Problem A. Max or Min

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Kevin has n integers $a_1, a_2, \ldots, a_n$ arranged in a **circle**. That is, the numbers a_i and a_{i+1} $(1 \le i < n)$ are neighbors. The numbers a_1 and a_n are neighbors as well. Therefore, each number has exactly two neighbors.

In one minute, Kevin can set a_i to the minimum among three numbers: a_i and it's two neighbors. Alternatively, Kevin can set a_i to the maximum among the same numbers. For example, if $a_i = 5$ and a_i has two neighbors 3 and 2, and Kevin performs the minimum operation, a_i will be equal to 2. However, if he performs the maximum operation, a_i will remain 5.

For each x from 1 to m, find the minimum number of minutes to make all numbers equal x, or determine that it is impossible to do so.

Input

The first line contains two integers n and m $(3 \le n \le 2 \cdot 10^5, 1 \le m \le 2 \cdot 10^5)$ — the number of integers in the circle, and the number of integers you need to find answers for.

The second line contains n integers $a_1, a_2, \ldots, a_n$ $(1 \le a_i \le m)$ — the integers in the circle.

Output

Print m integers. The i-th integer should be equal to the minimum number of minutes that are needed to make all numbers equal i or -1 if it's impossible.

Example

standard input	standard output
7 5 2 5 1 1 2 3 2	5 5 7 -1 6

Note

To make all numbers equal 2 Kevin needs at least 5 minutes. One of the possible sequence of operations:

1. Apply min operation to a_6, it will be equal to 2.
2. Apply max operation to a_4, it will be equal to 2.
3. Apply max operation to a_3, it will be equal to 5.
4. Apply min operation to a_2, it will be equal to 2.
5. Apply min operation to a_3, it will be equal to 2.

Problem B. Level Up

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

Being a fan of MMORPGs, Steve was really excited when World of Warcraft Classic was announced. He started playing from the first day and now has only two levels to go till the maximum level. Of course, he doesn't have as much time as he used to have when the game first appeared, so he really wants to finish these two levels as fast as possible.

In order to pass the first level, Steve needs s_1 experience. Only after he gains it, he can move to the second level, in which he needs to get other s_2 experience to pass it.

Steve has the list of n available quests. He knows that he may finish two levels with these quests. In order to pass the i-th quest, Steve needs t_i minutes. As a result, he will get x_i experience for this quest.

When Steve finishes a quest that takes him to the next level, the extra experience overflow is subtracted from the next level's required experience. Once he levels up, all the quests that are left will offer less experience y_i, but they will also require less time r_i.

Note that if Steve finishes a quest, he **can't** repeat this quest anymore (even in another level).

Given the list of quests, help Steve choose the order in which he will be doing the quests in order to finish the last two levels as fast as possible.

Input

The first line contains three integers n, s_1, s_2 ($1 \le n, s_1, s_2 \le 500$) — the number of quests, experience required for the first level and experience required for the second level.

Each of the next n lines contains four integers x_i, t_i, y_i, r_i ($1 \le y_i < x_i \le 500$, $1 \le r_i < t_i \le 10^9$) where x_i and y_i are the experience you will gain from the i-th quest on the 1-st and 2-nd level respectively and t_i and r_i are the number of minutes you need to spend in order to pass this quest on the 1-st and 2-nd level respectively,

Output

Print one number, representing the minimum number of minutes that Steve needs to finish two levels, or -1 if there is no way to complete quests to finish both levels.

Examples

standard input	standard output
2 100 100 100 100 10 10 101 11 100 10	110
4 20 20 40 1000 20 20 6 6 5 5 10 10 1 1 10 10 1 1	40
2 20 5 10 10 5 5 10 10 5 5	-1

Problem C. Find the Array

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

This is an interactive problem.

There is an array a of length n, consisting of **distinct** integers. It is guaranteed that every element of the array is a positive integer less than or equal to 10^9. You have to find out the values of all the elements of it.

