Problem Set

Please check that you have 12 problems that are spanned across 25 pages in total (including this page).

Problem A
Card Game
Time Limit: 1.0 Seconds

Alice and Bob play a game of taking turns removing cards from the grid board. At the beginning of the game, there is one card in each cell of the $N \times M$ sized grid board, and each card is painted in one of three colors: red, black, or green. In the grid, the position of the upper-left cell is indicated by $(1,1)$, and the position of the lower-right cell is indicated by (N, M).

Alice and Bob choose one of the cards placed on the grid, and then remove the cards according to the rules below.

- If the color of the chosen card is red, all 'connected cards' placed on a diagonal with a slope of 1 based on it are removed.
- If the color of the chosen card is blue, all 'connected cards' placed on a diagonal with a slope of -1 based on it are removed.
- If the color of the chose card is green, all 'connected cards' placed on the diagonal in both directions based on it are removed.

'Connected cards' to the chosen card are consecutively adjacent cards along a diagonal with a slope of 1 or -1 including the chosen card.

For example, when the current board situation during the game is as in Figure A.1, let the chosen card be a red card placed at (4,3). As shown in Figure A.1, 'connected cards' placed on the diagonal line with a slope of 1 refer to the cards placed in the oval circle, which should be removed. That is, cards placed in the cells on the movement path while moving diagonally in both directions from the position (4,3) are 'connected cards'. However, while moving in both directions along the diagonal at the chosen cell (4,3), if it encounters a grid boundary or a blank cell, the movement stops.

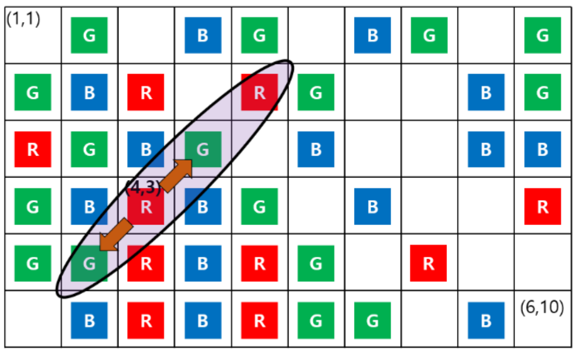


Figure A.1. An example to illustrate connected cards to the red card at $(\mathbf{4}, \mathbf{3})$

Similarly, when the current board situation during the game is as shown in Figure A.2, let the chosen card be the blue card placed at (3,5). As shown in Figure A.2, 'connected cards' placed on the diagonal line with a slope of -1 refer to the cards placed in the oval circle, which should be removed.

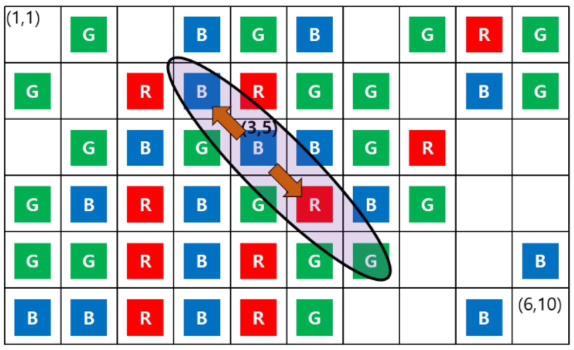


Figure A.2. An example to illustrate connected cards to the blue card at $(\mathbf{3}, \mathbf{5})$

Figure A.3 shows the cards to be removed when the chosen card is green card placed at (4,5).

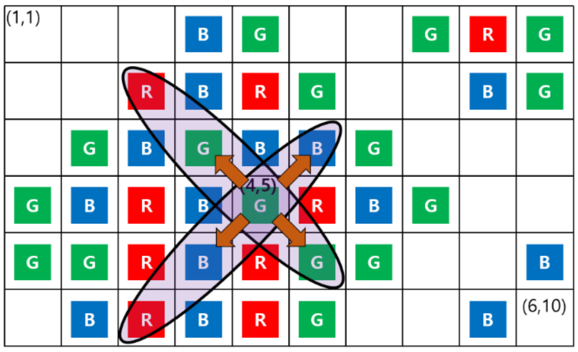


Figure A.3. An example to illustrate connected cards to the green card at $(\mathbf{4}, \mathbf{5})$

Alice and Bob alternately choose a card from the grid, and according to the color of the chosen card, remove the 'connected cards' according to the rules described above. Whoever removes the last card wins the game. That is, the player who cannot remove any card because there are no cards to choose from on the grid loses the game. Both Alice and Bob have a good understanding of the strategy of how to win the game and do their best to win.

Given the size of the grid board and the information on color of the cards placed on the board, write a program to determine whether Alice can win when she starts the game.

Input

Your program is to read from standard input. The input starts with a line containing two integers, N and M ($1 \le N, M \le 25$), where N is the number of rows and M is the number of columns of the grid. In the following N lines, the i-th line contains a string of length M, which represents the colors of the M cards in the i-th row in the grid. Every character in the string is either '`R`', '`B`', or '`G`', which stands for red, blue, or green, respectively.

Output

Your program is to write to standard output. Print exactly one line. The line should contain an upper-case letter: either '`W`' if Alice wins or '`L`' if Alice loses.

The following shows sample input and output for three test cases.

