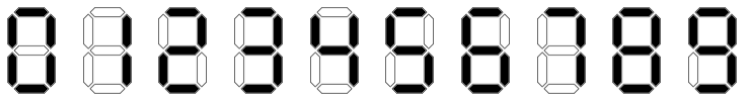
Problem F

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 G

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:

⟨formula⟩ ::= ⟨clause⟩ | ⟨formula⟩ ∨ ⟨clause⟩
⟨clause⟩ ::= ⟨literal⟩
⟨literal⟩ ::= ⟨variable⟩ | ¬ ⟨variable⟩
⟨variable⟩ ::= *A . . . Z* | *a . . . z*

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 H

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. bad, pot).

Some letters can be used to express both vowel and consonant sounds: for example, y may be used as a vowel (e.g. silly) or as a consonant (e.g. yellow). The letter w, usually used as a consonant (e.g. wet) could produce a vowel after another vowel (e.g. growth) 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

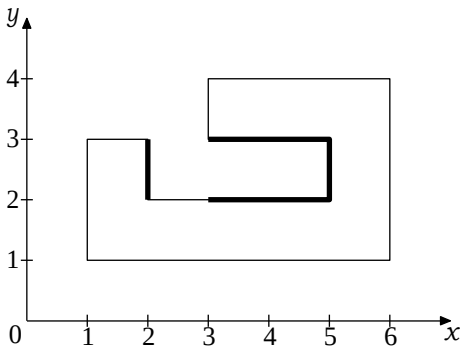
Sample Input	Sample Output
consonants	CoNsoNaNts
destruction	dEsTrUcTiOn
strength	StRenGtH

Problem I

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

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