To do so, you can make up to 30 queries of the following two types:

- "`1` i" ($1 \le i \le n$) — ask the value of a_i
- "`2` k $i_1, i_2, \ldots, i_k$" ($2 \le k \le n$, $1 \le i_j \le n$, all i_j must be distinct) — the number k and k positions in the array. As the answer to this query you will receive $\frac{k \cdot (k-1)}{2}$ integers — $| a_{i_c} - a_{i_d} |$ for every $c < d$. In other words, you will receive $\frac{k \cdot (k-1)}{2}$ absolute values of differences between all pairs of elements that are on positions $i_1, i_2, \ldots, i_k$. Note that the answer on query 2 is randomly shuffled.

Once you know the answer, print it using the following query:

- "`3` $a_1, a_2, \ldots, a_n$" ($1 \le a_i \le 10^9$) — the elements of the array a. After this query, your program must terminate. This query doesn't count (i.e. you can make up to 30 queries of either of the first two types plus 1 query of the third type).

Interaction Protocol

At the beginning, your program should read one integer n ($1 \le n \le 250$) — the number of elements.

In order to make a query of the first type, print "`1` i" ($1 \le i \le n$). After this query, read one integer — the value of a_i.

In order to make a query of the second type, print "`2` k" ($2 \le k \le n$). Then in the same line print k space-separated distinct integers — $i_1, i_2, \ldots, i_k$, ($1 \le i_j \le n$). After this query read $\frac{k \cdot (k-1)}{2}$ integers — $| a_{i_c} - a_{i_d} |$ for every $c < d$. These values will be given in random order.

Once you know the answer, print "`3`". Then in the same line, print n space-separated integers — $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$). Your program must terminate after this query.

If for either of two first queries you get one number -1 as the answer, then it means that you made more queries than allowed, or made an invalid query. Your program should immediately terminate (for example, by calling `exit(0)`). You will receive "`Wrong Answer`". If you ignore this, you can get other verdicts since your program will continue to read from a closed stream.

After printing a query do not forget to output end of line and flush the output. Otherwise, you will get `Wall time-limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;

- `stdout.flush()` in Python.

Precisely follow this format of interaction.

Example

standard input	standard output
3 1 2 4 3 1	 1 1 1 2 2 3 1 2 3 3 1 2 5

Note

In the first query of type 1, we ask the value of a_1 and receive 1 as the answer.

In the second query of type 2, we ask the value of a_2 and receive 2 as the answer.

In the query of type 2, we ask all the possible differences between the elements of array with indexes 1, 2 and 3. And we get array $4, 3, 1$ as the result. We know that the array contains values $|a_1 - a_2|, |a_1 - a_3|, |a_2 - a_3|$. Since we already know that $|a_2 - a_1| = 1$, one of the following is true: $|a_1 - a_3| = 3$ and $|a_2 - a_3| = 4$ or $|a_2 - a_3| = 3$ and $|a_1 - a_3| = 4$. The only case that is possible, taking into account the constraints of the problem, is when $|a_1 - a_3| = 4$ and $|a_2 - a_3| = 3$ with $a_3 = 5$.

Since we know the values of all the elements of the array, we print them in the last query.

Problem D. Cycle String?

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Great wizard gave Alice and Bob a cycle string of length $2 \cdot n$, which has no repeated substrings of length n. In a cycle string, character s_{i+1} comes after s_i. Also, s_1 comes after s_{2n}.

Unfortunately, evil gin shuffled all the symbols of the string. Help Alice and Bob restore the original string so that the above condition is satisfied.

Input

The first line contains one string s of length $2 \cdot n$ ($2 \le 2 \cdot n \le 1\,000\,000$) which consists only of the lowercase Latin letters.

Output

Print "`NO`" (without quotes) to the first line if it is impossible to restore the string so that the condition is satisfied. Otherwise, in the first line print "`YES`" (without quotes).

In the second line print one string — the restored string.

If there are multiple answers, print any.

Examples

standard input	standard output
`cbbabcacbb`	`YES` `abbabcbccb`
`aa`	`NO`
`afedbc`	`YES` `afedbc`

Note

In the first example, substrings of the restored string are: "`abbab`", "`bbabc`", "`babcb`", "`abcbc`", "`bcbcc`", "`cbccb`", "`bccba`", "`ccbab`", "`cbabb`", "`babba`".

