Problem A. Bridges, Edges and Leaves

Input file:	bel.in
Output file:	bel.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Research Institute of Leaves and Edges (RILE) and Research Institute of Bridges (RIB) have a lot in common. They often work in tight collaboration. But today they faced a problem that lies exactly between their research fields. They need to build an undirected graph with B bridges, E edges and L leaves. After a lot of study, they invited Vasya to deal with this problem. But as you know, Vasya is specialist in strings, so he asked for your help.

Remember that, for a given undirected graph, *bridge* is an edge such that deleting it increases the number of connected components, and *leaf* is a vertex of degree 1.

Input

The input file contains three integers B, E and L such that $0 \le B, E, L \le 1000$.

Output

You must output the description of any graph consisting V vertices, with exactly B bridges, E edges and L leaves, or `IMPOSSIBLE` if there is no such graph. Here V is non-negative integer number which does not exceed 2000. The first line must contain V. The following E lines contain pairs of numbers u_i and v_i indicating that vertices u_i and v_i ($1 \le u_i, v_i \le V$) are connected by a single edge. Loops, duplicate edges and vertices with zero degree are not allowed.

Examples

bel.in	bel.out
1 4 1	4 1 2 2 3 3 4 2 4
5 4 3	IMPOSSIBLE
2 2 4	4 1 2 3 4

Problem B. Bisectors, Medians and Heights

Input file:	bmh.in
Output file:	bmh.out
Time limit:	2 seconds
Memory limit:	256 megabytes

RING (Research Institute of Nontrivial Geometry) asked Vasya to solve a new problem.

Let $\mathcal{A}$ be a multiset of natural numbers. *Multiset* is just a set of numbers which can contain equal elements more than once.

Let $\mathcal{T}$ be a set of all possible triangles with sides of lengths a, b and c such that $\{a, b, c\}$ is some multisubset of $\mathcal{A}$ and $a \le b \le c < a + b$.

Finally, let us define three ordinary sets of real numbers $\mathcal{B}, \mathcal{M}, \mathcal{H}$: $\mathcal{B}$ contains lengths of all bisectors for all triangles from $\mathcal{T}$, $\mathcal{M}$ contains lengths of all medians, and $\mathcal{H}$ contains lenghts of all heights.

For example, if $\mathcal{A} = \{3, 4, 5\}$ then $\mathcal{B} = \{\frac{\sqrt{288}}{7}, \frac{\sqrt{45}}{2}, \frac{\sqrt{160}}{3}\}$, $\mathcal{M} = \{\frac{5}{2}, \sqrt{13}, \frac{\sqrt{73}}{2}\}$, $\mathcal{H} = \{\frac{12}{5}, 3, 4\}$.

The problem is to find the number of different elements in $(\mathcal{B} \cup \mathcal{M} \cup \mathcal{H})$.

Since Vasya doesn't know formulae for lengths of bisectors, medians and heights, he asks for your help.

Input

First line of input file contains a single integer N ($3 \le N \le 100$) — cardinality of $\mathcal{A}$. Second line contains N natural numbers not greater than 100 separated by single spaces — the elements of $\mathcal{A}$.

Output

Output file should contain a single integer — cardinality of $(\mathcal{B} \cup \mathcal{M} \cup \mathcal{H})$.

Examples

bmh.in	bmh.out
3 1 1 1	1
3 3 4 5	9
6 2 2 2 3 3 3	10

Problem C. How Many Parallelograms?

Input file:	howmany.in
Output file:	howmany.out
Time limit:	2 seconds
Memory limit:	256 megabytes

The Research Institute of Polygons and Polyhedra Enumeration and Description (RIPPED) now conducts their experiments on parallelograms. They cut a sheet of squared paper of size $M \times N$ (each cell of a paper has size 1×1) and now they need to count the number of non-degenerate parallelograms such that their vertices are corners of the cells of this sheet (vertices are allowed to lie on the side or in the corner of the sheet itself).

But the number appeared to be very large, so scientists called Vasya to help them.

