Codeforces Round #455 (Div. 2)

A. Generate Login

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

The preferred way to generate user login in Polygon is to concatenate a prefix of the user's first name and a prefix of their last name, in that order. Each prefix must be non-empty, and any of the prefixes can be the full name. Typically there are multiple possible logins for each person.

You are given the first and the last name of a user. Return the alphabetically earliest login they can get (regardless of other potential Polygon users).

As a reminder, a prefix of a string s is its substring which occurs at the beginning of s: "a", "ab", "abc" etc. are prefixes of string "{abcdef}" but "b" and 'bc" are not. A string a is alphabetically earlier than a string b, if a is a prefix of b, or a and b coincide up to some position, and then a has a letter that is alphabetically earlier than the corresponding letter in b: "a" and "ab" are alphabetically earlier than "ac" but "b" and "ba" are alphabetically later than "ac".

Input
The input consists of a single line containing two space-separated strings: the first and the last names. Each character of each string is a lowercase English letter. The length of each string is between 1 and 10, inclusive.

Output
Output a single string — alphabetically earliest possible login formed from these names. The output should be given in lowercase as well.

Examples

input
harry potter
output
hap

input
tom riddle
output
tomr

B. Segments

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

You are given an integer N. Consider all possible segments on the coordinate axis with endpoints at integer points with coordinates between 0 and N, inclusive; there will be $\frac{n(n+1)}{2}$ of them.

You want to draw these segments in several layers so that in each layer the segments don't overlap (they might touch at the endpoints though). You **can not** move the segments to a different location on the coordinate axis.

Find the minimal number of layers you have to use for the given N.

Input

The only input line contains a single integer N ($1 \le N \le 100$).

Output

Output a single integer - the minimal number of layers required to draw the segments for the given N.

Examples

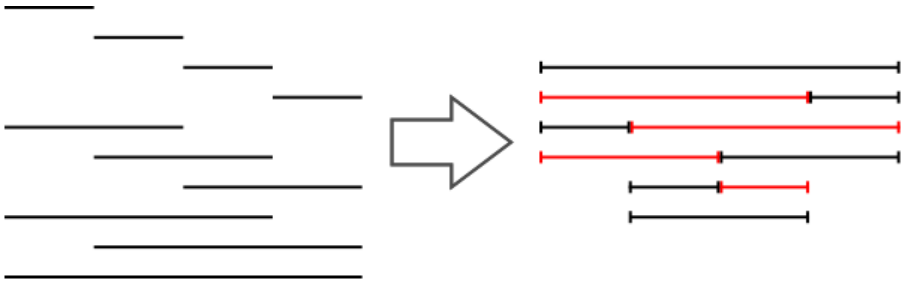
input
2
output
2

input
3
output
4

input
4
output
6

Note

As an example, here are the segments and their optimal arrangement into layers for $N = 4$.



C. Python Indentation

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

In Python, code blocks don't have explicit begin/end or curly braces to mark beginning and end of the block. Instead, code blocks are defined by indentation.

We will consider an extremely simplified subset of Python with only two types of statements.

Simple statements are written in a single line, one per line. An example of a simple statement is assignment.

For statements are compound statements: they contain one or several other statements. For statement consists of a header written in a separate line which starts with "for" prefix, and loop body. Loop body is a block of statements indented one level further than the header of the loop. Loop body can contain both types of statements. Loop body can't be empty.

You are given a sequence of statements without indentation. Find the number of ways in which the statements can be indented to form a valid Python program.

Input

The first line contains a single integer N ($1 \le N \le 5000$) — the number of commands in the program. N lines of the program follow, each line describing a single command. Each command is either "f" (denoting "for statement") or "s" ("simple statement"). It is guaranteed that the last line is a simple statement.

Output

Output one line containing an integer - the number of ways the given sequence of statements can be indented modulo $10^9 + 7$.

Examples

input
4 s f f s
output
1

input
4 f s f s
output
2

Note

In the first test case, there is only one way to indent the program: the second for statement must be part of the body of the first one.

```
simple statement
for statement
    for statement
        simple statement
```

In the second test case, there are two ways to indent the program: the second for statement can either be part of the first one's body or a separate statement following the first one.

```
for statement
    simple statement
    for statement
        simple statement
```

or

```
for statement
    simple statement
for statement
    simple statement
```

D. Colorful Points

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

You are given a set of points on a straight line. Each point has a color assigned to it. For point a, its neighbors are the points which don't have any other points between them and a. Each point has at most two neighbors - one from the left and one from the right.