Sample Input 1	Output for the Sample Input 1

`1 3`
`BBB`
	`W`

Sample Input 2	Output for the Sample Input 2
2 3 BBG RGR	W

Sample Input 3	Output for the Sample Input 3
2 2 GG GG	L

Problem B
Castle Design
Time Limit: 1.0 Seconds

The ICPC kingdom has decided to build a new castle. The boundary of the castle is designed as a *rectilinear* polygon with edges parallel to the x-axis or to the y-axis. To minimize the damage exposed by the enemy attack, the kingdom wants to minimize the perimeter of the rectilinear polygon. Let us go into more detail.

A rectilinear polygon P of n vertices with integer coordinates realizes a *turn sequence* S of length n of two letters `L` and `R` if there is a counterclockwise traverse along the boundary of P such that the turns at vertices of P, encountered during the traverse, form the turn sequence S; the left turn at a convex vertex corresponds to `L` and the right turn at a reflex vertex corresponds to `R`. For example, the rectilinear polygon in Figure B.1(a) realizes the turn sequence S = `RLLRLLLRRLLRLRLL` of length 16. Another turn sequence S = `LLRLLRLLRLRLLR` of length 14 can be realized by rectilinear polygons in Figure B.1(b) and B.1(c). Note that a turn sequence can have infinitely many realizations of rectilinear polygons in the integral plane.

A polygon is *simple* if there are no two edges that intersect except at the end vertices of adjacent edges. A polygon is *monotone* to an axis if its intersection with a line orthogonal to the axis is at most one segment. The monotone polygon is called 2-*monotone* if it is monotone to both x-axis and the y-axis, and 1-*monotone* if it is monotone to the one axis but not to the other axis. For example, the polygon in Figure B.1(a) is 1-monotone because it is monotone to only one axis, the x-axis, while the polygons in Figure B.1(b) and B.1(c) are 2-monotone. A turn sequence is also said to be t-*monotone* if any simple rectilinear polygon realizing the turn sequence is t-monotone where $t = 1, 2$.

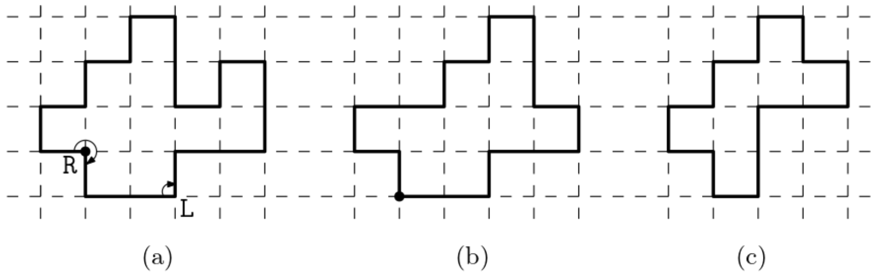


Figure B.1 (a) A simple 1-monotone rectilinear polygon realizing an 1-monotone turn sequence `RLLRLLLRRLLRLRLL` (starting from the marked vertex). (b) A simple 2-monotone rectilinear polygon realizing a 2-monotone turn sequence `LLRLLRLLRLRLLR` (starting from the marked vertex). (c) The 2-monotone rectilinear polygon with the minimum perimeter for the turn sequence in (b).

The perimeter of a rectilinear polygon is the sum of the length of its edges. The perimeter of the polygon in Figure B.1(b) is 18, but this is not minimum for `LLRLLRLLRLRLLR`. Its minimum perimeter should be 16 as in Figure B.1(c).

Given a t-monotone turn sequence of n turns for $t = 1, 2$, write a program to compute the minimum perimeter of simple t-monotone rectilinear polygons that realize the input t-monotone turn sequence.

Input

Your program is to read from standard input. The input is one line containing a string of n turns of two uppercase letters `L` and `R` that is a t-monotone turn sequence, where $t = 1, 2$ and $4 \leq n \leq 10^{t+3}$.

Output

Your program is to write to standard output. Print exactly one line. The line should contain the positive integer that is the minimum perimeter of simple t-monotone rectilinear polygons that realize the input t-monotone turn sequence for $t = 1, 2$.

The following shows sample input and output for four test cases.

Sample Input 1	**Output for the Sample Input 1**
`LLRLLRLLRLRLLR`	`16`

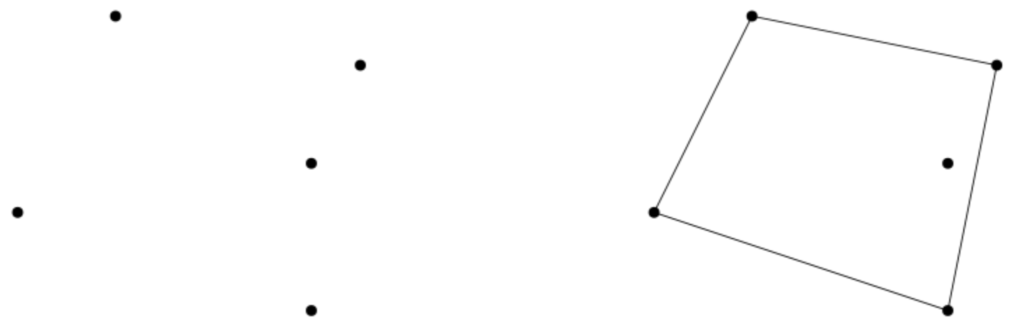
Sample Input 2	**Output for the Sample Input 2**
`RLLRLLLRRLLRLRLL`	`20`

Sample Input 3	**Output for the Sample Input 3**
`LLRRLLLLRRLL`	`16`

Sample Input 4	**Output for the Sample Input 4**
`RLLRLLRLLRLL`	`12`

Problem C
Empty Quadrilaterals
Time Limit: 2.0 Seconds

A *quadrilateral* is a polygon with exactly four distinct corners and four distinct sides, without any crossing between its sides. In this problem, you are given a set P of n points in the plane, no three of which are collinear, and asked to count the number of all quadrilaterals whose corners are members of the set P and whose interior contains no other points in P.



