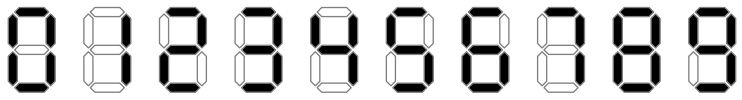
Problem A. Auxiliary Project

Input file:	`auxiliary.in`	Time limit:	3 seconds
Output file:	`auxiliary.out`	Memory limit:	512 megabytes

Anna has just finished her course project. She has a lot of seven-segment LED displays as leftovers and a small power source. Each display consumes power proportionally to the number of lit segments, e.g. '9' consumes twice more power than '7'.



Anna wonders what is the maximum possible sum of digits she is able to achieve, if her power source is able to light n segments, and she wants to light exactly n segments.

Input

The single line of the input contains one integer n — the number of segments that should be lit ($2 \le n \le 10^6$).

Output

Output a single integer — the maximum possible sum of digits that can be displayed simultaneously.

Examples

auxiliary.in	auxiliary.out
4	4
7	11
6	14

In the first example, a single '4' should be displayed ('7' has greater value, but has only three segments). In the second example '4' and '7' should be displayed, in the third one — two '7's.

Problem B. Boolean Satisfiability

Input file:	`boolean.in`	Time limit:	3 seconds
Output file:	`boolean.out`	Memory limit:	512 megabytes

Boolean satisfiability problem (SAT) is known to be a very hard problem in computer science. In this problem you are given a Boolean formula, and you need to find out if the variables of a given formula can be consistently replaced by the values `true` or `false` in such a way that the formula evaluates to `true`. SAT is known to be NP-complete problem. Moreover, it is NP-complete even in case of 3-CNF formula (3-SAT). However, for example, SAT problem for 2-CNF formulae (2-SAT) is in P.

#SAT is the extension of SAT problem. In this problem you need to check if it is possible, and count the number of ways to assign values to variables. This problem is known to be #P-complete even for 2-CNF formulae. We ask you to solve #1-DNF-SAT, which is #SAT problem for 1-DNF formulae.

You are given a Boolean formula in 1-DNF form. It means that it is a disjunction (logical or) of one or more clauses, each clause is exactly one literal, each literal is either variable or its negation (logical not).

Formally:

$$\begin{aligned} \langle \text{formula} \rangle &::= \langle \text{clause} \rangle \mid \langle \text{formula} \rangle \vee \langle \text{clause} \rangle \\ \langle \text{clause} \rangle &::= \langle \text{literal} \rangle \\ \langle \text{literal} \rangle &::= \langle \text{variable} \rangle \mid \neg \langle \text{variable} \rangle \\ \langle \text{variable} \rangle &::= A \ldots Z \mid a \ldots z \end{aligned}$$

Your task is to find the number of ways to replace all variables with values `true` and `false` (all occurrences of the same variable should be replaced with same value), such that the formula evaluates to `true`.

Input

The only line of the input file contains a logical formula in 1-DNF form (not longer than 1000 symbols). Logical operations are represented by '`|`' (disjunction) and '`~`' (negation). The variables are '`A`' ... '`Z`' and '`a`' ... '`z`' (uppercase and lowercase letters are different variables). The formula contains neither spaces nor other characters not mentioned in the grammar.

Output

Output a single integer — the answer for #SAT problem for the given formula.

Examples

boolean.in	boolean.out
a	1
B|~B	2
c|~C	3
i|c|p|c	7

Problem C. Consonant Fencity

Input file:	`consonant.in`	Time limit:	3 seconds
Output file:	`consonant.out`	Memory limit:	512 megabytes

There are two kinds of sounds in spoken languages: vowels and consonants. Vowel is a sound, produced with an open vocal tract; and consonant is pronounced in such a way that the breath is at least partly obstructed. For example, letters `a` and `o` are used to express vowel sounds, while letters `b` and `p` are the consonants (e.g. `b`ad, `p`ot).

Some letters can be used to express both vowel and consonant sounds: for example, `y` may be used as a vowel (e.g. sill`y`) or as a consonant (e.g. `y`ellow). The letter `w`, usually used as a consonant (e.g. `w`et) could produce a vowel after another vowel (e.g. gro`w`th) in English, and in some languages (e.g. Welsh) it could be even the only vowel in a word.