Input

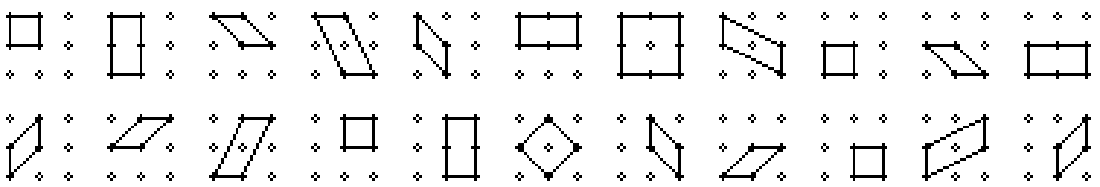
The input file contains two positive integers $N, M \leq 2000$.

Output

Output the number of non-degenerate parallelograms on an $N \times M$ sheet of squared paper.

Example

howmany.in	howmany.out
2 2	22



Problem D. (High Division Only!) Life 2

Input file:	life2.in
Output file:	life2.out
Time limit:	2 seconds
Memory limit:	256 megabytes

RIANT (Research Institute of Analysis of Nature Transformations) discovered another strange colony of microorganisms. One famous scientist deduced recurrent formulae for $T(N)$ — living time of a colony in seconds depending on a positive integer N, the characteristic number of a colony.

$$\begin{aligned} T(1) &= 0 \\ T(2k) &= T(k) + 1 \qquad \forall k \geq 1 \\ T(2k+1) &= T(6k+4) + 1 \qquad \forall k \geq 1 \end{aligned}$$

RIANT asked Vasya to write a program which will find the living time of a colony given the characteristic number N. Since the program should work for very large values of N, Vasya needs your help.

Input

The input file contains a single positive integer N ($N < 2^{100\,000}$). It is guaranteed that for given N, the value of $T(N)$ is not greater than 1 500 000.

Output

Output file must contain a single integer — $T(N)$.

Example

life2.in	life2.out
27	111

Problem E. Linear Device

Input file:	linear.in
Output file:	linear.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Research Institute of Graph Handling Theory (RIGHT) needs to build a special device to support their new experiment. The scheme of this device contains three types of elements: *generators*, *resistors* and *transistors*. Each **generator** is connected with exactly **one** other element, **resistor** is connected with **two** other elements, and **transistor** is connected with **three** of them. There is also a special condition, that **no transistor is connected to another transistor**.

All elements must be positioned on a metal string in such way that any two subsequent elements must be connected. Two non-subsequent elements are always allowed to be connected by an extra wire. To position all elements on a string in such a way, specialists of RIGHT invited Vasya from RIGS (Research Institute of Given Strings). But this is not a string Vasya used to deal with... so he asked you for help.

Input

The first line contains the number of elements in the device N ($2 \le N \le 100\,000$). The next N lines contain descriptions of elements and their connections. Each of them starts with a single letter `G`, `R` or `T` which stands for Generator, Resistor or Transistor respectively. One, two or three numbers follow, separated by spaces, indicating the numbers of elements connected with the corresponding element. Elements are enumerated from 1 to N in order of their description in input. Element cannot be connected to itself, and two elements are not allowed to be connected more than once.

Output

The only line of the output must contain N integer numbers — a permutation of elements that allows to position them all on a metal string such that any two consecutive elements are connected. If there is more than one solution, any solution is accepted. If there is no solution, output a single word `IMPOSSIBLE`.

Examples

linear.in	linear.out
5 G 2 R 1 3 T 2 4 5 R 3 5 R 3 4	5 4 3 2 1
7 T 2 3 4 G 1 G 1 R 1 7 G 7 G 7 T 4 5 6	IMPOSSIBLE

Problem F. (High Division Only!) Chess Playing Monkeys

Input file:	monkey.in
Output file:	monkey.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Research Institute of Supervision and Investigation of Baboons, Lemurs and Eagles (also known as RISIBLE) has a task for Vasya.

There are two very smart baboons in RISIBLE, because they know some chess playing rules! Moreover, they can play a self invented game on a chess board. At the beginning of each game, each baboon selects one chess piece (queen, rook, bishop or knight) and places it at a random chosen cell; the selected pieces have different color. The baboon who owns the white piece starts the game. They move their pieces according to the chess rules. Baboons alternate their moves. The baboon who takes the opponent's piece becomes the winner of the game.

After long observations, RISIBLE scientists discovered that in any position, baboons make each of the available moves with equal probability!

