Task 1: Fraud (`fraud`)

You have been put in charge of the 24^{th} National Olympiad in Informatics!

This year, the competition consisted of N contestants and 2 rounds. The i^{th} contestant scored A_i points in the first round and B_i points in the second round.

Furthermore, the rounds have associated **positive integer** weightages X and Y respectively. The i^{th} contestant's final score S_i is given by $S_i = A_i \times X + B_i \times Y$.

As the chairman, you have been given the freedom to select the values of X and Y as you wish.

Alas, Squeaky the Mouse has bribed you into committing fraud. To be exact, he has promised to reward you handsomely if you select some X and Y such that $S_i > S_j$ for all $1 \leq i < j \leq N$.

But is it even possible to do so?

Input

Your program must read from standard input.

The first line contains a single integer N, the number of contestants.

The second line contains N space-separated integers, $A_1, \ldots, A_N$.

The third line contains N space-separated integers, $B_1, \ldots, B_N$.

Output

Your program must print to standard output.

Output `YES` if it is possible to commit fraud and `NO` otherwise.

Implementation Note

As the input lengths for subtasks 1 and 4 may be very large, you are recommended to use C++ with fast input routines to solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Fraud.java` and place your main function inside `class Fraud`.

Subtasks

The maximum execution time on each instance is 1.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $2 \leq N \leq 3 \times 10^5$
- $0 \leq A_i, B_i \leq 10^6$

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Points	Additional Constraints
1	10	$B_i = 0$
2	25	$N = 2$
3	50	$2 \leq N \leq 10^4$
4	15	-

Sample Testcase 1

This testcase is valid for subtasks 2, 3, and 4 only.

Input	Output
2 1 2 2 1	YES

Sample Testcase 1 Explanation

A possible solution is $X = 1$ and $Y = 2$.

This is since $S_1 = 1 \times 1 + 2 \times 2 = 5 > S_2 = 2 \times 1 + 1 \times 2 = 4$.

Sample Testcase 2

This testcase is valid for subtasks 3 and 4 only.

Input	Output
3 2 4 3 4 2 3	NO

Sample Testcase 3

This testcase is valid for all subtasks.

Input	Output
2 5 1 0 0	YES

Task 2: Archaeologist (`archaeologist`)

A team of archaeologists are exploring some ruins, but many of them are running late, hence the earlier archaeologists are entering the ruins before the others arrive.

The K archaeologists each have a map of the ruins, and they note that there are N rooms in the ruins, labelled from 0 to $N-1$ inclusive, and some pairs of rooms are directly connected by bidirectional corridors. Furthermore, there are no cycles in the ruins, and all rooms are reachable from the entrance, which is directly connected to room 0.

Each archaeologist does not have time to backtrack (i.e. they cannot take a corridor that they have previously used in the opposite direction), nor do they have time to check how many archaeologists have already entered the ruins. However, they can change the illumination level of the room that they are standing in to any level from 0 to L inclusive before taking a corridor to the next room. All rooms in the ruins have an illumination level of 0 before the first archaeologist arrives, and will remain so until some archaeologist changes it. When currently inside a room, they cannot tell which room each corridor leads to (apart from the corridor from which they came, which they are not allowed to take), but they can tell which room they are currently in, and can see the illumination level of the room at the end of each corridor.

You may assume that each archaeologist arrives at the final room that they want to visit before any subsequent archaeologist enters the ruins.

Help the archaeologists come up with a strategy to be able to explore all rooms of the ruins!

Implementation Details

This is a communication task. You need to implement the strategy that each archaeologist should follow in order to collectively explore all the rooms. Remember that each archaeologist does not know how many other archaeologists have already entered the ruins.

You should implement the following procedure:

```
void archaeologist(int N, int K, int L, int[] map, int lightlevel, int[]
   paths)
```

- N: the number of rooms in the ruins
- K: the number of archaeologists
- L: the maximum illumination level
- map: an array of size N, where $map[0] = -1$ and for all $i \geq 1$ room i is directly connected to room $map[i]$
- $lightlevel$: the current illumination level of room 0

- $paths$: an array containing the illumination level at the end of each corridor of room 0 (other than the corridor from which the archaeologist entered room 0) — this array is shuffled arbitrarily before being given to you

Each invocation of the `archaeologist` procedure represents the behaviour of one archaeologist — all archaeologists start from room 0. The `archaeologist` procedure will be called exactly K times in sequence.