In this task, we consider `y` and `w` as vowels, so there are seven vowels in English alphabet: `a`, `e`, `i`, `o`, `u`, `w` and `y`, all other letters are consonants.

Let's define the *consonant fencity* of a string as the number of pairs of consecutive letters in the string which both are consonants and have different cases (lowercase letter followed by uppercase or vice versa). For example, the consonant fencity of a string `CoNsoNaNts` is 2, the consonant fencity of a string `dEsTrUcTiOn` is 3 and the consonant fencity of string `StRenGtH` is 5.

You will be given a string consisting of lowercase English letters. Your task is to change the case of some letters in such a way that all equal letters will be of the same case (that means, no letter can occur in resulting string as both lowercase and uppercase), and the consonant fencity of resulting string is maximal.

Input

The only line of the input contains non-empty original string consisting of no more than 10^6 lowercase English letters.

Output

Output the only line: the input string changed to have maximum consonant fencity.

Examples

consonant.in	consonant.out
consonants	CoNsoNaNts
destruction	dEsTrUcTiOn
strength	StRenGtH

Problem D. Dividing Marbles

Input file:	`dividing.in`	Time limit:	3 seconds
Output file:	`dividing.out`	Memory limit:	512 megabytes

Debbie, Debby, Debra and Deborah are going to play a game with marbles together. Debbie has brought 2^{d_1} marbles, Debby — 2^{d_2} marbles, Debra — 2^{d_3} marbles, while Deborah — 2^{d_4} marbles. The kids have gathered their marbles into a single *pile* containing $2^{d_1} + 2^{d_2} + 2^{d_3} + 2^{d_4}$ marbles, and the game is starting.

The game consists of turns. Each turn consists of two steps:

- The kids choose any of their piles with more than one marble and divide it into two non-empty piles. That is, if the chosen pile contains $m \geq 2$ marbles, the new piles must contain m_1 and m_2 marbles where m_1 and m_2 are positive integers, and $m_1 + m_2 = m$.
- If there are several piles with the same number of marbles, only one of these piles is kept, while all the others with this number of marbles are discarded (thrown away).

The game ends when only one pile is left, and this pile contains a single marble. The goal of the game is to end it in the least possible number of turns. Note that the game is cooperative, that is, the kids aren't playing against each other, but trying to reach a common goal together.

Help the kids and find the best way to play.

Input

The first line of the input contains a single integer T — the number of test cases ($1 \leq T \leq 500$).

Each of the next T lines describes one test case and contains four non-negative integers d_1, d_2, d_3, d_4 ($0 \leq d_i \leq 20$).

Output

For each test case, output an integer t — the smallest number of turns required to end the game.

Then, output t turn descriptions, in the order the turns should be made. Each description should consist of three integers m, m_1, m_2 — the size of the divided pile and the sizes of the new piles, respectively ($m \geq 2$; $m_1 > 0$; $m_2 > 0$; $m_1 + m_2 = m$). Note that a pile of size m must exist at that moment, and at the end of the game there should be only one pile left and that pile should contain a single marble.

Example

dividing.in	dividing.out
2 1 0 1 0 0 1 2 3	3 6 2 4 4 2 2 2 1 1 5 15 10 5 10 5 5 5 1 4 4 2 2 2 1 1

Consider the second example. Initially, there is a single pile containing $2^0 + 2^1 + 2^2 + 2^3 = 15$ marbles. After the first turn, there are two piles containing 10 and 5 marbles. After the second division, there are three piles containing 5 marbles each, and two of these piles are discarded, so only one pile with 5 marbles is left. After the third turn, there are two piles containing 1 and 4 marbles. After the fourth turn, there are two piles containing 1 and 2 marbles (the other pile with 2 marbles is discarded). Finally, after the fifth turn, there is just one pile with 1 marble (the other two piles with 1 marble are discarded).

Problem E. Equal Numbers

Input file:	`equal.in`	Time limit:	3 seconds
Output file:	`equal.out`	Memory limit:	512 megabytes

You are given a list of n integers $a_1, \ldots, a_n$. You can perform the following operation: choose some a_i and multiply it by any positive integer.