For some starting positions, RISIBLE asked Vasya to find the winning probability for baboon playing the white piece in each of these positions.

Input

The first line of input file contains the type of white piece — '`knight`', '`bishop`', '`rook`' or '`queen`' (without quotes). The second line contains the type of black piece in the same format. The third line contains natural number N — number of starting positions in the input file. Next N lines contain starting positions, one line per position. Every line contains four integers x_1, y_1, x_2, y_2 ($1 \le x_1, y_1, x_2, y_2 \le 8$) separated by single spaces. Here, (x_1, y_1) are coordinates of white piece, and (x_2, y_2) are coordinates of black piece. White and black pieces occupy different cells.

Output

Output file should contain N lines; i^{th} line should contain winning probability of baboon playing white in i^{th} starting position. Probabilities will be checked with precision 10^{-6}; we recommend to output real numbers as precisely as possible to prevent any precision issues.

If for some starting position neither side can win, your program should output zero for that position.

Examples

monkey.in	monkey.out
bishop queen 1 1 1 2 2	1
knight rook 1 1 1 2 3	0.73324215235465

Problem G. PCX

Input file:	pcx.in
Output file:	pcx.out
Time limit:	2 seconds
Memory limit:	256 megabytes

This time, Vasya has decided to write a computer game. In his game Vasya wants to use a lot of sprites — 256 color bitmaps. For sprites storage, Vasya has chosen **PCX** format, since the graphics library which he plans to use in game development supports only this format. Vasya wants the sprites to require as small disk space as possible, so he decided to learn **PCX** encoding/decoding algorithm. On the web site of Research Institute of Funny Language Examples (RIFLE), Vasya has found **PCX** image stream decoding program.

```
decode_pcx :: [Int] -> [Int]
decode_pcx [] = []
decode_pcx (x:xs)
    | x < 192 = x : (decode_pcx xs)
    | x>= 192 = (take (x-191) (repeat (head xs))) ++ (decode_pcx (tail xs))
```

After reading the functional language manual, Vasya understood that first line of the program defines that the function `decode_pcx` has one input parameter — list of integers and returns another list of integers. Second line says "if input list is empty then `decode_pcx` also returns an empty list". 3^{rd}, 4^{th} and 5^{th} lines describe behavior of function `decode_pcx` when x is first member of the input list and xs is the remainder (possibly empty) of input list.

Vasya has composed the table with the most important functions and operators:

x : xs	insert x to the front of list xs
xs ++ ys	returns concatenation of lists xs and ys
head xs	extract the first element of a list xs
tail xs	extract the elements after the head of a list xs
take n xs	applied to a list xs, returns the prefix of xs of length n, or xs itself if $n > \|xs\|$
repeat x	is an infinite list, with x being the value of every element

Finally Vasya understood how decoding algorithm works. Also he noticed that if we permute colors in the 256-color palette and recode the sprite using new color numbers, and then encode it to **PCX**, its size can be different than the size of original sprite encoded to **PCX**! Now, he wants to find the smallest encoded sprite such that after decoding it differs from the original sprite only by permutation of colors.

Input

The first line of input file contains a single positive integer N ($N \le 10\,000$) — number of pixels in sprite. The second line contains N integers a_i ($0 \le a_i \le 255$); here, a_i is the color of i^{th} pixel in sprite.

Output

Output file should contain optimal compressed sprite. After decompression, it should be possible to permute colors in the palette of the decompressed sprite so that it becomes equal to the given sprite.

First line should contain a single natural number M — number of bytes in the compressed sprite. Second line should contain M integers b_i ($0 \le b_i \le 255$) separated by single space. Each b_i describes i^{th} byte in compressed sprite. In case of multiple solutions, output any of them. Note that the palette always consists of all 256 colors.

Example

pcx.in	pcx.out
3 1 1 1	2 194 0

Problem H. Phone

Input file:	phone.in
Output file:	phone.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Research Institute of Friendly Telephones (RIFT) produces phones. In their current top model, keyboard was replaced by a modern touch-sensitive display with the keys drawn on it. One interesting feature of this model is a possibility to change keyboard layout — that is, to reorder the keys for user's convenience. Since this model was very cheap, Vasya bought it.