From within the `archaeologist` procedure, you may call the following two procedures, which are implemented for you in the grader:

```
void set_light(int level)
```

- This procedure sets the illumination level of the current room.
- $level$: the new illumination level, which must be between 0 and L inclusive

```
(int, int[]) take_path(int corridor)
```

- This procedure is called to make the archaeologist take a corridor into an adjacent room.
- $corridor$: the index of the corridor to take
- This procedure returns two values — the first value is the label of the new room that the archaeologist ends up in, and the second value is an array containing the illumination level at the end of each corridor of this new room (other than the corridor from which the archaeologist came from). Similarly, the second value is an array that is shuffled arbitrarily before being given to you.

The index of the corridor that is passed into `take_path` is the index into the $paths$ array of `archaeologist` (if we are currently in room 0) or array returned by the previous invocation of `take_path` (if we are currently in some room other than room 0).

Note: Invocations of the `archaeologist` procedure may be made on different processes, but we do not guarantee that every invocation is made on a different process from all other invocations. You should not rely on data that may be modified by previous invocations of the `archaeologist` procedure, such as global variables. Furthermore, you must not prematurely end execution of the process, e.g. by calling `exit()`.

Implementation Note

C++ and Java templates have been provided in the attachment. A grader file that reads from standard input is also provided, to aid you in your testing. We use a different grader implementation for grading, but with the same interface between our grader and your solution.

You are recommended to use C++ to solve this problem, even though the scientific committee has solutions in both C++ and Java. **It is not possible to make any submissions in Python.**

If you are implementing your solution in Java, please name your file `Archaeologist.java` and place your `archaeologist` function inside `class Archaeologist`.

Grader Input

A grader is provided for both C++ and Java (called `stub.cpp` and `stub.java` respectively), which reads from standard input. This grader is meant only to aid you when implementing your solution.

The input format for this grader is described below.

The first line contains three integers, N, K, and L, as described in the statement.

The second line contains $N - 1$ integers. The i^{th} integer is $map[i]$. $map[0]$ is implicitly set to -1, and hence is not part of the input file.

The grader prints debugging output when each archaeologist is invoked, and when `set_light` or `take_path` is called. The final line of the grader output will be `All rooms explored successfully` if your strategy managed to explore all rooms, or `Not all rooms were explored` otherwise.

Subtasks

The maximum execution time on each instance is 12.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $1 \leq N \leq 1000$
- K is equal to the number of dead-end rooms (i.e. rooms with no corridors from it apart from the corridor from which the archaeologist entered the room)

Note that the time limit is sufficient for each archaeologist to call `set_light` once in every room that they visit.

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	**Points**	**Value of L**	**Additional Constraints**
1	6	$L = 1$	$map[i] = 0$ for all $1 \leq i < N$
2	14	$L = N$	-
3	15	L is the number of corridors between room 0 and the furthest room from it	The map forms a perfect binary tree
4	18		The map array has no unique elements except for the value -1
5	47		-

Note: The distance of a room x from room 0 is the number of corridors along the shortest path from room 0 to room x. The furthest room from room 0 is the room with largest distance from room 0. When we say that the map forms a perfect binary tree, we mean that each dead-end room has the same distance from room 0, and at each non-dead-end room the archaeologist has exactly two corridors to choose from (apart from the corridor from which they entered the room).

Sample Interaction

This testcase is valid for subtasks 3, 4, and 5.