For example, assume that P consists of five points as shown in the left of the figure above. There are nine distinct quadrilaterals in total whose corners are members of P, while only one of them contains a point of P in its interior, as in the right of the figure above. Therefore, there are exactly eight quadrilaterals satisfying the condition and your program must print out `8` as the correct answer.

Input
Your program is to read from standard input. The input starts with a line containing an integer n ($1 \le n \le 300$), where n denotes the number of points in the set P. In the following n lines, each line consists of two integers, ranging from -10^9 to 10^9, representing the coordinates of a point in P. There are no three points in P that are collinear.

Output
Your program is to write to standard output. Print exactly one line consisting of a single integer that represents the number of quadrilaterals whose corners are members of the set P and whose interior contains no other points in P.

The following shows sample input and output for three test cases.

Sample Input 1	Output for the Sample Input 1

5
0 0
2 4
6 2
6 -2
7 3
	8

Sample Input 2

```
4
0 0
10 0
5 10
3 2
```

Output for the Sample Input 2

```
3
```

Sample Input 3

```
10
10 10
1 0
4 8
-1 -4
-7 -4
-3 2
5 -10
-10 -5
1 1
5 -3
```

Output for the Sample Input 3

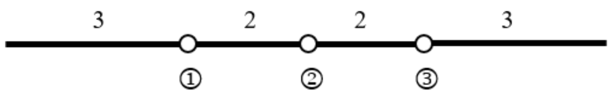
```
170
```

Problem D
Folding Stick
Time Limit: 0.4 Seconds

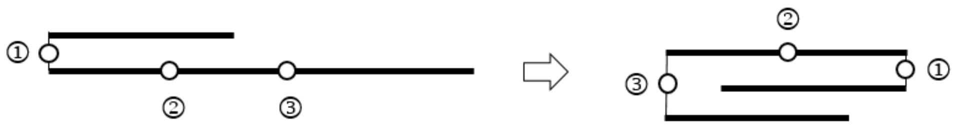
There is a folding stick made up of n segments of positive length. The segments are connected by hinges (stretchable strings), allowing the segments to be folded 180 degrees at each hinge. The *wrapping length* is said to be the length of the folded stick after the stick is folded at hinges one or more. Depending on the folding strategy, the wrapping length may be different.

You are to find the minimum wrapping length under the condition that the stick is folded in the following way: First, place the segments of the stick along a horizontal line. Then, fold the stick clockwise from left to right. During folding, the segment attached to the left side of each hinge rotates 180 degrees clockwise or not at all.

For example, the figure below shows a four-segment stick with a sum of segment lengths of 10. In the figure, the lengths of the segments are 3, 2, 2, and 3 from left to right, and the hinges are marked with ①, ②, ③.



In this example, the stick cannot be folded at both hinges ① and ②. This is because if the stick is folded at hinge ① and then at hinge ②, the segment with length 3 passing over the hinge ② will be broken. If it is folded only at hinge ②, the wrapping length is 5. If it is folded at hinges ① and ③ in order, the wrapping length is 4 as shown in the figure below.



Given a sequence of segments lengths of a folding stick, write a program to output the minimum wrapping length of the stick.

Input
Your program is to read from standard input. The input starts with a line containing an integer, n ($2 \leq n \leq 100{,}000$), where n is the number of segments of a folding stick. The next line contains n positive integers which represent a sequence of lengths of segments from the leftmost one to rightmost one of the stick. Each segment length is no more than 20,000.

Output
Your program is to write to standard output. Print exactly one line. The line should contain the positive integer representing the minimum wrapping length.

The following shows sample input and output for three test cases.

Sample Input 1	Output for the Sample Input 1
4 3 2 2 3	4

Sample Input 2	Output for the Sample Input 2
5 1 1 1 1 1	1

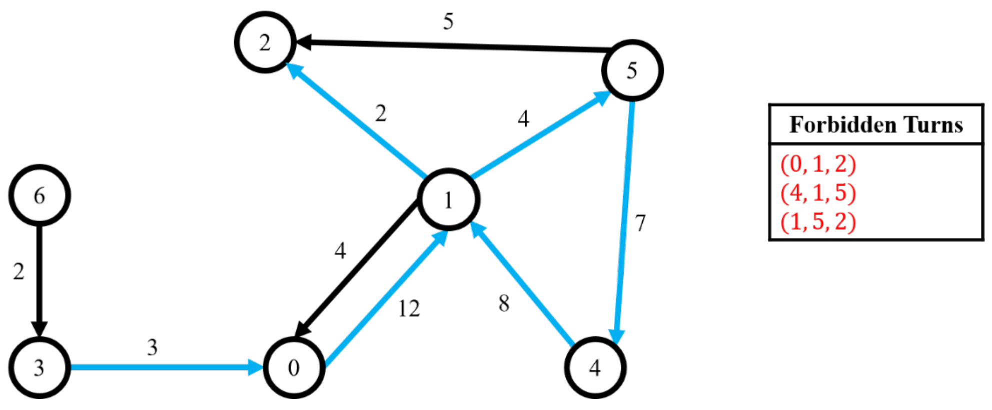
Sample Input 3	Output for the Sample Input 3
7 1 3 2 3 4 2 2	6