Your task is to compute the minimum number of different integers that could be on the list after k operations for all $0 \le k \le n$.

Input

The first line of the input contains single integer n ($1 \le n \le 3 \cdot 10^5$). The second line of the input contains n integers a_i ($1 \le a_i \le 10^6$).

Output

Output a single line that contains $n + 1$ integers. The i-th integer should be the minimum possible number of different integers in the list after $i - 1$ operations.

Example

equal.in	equal.out
6 3 4 1 2 1 2	4 4 3 3 2 2 1

Problem F. Fygon 2.0

Input file:	`fygon20.in`	Time limit:	3 seconds
Output file:	`fygon20.out`	Memory limit:	512 megabytes

The new version of beloved programming language Fygon has been released! The brand new Fygon 2.0 still has only two statements. The first statement is `lag`. It substitutes almost any other statement. Second statement is a `for` loop:

```
for <variable> in range(<from>, <to>):
    <body>
```

- The `for` loop makes `<variable>` iterate from `<from>` to `<to>`, **both inclusive**.
- If `<from>` is greater than `<to>`, `<body>` is not executed at all.
- `<variable>` is a lowercase letter from `a` to `z`, except for `n`, which is a variable that is defined prior to the given code snippet.
- `<from>` and `<to>` can be equal to any variable defined in outer loop. In addition to that, `<from>` can be `1` and `<to>` can be `n`.
- The `<body>` of the loop is indented by four spaces and contains at least one statement.

If you are familiar with Fygon 1.0, you can notice that, in the spirit of the best programming practices, Fygon 2.0 is not backwards compatible, since the `range` function now requires two parameters.

The performance of the new version is significantly improved, so you can write more nested `for` loops. That is why we are no longer interested in the exact number of operations, but in the *asymptotic complexity* of the program instead. For simplicity, all `for` loops are nested in a single chain and there is exactly one `lag` statement that is inside all `for` loops. All loop variables are different and are not equal to `n`.

Let's define $f(n)$ as the number of `lag` operations exectuted by a given Fygon program as the function of n. For non-negative integer k and positive rational number C we say that $C \cdot n^k$ is the *asymptotic complexity* of the program if

$$\lim_{n \to \infty} \frac{f(n)}{C \cdot n^k} = 1.$$

Given a Fygon 2.0 program, find its asymptotic complexity.

Input

The first line of the input contains single integer m — the number of lines in Fygon 2.0 program. Next m lines contain the program itself. The program has at least 1 and at most 20 `for` statements. Each `for` statement contains either single nested `for` statement or `lag` statement.

Output

Output numbers k and C. C should be output in the form of irreducible fraction p/q, where p and q are coprime.

Example

fygon20.in	fygon20.out

4
for i in range(1, n):
    for j in range(1, i):
        for k in range(j, n):
            lag
	3 1/3

Problem G. Grand Test

Input file:	grand.in	Time limit:	3 seconds
Output file:	grand.out	Memory limit:	512 megabytes

Jeremy, Richard and James like to test cars. It is always hard for them to decide where they should do it. Usually car test looks like this. They choose a country and examine its cities and two-way roads that connect them. To perform a test, they need to choose two different cities S and F, such that there exist three routes between them. Moreover, each city except S and F should be visited by at most one route, and none of the roads may be used twice.

Then each of them takes a car in city S, drives along one of those routes and tries to get to city F faster than others.

You are given a description of multiple countries. For each country you should decide if it is possible to choose two cities and three routes between them in a way described above.

Input

The first line of the input contains a single integer T — number of countries ($1 \le T \le 100\,000$). It is followed by T country descriptions.

The first line of each country description contains two integers n and m — the number of its cities and roads ($1 \le n, m \le 100\,000$). The following m lines contain two integer numbers each: u_i and v_i — the cities at the ends of the road ($1 \le u_i < v_i \le n$). All roads are two-way. Each pair of cities is connected by at most one road.

Both the total number of cities and roads in all countries does not exceed 100 000.

Output

Output the answer for each country in the order they are given in the input.