Grader input

Input
`7 4 2` `0 0 1 1 2 2`

Sample interaction for archaeologist #1

Grader calls `archaeologist(7, 4, 2, [-1, 0, 0, 1, 1, 2, 2], 0, [0, 0])`	
Function call	**Return value**
`set_light(1)`	-
`take_path(0)`	`(1, [0, 0])`
`take_path(1)`	`(4, [])`
`set_light(2)`	-

Sample interaction for archaeologist #2

Grader calls `archaeologist(7, 4, 2, [-1, 0, 0, 1, 1, 2, 2], 1, [0, 0])`	
Function call	**Return value**
`set_light(0)`	-
`take_path(0)`	`(2, [0, 0])`
`set_light(1)`	-
`take_path(1)`	`(6, [])`
`set_light(1)`	-

Sample interaction for archaeologist #3

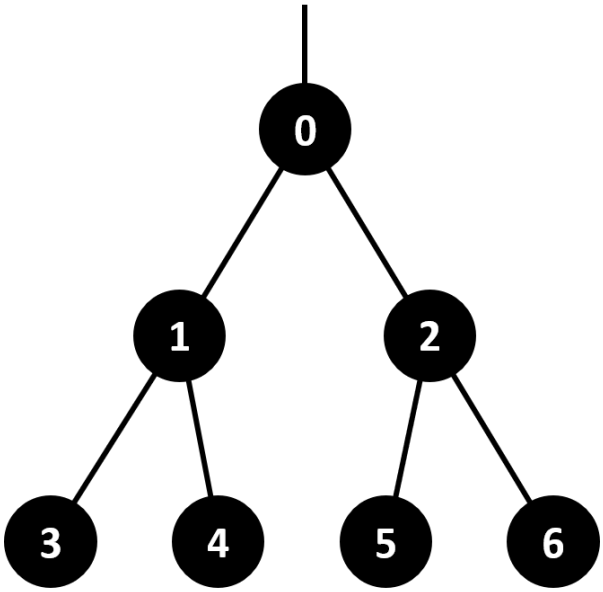
Grader calls `archaeologist(7, 4, 2, [-1, 0, 0, 1, 1, 2, 2], 0, [1, 0])`	
Function call	**Return value**
`set_light(2)`	-
`take_path(1)`	`(1, [2, 0])`
`set_light(0)`	-
`take_path(1)`	`(3, [])`

Sample interaction for archaeologist #4

Grader calls `archaeologist(7, 4, 2, [-1, 0, 0, 1, 1, 2, 2], 2, [1, 0])`	
Function call	**Return value**
`set_light(1)`	-
`take_path(0)`	`(2, [0, 1])`
`take_path(0)`	`(5, [])`
`set_light(1)`	-

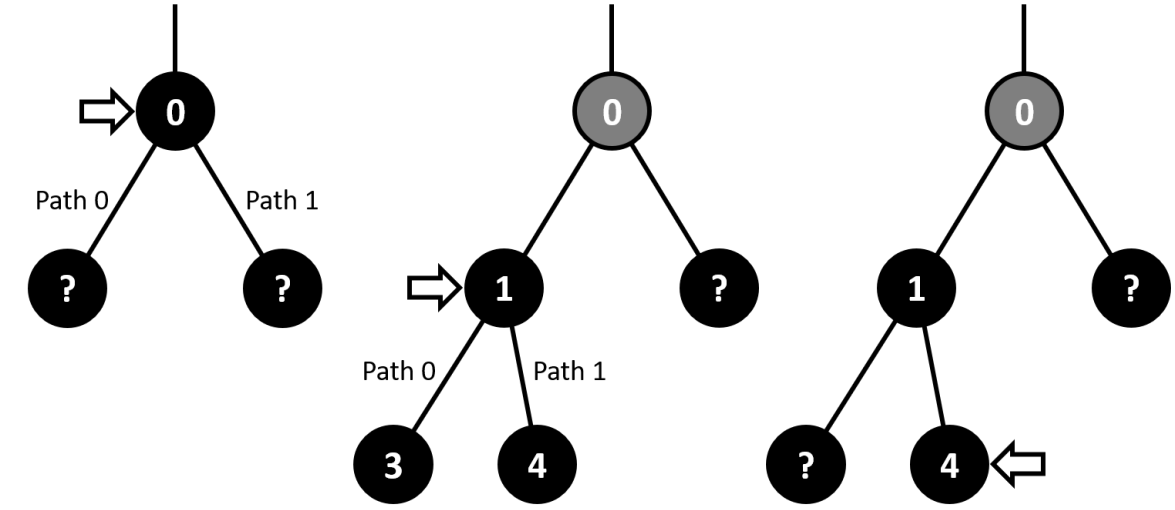
Sample Interaction Explanation

The map of the ruins looks like this, where the entrance is directly connected to room 0:

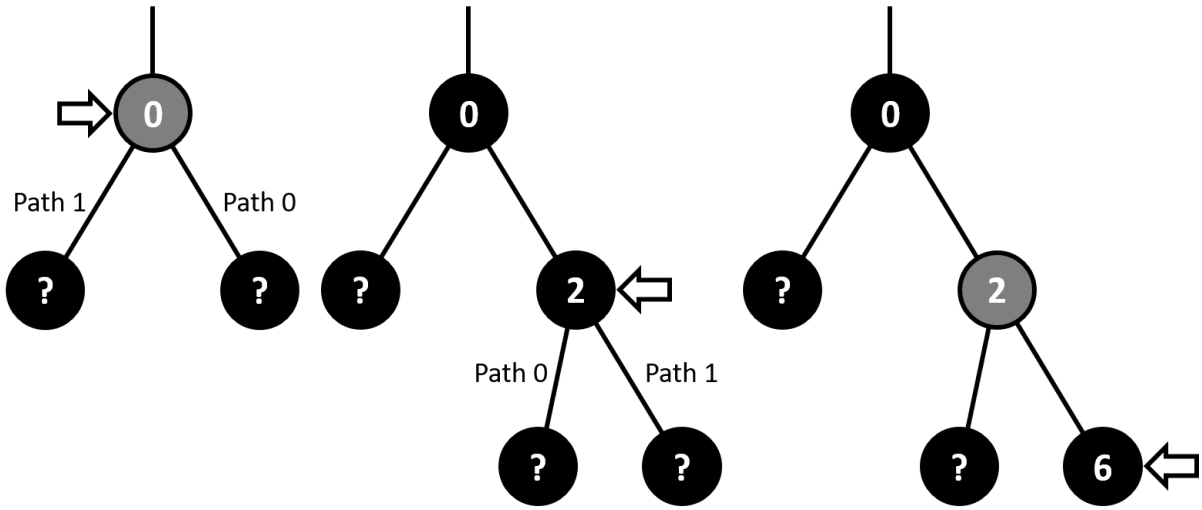


The diagrams that follow show, for each archaeologist, initially and after each call of `take_path`, where the archaeologist is and the information that they have been given. The diagrams are ordered from left to right, as the archaeologist goes deeper into the ruins. The arrow denotes the current location of the archaeologist. Black circles represent a room with an illumination level of 0, gray circles represent a room with an illumination level of 1, and white circles represent a room with an illumination level of 2, the maximum allowed. The numbers in the circles represent the room number, with question marks representing room numbers that the archaeologist does not yet know. Remember that the list of illumination levels presented to your program may be shuffled arbitrarily.