Problem E
Forbidden Turns
Time Limit: 2.0 Seconds

A GPS navigation company ICPC(International Control Perfection Company) designs a car navigation system for a transportation network. The system abstracts the transportation network as a directed graph $G(V, E)$ with edge cost c. For a directed edge $(v, w) \in E$, $c(v, w)$ denotes the distance from a place $v \in V$ to another place $w \in V$. The company wants to implement the shortest path module in the system. To reflect the normal situation that we cannot turn to some directions in a junction of transportation network, we want to find the shortest path that does not contain forbidden turns as a subpath.

A path from v to w is a sequence of vertices $(v_1, v_2, \cdots, v_k)$ where $v_1 = v$, $v_k = w$, $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k-1$. Unlike the common definition of the path, you are here allowed to repeat the same vertices in a path one or more. A subpath of a path is a contiguous subsequence of the sequence that corresponds to the path. A forbidden turn is a path (i.e., triplet) (x, y, z) such that $x, y, z \in V$ and $(x, y) \in E$ and $(y, z) \in E$. The distance of a path $(v_1, v_2, \cdots, v_k)$ is defined as $\sum_{i=1}^{k-1} c(v_i, v_{i+1})$. The shortest path from $v \in V$ to $w \in V$ is a path from v to w with the minimum distance. The company wants to find the distance of the shortest path that avoids the forbidden turns between two designated vertices. Note that the shortest path from $v \in V$ to $v \in V$ has distance 0 and it avoids all the forbidden turns.

Let's see the following example in the figure below. Each edge cost lies beside each edge and the list of three forbidden turns are in the right box. The shortest path without forbidden turns from the vertex 3 to the vertex 2 is $(3, 0, 1, 5, 4, 1, 2)$ which is denoted as blue arrows in the following figure. The distance of the shortest path is $3 + 12 + 4 + 7 + 8 + 2 = 36$. Note that we cannot take the shorter paths $(3, 0, 1, 2)$ and $(3, 0, 1, 5, 2)$ since they contain forbidden turns $(0, 1, 2)$ and $(1, 5, 2)$, respectively.



Forbidden Turns
$(0, 1, 2)$
$(4, 1, 5)$
$(1, 5, 2)$

Given a directed graph $G(V, E)$ with the edge cost c, a set of forbidden turns F, and two vertices v and w, write a program to output the distance of the shortest path from v to w that avoids all the forbidden turns. We assume that out-degree of each vertex v, i.e., the number of edges that starts from v is at most 10.

Input

Your program is to read from standard input. The input starts with a line containing three integers, m, n, and k. ($0 \le m \le 10n, 1 \le n \le 30{,}000, 0 \le k \le 500{,}000$), where m is the number of directed edges, n is the number of vertices, and k is the number of forbidden turns of the given directed graph $G(V, E)$. Here, k is less than or equal to the number of all the possible forbidden turns in the given directed graph $G(V, E)$. The vertices are numbered from 0 to $n - 1$. The second line contains two integers v and w which denote the source and destination vertices, respectively. In the following m lines, the i-th line contains three integers x_i, y_i, and c_i ($0 \le x_i \ne y_i \le n - 1$ and $0 \le c_i \le 10^3$) which denotes an edge $(x_i, y_i) \in E$ and its cost, respectively. In the following k lines, the i-th line contains three integers x_i, y_i, and z_i which denote a forbidden turn (x_i, y_i, z_i).

Output

Your program is to write to standard output. Print exactly one line. The line should contain an integer that represents the distance of the shortest path from v to w which avoids all the forbidden turns. If such a path does not exist, the line should contain -1.

The following shows sample input and output for three test cases.

Sample Input 1

```
9 7 3
3 2
6 3 2
3 0 3
0 1 12
1 0 4
1 2 2
1 5 4
4 1 8
5 4 7
5 2 5
0 1 2
4 1 5
1 5 2
```

Output for the Sample Input 1

```
36
```

Sample Input 2

```
4 4 1
0 3
0 1 2
1 2 3
0 2 7
2 3 10
0 1 2
```

Output for the Sample Input 2

```
17
```

Sample Input 3

```
4 4 0
0 3
0 1 2
1 2 3
0 2 7
2 3 10
```

Output for the Sample Input 3

```
15
```

Problem F
Frog Jump
Time Limit: 1.0 Seconds

A frog is living in a beautiful lake. On the lake, there are a lot of lotus leaves floating in a row, which are represented by closed intervals on the line. The frog likes to be on lotus leaves and moves between them.