If it is not possible to test cars in this country, the answer is -1. Otherwise the first line of the answer should contain two integers S and F — start and finish cities. The next three lines should contain three distinct routes. Each route is described by an integer k — the number of cities it visits, and k numbers $v_1, v_2, ..., v_k$ — the cities, where $v_1 = S$, $v_k = F$, and there is a road between cities v_i and v_{i+1} for all $1 \le i \le k - 1$.

Example

grand.in	grand.out
2 6 6 3 6 3 4 1 4 1 2 1 3 2 3 3 1 1 2	1 3 3 1 2 3 2 1 3 3 1 4 3 -1

Problem H. Hidden Supervisors

Input file:	`hidden.in`	Time limit:	3 seconds
Output file:	`hidden.out`	Memory limit:	512 megabytes

Helena works in a big company as a psychologist. Her task is to organize a team building game to enhance social relations between employees. Each employee except the Big boss has a single supervisor. So, employees of the company form a tree where each employee is a node, and the parent of that node is their supervisor. The root of the tree is the Big boss.

A team building game requires teams of two people. Every team should consist of an employee and their supervisor.

Helena asked every employee except the Big boss to send their supervisor ID. Some of them didn't reply. She is going to assign a fake supervisor to every employee that didn't reply, so that she could arrange as many teams as possible. And, of course, fake and real supervisors must form a tree.

Helena had a difficult, but a successful day organizing the event. Will you be able to assign fake supervisors?

Input

The first line of the input contains a single integer n — the number of employees in the company ($2 \le n \le 100\,000$).

The following line contains $n - 1$ integers $p_2, p_3, \ldots, p_n$, where p_i is the supervisor of employee i ($0 \le p_i \le n$). If employee i didn't reply to Helena, p_i equals zero, and she needs to assign a fake supervisor to that employee. The Big boss has the number 1.

It's possible to assign a fake supervisor to each employee that didn't reply to Helena so that all employees will form a tree having the Big boss as a root.

Output

In the first line output a single integer m — the maximum possible number of arranged teams.

The next line should contain supervisors: $n-1$ integers, i-th of which denoting the supervisor of employee $i+1$ (either fake or real). Of course, all real supervisors should be preserved, and employees must form a tree. It should be possible to arrange m teams using specified supervisors.

Examples

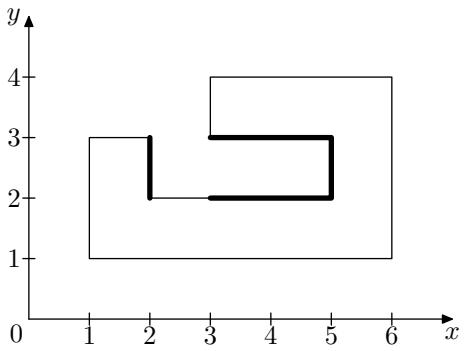
hidden.in	hidden.out
6 3 1 0 4 4	2 3 1 2 4 4
6 3 1 0 6 4	3 3 1 1 6 4

Problem I. Intelligence in Perpendicularia

Input file:	`intel.in`	Time limit:	3 seconds
Output file:	`intel.out`	Memory limit:	512 megabytes

There are only two directions in Perpendicularia: vertical and horizontal. Perpendicularia government are going to build a new secret service facility. They have some proposed facility plans and want to calculate total secured perimeter for each of them.

The total secured perimeter is calculated as the total length of the facility walls invisible for the perpendicularly-looking outside observer. The figure below shows one of the proposed plans and corresponding secured perimeter.



Write a program that calculates the total secured perimeter for the given plan of the secret service facility.

Input

The plan of the secret service facility is specified as a polygon.

The first line of the input contains one integer n — the number of vertices of the polygon ($4 \le n \le 1000$). Each of the following n lines contains two integers x_i and y_i – the coordinates of the i-th vertex ($-10^6 \le x_i, y_i \le 10^6$). Vertices are listed in the consecutive order.

All polygon vertices are distinct and none of them lie at the polygon's edge. All polygon edges are either vertical ($x_i = x_{i+1}$) or horizontal ($y_i = y_{i+1}$) and none of them intersect each other.

Output

Output a single integer — the total secured perimeter of the secret service facility.