Note that the first example does not contain repetitions, however it can be rewritten as another cycle with no repetitions. Thus, the solution is not unique — the given example is also a correct solution.

In the second example, it is impossible to restore the string so that no repetition exists.

In the third example, there is no need to change anything.

Problem E. Life Transfer

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Note: "feli" is the local currency.

In the great city of Nekoresti, there are n people for which we know their ages: a_i is the age of the i-th person. Currently, they are on vacation, so they decided to go on a trip to Pisiev to visit a Koshkseum, a famous museum. They can go either by car or by motorcycle:

- a **car** can transport k people (one driver which has to be at least l_c years old and $k-1$ passengers). The cost to rent a car is p_c feli.
- a **motorcycle** can transport only one person (which has to be at least l_m years old). The cost to rent a motorcycle is p_m feli.

Unfortunately, people have money issues, so they decided to consult Mewlin, the great local magician from the city. Using a formidable spell called Mucadabra, Mewlin can transfer age from one person to another. Formally, he can reduce the age x of a person and increase the age y of another person by the same amount (so the sum of ages is constant). The cost to transfer 1 unit of age is t feli. For magic medical reasons, the age of a person cannot be changed by more than d years (if the initial age is x, his age must be at least $x-d$ and at most $x+d$ at all times). Also, the age cannot go below 1 year old.

Help the people from Nekoresti to spend as little money as possible, so they can arrive in Pisiev.

Input

The first line contains two integers n and k ($1 \le n, k \le 10^5$) — the number of people and the maximum number of people that can be in one car.

The second line contains four integers l_c, p_c, l_m and p_m ($1 \le l_m < l_c \le 10^5$, $1 \le p_m < p_c \le 10^5$) — the minimum needed age to drive a car; the price of renting one car; the minimum needed age to drive a motorcycle and the price of renting one motorcycle.

The third line contains two integers t and d ($0 \le t, d \le 10^5$) — the price of transferring one year and the maximum number of times the spells can be applied per each person.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^5$) — the age of the i-th person.

Output

Print one number, the smallest amount of feli the people need to spend in order for them to reach their destination. If there is no such solution, print -1.

Examples

standard input	standard output
2 2 18 1000 16 1 5 3 16 15	1010
2 2 23 10 15 5 2 2 9 20	-1

Problem F. Game on a Tree

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Alice and Bob play a game on a tree. Initially, all nodes are white.

Alice is the first to move. She chooses any node and put a chip on it. The node becomes black. After that players take turns. In each turn, a player moves the chip from the current position to an ancestor or descendant node, as long as the node is not black. This node also becomes black. The player who cannot move the chip looses.

Who wins the game?

An *ancestor* of a node v in a rooted tree is any node on the path between v and the root of the tree.

A *descendant* of a node v in a rooted tree is any node w such that node v is located on the path between w and the root of the tree.

We consider that the root of the tree is 1.

Input

The first line contains one integer n ($1 \le n \le 100\,000$) — the number of nodes.

Each of the next $n - 1$ lines contains two integers u and v ($1 \le u, v \le n$) — the edges of the tree. It is guaranteed that they form a tree.

Output

In a single line, print "`Alice`" (without quotes), if Alice wins. Otherwise, print "`Bob`".

Examples

standard input	standard output
4 1 2 2 3 3 4	Bob
7 2 1 2 6 1 3 2 5 7 2 2 4	Alice

Note

In the first test case, the tree is a straight line and has 4 nodes, so Bob always can choose the last white node.

In the second test case, the optimal strategy for Alice is to place the chip on 3. This node will become black. Bob has to choose the node 1. Alice can choose any of 4, 5, 6, or 7. Bob can only choose 2. Alice chooses any of the white sons of 2, and Bob cannot make a move.

Problem G. Projection

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

Everybody knows that you are a `TensorFlow` fan. Therefore, you've been challenged to recreate the `TensorFlow` logo from two projections.

