A. Oh Those Palindromes, 2 seconds, 256 megabytes, standard input, standard output

A non-empty string is called *palindrome*, if it reads the same from the left to the right and from the right to the left. For example, "abcba", "a", and "abba" are palindromes, while "abab" and "xy" are not.

A string is called a *substring* of another string, if it can be obtained from that string by dropping some (possibly zero) number of characters from the beginning and from the end of it. For example, "abc", "ab", and "c" are substrings of the string "abc", while "ac" and "d" are not.

Let's define a *palindromic count* of the string as the number of its substrings that are palindromes. For example, the palindromic count of the string "aaa" is 6 because all its substrings are palindromes, and the palindromic count of the string "abc" is 3 because only its substrings of length 1 are palindromes.

You are given a string s. You can arbitrarily rearrange its characters. You goal is to obtain a string with the maximum possible value of palindromic count.

Input

The first line contains an integer n ($1 \le n \le 100\,000$) — the length of string s.

The second line contains string s that consists of exactly n lowercase characters of Latin alphabet.

Output

Print string t, which consists of the same set of characters (and each characters appears exactly the same number of times) as string s. Moreover, t should have the maximum possible value of palindromic count among all such strings strings.

If there are multiple such strings, print any of them.

input
```
5
oolol
```
output
```
ololo
```

input
```
16
gagadbcgghhchbdf
```
output
```
abccbaghghghgdfd
```

In the first example, string "ololo" has 9 palindromic substrings: "o", "l", "o", "l", "o", "olo", "lol", "olo", "ololo". Note, that even though some substrings coincide, they are counted as many times as they appear in the resulting string.

In the second example, the palindromic count of string "abccbaghghghgdfd" is 29.

B. Labyrinth, 2 seconds, 512 megabytes, standard input, standard output

You are playing some computer game. One of its levels puts you in a maze consisting of n lines, each of which contains m cells. Each cell either is free or is occupied by an obstacle. The starting cell is in the row r and column c. In one step you can move one square up, left, down or right, if the target cell is not occupied by an obstacle. You can't move beyond the boundaries of the labyrinth.

Unfortunately, your keyboard is about to break, so you can move left no more than x times and move right no more than y times. There are no restrictions on the number of moves up and down since the keys used to move up and down are in perfect condition.

Now you would like to determine for each cell whether there exists a sequence of moves that will put you from the starting cell to this particular one. How many cells of the board have this property?

Input

The first line contains two integers n, m ($1 \le n, m \le 2000$) — the number of rows and the number columns in the labyrinth respectively.

The second line contains two integers r, c ($1 \le r \le n, 1 \le c \le m$) — index of the row and index of the column that define the starting cell.

The third line contains two integers x, y ($0 \le x, y \le 10^9$) — the maximum allowed number of movements to the left and to the right respectively.

The next n lines describe the labyrinth. Each of them has length of m and consists only of symbols '.' and '*'. The j-th character of the i-th line corresponds to the cell of labyrinth at row i and column j. Symbol '.' denotes the free cell, while symbol '*' denotes the cell with an obstacle.

It is guaranteed, that the starting cell contains no obstacles.

Output

Print exactly one integer — the number of cells in the labyrinth, which are reachable from starting cell, including the starting cell itself.

input
```
4 5
3 2
1 2
.....
.***.
...**
*....
```
output
```
10
```

input
```
4 4
2 2
0 1
....
..*.
....
....
```
output
```
7
```

Cells, reachable in the corresponding example, are marked with '+'.

First example:

```
+++..
+***.
+++**
*+++.
```

Second example:

```
.++.
.+*.
.++.
.++.
```

C. Dwarves, Hats and Extrasensory Abilities,

2 seconds, 256 megabytes, standard input, standard output

This is an interactive problem.

In good old times dwarves tried to develop extrasensory abilities:

- Exactly n dwarves entered completely dark cave.
- Each dwarf received a hat — white or black. While in cave, none of the dwarves was able to see either his own hat or hats of other Dwarves.
- Dwarves went out of the cave to the meadow and sat at an arbitrary place one after the other. When a dwarf leaves the cave, he sees the colors of all hats of all dwarves that are seating on the meadow (i.e. left the cave before him). However, he is not able to see the color of his own hat and none of the dwarves can give him this information.
- The task for dwarves was to got diverged into two parts — one with dwarves with white hats and one with black hats.

After many centuries, dwarves finally managed to select the right place on the meadow without error. Will you be able to repeat their success?

You are asked to successively name n different integer points on the plane. After naming each new point you will be given its color — black or white. Your task is to ensure that the named points can be split by a line in such a way that all points of one color lie on the same side from the line and points of different colors lie on different sides. Moreover, no points can belong to the line. Also, you need to report any such line at the end of the process.

In this problem, the interactor is *adaptive* — the colors of the points in the tests are not fixed beforehand and the jury program can select them arbitrarily, in particular, depending on your program output.