Example

intel.in	intel.out
10 1 1 6 1 6 4 3 4 3 3 5 3 5 2 2 2 2 3 1 3	6

Problem J. Joker

Input file:	`joker.in`	Time limit:	3 seconds
Output file:	`joker.out`	Memory limit:	512 megabytes

Joker prepares a new card trick with a strong mathematical background. You are asked to help Joker with calculations.

There is a row of n cards with non-zero numbers a_i written on them. Let's call the sum of all positive numbers P and the sum of all negative numbers N. Every card i has a weight $w_i = \frac{a_i}{P}$ if $a_i > 0$ and $\frac{a_i}{|N|}$ otherwise.

Let's denote $s_i = \sum_{j=1}^{j \le i} w_j$. Joker needs to know positive i with the largest s_i. If there is more than one such i, he is interested in the smallest one.

But static tricks are boring, so Joker wants to change numbers on some cards, and after each change he needs to known where is the largest s_i is.

Input

The first line of the input contains two integers n and m — the number of cards and the number of changes ($1 \le n, m \le 50\,000$).

The second line consists of n integers a_i — numbers written on cards at the beginning ($-10^9 \le a_i \le 10^9$; $a_i \ne 0$).

The following m lines contain two integers each: p_i and v_i, that means value of card at position p_i is changed to v_i ($1 \le p_i \le n$; $-10^9 \le v_i \le 10^9$; $v_i \ne 0$).

It is guaranteed that at each moment there is at least one card with positive number and at least one card with negative number. The sum of all positive cards will never exceed 10^9 and the sum of all negative cards will never exceed -10^9.

Output

You should output $m+1$ integers. The first integer is the position of the largest s_i for the initial numbers. Next m numbers are positions of the largest s_i after each change.

Example

joker.in	joker.out
4 7 1 -5 3 -5 4 -1 2 -1 3 10 4 10 1 -1 2 1 3 -1	3 1 3 3 1 4 4 4

Problem K. Kotlin Island

Input file:	kotlin.in	Time limit:	3 seconds
Output file:	kotlin.out	Memory limit:	512 megabytes

There is an urban myth that Peter the Great wanted to make a rectangular channel-grid engineering masterpiece not only from Vasilyevskiy island, but also from Kotlin island (where the town of Kronstadt is located nowadays).

The following mathematical model was (allegedly) presented to the tsar. The island is considered a rectangular grid h cells high and w cells wide. Each cell is dry land initially but can become water.

Technologies of those days allowed engineers to dig a channel across the entire island. In that case an entire row or an entire column of cells became water. If some of these cells already were water, their status did not change.

Your task is to propose a plan of the island which has exactly n connected components of dry land cells.

Input

The only line of the input contains three integers h, w, and n — grid's height, width and the desired number of connected components ($1 \le h, w \le 100$; $1 \le n \le 10^9$).

Output

If there is no valid plan containing n connected components, output a single word "`Impossible`".

Otherwise output h lines of length w depicting the plan. Dot ('`.`') represents a dry land cell, hash ('`#`') represents a water cell.

Examples

kotlin.in	kotlin.out
3 5 4	..#.. ##### ..#..
2 1 1	# .
5 3 10	Impossible

Problem L. Little Difference

Input file:	little.in	Time limit:	3 seconds
Output file:	little.out	Memory limit:	512 megabytes

Little Lidia likes playing with numbers. Today she has a positive integer n, and she wants to decompose it to the product of positive integers.

Because Lidia is little, she likes to play with numbers with little difference. So, all numbers in decomposition should differ by at most one. And of course, the product of all numbers in the decomposition must be equal to n. She considers two decompositions the same if and only if they have the same number of integers and there is a permutation that transforms the first one to the second one.

Write a program that finds all decompositions, which little Lidia can play with today.

Input

The only line of the input contains a single integer n ($1 \le n \le 10^{18}$).

Output

In first line output the number of decompositions of n, or -1 if this number is infinite. If number of decompositions is finite, print all of them one per line. In each line first print number k_i of elements in decomposition. Then print k_i integers in this decomposition in any order. Don't forget that decompositions which are different only in order of elements are considered the same.

Examples

little.in	little.out
12	3 1 12 3 2 3 2 2 4 3
1	-1

In the second example 1 can be represented as product of any number of ones.