The n closed intervals, representing lotus leaves, on the line, that is, on the x-axis are given and the frog is initially on some interval I_0. The frog can move from an interval I to an interval J if they overlap. Two intervals overlap if they share a common point. So the frog can move through overlapping intervals. When the frog is moving to the right (left) through the overlapping intervals, it may reach an interval H, where it can no longer move to the right (left) from the right (left) endpoint of H. In this case, the frog can *jump* to the interval K with the smallest (largest) left (right) endpoint among intervals whose left (right) endpoint is greater (smaller) than the right (left) endpoint of H if they exist. Then, the *jump length* is defined to be the length between the right (left) endpoint of H and the left (right) endpoint of K. See Figure F.1.

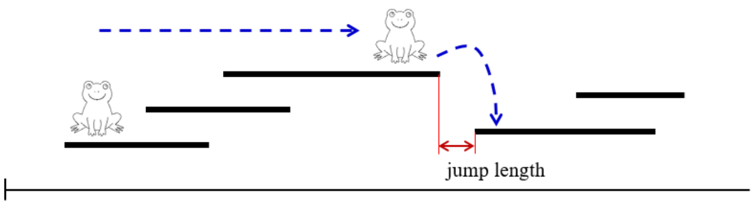


Figure F.1 Jump length

A sequence of k intervals $I_1, I_2, \dots, I_k$ is given and the frog should visit the intervals in order from the initial interval I_0. In this travel, the frog has to jump if necessary.

For example, in Figure F.2, eight intervals $[1,8]$, $[2,4]$, $[5,11]$, $[13,15]$, $[15,17]$, $[16,18]$, $[19,22]$ and $[20,22]$ are given and numbered from 1 and 8. The frog is initially on interval 1. Intervals 3, 7, 4, 6, 3 which the frog should visit in a sequence are given. Then the frog moves from interval 1 to 3 with no jump, and it moves from 3 to 7 with two jumps, say, $3 \to 4$ and $6 \to 7$ whose jump length is 3 totally. In this movement, the frog passes through the interval 4. Nevertheless, it should visit the interval 4 after the interval 7. Then, there are two jumps during the movements from 7 to 4 and from 6 to 3 whose jump length is 3 totally. Thus after the frog visits all the given intervals, the total jump length is 6.

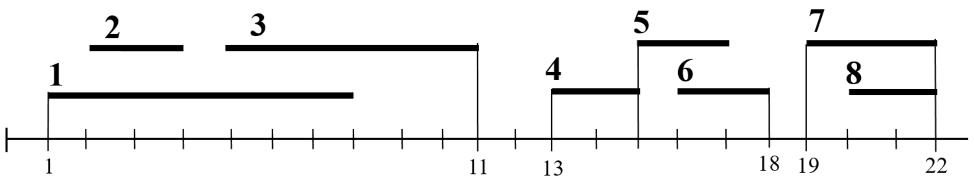


Figure F.2 The given eight intervals

Given n intervals on the line and a sequence of k intervals, write a program to output the total jump length during the travel that the frog visits the k intervals in order from its initial interval 1.

Input

Your program is to read from standard input. The input starts with a line containing two integers, n and k ($1 \leq n \leq 100{,}000$ and $1 \leq k \leq 1{,}000{,}000$), where n is the number of intervals and k is the number of intervals which the frog should visit. The intervals are numbered from 1 to n and the initial location of the frog is always 1. In the following n lines, the i-th line contains two integers a and b ($0 \leq a < b \leq 10^9$) that represent the left and right endpoints of interval i, respectively. The intervals are given in increasing order of their left endpoints – if they are same, then in increasing order of the right endpoints. Also the intervals are all distinct. The next line contains k integers that represent the intervals which the frog should visit in order. These integers are between 1 and n and can be in duplicate.

Output

Your program is to write to standard output. Print exactly one line. The line should contain the total jump length of frog when it visits the given k intervals in order.

The following shows sample input and output for three test cases.

Sample Input 1	Output for the Sample Input 1

4 3
0 2
0 3
3 5
6 7
4 2 3
	2

Sample Input 2	Output for the Sample Input 2

4 3
0 2
0 3
3 5
6 7
2 3 2
	0

Sample Input 3	Output for the Sample Input 3

8 5
1 8
2 4
5 11
13 15
15 17
16 18
19 22
20 22
3 7 4 6 3
	6

Problem G
Linear Regression
Time Limit: 5.0 Seconds

Chansu is a graduate student at University of ICPC, working in a laboratory for his master's degree. His research theme is to reveal a relation between the obesity and the yearly income of individuals in a certain group G.

Chansu collected data of the form (x_i, y_i) from n persons in G, where x_i and y_i denote the obesity index and the yearly income of the i-th person, and made an apparent hypothesis:

There is a linear dependency between the obesity and the yearly income of individuals in group G.

To prove his hypothesis, Chansu tried to find an optimal linear function $f^*(x)$ with real coefficients such that the error with respect to the collected data is minimized. More specifically, the error of f with respect to the data is defined to be the maximum of $|y_i - f(x_i)|$ over all $i = 1, \dots, n$.

However, the result was disappointing because the error of the optimal function $f^*(x)$ was unexpectedly big. This means that his hypothesis cannot be proven in this way.

Chansu tried to figure out the reason of the big errors. One day, he plotted the data (x_i, y_i) as points on the coordinated plane and realized that there are a small number k of points that are unusually far from the others, so the error of the optimal function can be drastically reduced after removing them.