Consider that you have a `3D` volume, $n \times m \times h$, and two projections (two matrices with dimensions $n \times m$ and $n \times h$ with elements 0 and 1). You are asked to compute a possible sets of cubes that must be placed inside the `3D` volume such that the `3D` object created with the cubes throws the shadows specified by the projection-matrices, when the light comes from left and front. If it is not possible, just print -1. If it is possible you must find exactly two sets, one with the **maximum** amount of cubes and one with the **minimum** amount. You can assume there is no gravitation (the cubes are located inside the `3D` volume exactly where they are placed, without requiring any support). We assume that 1 represents shadow and 0 represents light.

If there are multiple such solutions, you must output the minimum lexicographic one. One solution A is lexicographically smaller than another solution b if the first number that differs between the two solutions is smaller in a than in b.

For example, solution $[(0,0,0),(1,1,1)]$ is smaller than $[(1,1,1),(0,0,0)]$.

Input

The first line contains three integers separated by a single space n, m, h ($1 \leq n, m, h \leq 100$) — the volume dimensions.

Each of the next n lines contains m characters, each being either 1 or 0 representing either a shadow area (1) or a light area (0), describing the projection from the light in the front.

Each of the next n lines contains h characters, with the same format as above, describing the projection from the light on the left.

Output

The output should contain on the first line one number, either -1 if there is no solution or k_{max} representing the maximum number of cubes we can assign in the volume that will generate the two projections given in the input.

The next k_{max} lines should contain triplets of numbers x, y, z ($0 \leq x < n$, $0 \leq y < m$, $0 \leq z < h$) representing the cubes chosen in the lexicographically smallest solution with maximum number of cubes.

Then, only if there is a solution, one more line follows containing k_{min}, the minimum number of cubes we can assign in the volume that will generate the two projections given in the input.

After that, the next k_{min} lines should contain triplets of numbers x, y, z ($0 \leq x < n$, $0 \leq y < m$, $0 \leq z < h$) representing the cubes in the lexicographically smallest solution with minimum number of cubes.

Examples

standard input	standard output
5 3 3 111 010 010 010 010 111 100 110 100 100	14 0 0 0 0 0 1 0 0 2 0 1 0 0 1 1 0 1 2 0 2 0 0 2 1 0 2 2 1 1 0 2 1 0 2 1 1 3 1 0 4 1 0 8 0 0 0 0 1 1 0 2 2 1 1 0 2 1 0 2 1 1 3 1 0 4 1 0
2 2 2 00 00 11 11	-1
2 3 2 101 011 10 11	6 0 0 0 0 2 0 1 1 0 1 1 1 1 2 0 1 2 1 4 0 0 0 0 2 0 1 1 0 1 2 1

Note

A cube at coordinates (x, y, z) will generate a shadow at line x and column y in the $n \times m$ projection and line x and column z in the $n \times h$ projection (indexed from 0).

Problem H. Tree Permutations

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Once upon a time, Mr. Cool created a tree (an undirected graph without cycles) of n vertices, by assigning to each vertex $i > 1$ two numbers: $p_i < i$ — the direct ancestor of vertex i and w_i — the weight of the edge between vertex i and p_i. Vertex 1 is the root, so it does not have any ancestors.

You wanted to know what tree did Mr. Cool build, but Mr. Cool refused to tell this, but he gave you a tip:

He wrote all these numbers in one line. That's how he got array b of length $2 \cdot n - 2$.

$$b = [p_2, w_2, p_3, w_3, \ldots, p_{n-1}, w_{n-1}, p_n, w_n]$$

Then he randomly shuffled it. That's how he got array a, and Mr. Cool presented you with it.

Since it is impossible to restore the tree knowing only values of array a, you decided to solve a different problem.

Let's call a tree `k-long`, if there are exactly k edges on the path between vertex 1 and n.

Let's call a tree `k-perfect`, if it is k-long and the sum of the weights of the edges on the path between vertex 1 and vertex n is maximal among all possible k-long trees that Mr. Cool could build.

Your task is to print the sum of the weights of the edges on the path between vertex 1 and vertex n for all possible k-perfect trees or print -1 if a certain k-long tree could not be built by Mr. Cool.

