A. Home Numbers

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

The main street of Berland is a straight line with n houses built along it (n is an even number). The houses are located at both sides of the street. The houses with odd numbers are at one side of the street and are numbered from 1 to $n - 1$ in the order from the beginning of the street to the end (in the picture: from left to right). The houses with even numbers are at the other side of the street and are numbered from 2 to n in the order from the end of the street to its beginning (in the picture: from right to left). The corresponding houses with even and odd numbers are strictly opposite each other, that is, house 1 is opposite house n, house 3 is opposite house $n - 2$, house 5 is opposite house $n - 4$ and so on.

Vasya needs to get to house number a as quickly as possible. He starts driving from the beginning of the street and drives his car to house a. To get from the beginning of the street to houses number 1 and n, he spends exactly 1 second. He also spends exactly one second to drive the distance between two neighbouring houses. Vasya can park at any side of the road, so the distance between the beginning of the street at the houses that stand opposite one another should be considered the same.

Your task is: find the minimum time Vasya needs to reach house a.

Input

The first line of the input contains two integers, n and a ($1 \le a \le n \le 100\,000$) — the number of houses on the street and the number of the house that Vasya needs to reach, correspondingly. It is guaranteed that number n is even.

Output

Print a single integer — the minimum time Vasya needs to get from the beginning of the street to house a.

Examples

input
4 2
output
2

input
8 5
output
3

Note

In the first sample there are only four houses on the street, two houses at each side. House 2 will be the last at Vasya's right.

The second sample corresponds to picture with $n = 8$. House 5 is the one before last at Vasya's left.

B. Making Genome in Berland

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

Berland scientists face a very important task - given the parts of short DNA fragments, restore the dinosaur DNA! The genome of a berland dinosaur has noting in common with the genome that we've used to: it can have 26 distinct nucleotide types, a nucleotide of each type can occur **at most once**. If we assign distinct English letters to all nucleotides, then the genome of a Berland dinosaur will represent a non-empty string consisting of small English letters, such that each letter occurs in it at most once.

Scientists have n genome fragments that are represented as **substrings** (non-empty sequences of consecutive nucleotides) of the sought genome.

You face the following problem: help scientists restore the dinosaur genome. It is guaranteed that the input is not contradictory and at least one suitable line always exists. When the scientists found out that you are a strong programmer, they asked you in addition to choose the one with the minimum length. If there are multiple such strings, choose any string.

Input

The first line of the input contains a positive integer n ($1 \le n \le 100$) — the number of genome fragments.

Each of the next lines contains one descriptions of a fragment. Each fragment is a non-empty string consisting of distinct small letters of the English alphabet. It is not guaranteed that the given fragments are distinct. Fragments could arbitrarily overlap and one fragment could be a substring of another one.

It is guaranteed that there is such string of distinct letters that contains all the given fragments as substrings.

Output

In the single line of the output print the genome of the minimum length that contains all the given parts. All the nucleotides in the genome must be distinct. If there are multiple suitable strings, print the string of the minimum length. If there also are multiple suitable strings, you can print any of them.

Examples

input
3 bcd ab cdef
output
abcdef

input
4 x y z w
output
xyzw

C. Road Improvement

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

In Berland there are n cities and $n - 1$ bidirectional roads. Each road connects some pair of cities, from any city you can get to any other one using only the given roads.

In each city there is exactly one repair brigade. To repair some road, you need two teams based in the cities connected by the road to work simultaneously for one day. Both brigades repair one road for the whole day and cannot take part in repairing other roads on that day. But the repair brigade can do nothing on that day.

Determine the minimum number of days needed to repair all the roads. The brigades cannot change the cities where they initially are.

Input

The first line of the input contains a positive integer n ($2 \le n \le 200\,000$) — the number of cities in Berland.

Each of the next $n - 1$ lines contains two numbers u_i, v_i, meaning that the i-th road connects city u_i and city v_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$).