The default keyboard layout looks as follows:

Standard Keyboard Layout

0	**1**	**2**
3	**4**	**5**
6	**7**	**8**
*	**9**	#

Vasya wants to minimize his average phone number dialing time by changing the keyboard layout.

Vasya uses only one finger during dialing. For a given phone number, Vasya deduced three rules for evaluating its dialing time:

1. moving finger to the first key happens instantly
2. moving finger from cell (x_i, y_i) to cell (x_j, y_j) needs $|x_i - x_j| + |y_i - y_j|$ seconds
3. key pressing happens instantly

Vasya's phone book contains N phone numbers. Vasya dials these numbers with equal probability.

Note that due to a bug in RIFT software, it is not possible to change the positions of special keys '*' and '#'.

Vasya wrote a program to find the optimal layout, but it's too slow, so he needs your help.

Input

The first line of the input file contains a single positive integer $N \le 50\,000$ — how many phone numbers Vasya's phone book contains. Next N lines contain phone numbers, one number on each line. A phone number is a string which contains at least 2 and at most 20 decimal digits.

When dialing a number, Vasya picks each of the N lines in his phone book with equal probability $1/N$, even if some of these lines contain equal numbers.

Output

Output the optimal layout. Adhere to the output format shown in the example below. Layout should contain each character only once. Positions of special keys should be the same as in the standard layout. In case of multiple solutions, output any of them.

Example

phone.in	phone.out
1 1919191	190 234 567 *8#

Problem I. Another Prime Problem

Input file:	primpart.in
Output file:	primpart.out
Time limit:	2 seconds
Memory limit:	256 megabytes

R.I.P. (Research Institute of Primes) calls for help again!

Let N be a natural number. Consider all its non-repeating prime partitions — that is, partitions $N = p_1 + p_2 + \ldots + p_k$ such that p_i are different positive prime numbers. Among them, find a partition where the number of primes k is the largest possible.

Input

Input file contains a single natural number N ($N \le 1\,000\,000$).

Output

If such partition can't be constructed, output "IMPOSSIBLE" (without quotes).

Otherwise, first line must contain a single integer — the number of primes in partition. Second line must contain primes themselves, separated by spaces. You can output primes in any order.

If there are multiple partitions with the same number of primes, you can output any of them.

Examples

primpart.in	primpart.out
4	IMPOSSIBLE
30	5 2 3 5 7 13

Problem J. Progressions

Input file:	progressions.in
Output file:	progressions.out
Time limit:	2 seconds
Memory limit:	256 megabytes

RIPS (Research Institute of Progressions and Sequences) studies all about progressions and sequences. Today they just need to compare two progressions, so they invited Vasya to deal with it.

An *arithmetic progression* (or *arithmetic sequence*) is an arbitrary sequence of real numbers a_0, a_1, ..., a_k, ... such that there exists a fixed real number d such that $a_i = a_{i-1} + d$ for any $i > 0$.

A *geometric progression* (or *geometric sequence*) is an arbitrary sequence of real numbers b_0, b_1, ..., b_k, ... such that there exists a fixed real number q such that $b_i = b_{i-1} \cdot q$ for any $i > 0$.

Vasya needs to compare $S_1 = a_0 + a_1 + \ldots + a_k$ and $S_2 = b_0 + b_1 + \ldots + b_k$ for some arithmetic progression $\{a_i\}$ and geometric progression $\{b_i\}$ such that $a_0 = b_0 = x$ and $a_k = b_k = y$. But the task appeared to be very hard so he asked you to help him.

Input

Input file contains three non-negative integers x, y and k less than 10.

Output

If for any two progressions $\{a_i\}$ and $\{b_i\}$ satisfiying the given constraints $a_0 = b_0 = x$ and $a_k = b_k = y$, S_1 is less than S_2, output one character '<'. If for any two $\{a_i\}$ and $\{b_i\}$ satisfying the constraints, S_1 appears to be greater than S_2, output '>'. If for any two $\{a_i\}$ and $\{b_i\}$ satisfiying the constraints, S_1 is always equal to S_2, output '='. If the result of comparison may differ depending on choice of $\{a_i\}$ or $\{b_i\}$ satisfying the constraints, output '?'. If there are no such examples of $\{a_i\}$ or there are no such examples of $\{b_i\}$ satisfying the given constraints, output '#'.