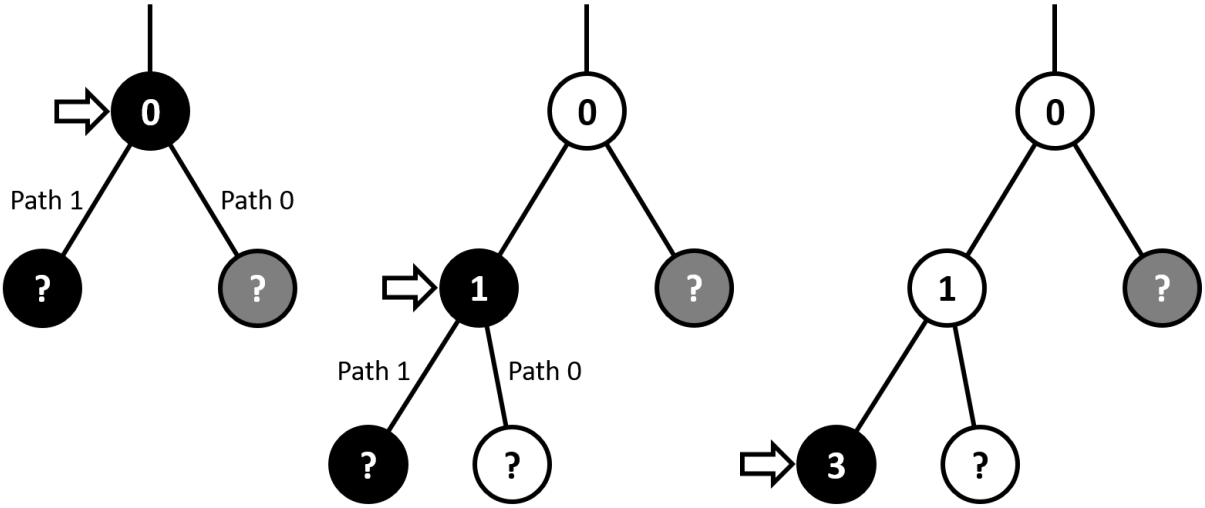
Diagrams for archaeologist #1



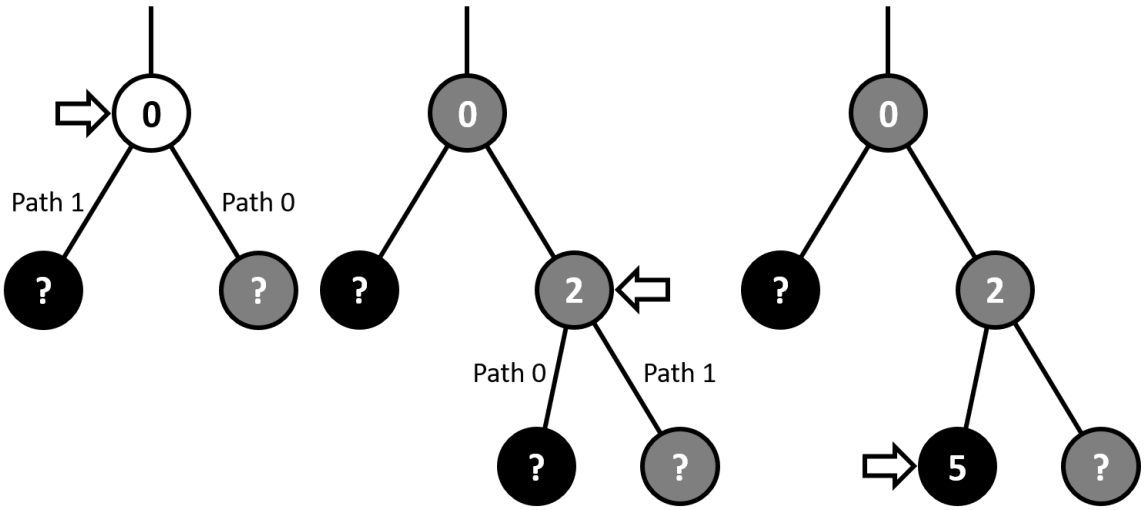
Diagrams for archaeologist #2



Diagrams for archaeologist #3



Diagrams for archaeologist #4



After all four archaeologists have executed their strategies, all rooms will have been explored by at least one archaeologist.

Task 3: Password (`password`)

The elusive Phantom Thief Lupin the Fourth (usually addressed as just Lupin) has taken on the daunting task of breaking into the vault of the largest, most majestic, most technologically advanced castle in the world: The Castle of Gagliostro. Equipped with state of the art technology, it goes without saying that the castle has top-class security which fully utilises the capabilities of man and machine. Nevertheless, Lupin was unfazed even when faced when such a challenge. He stealthily evaded the guards and security cameras positioned around the castle, fought through numerous traps, and solved many puzzles before finally arriving at the vault, where he hacked the security cameras to show a feed that is not at all suspicious and drugged the security guards to render all of them unconscious.

The final obstacle standing in his way is an electronic password system which will open the vault once the correct password is entered. Initially, N non-negative integers between 0 and K inclusive are shown on the screen, the i^{th} of which is equal to A_i. The password also consists of N numbers, each of which is also a non-negative integer between 0 and K inclusive. Through his sheer, blinding brilliance in both mind and body, Lupin determined that the i^{th} number in the password is P_i for each $1 \leq i \leq N$. Upon figuring out the password, Lupin was elated and all ready to complete his mission. To his dismay, however, the way the password is input is rather peculiar, and very time consuming, involving operations which alter the numbers shown on the screen slowly but surely.

