Codeforces Round #138 (Div. 1)

A. Bracket Sequence

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A *bracket sequence* is a string, containing only characters "(", ")", "[" and "]".

A *correct bracket sequence* is a bracket sequence that can be transformed into a correct arithmetic expression by inserting characters "1" and "+" between the original characters of the sequence. For example, bracket sequences "()[]", "([])" are correct (the resulting expressions are: "(1)+[1]", "([1+1]+1)"), and "](" and "[" are not. **The empty string is a correct bracket sequence by definition.**

A *substring* $s[l... r]$ $(1 \le l \le r \le |s|)$ of string $s = s_1 s_2 ... s_{|s|}$ (where $|s|$ is the length of string s) is the string $s_l s_{l+1} ... s_r$. **The empty string is a substring of any string by definition**.

You are given a bracket sequence, not necessarily correct. Find its substring which is a correct bracket sequence and contains as many opening square brackets «[» as possible.

Input

The first and the only line contains the bracket sequence as a string, consisting only of characters "(", ")", "[" and "]". It is guaranteed that the string is non-empty and its length doesn't exceed 10^5 characters.

Output

In the first line print a single integer — the number of brackets «[» in the required bracket sequence. In the second line print the optimal sequence. If there are more than one optimal solutions print any of them.

Sample test(s)

input
([])
output
1 ([])

input
(((
output
0

B. Two Strings

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A *subsequence* of length $|x|$ of string $s = s_1 s_2 \ldots s_{|s|}$ (where $|s|$ is the length of string s) is a string $x = s_{k_1} s_{k_2} \ldots s_{k_{|x|}}$ $(1 \le k_1 < k_2 < \ldots < k_{|x|} \le |s|)$.

You've got two strings — s and t. Let's consider all subsequences of string s, coinciding with string t. Is it true that each character of string s occurs in at least one of these subsequences? In other words, is it true that for all i $(1 \le i \le |s|)$, there is such subsequence $x = s_{k_1} s_{k_2} \ldots s_{k_{|x|}}$ of string s, that $x = t$ and for some j $(1 \le j \le |x|)$ $k_j = i$.

Input

The first line contains string s, the second line contains string t. Each line consists only of lowercase English letters. The given strings are non-empty, the length of each string does not exceed $2 \cdot 10^5$.

Output

Print "`Yes`" (without the quotes), if each character of the string s occurs in at least one of the described subsequences, or "`No`" (without the quotes) otherwise.

Sample test(s)

input
abab ab
output
Yes

input
abacaba aba
output
No

input
abc ba
output
No

Note

In the first sample string t can occur in the string s as a subsequence in three ways: **ab**ab, **a**ba**b** and ab**ab**. In these occurrences each character of string s occurs at least once.

In the second sample the 4-th character of the string s doesn't occur in any occurrence of string t.

In the third sample there is no occurrence of string t in string s.

C. Partial Sums

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You've got an array a, consisting of n integers. The array elements are indexed from 1 to n. Let's determine a two step operation like that:

1. First we build by the array a an array s of partial sums, consisting of n elements. Element number i ($1 \le i \le n$) of array s equals $s_i = \left(\sum_{j=1}^{i} a_j \right) \mod (10^9 + 7)$. The operation $x \ mod \ y$ means that we take the remainder of the division of number x by number y.
2. Then we write the contents of the array s to the array a. Element number i ($1 \le i \le n$) of the array s becomes the i-th element of the array a ($a_i = s_i$).

You task is to find array a after exactly k described operations are applied.

Input

The first line contains two space-separated integers n and k ($1 \le n \le 2000, 0 \le k \le 10^9$). The next line contains n space-separated integers $a_1, a_2, ..., a_n$ — elements of the array a ($0 \le a_i \le 10^9$).

Output

Print n integers — elements of the array a after the operations are applied to it. Print the elements in the order of increasing of their indexes in the array a. Separate the printed numbers by spaces.

Sample test(s)

input
3 1 1 2 3
output
1 3 6

input
5 0 3 14 15 92 6
output
3 14 15 92 6

D. Spider

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A plane contains a not necessarily convex polygon without self-intersections, consisting of n vertexes, numbered from 1 to n. There is a spider sitting on the border of the polygon, the spider can move like that:

1. *Transfer.* The spider moves from the point p_1 with coordinates (x_1, y_1), lying on the polygon border, to the point p_2 with coordinates (x_2, y_2), also lying on the border. The spider can't go beyond the polygon border as it transfers, that is, the spider's path from point p_1 to point p_2 goes along the polygon border. It's up to the spider to choose the direction of walking round the polygon border (clockwise or counterclockwise).
2. *Descend.* The spider moves from point p_1 with coordinates (x_1, y_1) to point p_2 with coordinates (x_2, y_2), at that points p_1 and p_2 must lie on one vertical straight line ($x_1 = x_2$), point p_1 must be not lower than point p_2 ($y_1 \geq y_2$) and segment p_1p_2 mustn't have points, located strictly outside the polygon (specifically, the segment can have common points with the border).