You, as a friend of Chansu, would love to help Chansu. Write a program that finds an optimal linear function minimizing the error after removing some k values from the given data $\{(x_1, y_1), \dots, (x_n, y_n)\}$ and prints out the error value, when the number k is given as part of input.

Input
Your program is to read from standard input. The input starts with a line containing two integers, n and k $(1 \le n \le 50{,}000,\ 0 \le k \le \min\{\frac{n}{2}, 300\})$, where n is the number of collected data values. In each of the following n lines, each data value (x_i, y_i) is given by two integers x_i and y_i $(-10^9 \le x_i,\ y_i \le 10^9)$ for $i = 1, \dots, n$. You can assume that no three of them are collinear when plotting them in the coordinated plane.

Output
Your program is to write to standard output. Print exactly one line. The line should contain a real number z representing the minimum possible error of a linear function with respect to the data after removing some k values. Your output z should be in the format that consists of its integer part, a decimal point, and its fractional part, and will be decided to be "correct" if it holds that $a - 10^{-6} < z < a + 10^{-6}$, where a denotes the exact answer.

The following shows sample input and output for four test cases.

Sample Input 1

```
6 0
0 0
5 -1
9 6
3 0
4 2
3 1
```

Output for the Sample Input 1

```
2.166667
```

Sample Input 2

```
6 1
0 0
5 -1
9 6
3 0
4 2
3 1
```

Output for the Sample Input 2

```
1.000000
```

Sample Input 3

```
6 2
0 0
5 -1
9 6
3 0
4 2
3 1
```

Output for the Sample Input 3

```
0.500000
```

Sample Input 4

```
6 3
0 0
5 -1
9 6
3 0
4 2
3 1
```

Output for the Sample Input 4

```
0.083333
```

Problem H
Longest Substring

Time Limit: 5.0 Seconds

For a string S of length $n \geq 1$ and a positive integer k $(1 \leq k \leq n)$, a non-empty substring of S is called a k-*substring* if the substring appears *exactly* k times. Such k occurrences are not necessarily disjoint, i.e., are possibly overlapping. For example, if $S =$ "ababa", the k-substrings of S for every $k = 1, \ldots, 5$ are as follows:

- There are four 1-substrings in S, "abab", "ababa", "bab", and "baba" because these substrings appear exactly once in S. Note that "aba" is not a 1-substring because it appears twice.
- There are four 2-substrings in S, "ab", "aba", "b", and "ba". The substring "ab" appears exactly twice without overlapping. Two occurrences of the substring "aba" are overlapped at a common character "a", but it does not appear three times or more.
- There is only one 3-substring in S, "a".
- Neither 4-substrings nor 5-substrings exist in S.

For a k-substring T of S, let $d(T)$ be the maximum number of the disjoint occurrences of T in S. For example, a 2-substring T = "ab" can be selected twice without overlapping, that is, the maximum number of the disjoint occurrences is two, so $d(T) = 2$. For a 2-substring T = "aba", it cannot be selected twice without overlapping, so $d(T) = 1$. For a 3-substring T = "a", it can be selected three times without overlapping, which is the maximum, so $d(T) = 3$.

Let $f(k)$ be the length of the longest one among all k-substring T with the largest $d(T)$ for $1 \leq k \leq n$. For example, $f(k)$ for $S =$ "ababa" and $k = 1, \ldots, 5$ is as follows:

- For $k = 1$, all 1-substrings T can be selected only once without overlapping, so $d(T) = 1$. Thus, the longest one among all 1-substrings with $d(T) = 1$ is "ababa", so $f(1) = 5$.
- For $k = 2$, $d(T) = 1$ for T = "aba", but $d(T) = 2$ for the other 2-substrings T = "ab", "b", "ba". Among 2-substrings with $d(T) = 2$, "ab" and "ba" are the longest ones, so $f(2) = 2$.
- For $k = 3$, $f(3) = 1$ because there is only one 3-substring "a".
- For $k = 4, 5$, there are no k-substrings, so $f(4) = 0$ and $f(5) = 0$.

Given a string S of length n, write a program to output n values of $f(k)$ from $k = 1$ to $k = n$. For the above example, the output should be 5 2 1 0 0.

Input

Your program is to read from standard input. The input starts with a line containing the string S consisting of n $(1 \leq n \leq 50{,}000)$ lowercase alphabets.

Output

Your program is to write to standard output. Print exactly one line. The line should contain exactly n non-negative integers, separated by a space, that represent $f(k)$ from $k = 1$ to $k = n$ in order, that is, $f(1)$ $f(2)$ … $f(n)$. Note that $f(k)$ should be zero if there is no k-substring for some k.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1
`ababa`	`5 2 1 0 0`

Sample Input 2	Output for the Sample Input 2
`aaaaaaaa`	`8 7 6 5 4 3 2 1`

Problem I
Palindrome Type
Time Limit: 1.0 Seconds

A *palindrome* string is a word which reads the same backward as forward, such as *madam* or *racecar*. In this problem we only consider strings with lowercase alphabets.

We newly define the types of palindromes. If a string is not a palindrome, we try to make it a palindrome by removing the minimum number of characters in the string. For a string w, if k is the minimum number of characters removed to make the string a palindrome, we call the string w *type-k palindrome*. Thus, if w is a palindrome, then w is a type-0 palindrome.