Suppose the i^{th} number currently shown on the screen is C_i ($1 \leq i \leq N$). Lupin can open the vault when $C_i = P_i$ for all $1 \leq i \leq n$. Otherwise, Lupin is allowed operations of the following form: choose two integers i, j with $1 \leq i \leq j \leq N$, and for all integers l with $i \leq l \leq j$, C_l changes to $C_l + 1$ (mod $K + 1$). In other words, change C_l to the remainder of $C_l + 1$ when it is divided by $K + 1$. Specifically, 0 changes to 1, 1 changes to 2, ..., $K - 1$ changes to K and K changes to 0.

As if the terrible password input system wasn't bad enough, Lupin has also received information that his archnemesis Inspector Zenigada is rushing over to catch him once and for all. In order to break into the vault and escape as quickly as possible, Lupin wants to use the minimum number of operations needed to change the initial numbers to become the password and open the vault; your task is to determine the minimum number of operations Lupin needs.

Input

Your program must read from standard input.

The first line of the input contains two integers, N and K.

The next line contains N integers $A_1, A_2, \ldots, A_N$, where A_i is the i^{th} number shown on the screen initially.

The third and final line contains N integers $P_1, P_2, \ldots, P_N$, where P_j is the j^{th} number in the password.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the minimum number of operations Lupin needs to change the numbers on the screen to become the password.

Implementation Note

As the input lengths for some subtasks may be very large, you are recommended to use C++ with fast input routines to solve this problem. The scientific committee does not have a solution written in Java or Python that can fully solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Password.java` and place your main function inside `class Password`.

Subtasks

The maximum execution time on each instance is 1.0s, and the maximum memory usage on each instance is 1GiB. For all test-cases, the input will satisfy the following bounds:

- $1 \leq N \leq 3 \times 10^5$
- $0 \leq K \leq 10^9$
- $0 \leq A[i], P[i] \leq K$ for all $1 \leq i \leq N$

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Points	Additional Constraints
1	6	$N \leq 3$
2	5	$A_i \leq A_{i+1}$ for all $1 \leq i \leq N-1$ $P_i = 0$ for all $1 \leq i \leq N$
3	9	$K \leq 1$
4	10	$N, K \leq 80$
5	13	$N \leq 400$
6	23	$N \leq 3000$
7	34	-

Sample Testcase 1

This testcase is valid for subtasks 1, 4, 5, 6, and 7.

Input	Output
3 4 1 2 0 2 1 2	4

Sample Testcase 1 Explanation

An optimal sequence of operations is to choose $(i, j) = (1, 3)$ once, $(2, 3)$ once and $(2, 2)$ twice.

Sample Testcase 2

This testcase is valid for subtasks 3, 4, 5, 6, and 7.

Input	Output
7 1 1 0 1 0 1 1 1 0 0 1 1 0 1 0	3

Sample Testcase 2 Explanation

An optimal sequence of operations is to choose $(i, j) = (1, 3)$ once, $(2, 6)$ once and $(6, 7)$ once.

Sample Testcase 3

This testcase is valid for subtasks 4, 5, 6, and 7.

Input	Output
7 9 1 5 3 4 8 3 2 7 4 8 3 2 3 1	18

Task 4: Tiles (`tiles`)

Having recently moved into a new house, Eustace the Sheep has decided to renovate his lavatory as he simply cannot stand the sight of its drab interior. At the moment, the toilet floor consists of a 3 by N grid of black and white squares in some initial pattern.

Eustace notices that he has a very large number of identical rectangular 1 by 2 tiles at his disposal. To preserve the aesthetic appeal of his washroom, each tile can be rotated but must be placed parallel to the walls of the toilet. Furthermore, the glue that he uses to secure the tiles in place cannot be applied to the black squares, meaning that tiles can only be placed on white squares.

Alas, the successful renovation of Eustace's bathroom is contingent on the availability of his contractor, who has unilaterally postponed their plans. In the midst of daydreaming, Eustace gazes wistfully at a section of his bathroom floor that spans columns a to b, wondering how many different patterns he can make by placing down some or none of the tiles within that region. Two patterns are considered different if two squares that share a tile in one pattern do not share a tile in the other.

