TASK	DIGITALNA	D'HONDT	POGODAK	ROBOT	PLAĆE	TRAKA

source code	`digitalna.pas`
	`digitalna.c`
	`digitalna.cpp`
		`dhondt.pas`
		`dhondt.c`
		`dhondt.cpp`
			`pogodak.pas`
			`pogodak.c`
			`pogodak.cpp`
				`robot.pas`
				`robot.c`
				`robot.cpp`
					`place.pas`
					`place.c`
					`place.cpp`
						`traka.pas`
						`traka.c`
						`traka.cpp`

input	standard input (*stdin*)					
output	standard output (*stdout*)					
time limit	1 second	1 second	1 second	1 second	1.5 seconds	1 second
memory limit	32 MB	32 MB	32 MB	32 MB	64 MB	128 MB
point value	**50**	**80**	**100**	**120**	**140**	**160**
	650					

A while ago, Mirko's grandparents had to switch from analog to digital TV receiver. On their old analog receiver they could watch only two channels: BLJTV1 and BLJTV2. Switching to digital receiver they gained dozens of new channels, but they still wanted to watch just the two channels they had before. They asked Mirko to adjust the remote control in a way that **BLJTV1 is on first and BLJTV2 on the second** key.

When switched on, digital receiver creates a list of available channels. All of the channels are distinct and the list **always** contains BLJTV1 and BLJTV2. Mirko found the remote and he started adjusting the receiver. Menu contains the channel list and a little arrow which is marking the **currently selected** channel. After entering the menu, the arrow is marking the **first channel** on the list.

There are four operations in the menu, denoted with numbers 1 to 4:

1. move the arrow one place down (from channel **i** to channel **i**+1)
2. move the arrow one place up (from channel **i** to channel **i**-1)
3. move the arrow and selected channel one place down (channel **i** and channel **i**+1 are switching places, arrow ends up in place **i**+1)
4. move the arrow and selected channel one place up (channel **i** and channel **i**-1 are switching places, arrow ends up in place **i**-1)

Menu is robust, so invalid commands (such as a command which would move the arrow outside of the channel list) are just **ignored**.

Write a program which will, given a channel list, output a sequence of **any** operations such that, after they are executed, BLJTV1 will be on the first, and BLJTV2 on the second place in channel list. Additionally, the sequence length should be less than **500.** Ordering of all other channels is arbitrary.

INPUT

First line of input contains a positive integer **N** ($2 \leq \mathbf{N} \leq 100$), number of channels.

Next **N** lines contain a list of channels generated by the digital receiver. Each line contains the name of one channel. Channel name is a sequence of at most 10 characters - capital letters of English alphabet and digits.

Input sequence is created in such way that it will **always** be required to make **at least one** operation.

OUTPUT

First and only line of output should contain the sequence of Mirko's commands with no spaces in between.

SAMPLE TESTS

input	**input**	**input**
3 ABC BLJTV1 BLJTV2	4 ABC1 ABC02 BLJTV2 BLJTV1	4 ABC1 ABC02 BLJTV2 BLJTV1
output	**output**	**output**
33	11144411144	33144413

In the beginning of December, parliamentary elections were held in our country. Croatia is divided in 10 ***election regions***. From each region, **14 parliamentary representatives** are elected. Each of the voters is voting for one of the few **parties**. After voting, the representatives are elected using the D'Hondt (D"Ont) method.

By this method, **first** we select parties which gathered **at least 5%** of the votes. Number of votes of each of the selected parties is then **divided** by every number from 1 to 14. In this way we assign 14 rational numbers - let's call them 'scores' - to each of the parties.

First of the 14 representatives in a region is chosen from a party with the largest score. Second representative is selected from a party with the second largest score. The third... This procedure continues until all of the 14 places are elected. **Remark:** There will always be a unique way to elect the representatives, i.e. no two scores will be equal.

Write a program that, given the **total number of voters** and number of votes each party gained, determines how many politicians were elected as region representatives from each party. Some parties have gained negligible number of votes and will not be in the input - that is the reason that the total number of voters might not be equal to the sum of list votes in the input.

INPUT

First line of input contains a positive integer **X** ($1 \leq \mathbf{X} \leq 2\ 500\ 000$), total number of voters in the region. Second line of input contains a positive integer **N** ($0 \leq \mathbf{N} \leq 10$), number of parties we are considering. Next **N** lines contain two positive integers divided by a single space: party identifier (capital letter of English alphabet) and a positive integer **G** ($0 \leq \mathbf{G} \leq 250\ 000$), number of votes gained by that party.