Given a string w, write a program to determine if w is a type-k palindrome where $k = 0, 1, 2, 3$.

Input
Your program is to read from standard input. The input is a single line containing a string w with length n ($5 \leq n \leq 10^5$) of lowercase alphabets.

Output
Your program is to write to standard output. Print exactly one line. The line should contain a number k among $\{0, 1, 2, 3, -1\}$ if the input string is a type-k palindrome where $k = 0, 1, 2, 3$ and otherwise -1. The negative number -1 means the input string is not a type-k palindrome where $k = 0, 1, 2, 3$.

The following shows sample input and output for three test cases.

Sample Input 1	**Output for the Sample Input 1**
`aababaa`	`0`

Sample Input 2	**Output for the Sample Input 2**
`abccbbab`	`2`

Sample Input 3	**Output for the Sample Input 3**
`acmicpc`	`-1`

Problem J

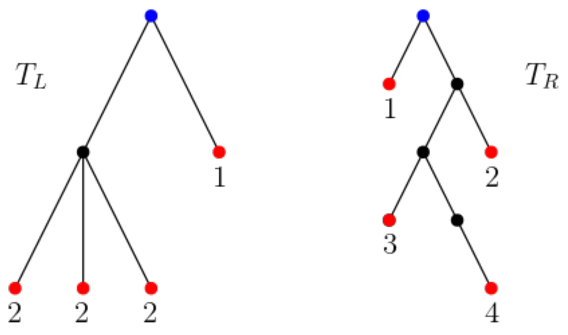
Parentheses Tree

Time Limit: 1.0 Seconds

A rooted ordered tree T can be expressed as a string of matched parentheses $p(T)$. The string representation $p(T)$ can be defined recursively. As a base case, a tree consisting of a single node is expressed by a pair of parentheses `()`. When a rooted ordered tree T consists of a root node and k ordered subtrees $T_1, T_2, \dots, T_k$ having their roots as child nodes of the root node, the string representation $p(T)$ is defined as follows:

$$p(T) := (+ p(T_1) + p(T_2) + \cdots + p(T_k) +)$$

In the above expression, the operator $+$ means the concatenation of two strings. The figure below shows two examples of rooted ordered trees. The string representations $p(T_L)$ and $p(T_R)$ are `((()()())())` and `(()((()(()))()))`, respectively.



The distance from the root node to a leaf node is defined as the number of edges to be traversed to reach the leaf from the root. In the figure above, the root nodes are colored in blue, and the distances from the root node to all leaf nodes are shown. For trees T_L and T_R, the sum of the distances from the root to all leaf nodes are 7 and 10, respectively.

Given a string of matched parentheses representing only one rooted ordered tree, write a program to output the sum of the distances from the root of the tree to all leaf nodes.

Input

Your program is to read from standard input. The input consists of one line containing a string of matched parentheses which represents only one rooted ordered tree. The input does not contain any characters other than parentheses, and the length of string is at least 2 and no more than 10^7.

Output

Your program is to write to standard output. Print exactly one line. The line should contain the sum of the distances from the root of the rooted ordered tree to all leaf nodes.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1
`((()()())())`	`7`

Sample Input 2	Output for the Sample Input 2
`(()((()(()))()))`	`10`

Problem K
Shuffle Game
Time Limit: 1.5 Seconds

Shuffle Game is a simple card game between the dealer and the player. Initially, the same deck of n cards is given to both the dealer and the player. Each card in the deck suits with one of the four symbols (C, D, H, or S), followed by the one of 13 kinds (2, 3, 4, 5, 6, 7, 8, 9, 10, A, J, K or Q). Therefore, there are 52 different types of cards and the same cards can exist in the deck. After the cards are given to the dealer and the player, the dealer first creates their own deck X from the deck given to the dealer using any shuffling method and shows X to the player. After that, the player creates the deck Y by the following steps: Y is initially empty.

Step 1. Create two decks $P1$ and $P2$ from the deck given to the player. The number of cards in $P1$ and $P2$ can be different.

Step 2. Interleave $P1$ and $P2$. That is, move a card at the bottom of $P1$ or $P2$ to the current top of Y, until there is no card on both $P1$ and $P2$. Note that the player does not need to move the cards in $P1$ and $P2$ alternately to Y. Also, since both the dealer and the player create their own deck from the same deck of n cards, Y always consists of the same cards as X.

We define a sequence of a deck as the sequence of the cards in the deck from bottom to top. Then the player's score is defined as the length of the longest common subsequence between the sequences X and Y. For example, suppose the deck of $n = 5$ cards, $(C2,\ CJ,\ D5,\ HA,\ S7)$ is given to both the dealer and the player (here, we represent the deck as its sequence). Then the dealer creates the deck $X = (CJ,\ D5,\ HA,\ C2,\ S7)$ and shows X to the player. After that, the player creates their deck by (i) creating two decks $P1 = (D5,\ HA)$ and $P2 = (CJ,\ S7,\ C2)$ from the given deck and (ii) create $Y = (D5,\ CJ,\ S7,\ HA,\ C2)$ by interleaving $P1$ and $P2$. In this example, the player's score is 3 since $(CJ,\ HA,\ C2)$ is the longest common subsequences between the sequences of X and Y. Now, after finishing Step 1, the player wants to know the maximum possible score to that the player can achieve after applying Step 2. For example, the maximum possible score from X and Y in the previous example is 4 since it is possible to create Y from $P1$ and $P2$ as $(CJ,\ D5,\ HA\ S7,\ C2)$.