Initially the spider is located at the polygon vertex with number s. Find the length of the shortest path to the vertex number t, consisting of transfers and descends. The distance is determined by the usual Euclidean metric $|p_1p_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Input

The first line contains integer n ($3 \leq n \leq 10^5$) — the number of vertexes of the given polygon. Next n lines contain two space-separated integers each — the coordinates of the polygon vertexes. The vertexes are listed in the counter-clockwise order. The coordinates of the polygon vertexes do not exceed 10^4 in their absolute value.

The last line contains two space-separated integers s and t ($1 \leq s, t \leq n$) — the start and the end vertexes of the sought shortest way.

Consider the polygon vertexes numbered in the order they are given in the input, that is, the coordinates of the first vertex are located on the second line of the input and the coordinates of the n-th vertex are on the $(n + 1)$-th line. It is guaranteed that the given polygon is simple, that is, it contains no self-intersections or self-tangencies.

Output

In the output print a single real number — the length of the shortest way from vertex s to vertex t. The answer is considered correct, if its absolute or relative error does not exceed 10^{-6}.

Sample test(s)

input
```
4
0 0
1 0
1 1
0 1
1 4
```
output
```
1.0000000000000000e+000
```

input
```
4
0 0
1 1
0 2
-1 1
3 3
```
output
```
0.0000000000000000e+000
```

input
```
5
0 0
5 0
1 4
0 2
2 1
3 1
```
output
```
5.650281539872884700e+000
```

Note

In the first sample the spider transfers along the side that connects vertexes 1 and 4.

In the second sample the spider doesn't have to transfer anywhere, so the distance equals zero.

In the third sample the best strategy for the spider is to transfer from vertex 3 to point (2,3), descend to point (2,1), and then transfer to vertex 1.

E. Planar Graph

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

A graph is called *planar*, if it can be drawn in such a way that its edges intersect only at their vertexes.

An *articulation point* is such a vertex of an undirected graph, that when removed increases the number of connected components of the graph.

A *bridge* is such an edge of an undirected graph, that when removed increases the number of connected components of the graph.

You've got a connected undirected planar graph consisting of n vertexes, numbered from 1 to n, drawn on the plane. The graph has no bridges, articulation points, loops and multiple edges. You are also given q queries. Each query is a cycle in the graph. The query response is the number of graph vertexes, which (if you draw a graph and the cycle on the plane) are located either inside the cycle, or on it. Write a program that, given the graph and the queries, will answer each query.

Input

The first line contains two space-separated integers n and m $(3 \le n, m \le 10^5)$ — the number of vertexes and edges of the graph. Next m lines contain the edges of the graph: the i-th line contains two space-separated integers u_i and v_i $(1 \le u_i, v_i \le n)$ — the numbers of vertexes, connecting the i-th edge. The next n lines contain the positions of the planar graph vertexes on the plane: the i-th line contains a pair of space-separated integers x_i and y_i $(|x_i|, |y_i| \le 10^9)$ — the coordinates of the i-th vertex of the graph on the plane.

The next line contains integer q $(1 \le q \le 10^5)$ — the number of queries. Then follow q lines that describe the queries: the i-th line contains the sequence of space-separated integers $k_i, a_1, a_2, ..., a_{k_i}$ $(1 \le a_j \le n;\ k_i > 2)$, where k_i is the cycle length in the i-th query, a_j are numbers of the vertexes that form a cycle. The numbers of vertexes in the cycle are given in the clockwise or counterclockwise order. The given cycles are simple, that is they cannot go through a graph vertex more than once. The total length of all cycles in all queries does not exceed 10^5.

It is guaranteed that the given graph contains no bridges, articulation points, loops and multiple edges. It is guaranteed that the edge segments can have common points only at the graph's vertexes.

Output

For each query print a single integer — the number of vertexes inside the cycle or on it. Print the answers in the order, in which the queries follow in the input. Separate the numbers by spaces.

Sample test(s)

input

```
3 3
1 2
2 3
3 1
0 0
1 0
0 1
1
3 1 2 3
```

output

```
3
```

input

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

output

```
5
```

input

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

```
0 0
1 0
1 1
0 1
3
3 1 2 4
3 4 2 3
4 1 2 3 4
```

output

```
3
3
4
```