OUTPUT

Output is consisted of number of lines equal to the number of parties which had at least 5% of the votes. For each of these parties, print a party identifier and a number of parliamentary representatives elected from that party. Lines should be sorted by identifiers, alphabeticaly.

SCORING

In 30% of the test cases, every party in the input will have at least 5% of the votes.

SAMPLE TESTS

input

```
235217
3
A 107382
C 18059
B 43265
```

output

```
A 9
B 4
C 1
```

input

```
245143
4
F 14845
A 104516
B 52652
C 14161
```

output

```
A 8
B 4
C 1
F 1
```

input

```
206278
5
D 44687
A 68188
C 7008
B 48377
G 9665
```

output

```
A 6
B 4
D 4
```

Mirko doesn't like Latin homeworks so he made a bet with Slavko. Loser will be writing homeworks for both of them the entire month. Mirko wants to win so he designed this problem they could have something to bet on.

At his desk he found a cube, with numbers 1 to 6 on its faces. Cube is shown on the picture. Additionally, sum of the numbers on opposing faces is equal to 7. That means that 6 is on the opposite face of 1, 5 on the opposite of 2 and 4 on the opposite face of 3.

Mirko has put the cube in the upper left field of the matrix of **R** rows and **C** columns. The cube is initially oriented in a way that upper side is showing number 1, and side to the right number 3.

Mirko now makes te following moves:

1. He is rolling the cube to the right, until it reaches the last column
2. Then he rolls it down (to the next row)
3. Now he rolls the cube to the left, until it reaches first column
4. Like in step 2, he rolls it down (to the next row)

Mirko is repeating these steps for as long as he can, i.e. as long as he can roll the cube in the next row. When a cube reaches some field, Mirko writes down the number on the top of the cube. In the end he sums all of the numbers he had written.

Mirko made a bet with Slavko that he could calculate that sum without error. Help Slavko verifying Mirko's solution!

INPUT

First and only line of input contains two positive integers. **R** and **C** ($1 \leq$ **R**, C $\leq$ 100 000), matrix dimensions.

OUTPUT

First and only line of input should contain the sum described in the task.

SCORING

In test cases worth 50% of total points, **R** and **C** will be less than or equal to 100.

SAMPLE TESTS

input	**input**	**input**
3 2	3 4	737 296
output	**output**	**output**
19	42	763532

First sample description: numbers Mirko wrote down are:

1	4
1	5
3	5

Mirko created a new robot and decided to test it on a giant test track. We can imagine the test track as 2D coordinate system. The robot **starts at a point (0, 0)** and receives a set of instructions denoted by letters S, J, I, Z, each of them marking a direction in which robot should be moving.

More precisely, if a robot is located in (**x**, **y**), S ("north") means it should move to (**x**, **y**+1), J ("south") means it should move to (**x**, **y**-1), I ("east") means it should move to (**x**+1, **y**) and Z ("west") means it should move to (**x**-1, **y**).

While robot is receiving instructions and moves through the test track, Mirko is verifying its position in the following manner. Test track contains **N** fixed **control points**. After each instruction is made, each of the control points measures **manhattan-distance** to the robot. Distances from all control points are then summed and sent to Mirko.

Assuming that robot moves by the instructions without error, calculate the sum of distances to all control points after each instruction.

Remark: manhattan-distance of the points (**x1**, **y1**) and (**x2**, **y2**) is equal to |**x1** - **x2**| + |**y1** - **y2**|.

INPUT

First line of input contains positive integers **N** (number of control points, $1 \leq \mathbf{N} \leq 100\,000$) and **M** (number of instructions, $1 \leq \mathbf{M} \leq 300\,000$), separated by a single space.

Each of the following **N** lines contains coordinates of one control point: two space-separated integers **x**, **y**, with absolute value less than 1 000 000 (million). It is possible that two control points have the same coordinates - distance towards each of them is added to the sum.

The following line contains a string of **M** characters from the set {S, J, I, Z}, the sequence of instructions sent to the robot.

OUTPUT

Output **M** lines: **i**-th line of output should contain the described number after **i**-th instruction.

SAMPLE TESTS

input

```
1 3
0 -10
ISI
```

output

```
11
12
13
```

input

```
3 5
0 0
1 1
1 -1
SIJJZ
```

output

```
5
4
3
4
5
```

Mirko loves cars and he finally managed to start his own car factory! Factory has **N** employees, each of them has exactly one superior (except Mirko - he is by default everybody's superior). Mirko is denoted by number 1, and the rest of the employees with numbers 2 to **N**.