Input

The first line contains one integer n $(2 \le n \le 10^5)$ — the number of the vertices in the tree.

The second line contains $2 \cdot n - 2$ integers $a_1, a_2, \ldots, a_{2n-2} (1 \le a_i \le n - 1)$ — the elements of array a.

Output

In one line, print $n - 1$ space-separated integers $w_1, w_2, w_3, \ldots, w_{n-1}$, where w_k — the sum of the weights of the edges on the path between vertex 1 and vertex n in a k-perfect tree. If there is no i-long tree, then w_i should be equal to -1.

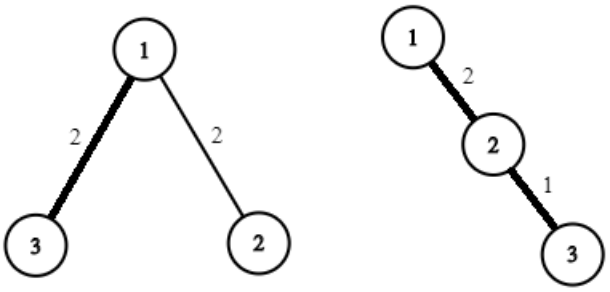
Examples

standard input	standard output
3 1 1 2 2	2 3
3 2 2 2 2	-1 -1
6 1 4 5 4 4 4 3 4 4 2	-1 -1 -1 17 20

Note

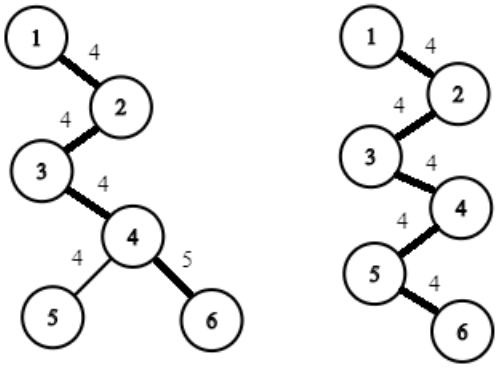
In the first example, the 1-perfect tree is defined by array $[1, 2, 1, 2]$ (i.e. $p_2 = 1$, $w_2 = 2$, $p_3 = 1$, $w_3 = 2$). The 2-perfect tree is defined by array $[1, 2, 2, 1]$ (i.e $p_2 = 1$, $w_2 = 2$, $p_3 = 2$, $w_3 = 1$). Here are illustrations

of the 1-perfect tree and the 2-perfect tree respectively (path from vertex 1 to vertex n is drawn with bold lines):



In the second example, there are no k-perfect trees, that can be obtained by permuting array a.

In the third example, only 4-perfect tree and 5-perfect tree can be obtained. These are defined by arrays $[1, 4, 2, 4, 3, 4, 4, 4, 4, 5]$ and $[1, 4, 2, 4, 3, 4, 4, 4, 5, 4]$ respectively. Here are illustrations of them:



Problem I. Absolute Game

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Alice and Bob are playing a game. Alice has an array a of n integers, Bob has an array b of n integers. In each turn, a player removes one element of his array. Players take turns alternately. Alice goes first.

The game ends when both arrays contain exactly one element. Let x be the last element in Alice's array and y be the last element in Bob's array. Alice wants to maximize the absolute difference between x and y while Bob wants to minimize this value. Both players are playing optimally.

Find what will be the final value of the game.

Input

The first line contains a single integer n ($1 \le n \le 1\,000$) — the number of values in each array.

The second line contains n space-separated integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the numbers in Alice's array.

The third line contains n space-separated integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le 10^9$) — the numbers in Bob's array.

Output

Print the absolute difference between x and y if both players are playing optimally.

Examples

standard input	standard output
4 2 14 7 14 5 10 9 22	4
1 14 42	28

Note

In the first example, the $x = 14$ and $y = 10$. Therefore, the difference between these two values is 4.

In the second example, the size of the arrays is already 1. Therefore, $x = 14$ and $y = 42$.

Problem J. Graph and Cycles

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

There is an undirected weighted complete graph of n vertices where n is odd.