Given $n, X, P1$ and $P2$, write a program to compute the maximum possible score that the player can achieve.

Input

Your program is to read from standard input. The input starts with a line containing three positive integers n, p and q ($3 \le n \le 500,\ p + q = n$), where n is the number of cards in the initial deck, and p and q are the number of cards in $P1$ and $P2$, respectively. In the following three lines, the dealer's deck X consisting of n cards, and the player's two decks $P1$ and $P2$ consisting of p and q cards, respectively, are given. Each card in X, $P1$, and $P2$ is represented as its suit (uppercase alphabet C, D, H, or S) followed by its kind (2, 3, 4, 5, 6, 7, 8, 9, 10, uppercase alphabet A, J, K or Q). The cards in the same line are ordered from bottom to top of the corresponding deck.

Output

Your program is to write to standard output. Print exactly one line. The line should contain the maximum possible score that the player can achieve from $X, P1$ and $P2$ after applying Step 2.

The following shows sample input and output for three test cases.

Sample Input 1

```
5 2 3
CJ D5 HA C2 S7
D5 HA
CJ S7 C2
```

Output for the Sample Input 1

```
4
```

Sample Input 2

```
6 3 3
C9 HK SQ SQ H2 CA
CA HK SQ
H2 C9 SQ
```

Output for the Sample Input 2

```
4
```

Sample Input 3

```
7 3 4
S9 C10 DJ S6 S7 SA DQ
DJ S6 S7
S9 C10 SA DQ
```

Output for the Sample Input 3

```
7
```

Problem L
Two Choreographies
Time Limit: 2.5 Seconds

Somin and Eunjoo are famous dancers and very talented choreographers, but they haven't won a contest recently. To win the contest this year, they are trying to help each other to make new choreographies. Actually, nobody has tried smoothly appending static motions, and they are going to give it a try for the first time!

Somin and Eunjoo want to make two choreographies consisting of n static motions for each of them. They have a good understanding of how to smoothly append static motions, and they concluded that exactly $2n - 3$ *unordered* pairs of static motions are enough for them to perform freely. The order of static motions in a pair $\{A, B\}$ does not matter, i.e., if motion B can be appended after motion A, then A can also be appended after B.

The choreographies which Somin and Eunjoo want to perform are as follows. The two choreographies last for the same amount of time, which means that each one should consist of the same number of static motions. Each choreography should end at its first static motion. More precisely, two choreographies C_1 and C_2 are sequences of distinct l static motions, $C_1 = (a_0, a_1, \dots, a_l)$ and $C_2 = (b_0, b_1, \dots, b_l)$ where $a_0 = a_l$ and $b_0 = b_l$. For the entertainment of the audience, C_1 and C_2 should be different, that is, there should be some $0 \le i \le l - 1$ which $\{a_i, a_{i+1}\}$ in C_1 is not equal to any of $\{b_j, b_{j+1}\}$ in C_2 for $0 \le j \le l - 1$. (For example, (1,2,3,4,5,1) and (3,4,5,2,1,3) are different but (1,2,3,4,5,1) and (3,4,5,1,2,3) are not.) Also, the audience easily gets bored, so the choreography should not be too short, and contain at least 3 distinct static motions, that is, $l \ge 3$.

For this, you are given $2n - 3$ unordered pairs P of static motions from n distinct static motions $m_1, \dots, m_n$ that two dancers can perform. For a pair $\{m_i, m_j\}$ where $i \ne j$, one of m_i and m_j can appear after the other in the sequence; there is no specific order between them. You should write a program to find two different choreographies $C_1 = (a_0, a_1, \dots, a_l)$ and $C_2 = (b_0, b_1, \dots, b_l)$ of the same length $l \ge 3$ such that $\{a_i, a_{i+1}\} \in P$, $\{b_i, b_{i+1}\} \in P$ for any $0 \le i \le l - 1$, and $a_0 = a_l$ and $b_0 = b_l$.

Input
Your program is to read from standard input. The input starts with a line containing a single integer, n ($4 \le n \le 100{,}000$), where n is the number of static motions two dancers can represent. Each static motion is numbered as an integer from 1 to n. The following $2n - 3$ lines represent $2n - 3$ unordered pairs of stack motions, P. Each line contains two distinct integers representing two static motions of a pair of P. Note that no two pairs in P are identical.

Output
Your program is to write to standard output. If you cannot find two choreographies of static motions, then print `-1`. If not, you should print exactly three lines. The first line contains an integer $l \ge 3$ which is the number of distinct static motions in each choreography. The second line contains exactly l integers, separated by a space, each representing a choreography of the l static motions in order. The last repeated motion should be omitted. The third line contains exactly l integers representing the other choreography.

The following shows sample input and output for three test cases.

Sample Input 1

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

Output for the Sample Input 1

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

Sample Input 2

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

Output for the Sample Input 2

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

Sample Input 3

```
7
1 2
3 4
5 6
5 2
3 1
6 1
4 2
4 5
2 6
3 6
1 5
```

Output for the Sample Input 3

```
4
6 1 5 2
4 2 1 3
```