Just as he has finished calculating the total number of possible patterns, he realises that some squares have been discoloured due to a variety of factors such as mould, mildew and the like. In particular, sometimes a single square in row x and column y may have its colour flipped from black to white and vice versa.

Help Eustace answer his questions by determining the number of possible patterns of tiles amid the ever-changing colour scheme of his bathroom floor!

As the answer may be large, output the remainder when the answer is divided by 1 000 000 007.

Input

Your program must read from standard input.

The first line of the input contains 2 integers N and Q denoting the length of Eustace's bathroom floor and the total number of queries and updates.

3 lines will follow, representing the initial pattern of squares. Each line contains a string of length N consisting solely of dots '.' and crosses 'x'. Here, a dot denotes a white square while a cross denotes a black square.

Q lines then follow, with each line taking on one of the following forms:

- $1\ x\ y$ which indicates an update where the colour of the square at row x and column y has been flipped.
- $2\ a\ b$ which represents a query for the number of patterns that can be formed if tiles are confined to columns a to b. Not placing any tiles also forms a pattern.

Output

Your program must print to standard output.

For each query, output on a new line the remainder when the number of possible patterns is divided by 1 000 000 007.

Implementation Note

As the input lengths for subtasks 2 and 4 may be very large, you are recommended to use C++ with fast input routines to solve this problem. The scientific committee does not have a solution written in Java or Python that can fully solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Tiles.java` and place your main function inside `class Tiles`.

Subtasks

The maximum execution time on each instance is 4.0s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $1 \leq N, Q \leq 30000$
- $1 \leq x \leq 3$
- $1 \leq y \leq N$
- $1 \leq a \leq b \leq N$

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Points	Additional Constraints
1	17	$1 \leq N, Q \leq 8$
2	23	There will never be any black squares.
3	26	$1 \leq N, Q \leq 7000$
4	34	-

Sample Testcase 1

This testcase is valid for subtasks 1, 3, and 4 only.

Input	Output
4 5 .x.x xx.. ...x 2 1 4 2 3 3 1 2 3 2 1 4 2 3 3	11 3 3 1

Sample Testcase 1 Explanation

Using $--$ to denote a tile, the 11 patterns for the first query are:

```
.x.x    .x.x    .x.x    .x.x
xx..    xx..    xx--    xx|.
...x    .--x    --.x    ..|x

.x.x    .x.x    .x|x    .x|x
xx..    xx--    xx|.    xx|.
--.x    .--x    .--x    ...x

.x.x    .x.x    .x|x
xx--    xx|.    xx|.
...x    --|x    --.x
```

For the second query, tiles are restricted to column 3. These are the 3 patterns:

```
.    .    |
.    |    |
.    |    .
```

After the first update, the bathroom floor will look like this:

```
.x.x
xxx.
...x
```

There are only 3 possible patterns now for the third query:

```
.x.x     .x.x     .x.x
xxx.     xxx.     xxx.
...x     --.x     .--x
```

For the last query, no tile can be placed in column 3 alone. There is only one pattern, the current one.

Sample Testcase 2

This testcase is valid for subtasks 1, 3, and 4 only.

Input	Output
2 1 .. .. xx 2 1 2	7

Sample Testcase 2 Explanation

Using −− to denote a tile, the 7 patterns are:

```
..     ..     .|     --     |.     --     ||
..     --     .|     ..     |.     --     ||
```

Sample Testcase 3

This testcase is valid for subtasks 2, 3, and 4 only.

Input	Output
14 2 .... 2 2 11 2 1 14	47177097 254767228

Task 5: Pond (pond)

Syrup the Turtle often swims in a pond next to his house. Having been carved out by glacial movements long ago, the pond is narrow and straight — shaped almost like a river, but with waters calm and still enough to allow a turtle to swim both ways unimpeded.

Today, Syrup was in the pond as usual when he caught a glance of the dreaded green speck — a blooming algal spore. After bouts of heavy rain, the rich soil washed into the pond gradually disintegrates and provides the nutrition for the normally benign local algae to grow at a massively accelerated rate. If left unchecked, these blooms could expand to the point where they block out sunlight from reaching the lake-bed plants below; setting the stage for ecological imbalances which could mar the water for months on end.

