- **Subtask 6** [**30 points**]: No additional limitations.

Examples

input.txt	output.txt
5 2 1 0 1 0 0 1	1
12 4 2 0 0 0 0 1 0 1 0 0 0 1 0	3

Explanation

In the **first sample case**, it is sufficient to lit up the lantern with $i = 2$ obtaining the following configuration:

0 1 1 0 1

In the **second sample case**, it is sufficient to lit up the lanterns with $i = 2, 3, 8$ obtaining the following configuration:

0 0 1 1 1 0 1 0 1 0 1 0

Wheel of Fortune (`wheel`)

William is playing the Wheel of Fortune, a famous game show. The wheel is a "sliced" circle, with numbers on each slice. It can be seen as a circular array of $2N$ elements.

Figure 1: A valid wheel having $N = 3$.

The rules are as follows. William can "turn" the wheel by rotating the array, i.e. moving each of its elements ahead by some positions (possibly making several complete turns). For example, if the array is `2 1 5 3`, a rotation of 1 would yield the following array: `3 2 1 5`. Rotating again by 2 would yield `1 5 3 2`. Rotating again by 1 would yield the original array.

In order to win the game, William needs some information about the wheel everytime he turns it. Specifically: he needs to know the **min** among the first N elements and the **max** among the last N elements. Initially, in the previous example, the min and max values are 1 and 5. After the first rotation they become 2 and 5, then they become 1 and 3, and finally they become 1 and 5 again when the original ordering is restored.

Note: the "turns" add up, that is, the wheel is *not* reset after each turn.

☞ Among the attachments of this task you may find a template file `wheel.*` with a sample incomplete implementation.

Input

The first line contains a single integer N: half of the length of the wheel. The second line contains $2N$ integers W_i. The third line contains a single integer Q: the number of turns. The fourth line contains Q space-separated integers K_i, the list of "turns" William will do.

Output

You need to write Q lines, each of them corresponding to a turn and containing two integers:

- The **min** of the first half of the wheel after performing the turn.
- The **max** of the second half of the wheel after performing the turn.

Constraints

- $1 \leq N, Q \leq 40\,000$.
- $1 \leq W_i \leq 40\,000$ for each $i = 0 \ldots N-1$.
- $1 \leq K_i \leq 40\,000$ for each $i = 0 \ldots Q-1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**25 points**]: $N, Q \leq 10$.
- **Subtask 3** [**40 points**]: $N, Q \leq 100$.
- **Subtask 4** [**30 points**]: No additional limitations.

Examples

input.txt	output.txt
2 2 1 5 3 3 1 2 1	2 5 1 3 1 5
3 2 9 8 3 5 5 3 2 1 8	2 9 3 9 3 5

Explanation

The **first sample case** is the same example presented earlier.

In the **second sample case** the wheel 2 9 8 3 5 5 gets turned by 2 and becomes 5 5 2 9 8 3, then 1 and becomes 3 5 5 2 9 8, then 8 and becomes 9 8 3 5 5 2.