Problem A. Achromatic Number

Input file:	achromatic.in
Output file:	achromatic.out
Time limit:	1 second
Memory limit:	256 megabytes

Chromatic number of the graph is the minimal number of colors needed to color its vertices so that no two vertices connected by an edge had the same color. Such coloring is called *proper*. It is well known that the problem of finding the chromatic number of a graph is NP-complete.

The proper coloring of the graph is called *achromatic* if each pair of distinct colors occurs at the ends of some edge. The achromatic number of a graph is the maximal number of colors that can be used to create the achromatic coloring.

For example, the achromatic number of a triangle is 3, because if we color all vertices of a triangle to different colors, each pair of colors would occur as the ends of an edge.

You are given n. Find the achromatic number of a cycle of length n and its corresponding achromatic coloring.

Input

Input file contains one integer number n ($3 \le n \le 1000$).

Output

The first line of the output file must contain one number a — the achromatic number of a cycle of length n. The second line must contain n integer numbers ranging from 1 to a and describe the corresponding proper achromatic coloring.

Examples

achromatic.in	achromatic.out
3	3 1 2 3

Problem B. Binary Search

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

Binary search is an algorithm that can be used to find an element in a sorted array. Consider the following code:

```
input: a[0..n - 1], x

l = 0;
r = n;
while (l < r - 1) {
    m = (l + r) / 2;
    if (a[m] <= x)
        l = m;
    else
        r = m;
}
if (a[l] == x)
    return true;
else
    return false;
```

Sometimes it happens that the binary search finds the occurence of an element in the array even if the array is not sorted. You are given n. Find the number of pairs $\langle a, x \rangle$ where a is the array of length n containing integer numbers from 1 to n, and x is an integer number from 1 to n, such that binary search above returns "`true`" if it is run on a and x as arguments.

Input

Input file contains one integer number n ($1 \le n \le 1000$).

Output

Output one integer number — the answer to the problem.

Examples

binary.in	binary.out
1	1
2	5

Problem C. Blind Flibs

Input file:	`blind.in`
Output file:	`blind.out`
Time limit:	1 second
Memory limit:	256 megabytes

Flibs are fantastic creatures intended to predict future. Flibs are often used in demonstration of genetic programming usage. In this problem we consider blind flibs that try to predict future but see nothing around them.

Future is presented as a word w consisting of zeroes and ones, concatenated to itself infinitely to get an infinite word w^*. For example if $w = 010011$, future is $w^* = 010011010011010011....$ Blind flib is a deterministic finite automaton that also prints a character on every transition.

Formally, blind flib can be in one of n states, from each state there is a transition marked with 0 and a transition marked with 1. A transition is a triple $\langle m, p, s \rangle$ where m is a character it is marked with, p is a character that is printed when taking this transition, and s is a state that the flib turns to after taking this transition.

Initially the flib prints some character (it can choose one) and turns to state 1. After that the flib acts as following. Let c be the last character printed by the flib. It chooses a transition from its current state that is marked with c and takes it. The process continues infinitely.

Let z be an inifinite word printed by the blind flib. Let $e(k)$ be the number of positions among first k that contain the same characters in z and w^*. The flib's prediction level is $P = \lim\limits_{n \to +\infty} \frac{e(n)}{n}$. The greater is prediction level — the better is the flib.

For example, clearly there exists blind flib with number of states equal to the length of w that has the prediction level 1.0. Its states correspond to positions in w, and it always correctly predicts the next character.

You are given a word w. For each n from 1 to length of w find the maximal prediction level of some blind flib with n states.

Input

The input file contains the given word w (length of w is between 1 and 200, inclusive).

Output

Print length of w numbers: n from 1 to length of w find the maximal prediction level of some blind flib with n states.

Examples

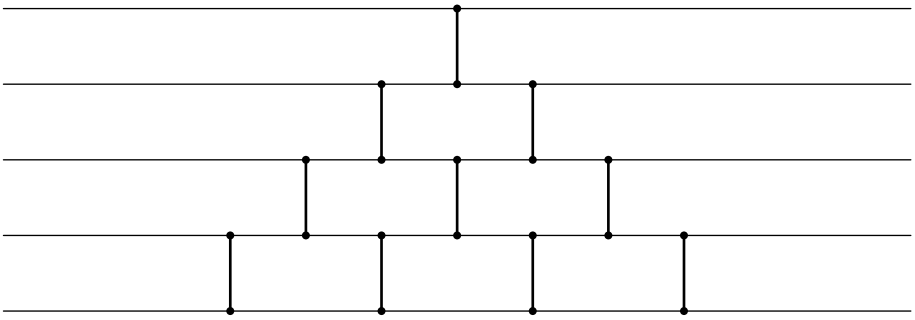
blind.in	blind.out
010011	0.6666666666666667 0.8333333333333334 1.0 1.0 1.0 1.0

Problem D. Bubble Sort

Input file:	`bubble.in`
Output file:	`bubble.out`
Time limit:	1 second
Memory limit:	256 megabytes

Consider n parallel horizontal wires each of which can carry a number from 1 to n. The numbers are considered to be travelling along the wires from left to right. You can put a comparator to connect two wires. If the numbers on the wires to the left of the comparator are x and y, after the comparator the numbers on the wires are sorted: $\min(x, y)$ is on the upper wire, and $\max(x, y)$ is not lower wire.