Examples

progressions.in	progressions.out
1 2 3	>
0 1 1	#
1 1 0	=

Problem K. Lev Tolstoy

Input file:	tolstoy.in
Output file:	tolstoy.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Research Institute of Lev Tolstoy (RILT) is interested in getting the fifth book of "War and peace", Lev Tolstoy's most famous writing. However, there is a problem — they don't have the book, simply because there are only four of them. What they do have is a bunch of 1 000 000 monkeys with typewriters. They know that, according to the probabilitity theory, monkeys will sooner or later type the text of fifth book exactly as Lev Tolstoy would have done it himself. So, they commanded the monkeys to start typing. The result was fantastic — they now have 1 000 000 variants of the book.

Inspired by their success, Research Institute of Absolutely Impossible Graphs (RIAIG) decided to use their experience, as well as borrow their monkeys, for a similar task. They want to find a graph with N vertices which contains no cycles. The idea is that one of the monkeys could perhaps draw such a graph with more than $N - 1$ edge.

First, each monkey is given a graph with N vertices and no edges. Then, she starts to draw edges, one by one. At each step, she randomly chooses a pair of different vertices not yet connected by an edge; the probability is the same for all such pairs. If an edge between these vertices would lie on a cycle, she does not draw it and stops her work, otherwise she draws it and proceeds to the next step.

Vasya has recently visited RIAIG, and said that it is a foolish experiment. He claimed that for every natural number K, he knows the probability for the number of edges in a monkey's graph to be exactly K. And there are no chances to get an acyclic graph with more than $N - 1$ edge.

But, because N was large, Vasya could not count probabilities in his mind, so he asked you to write a program for him.

Input

The input file contains only one integer number $3 \le N \le 500$ — the number of vertices in the graph.

Output

Output $N - 1$ real number, one per line. The number on K^{th} line should represent the probability of the fact that the monkey's graph will have exactly K edges. Lines are numbered starting from one. The probabilities must be accurate with relative precision 10^{-6}. That is, your answer should lie within the range $0.999999 \cdot x \le y \le 1.000001 \cdot x$, where x is the precise answer, and y is your output. Exponential form of output is also accepted.

Examples

tolstoy.in	tolstoy.out
3	0 1
4	0 0.2 0.8

Problem L. (First Division Only!) Two Heaps

Input file:	heaps.in
Output file:	heaps.out
Time limit:	2 seconds
Memory limit:	64 megabytes

Two players play with two heaps initially composed of A and B stones. Both players have their sets of possible moves: $a_1, a_2, \ldots, a_k$ for the first, and $b_1, b_2, \ldots, b_l$ for the second. The first player can take a_i stones from any heap by its move (each time for only one i in range $1 \le i \le k$), and the second can take b_j. Players move in turn. If a player cannot move, she loses. Your task is to determine the winner!

Input

First line contains integers A and B. The second line starts with integer k, followed by k numbers a_i. The third line has form l b_1 b_2 $\ldots$ b_l. All numbers satisfy the following requirements: $1 \le A, B \le 1000$, $1 \le k, l \le 10$, $1 \le a_i, b_j \le 1000$.

Output

If first player wins, output `First`. Otherwise output `Second`.

Example

heaps.in	heaps.out
2 2 2 1 2 1 1	First
2 2 1 1 2 1 2	Second

Problem M. (First Division Only!) Sum of Squares

Input file:	sum.in
Output file:	sum.out
Time limit:	2 seconds
Memory limit:	64 megabytes

There are many ways to find for a given integer N integers $a_1, a_2, \ldots, a_k$ such that $a_1^2 + a_2^2 + \ldots + a_k^2 = N$. Your task is to do it so that their sum $a_1 + a_2 + \ldots + a_k$ would be minimal possible.

Input

The only integer in the input file is N. It does not exceed 10 000 by an absolute value.

Output

In the first line output k and minimal possible value of the sum. Then output k numbers a_i in following k lines. If there is more than one optimal solution, you must output one with the least possible k (but k must be at least one). If there is no solution, output file must consist of a single zero.

Example

sum.in	sum.out
0	1 0 0