Output

First print number k — the minimum number of days needed to repair all the roads in Berland.

In next k lines print the description of the roads that should be repaired on each of the k days. On the i-th line print first number d_i — the number of roads that should be repaired on the i-th day, and then d_i space-separated integers — the numbers of the roads that should be repaired on the i-th day. The roads are numbered according to the order in the input, starting from one.

If there are multiple variants, you can print any of them.

Examples

input
4 1 2 3 4 3 2
output
2 2 2 1 1 3

input
6 3 4 5 4 3 2 1 3 4 6
output
3 1 1 2 2 3 2 4 5

Note

In the first sample you can repair all the roads in two days, for example, if you repair roads 1 and 2 on the first day and road 3 — on the second day.

D. Three-dimensional Turtle Super Computer

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

A super computer has been built in the Turtle Academy of Sciences. The computer consists of $n \cdot m \cdot k$ CPUs. The architecture was the paralellepiped of size $n \times m \times k$, split into $1 \times 1 \times 1$ cells, each cell contains exactly one CPU. Thus, each CPU can be simultaneously identified as a group of three numbers from the layer number from 1 to n, the line number from 1 to m and the column number from 1 to k.

In the process of the Super Computer's work the CPUs can send each other messages by the famous turtle scheme: CPU (x, y, z) can send messages to CPUs $(x+1, y, z)$, $(x, y+1, z)$ and $(x, y, z+1)$ (of course, if they exist), there is no feedback, that is, CPUs $(x+1, y, z)$, $(x, y+1, z)$ and $(x, y, z+1)$ cannot send messages to CPU (x, y, z).

Over time some CPUs broke down and stopped working. Such CPUs cannot send messages, receive messages or serve as intermediates in transmitting messages. We will say that CPU (a, b, c) *controls* CPU (d, e, f) , if there is a chain of CPUs (x_i, y_i, z_i), such that $(x_1 = a, y_1 = b, z_1 = c)$, $(x_p = d, y_p = e, z_p = f)$ (here and below p is the length of the chain) and the CPU in the chain with number i ($i < p$) can send messages to CPU $i+1$.

Turtles are quite concerned about the denial-proofness of the system of communication between the remaining CPUs. For that they want to know the number of critical CPUs. A CPU (x, y, z) is *critical*, if turning it off will disrupt some control, that is, if there are two distinctive from (x, y, z) CPUs: (a, b, c) and (d, e, f), such that (a, b, c) controls (d, e, f) before (x, y, z) is turned off and stopped controlling it after the turning off.

Input

The first line contains three integers n, m and k ($1 \le n, m, k \le 100$) — the dimensions of the Super Computer.

Then n blocks follow, describing the current state of the processes. The blocks correspond to the layers of the Super Computer in the order from 1 to n. Each block consists of m lines, k characters in each — the description of a layer in the format of an $m \times k$ table. Thus, the state of the CPU (x, y, z) is corresponded to the z-th character of the y-th line of the block number x. Character "1" corresponds to a working CPU and character "0" corresponds to a malfunctioning one. The blocks are separated by exactly one empty line.

Output

Print a single integer — the number of critical CPUs, that is, such that turning only this CPU off will disrupt some control.

Examples

input
2 2 3 000 000 111 111
output
2

input
3 3 3 111 111 111 111 111 111 111 111 111
output
19

input
1 1 10 0101010101
output
0

Note

In the first sample the whole first layer of CPUs is malfunctional. In the second layer when CPU $(2, 1, 2)$ turns off, it disrupts the control by CPU $(2, 1, 3)$ over CPU $(2, 1, 1)$, and when CPU $(2, 2, 2)$ is turned off, it disrupts the control over CPU $(2, 2, 3)$ by CPU $(2, 2, 1)$.

In the second sample all processors except for the corner ones are critical.

In the third sample there is not a single processor controlling another processor, so the answer is 0.