task	kontrola	stolni	kockice

source file	`kontrola.pas`
	`kontrola.c`
	`kontrola.cpp`
		`stolni.pas`
		`stolni.c`
		`stolni.cpp`
			`kockice.pas`
			`kockice.c`
			`kockice.cpp`

input data	stdin		
output data	stdout		
time limit (Intel Celeron 2.66Ghz)	1 sec		5 sec
memory limit (heap)	32 MB		64 MB
memory limit (stack)	8 MB		16 MB
points	**50**	**60**	**90**
	200		

A train is operating on a line that consists of N stops (including the first and last stops).

The train is empty **in the beginning** and **in the end,** and for each stop we know the number of passengers that leave the train and the number of passengers that enter the train. Each passenger is traveling for some number of stops and the same passenger never boards the same train more than once.

There is a ticket inspector in the train. He walks through the entire train **between the first and second stops** and inspects the tickets of **all** passengers currently aboard. After that, the inspector inspects tickets again **after every K stops** (hence he inspects the ticket between stops a*K+1 and a*K+2 for each integer a). It is therefore possible that some passengers enter and leave the train with their tickets **never inspected**.

Write a program that finds the **minimum** and **maximum** possible number of such passengers.

input data

The first line of input contains two integers N and K, $2 \leq N \leq 1000$, $1 \leq K \leq 1000$.

Each of the next N lines contains two integers – the number of passengers that leave and the number of passengers that enter on that particular stop (**from the first to the last stop**). These numbers will be greater than or equal to 0 and no greater than 1000.

output data

The first and only line of output should contain two integers – the minimum and maximum numbers from the task description.

examples

input

```
3 2
0 5
4 2
3 0
```

output

```
2 2
```

input

```
4 2
0 5
0 5
3 0
7 0
```

output

```
0 3
```

input

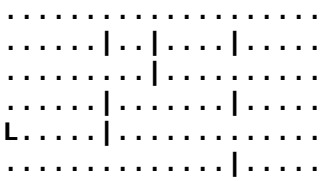
```
6 2
0 10
5 3
6 4
2 8
8 1
5 0
```

output

```
5 11
```

Two friends, Mirko and Slavko, are playing table soccer. Mirko has no players on the table, and Slavko's players are attached to vertical bars.

The ball is situated on the left edge of the table and Mirko shots the ball in the **up-right diagonal direction**. After that, the ball **moves in a straight line** across the table **deflecting from the upper and lower edges**.



If the ball hits some of the Slavko's players, Mirko fails to score, and if the ball reaches the right edge of the table, Mirko scores.

Slavko knows that he is a better player than Mirko and he wants to allow Mirko to score.

Write a program that finds **some arrangement** of the Slavko's players that **allows Mirko to score**. **Also, draw the path of the ball**.

Slavko can arrange the players by moving each column **up or down** a certain distance with the restriction that all players have to remain within the table.

input data

The first line of input contains two integers R and C, $2 \leq R,C \leq 100$, the numbers of rows and columns on the table.

Each of the following R lines contains C characters – the initial layout of the table.

The ball is labeled with '**L**', players with '**|**' (vertical line) and empty squares on the table with '**.**' (dot). There will be no players in the leftmost column on the table.

output data

Output the final layout of the table, together with the path of the ball.

Note: the test data will be such that a solution, although not necessarily unique, will always exist.

examples

input

```
6 19
............|.....
.|....|...|..|.....
.|........|........
L.....|...|........
......|......|.....
...................
```

output

```
.|.L......|..L.....
.|L.L.....|.L|L....
.L...L|...|L.|.L...
L.....L...L.....L..
......|L.L.......L.
......|.L....|....L
```

input

```
3 8
.||.....
L|.|.|..
..||.|..
```

output

```
.L||.L..
L|L|L|L.
.||L.|.L
```

input

```
5 9
L........
.||||||||
.........
.||||||||
.........
```

output

```
L|..|..|L
.L|...|L.
.|L|||L||
..|L.L|..
...|L|..|
```

Little Luka is playing an interesting tetris-like game.

In the beginning of the game, there are **three empty columns** on the screen. During the game, letters appear on the top of the screen. When a letter appears, Luka chooses one of the columns and puts this letter on top of the topmost letter in that column, or on the bottom if that column is empty.

When the game is over, we define the **total number of points** that Luka gets as sum of the points **for each column**. The number of points for a column is calculated in the following way: we first **find groups of identical consecutive letters** (borders between two groups are between different letters).

For example, if the letters in some column are:

A
A
A
B
C
C
C
A
A

then this column consists of 4 groups. Now, for each of the groups, we get some number of points that depends only on the number of letters in that group.

For example, if we get 3 points for the groups with size 1, 7 points for the groups with size 2, and 5 points for the groups with size 3, then the number of points for the example column would be 5+3+5+7=20.

Write a program that, given a sequence of letters and the assignment of scores to group sizes, finds the **maximum number of points** that Luka can get in this game.

input data

The first line of input contains five integers B_1, B_2, B_3, B_4 i B_5. For i=1,2,3,4 the number $\mathbf{B_i}$ is the number of points for groups consisting of **i** letters and the number $\mathbf{B_5}$ is the number of points for groups of **5 or more letters**. Each of these 5 numbers will be less than or equal to 100.

The second line contains an integer N, $1 \leq N \leq 1000$ – the number of letters appearing.

The third line contains a sequence of N uppercase letters of the English alphabet ('A'-'Z'), in the order in which they appear on the top of the screen during the game.

output data

The first and only line of output should contain a single integer – the maximum number of points from the task description.

examples

input

```
5 5 5 5 5
9
ABCABCABC
```

output

```
45
```

explanation

```
|   |
|CAB|
|BCA|
|ABC|
+---+
```

input

```
1 5 1 1 1
9
ABCABCABC
```

output

```
18
```

explanation

```
|   |
|BCA|
|ABC|
|ABC|
+---+
```

input

```
3 3 10 3 3
17
AAABBCCCAAACBAAAB
```

output

```
56
```

explanation

```
|   |
|A  |
|A  |
|A  |
|B  |
|A  |
|A  |
|A  |
|C  |
|ABC|
|ABC|
|ABC|
+---+
```