Comparator networks can be used to represent some sorting algorithms. For example, the network below represents bubble sorting algorithm of 5 numbers.



The comparator in a network can be active (if it swaps the numbers on its wires) or inactive (if it doesn't swap the numbers). For example, if the network above is run on input $\langle 5, 4, 3, 2, 1\rangle$ all comparators are active, but if it is run on input $\langle 2, 1, 3, 4, 5\rangle$ only the topmost one is.

You are given a comparator network that represents bubble sort and which comparators are active. Find out whether there exists a permutation of numbers from 1 to n such that if it reperesents the initial values on the wires, the given set of comparators is active.

Input

The first line of the input file contains n ($2 \le n \le 200$). After that $n(n-1)/2$ numbers follow, describing comparators from top to bottom, from left to right. Each number is 1 if the corresponding comparator is active, or 0 if it is not.

Output

If there exists a permutation forcing the given set of comparators to be active, print "`YES`" at the first line of the output file. The second line must contain the permutation itself. If there exists no such permutation, print "`NO`" to the output file.

Examples

bubble.in	bubble.out
3 1 1 1	YES 3 2 1
3 0 1 1	NO

Problem E. Word Cover

Input file:	`cover.in`
Output file:	`cover.out`
Time limit:	1 second
Memory limit:	256 megabytes

A word α is said to cover a word β if for each position in a word β there exists an occurance of α as a subword of β that contains this position. For example, the word "`aba`" covers the word "`abaabaababa`", but doesn't cover the word "`baba`". Clearly a word covers itself.

You are given a word w. For each prefix $w[1..k]$ of w find the length of the shortest word that covers this prefix.

Input

Input file contains w consisting of small letters of the English alphabet. The length of w doesn't exceed 250 000.

Output

For each k from 1 to length of w print the length of the shortest word that covers $w[1..k]$.

Examples

cover.in	cover.out
abaabaababa	1 2 3 4 5 3 4 5 3 10 3

Problem F. Permutations with Monotonic Segments

Input file:	`monotonic.in`
Output file:	`monotonic.out`
Time limit:	1 second
Memory limit:	256 megabytes

Consider a permutation π of n numbers from 1 to n. It is said to have monotonic segment $[i..j]$ if the following conditions are true:

- $\pi[i] > \pi[i+1] > \ldots > \pi[j]$;
- $i = 1$ or $\pi[i-1] < \pi[i]$;
- $j = n$ or $\pi[j] < \pi[j+1]$.

Given n, k and positive integer numbers $a_1, a_2, \ldots, a_k$ such that $a_1 + a_2 + \ldots + a_k = n$, find the number of permutations of n numbers that have monotonic segments $[1..a_1]$, $[a_1 + 1..a_1 + a_2]$, $\ldots$, $[a_1 + a_2 + \ldots + a_{k-1} + 1..a_1 + \ldots + a_k]$ (these segments have lengths $a_1, a_2, \ldots, a_k$, respectively).

The answer can be quite large, so find it modulo 1000000007

Input

The first line of the input file contains n and k ($1 \le n \le 300$, $1 \le k \le n$). The second line contains $a_1, a_2, \ldots, a_k$.

Output

Output the number of permutations of n numbers with the given monotonic segments.

Examples

monotonic.in	monotonic.out
4 2 2 2	5

In the given example the following permutations must be counted: $\langle 2143 \rangle$, $\langle 3142 \rangle$, $\langle 3241 \rangle$, $\langle 4132 \rangle$, $\langle 4231 \rangle$.

Problem G. Persistent Queue

Input file:	queue.in
Output file:	queue.out
Time limit:	1 second
Memory limit:	256 megabytes

Persistent data structures are designed to allow access and modification of any version of data structure. In this problem you are asked to implement persistent queue.

Queue is the data structure that maintains a list of integer numbers and supports two operations: push and pop. Operation push(x) adds x to the end of the list. Operation pop returns the first element of the list and removes it.

In persistent version of queue each operation takes one additional argument v. Initially the queue is said to have version 0. Consider the i-th operation on queue. If it is push(v, x), the number x is added to the end of the v-th version of queue and the resulting queue is assigned version i (the v-th version is not modified). If it is pop(v), the front number is removed from the v-th version of queue and the resulting queue is assigned version i (similarly, version v remains unchanged).