You perform a sequence of operations on this set of points. In one operation, you delete all points which have a neighbor point of a different color than the point itself. Points are deleted simultaneously, i.e. first you decide which points have to be deleted and then delete them. After that you can perform the next operation etc. If an operation would not delete any points, you can't perform it.

How many operations will you need to perform until the next operation does not have any points to delete?

Input

Input contains a single string of lowercase English letters 'a'-'z'. The letters give the points' colors in the order in which they are arranged on the line: the first letter gives the color of the leftmost point, the second gives the color of the second point from the left etc.

The number of the points is between 1 and 10^6.

Output

Output one line containing an integer - the number of operations which can be performed on the given set of points until there are no more points to delete.

Examples

input
aabb
output
2

input
aabcaa
output
1

Note

In the first test case, the first operation will delete two middle points and leave points "ab", which will be deleted with the second operation. There will be no points left to apply the third operation to.

In the second test case, the first operation will delete the four points in the middle, leaving points "aa". None of them have neighbors of other colors, so the second operation can't be applied.

E. Coprocessor

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a program you want to execute as a set of tasks organized in a dependency graph. The dependency graph is a directed acyclic graph: each task can depend on results of one or several other tasks, and there are no directed circular dependencies between tasks. A task can only be executed if all tasks it depends on have already completed.

Some of the tasks in the graph can only be executed on a coprocessor, and the rest can only be executed on the main processor. In one coprocessor call you can send it a set of tasks which can only be executed on it. For each task of the set, all tasks on which it depends must be either already completed or be included in the set. The main processor starts the program execution and gets the results of tasks executed on the coprocessor automatically.

Find the minimal number of coprocessor calls which are necessary to execute the given program.

Input

The first line contains two space-separated integers N ($1 \le N \le 10^5$) — the total number of tasks given, and M ($0 \le M \le 10^5$) — the total number of dependencies between tasks.

The next line contains N space-separated integers $E_i \in \{0, 1\}$. If $E_i = 0$, task i can only be executed on the main processor, otherwise it can only be executed on the coprocessor.

The next M lines describe the dependencies between tasks. Each line contains two space-separated integers T_1 and T_2 and means that task T_1 depends on task T_2 ($T_1 \neq T_2$). Tasks are indexed from 0 to N - 1. All M pairs (T_1, T_2) are distinct. It is guaranteed that there are no circular dependencies between tasks.

Output

Output one line containing an integer — the minimal number of coprocessor calls necessary to execute the program.

Examples

input
4 3 0 1 0 1 0 1 1 2 2 3
output
2

input
4 3 1 1 1 0 0 1 0 2 3 0
output
1

Note

In the first test, tasks 1 and 3 can only be executed on the coprocessor. The dependency graph is linear, so the tasks must be executed in order 3 -> 2 -> 1 -> 0. You have to call coprocessor twice: first you call it for task 3, then you execute task 2 on the main processor, then you call it for for task 1, and finally you execute task 0 on the main processor.

In the second test, tasks 0, 1 and 2 can only be executed on the coprocessor. Tasks 1 and 2 have no dependencies, and task 0 depends on tasks 1 and 2, so all three tasks 0, 1 and 2 can be sent in one coprocessor call. After that task 3 is executed on the main processor.

F. AND-permutations

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

Given an integer N, find two permutations:

1. Permutation p of numbers from 1 to N such that $p_i \neq i$ and $p_i \ \& \ i = 0$ for all $i = 1, 2, ..., N$.
2. Permutation q of numbers from 1 to N such that $q_i \neq i$ and $q_i \ \& \ i \neq 0$ for all $i = 1, 2, ..., N$.

$\&$ is the bitwise AND operation.

Input

The input consists of one line containing a single integer N ($1 \leq N \leq 10^5$).

Output

For each subtask, if the required permutation doesn't exist, output a single line containing the word "NO"; otherwise output the word "YES" in the first line and N elements of the permutation, separated by spaces, in the second line. If there are several possible permutations in a subtask, output any of them.

Examples

input
3

output
NO NO

input
6

output
YES 6 5 4 3 2 1 YES 3 6 2 5 1 4