Every employee can raise or lower the wages of **all** of his subordinates (both direct subordinates and those lower in the hiearchy tree). Mirko's role is to prevent abuse of such power, so from time to time he wants to know wage of a particular employee.

He is asking you to write a program which will help him monitor wage changes, given a sequence of commands described in the input section.

Remark: at any time, all of the wages will be positive integers and will fit in standard 32-bit integer type (int in C/C++, longint in Pascal).

INPUT

First line of input contains two space-separated positive integers **N** ($1 \leq \mathbf{N} \leq 500\,000$), number of employees, and **M** ($1 \leq \mathbf{M} \leq 500\,000$), number of wage changes and wage queries.

Next **N** lines contain the information about employees 1, 2, ..., **N** (respectively): starting wage and the identifier of his direct supervisor. **Remark:** Mirko has no supervisor, so his line will contain only his starting wage.

Next **M** lines contain one of the following:

1. **p A X** - employee **A** increases (or decreases in case of a negative **X**) wage of all his subordinates by the amount **X** ($-10\,000 \leq \mathbf{X} \leq 10\,000$);

2. **u A** - Mirko wants to know the wage of employee **A**.

OUTPUT

Output should contain one line for each '2' query in the input - the current wage of the given employee.

SAMPLE TESTS

input

```
2 3
5
3 1
p 1 5
u 2
u 1
```

output

```
8
5
```

input

```
5 5
4
2 1
6 1
7 1
3 4
u 3
p 1 -1
u 3
p 4 5
u 5
```

output

```
6
5
7
```

input

```
6 7
5
4 1
3 2
7 3
2 3
3 5
p 3 2
p 2 4
u 3
u 6
p 5 -2
u 6
u 1
```

output

```
7
9
7
5
```

As mentioned before, there are **N** workers in Mirko's factory. They are manufacturing cars on a conveyor belt, in a pipeline fashion. Workers are denoted by numbers 1 – leftmost, to **N** - rightmost. Each of the workers does his specific job and requires certain amount of time to complete it.

Production of a single car starts with worker #1 (Mirko). After he had finished with his part of the job, worker #2 takes over, after him #3... When worker #**N** finishes with his part, the car is finished. Mirko and his workers have to produce **M** cars and they **must** produce them in order 1 to **M**.

For every worker **i** we know $\mathbf{T_i}$ - time required for him to do his part of the job. For every car **j** we know factor of assembly complexity $\mathbf{F_j}$. Time in minutes for worker **i** to finish his part of he job on the car **j** is computed as a product $\mathbf{T_iF_j}$.

After some worker has finished working on a car, he has to give it to the next worker **instantly**, without any delay (weird company policy). For that reason, the worker receiving the car has to be free (he must not be working on some other car). In order to fulfill this condition, Mirko has to choose a good timing to start building a new car. To be efficient, he'll wait **minimum** number of minutes until he is certain that all of the conditions described are met.

Write a program which will, given worker times and factors of complexity for each car, compute total time required for producing all of the cars.

INPUT

First line of input contains space-separated positive integers **N** ($1 \leq \mathbf{N} \leq 100\,000$), number of workers, and **M** ($1 \leq \mathbf{M} \leq 100\,000$), number of cars.

i-th of the following **N** lines contains worker time $\mathbf{T_i}$ for the worker **i**.

j-th of the following **M** lines contains factor of complexity $\mathbf{F_j}$ for the car **j**.

These conditions hold: $1 \leq \mathbf{T_i} \leq 10\,000$, $1 \leq \mathbf{F_j} \leq 10\,000$.

OUTPUT

First and only line of output has to contain required number of minutes.

SCORING

In test cases worth 40% of total points, **N** and **M** will be at most 1000.

SAMPLE TESTS

input	input	input
3 3 2 1 1 2 1 1	3 3 2 3 3 2 1 2	4 5 3 2 2 2 3 1 2 1 2
output	output	output
11	29	55

First sample description: After four minutes, first worker finishes working on the first car. He might start working on the second car immediately, but that would violate a condition that cars have to be passed to next workers as soon as they're done (after seven minutes second worker would finish working on his part of second car, but the third worker would not be free to take over as he would still be working on the first car). That is the reason production of the second car is started after five minutes. Production of the third car starts after seven minutes. First car is finished after eight, second after nine and third after eleven seconds. Total time is then 11.