Let's define a *cycle-array* of size k as an array of edges $[e_1, e_2, \ldots, e_k]$ that has the following properties:

- k is greater than 1.
- For any i from 1 to k, an edge e_i has exactly one common vertex with edge e_{i-1} and exactly one common vertex with edge e_{i+1} and these vertices are distinct (consider $e_0 = e_k$, $e_{k+1} = e_1$).

It is obvious that edges in a cycle-array form a cycle.

Let's define $f(e_1, e_2)$ as a function that takes edges e_1 and e_2 as parameters and returns the maximum between the weights of e_1 and e_2.

Let's say that we have a cycle-array $C = [e_1, e_2, \ldots, e_k]$. Let's define the *price of a cycle-array* as the sum of $f(e_i, e_{i+1})$ for all i from 1 to k (consider $e_{k+1} = e_1$).

Let's define a *cycle-split* of a graph as a set of non-intersecting cycle-arrays, such that the union of them contains all of the edges of the graph. Let's define the *price of a cycle-split* as the sum of prices of the arrays that belong to it.

There might be many possible cycle-splits of a graph. Given a graph, your task is to find the cycle-split with the minimum price and print the price of it.

Input

The first line contains one integer n ($3 \le n \le 999$, n is odd) — the number of nodes in the graph.

Each of the following $\frac{n \cdot (n-1)}{2}$ lines contain three space-separated integers u, v and w ($1 \le u, v \le n, u \ne v, 1 \le w \le 10^9$), meaning that there is an edge between the nodes u and v that has weight w.

Output

Print one integer — the minimum possible price of a cycle-split of the graph.

Examples

standard input	standard output
3 1 2 1 2 3 1 3 1 1	3
5 4 5 4 1 3 4 1 2 4 3 2 3 3 5 2 1 4 3 4 2 2 1 5 4 5 2 4 3 4 2	35

Note

Let's enumerate each edge in the same way as they appear in the input. I will use e_i to represent the edge that appears i-th in the input.

The only possible cycle-split in the first sample is $S = \{[e_1, e_2, e_3]\}$. $f(e_1, e_2)+f(e_2, e_3)+f(e_3, e_1) = 1+1+1 = 3$.

The optimal cycle-split in the second sample is $S = \{[e_3, e_8, e_9], [e_2, e_4, e_7, e_{10}, e_5, e_1, e_6]\}$. The price of $[e_3, e_8, e_9]$ is equal to 12, the price of $[e_2, e_4, e_7, e_{10}, e_5, e_1, e_6]$ is equal to 23, thus the price of the split is equal to 35.

Problem K. Stranded Robot

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	256 megabytes

A robot is stranded in the wreckage of an interstellar spaceship. There is a teleporter somewhere in the wreckage that can take the poor robot to safety.

The spaceship is spinning out of control along all axes. A nearby sun is shining light on the wreckage. The ship is also equipped with an artificial gravity generator. The artificial gravity always pulls the robot away from the sun, irrespective of the ship's orientation.

The robot is equipped with solar panels and must rely on solar power to move through the wreckage. When parts of the wreckage block out the sun, the robot is immobile. However, the robot always anchors itself after every move and does not risk thrashing about or falling into the void of space.

The wreckage and the area around it is represented by a three-dimensional grid a of size $m \times n \times p$. Each individual block can either be taken up by part of the ship or by vacuum. The ship blocks can be disconnected.

The robot starts off anchored to a piece of the ship. The robot chooses on its own when to move and when to wait for the sun to shine from a convenient direction.

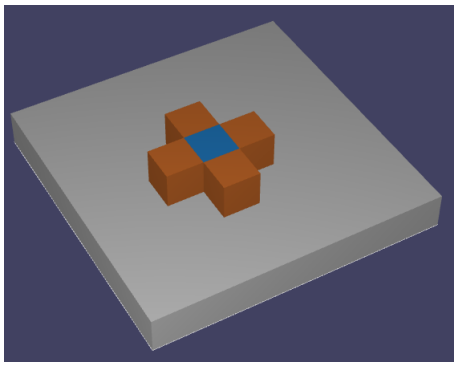
More formally, gravity can pull the robot in one of 6 directions, 2 directions along each of the 3 axes. A cell is lit by the sun if there is no wreckage from said cell in the direction opposite to gravity. Before making each move, the robot can effectively choose the direction in which it is pulled by the gravity. When making a move, both the original and the destination position must simultaneously be lit by the sun.