Given a sequence of operations on persistent queue, print the result of all pop operations.

Input

The first line of the input file contains n — the number of operations ($1 \le n \le 200\,000$). The following n lines describe operations. The i-th of these lines describes the i-th operation. Operation push(v, x) is described as "`1` v x", operation pop(v) is described as "`-1` v". It is guaranteed that pop is never applied to an empty queue. Elements pushed to the queue fit standard signed 32-bit integer type.

Output

For each pop operation print the element that was extracted.

Examples

queue.in	queue.out
10 1 0 1 1 1 2 1 2 3 1 2 4 -1 3 -1 5 -1 6 -1 4 -1 8 -1 9	1 2 3 1 2 4

Problem H. Sea Port

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

You are writing software for a large seaport. The fleet of n cargo ships numbered from 1 to n plans to arrive to the port in a next several days. For each ship the day of its arrival d_i is known. Each ship carries s_i tonnes of cargo of one of c possible types.

There are m cranes numbered from 1 to m working in the port. Each crane can unload ships with some predefined types of cargo. The time it takes for the crane to unload a ship carrying s_i tonnes of cargo is $\lceil s_i/v_i \rceil$ days, where v_i is the speed of the crane.

The software you are requested to write must distribute jobs of unloading ships between cranes. Initially all cranes are idle. When the ship arrives the idle crane is selected to unload the ship. If there are several idle cranes avaiable that can unload the type of cargo that the ship carries, the one is selected which became idle for the last time earliest among them. If there are several cranes that became idle for the last time at the same moment, the one is selected which has the smaller number. If several ships come at the same time, ships are assigned crane in order of numbers of these ships. If there are no cranes available to unload the ship it is put to the dock.

When the crane finishes unloading the ship, if there is a ship in the dock that it can unload, it immediately starts unloading such ship. If there are several ships, the ship which arrived earliest among them is served. If there are several ships that arrived at the same time, the ship which has smaller number is served. If there are no ships available, the crane becomes idle. If several cranes finish their work at the same time, they choose ships in order of numbers of these cranes.

If the crane finished its current job at the same time the ship comes to the port, it can be selected for unloading this ship.

For each ship you must detect which crane would unload it.

Input

The first line of the input file contains three integer numbers: n, m and c ($1 \le n, m \le 100\,000$, $1 \le c \le 10$).

The following n lines contain three integer numbers each and describe ships. Each ship is described by the day it arrives d_i ($0 \le d_i \le 10^9$), its cargo type c_i ($1 \le c_i \le c$) and the size of its cargo s_i ($1 \le s_i \le 10^9$).

The following m lines describe cranes. Each crane is described by its speed v_i ($1 \le v_i \le 10^9$) followed by t_i — the number of types of cargo this crane can unload, followed by t_i numbers — the types of cargo themselves.

It is guaranteed that each cargo type can be unloaded by at least one crane.

Output

Output n lines — for each ship print the number of the crane that would unload it.

Examples

seaport.in	seaport.out
5 3 2 0 1 10 1 1 10 2 2 5 4 1 10 5 1 12 1 1 1 2 1 2 2 2 1 2	1 3 2 3 1

Problem I. Shooting Game

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

Alice and Bob play shooting game on a plane.

The game proceeds as follows. There are n obstacles on the plane, each obstacle is a segment. First, Alice selects some point on the plane that doesn't belong to a line containing an obstacle and stands there. Then Bob selects some other point and stands at it. After that Alice shoots at Bob from the gun.

Bob always selects such point that there are as many obstacles as possible between him and Alice. If the line connecting Alice and Bob passes through the end of an obstacle, this obstacle is considered to be between Alice and Bob.

Alice would like to shoot Bob very much, so she tries to select such point that Bob couldn't hide behind too many obstacles. Help Alice to choose such point on the plane, that after Bob selects his point, the number of obstacles between Alice and Bob was minimal possible.

Input

The first line of the input file contains n — the number of obstacles ($1 \le n \le 8$). The following n lines contain four integer numbers x_1, y_1, x_2, y_2 each — the coordinates of the ends of the corresponding obstacle. Obstacles have no common points. Coordinates do not exceed 100 by their absolute values.

Output

The first line of the output file must contain k — the minimal number of obstacles Alice can ensure to be between her and Bob. The second line of the output file must contain two real numbers — coordinates of the point Alice should choose. The point must not belong to any line containing an obstacle. Print as many digits after the decimal point as possible.

Examples

shooting.in	shooting.out
2 0 0 2 0 0 2 2 2	1 1.000000000000000000 1.000000000000000000