Interaction

The first line of the standard input stream contains an integer n $(1 \le n \le 30)$ — the number of points your program should name.

Then n times your program must print two integer coordinates x and y $(0 \le x \le 10^9, 0 \le y \le 10^9)$. All points you print must be distinct.

In response to each coordinate pair your program will receive the string "black", if the point is black, or "white", if the point is white.

When all n points are processed, you need to print four integers x_1, y_1, x_2 and y_2 $(0 \le x_1, y_1 \le 10^9, 0 \le x_2, y_2 \le 10^9)$ — coordinates of points (x_1, y_1) and (x_2, y_2), which form a line, which separates n points into black and white. Points (x_1, y_1) and (x_2, y_2) should not coincide.

Hacks

To hack solution use the following format. The first line must contain word "hack", the second line should contain the number n and the last line should contain the sequence of 0 and 1 — colors of points, which will be reported to the solution. Unlike the jury tests, colors of points in hacks are always fixed in advance. Of course, the hacked solution wouldn't be able to get the information about the colors in advance.

For example, the hack corresponding to sample test will look like this:

```
hack
5
0 0 1 1 0
```

input

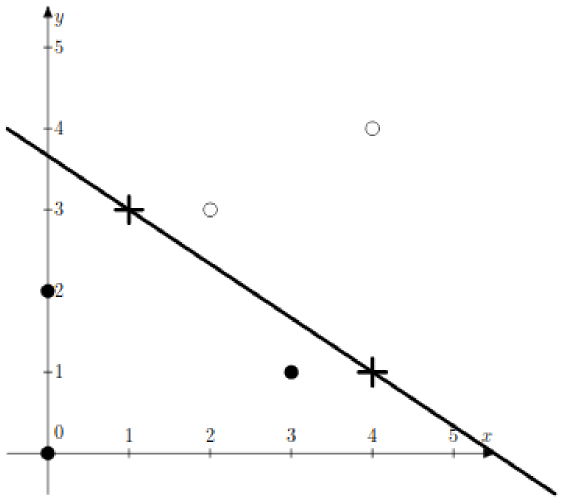
```
5

black

black

white

white

black
```

output

```

0 0

3 1

2 3

4 4

0 2

1 3 4 1
```

In the sample input and output values are aligned only for simplicity of interpreting them chronologically. In real interaction no "extra" line breaks should appear.

The following picture illustrates the first test.



D. Candies for Children,

1 second, 256 megabytes, standard input, standard output

At the children's festival, children were dancing in a circle. When music stopped playing, the children were still standing in a circle. Then Lena remembered, that her parents gave her a candy box with exactly k candies "Wilky May". Lena is not a greedy person, so she decided to present all her candies to her friends in the circle. Lena knows, that some of her friends have a sweet tooth and others do not. Sweet tooth takes out of the box two candies, if the box has at least two candies, and otherwise takes one. The rest of Lena's friends always take exactly one candy from the box.

Before starting to give candies, Lena step out of the circle, after that there were exactly n people remaining there. Lena numbered her friends in a clockwise order with positive integers starting with 1 in such a way that index 1 was assigned to her best friend Roma.

Initially, Lena gave the box to the friend with number l, after that each friend (starting from friend number l) took candies from the box and passed the box to the next friend in clockwise order. The process ended with the friend number r taking the last candy (or two, who knows) and the empty box. Please note that it is possible that some of Lena's friends took candy from the box several times, that is, the box could have gone several full circles before becoming empty.

Lena does not know which of her friends have a sweet tooth, but she is interested in the maximum possible number of friends that can have a sweet tooth. If the situation could not happen, and Lena have been proved wrong in her observations, please tell her about this.

Input

The only line contains four integers n, l, r and k ($1 \le n, k \le 10^{11}$, $1 \le l, r \le n$) — the number of children in the circle, the number of friend, who was given a box with candies, the number of friend, who has taken last candy and the initial number of candies in the box respectively.

Output

Print exactly one integer — the maximum possible number of sweet tooth among the friends of Lena or "-1" (quotes for clarity), if Lena is wrong.

input
```
4 1 4 12
```
output
```
2
```

input
```
5 3 4 10
```
output
```
3
```

input
```
10 5 5 1
```
output
```
10
```

input
```
5 4 5 6
```
output
```
-1
```

In the first example, any two friends can be sweet tooths, this way each person will receive the box with candies twice and the last person to take sweets will be the fourth friend.

In the second example, sweet tooths can be any three friends, except for the friend on the third position.

In the third example, only one friend will take candy, but he can still be a sweet tooth, but just not being able to take two candies. All other friends in the circle can be sweet tooths as well, they just will not be able to take a candy even once.

In the fourth example, Lena is wrong and this situation couldn't happen.

E. Lasers and Mirrors, 1 second, 256 megabytes, standard input, standard output

Oleg came to see the maze of mirrors. The maze is a n by n room in which each cell is either empty or contains a mirror connecting opposite corners of this cell. Mirrors in this maze reflect light in a perfect way, which causes the interesting visual effects and contributes to the loss of orientation in the maze.