Fortunately, Syrup is no stranger to this game and has a simple but effective answer to this infrequent problem — eating it. He has identified N soil runoff points in the linear pond where algae is starting to bloom, which can be numbered from one end to the other as 1 through N. The i^{th} and $(i+1)^{\text{th}}$ points are separated by a distance of D_i metres, and Syrup is currently at the K^{th} spot alongside the spore he first noticed. He will now swallow down that very spore, before swimming off in one of the two directions at a speed of 1 metre per second and eating up every cluster of algae he passes until all blooms are gone.

Each of the N runoff points starts off with 0 algal strands, and will gain 1 strand every second until Syrup reaches it. Turtles are robust and Syrup has no difficulty eating any number of algal strands. However, as overgrown algae tastes no good, he would prefer to minimise the number of strands eaten over his trip. Your task is to find the fewest number of total algae strands Syrup has to eat to clear the pond of algae, given that he takes the best route along it.

Input

Your program must read from standard input.

The first line contains two integers, N and K.

The second line contains $N-1$ integers. The i^{th} integer represents D_i, the distance in metres between runoff points i and $i+1$.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the minimum possible total algal strands Syrup must eat to remove all the algae from the pond.

Implementation Note

As the input lengths for subtasks 3, 4, 5, 6, and 7 may be very large, you are recommended to use C++ with fast input routines to solve this problem. The scientific committee does not have a solution written in Java or Python that can fully solve this problem.

C++ and Java source files containing fast input/output templates have been provided in the attachment. You are strongly recommended to use these templates.

If you are implementing your solution in Java, please name your file `Pond.java` and place your `main` function inside `class Pond`.

Subtasks

The maximum execution time on each instance is 1.5s, and the maximum memory usage on each instance is 1GiB. For all testcases, the input will satisfy the following bounds:

- $2 \leq N \leq 3 \times 10^5$
- $1 \leq K \leq N$
- $1 \leq D_i \leq 10^6$

Your program will be tested on input instances that satisfy the following restrictions:

Subtask	Points	Additional Constraints
1	7	$N \leq 100$
2	11	$N \leq 2000$
3	10	$1 \leq K \leq \min(N, 20)$
4	6	$D_i = 1$
5	12	$1 \leq K \leq \min(N, 2000)$ $D_i \geq D_{i+1}$ for all $i \not\equiv 0 \pmod{100}$
6	25	$1 \leq K \leq \min(N, 2000)$
7	29	-

Sample Testcase 1

This testcase is valid for subtasks 1, 2, 3, 6, and 7.

Input	Output
7 3 5 2 4 2 2 5	86

Sample Testcase 1 Explanation

The optimal route is to swim between the points in the order $3 \longrightarrow 2 \longrightarrow 4 \longrightarrow 5 \longrightarrow 6 \longrightarrow 7 \longrightarrow 1$, for a total of $0 + 2 + 8 + 10 + 12 + 17 + 37 = 86$ algal strands eaten.

Sample Testcase 2

This testcase is valid for subtasks 1, 2, 3, 6, and 7.

Input	Output
9 5 4 3 2 1 1 3 6 10	129

Sample Testcase 2 Explanation

The optimal route is to swim between the points in the order $5 \longrightarrow 6 \longrightarrow 4 \longrightarrow 3 \longrightarrow 2 \longrightarrow 1 \longrightarrow 7 \longrightarrow 8 \longrightarrow 9$, for a total of $0 + 1 + 3 + 5 + 8 + 12 + 26 + 32 + 42 = 129$ algal strands eaten.

Sample Testcase 3

This testcase is valid for all subtasks.

Input	Output
6 4 1 1 1 1 1	21

Sample Testcase 3 Explanation

One optimal route is to swim between the points in the order $4 \longrightarrow 3 \longrightarrow 2 \longrightarrow 1 \longrightarrow 5 \longrightarrow 6$, for a total of $0 + 1 + 2 + 3 + 7 + 8 = 21$ algal strands eaten.