The following moves are allowed: (the light always shines from above in the following images; the blue block (or darker block in black-and-white printings) represents the robot, and the orange blocks (or lighter blocks) are possible destinations)

1. Moving across the floor

If the robot is resting on top of a block, it can move to an adjacent position, granted that it is at the same height.

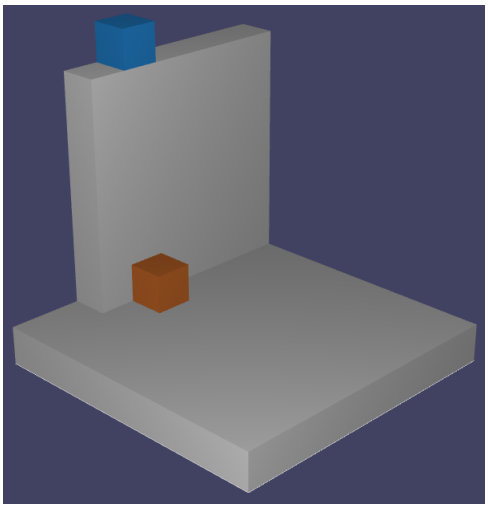
The robot can't move diagonally. The destination must also be lit by the sun.



2. Jumping off a cliff

The robot can take a step off a heightened position and subsequently drop down. There is no restriction on how long the drop is.

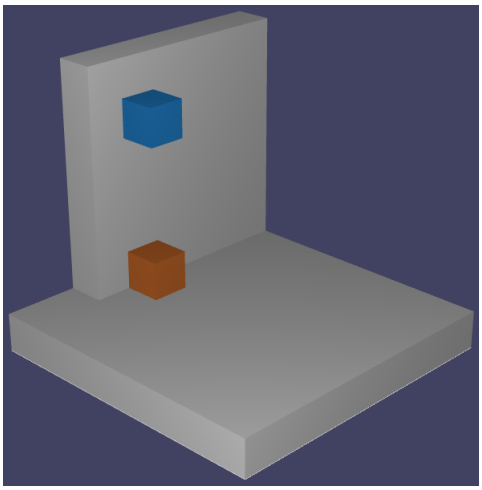
The robot can't drop down into the void of space, or to an unlit position.



3. Dropping down

If the robot is lit by the sun and it finds itself hanging, it can drop down. This can happen if the gravity's direction has changed.

The robot can't drop down into the void of space.



The goal is to reach the teleporter, if at all possible, using the smallest number of moves. The robot must be firmly anchored to the ship for the teleporter to work. In other words, the robot must find itself at the teleporter at the end of a valid move, and dropping down through it does not work. The teleporter doesn't block the sun or the robot's movement.

Input

The first line contains three integers m, n, p ($1 \leq m, n, p \leq 500$).

The spaceship and the area around the spaceship is described in p blocks.

The k-th block describes the block located on the k-th height. Each block consist of n lines.

The i-th line of the k-th block consists of m symbols. The j-th symbol is called a_{ijk}.

- If a_{ijk} is "`*`", then it is a full block.
- If a_{ijk} is "`-`", then it is an empty block.
- If a_{ijk} is "`R`", then this block contains the robot. It is guaranteed that there is only one such block. It is guaranteed that the robot is connected to a full block.
- If a_{ijk} is "`T`", then this block contains the teleporter. It is guaranteed that there is only one such block.

Output

Print the minimal number of moves required to reach the teleporter, or -1 if the teleporter is unreachable.

Examples

standard input	standard output
2 5 1 R- *- *- *T **	1
3 2 1 R-T ***	2
3 3 1 -R- -*- -T-	-1
5 4 2 -R--- -**** -**** -**** ----- ----- *T--- ----*	5