Oleg is a person of curious nature, so he decided to install n lasers facing internal of the maze on the south wall of the maze. On the north wall of the maze, Oleg installed n receivers, also facing internal of the maze. Let's number lasers and receivers from west to east with distinct integers from 1 to n. Each laser sends a beam of some specific kind and receiver with number a_i should receive the beam sent from laser number i. Since two lasers' beams can't come to the same receiver, these numbers form a *permutation* — each of the receiver numbers occurs exactly once.

You came to the maze together with Oleg. Help him to place the mirrors in the initially empty maze so that the maximum number of lasers' beams will come to the receivers they should. There are no mirrors outside the maze, so if the laser beam leaves the maze, it will not be able to go back.

Input

The first line contains a single integer n ($1 \le n \le 1000$) — the size of the maze.

The second line contains a permutation of n integers a_i ($1 \le a_i \le n$), where a_i defines the number of the receiver, to which the beam from i-th laser should come.

Output

In the first line print the maximum possible number of laser beams, which can come to the receivers they should.

In the next n lines of length n print the arrangement of mirrors, causing such number of laser beams to come where they should. If the corresponding cell is empty, print ".", otherwise print "/" or "\", depending on the orientation of the mirror.

In your output north should be above, south should be below, and west and east should be on left and on the right respectively.

It is allowed for laser beams to come not to the receivers they correspond to, but they are not counted in the answer.

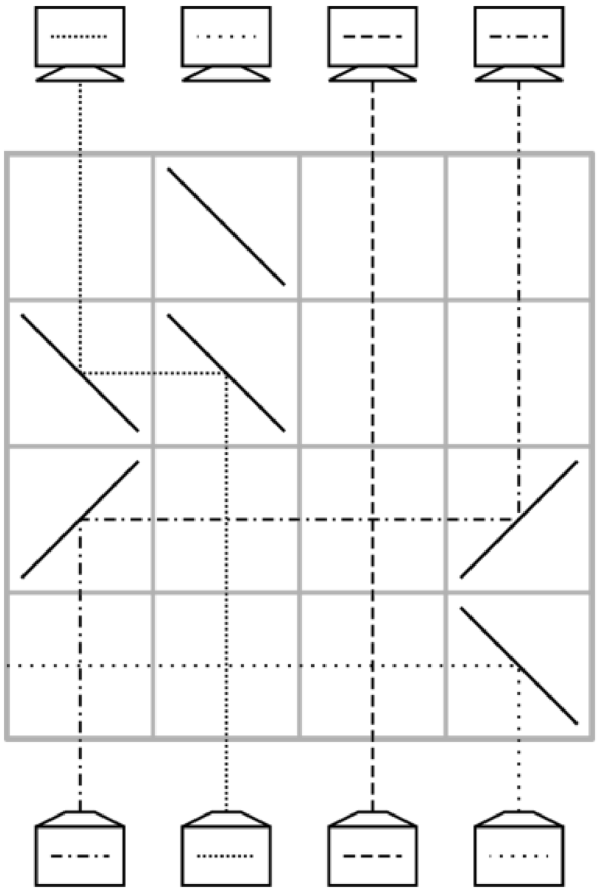
If there are multiple arrangements of mirrors leading to the optimal answer — print any of them.

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

output

```
3
.\..
\\..
/../
...\
```

The picture illustrates the arrangements of the mirrors in the first example.



F. String Journey, 5 seconds, 512 megabytes, standard input, standard output

We call a sequence of strings $t_1, \ldots, t_k$ a *journey* of length k, if for each $i > 1$ t_i is a substring of t_{i-1} and length of t_i is strictly less than length of t_{i-1}. For example, $\{ab, b\}$ is a journey, but $\{ab, c\}$ and $\{a, a\}$ are not.

Define a *journey on string* s as journey $t_1, \ldots, t_k$, such that all its parts can be nested inside s in such a way that there exists a sequence of strings $u_1, \ldots, u_{k+1}$ (each of these strings can be empty) and $s = u_1\ t_1\ u_2\ t_2 \ldots u_k\ t_k\ u_{k+1}$. As an example, $\{ab, b\}$ is a journey on string abb, but not on bab because the journey strings t_i should appear from the left to the right.

The *length* of a journey on a string is the number of strings in it. Determine the maximum possible length of a journey on the given string s.

Input

The first line contains a single integer n $(1 \le n \le 500\ 000)$ — the length of string s.

The second line contains the string s itself, consisting of n lowercase Latin letters.

Output

Print one number — the maximum possible length of string journey on s.

input

```
7
abcdbcc
```

output

```
3
```

input

```
4
bbcb
```

output

```
2
```

In the first sample, the string journey of maximum length is $\{abcd, bc, c\}$.

In the second sample, one of the suitable journeys is $\{bb, b\}$.