Problem A. A Journey to Greece

Source file name:	journey.c, journey.cpp, journey.java
Input:	`Standard`
Output:	`Standard`

For a long time Tim wanted to visit Greece. He has already purchased his flight to and from Athens. Tim has a list of historical sites he wants to visit, e.g., Olympia and Delphi. However, due to recent political events in Greece, the public transport has gotten a little complicated. To make the Greek happy and content with their new government, many short-range bus and train lines have been created. They shall take the citizens around in their neighborhoods, to work or to their doctor. At the same time, long-range trains that are perfect for tourists have been closed down as they are too expensive. This is bad for people like Tim, who really likes to travel by train. Moreover, he has already purchased the Greece' Card for Public Conveyance (GCPC) making all trains and buses free for him.

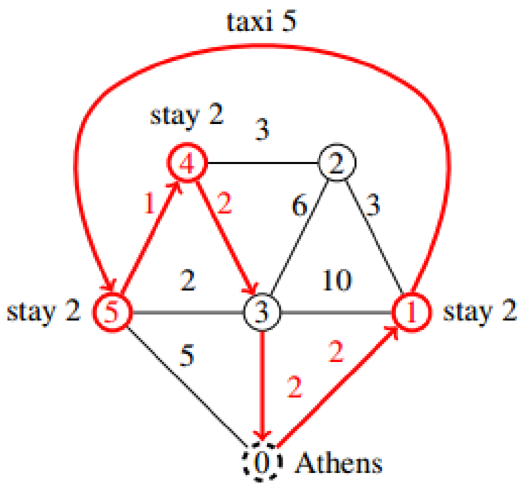


Figure A.1: Visual representation of the Sample Input: Tim's tour has length 18.

Despite his preferred railway lines being closed down, he still wants to make his travel trough Greece. But taking all these local bus and train connections is slower than expected, so he wants to know whether he can still visit all his favorite sites in the timeframe given by his flights. He knows his schedule will be tight, but he has some emergency money to buy a single ticket for a special Greek taxi service. It promises to bring you from any point in Greece to any other in a certain amount of time. For simplicity we assume, that Tim does never have to wait for the next bus or train at a station. Tell Tim, whether he can still visit all sites and if so, whether he needs to use this taxi ticket.

Input

The first line contains five integers N, P, M, G and T, where N denotes the number of places in Greece, P the number of sites Tim wants to visit, M the number of connections, G the total amount of time Tim can spend in Greece, and T the time the taxi ride takes ($1 \leq N \leq 2 \cdot 10^4$, $1 \leq P \leq 15$, $1 \leq M, G \leq 10^5$,$1 \leq T \leq 500$).

Then follow P lines, each with two integers p_i and t_i , specifying the places Tim wants to visit and the time Tim spends at each site ($0 \leq p_i < N$, $1 \leq t_i \leq 500$). The sites pi are distinct from each other.

Then follow M lines, each describing one connection by three integers s_i , d_i and t_i , where s_i and d_i specify the start and destination of the connection and t_i the amount of time it takes ($0 \leq s_i$, $d_i < N$; $1 \leq t_i \leq 500$).

All connections are bi-directional. Tim's journey starts and ends in Athens, which is always the place 0.

Output

Print either "impossible", if Tim cannot visit all sites in time, "possible without taxi", if he can visit all sites without his taxi ticket, or "possible with taxi", if he needs the taxi ticket.

Example

Input	Output

6 3 10 18 5
1 2
4 2
5 2
0 1 2
1 2 3
2 4 3
1 3 10
2 3 6
0 3 2
3 4 2
4 5 1
3 5 2
0 5 5
	possible with taxi

Problem B. Bounty Hunter II

Source file name:	bounty.c, bounty.cpp, bounty.java
Input:	`Standard`
Output:	`Standard`

Spike the bounty hunter is tracking another criminal through space. Luckily for him hyperspace travel has made the task of visiting several planets a lot easier. Each planet has a number of Astral Gates; each gate connects with a gate on another planet. These hyperspace connections are, for obvious safety reasons, one-way only with one gate being the entry point and the other gate being the exit point from hyperspace. Furthermore, the network of hyperspace connections must be loop-free to prevent the Astral Gates from exploding, a tragic lesson learned in the gate accident of 2022 that destroyed most of the moon.

While looking at his star map Spike wonders how many friends he needs to conduct a search on every planet. Each planet should not be visited by more than one friend otherwise the criminal might get suspicious and flee before he can be captured. While each person can start at a planet of their choosing and travel along the hyperspace connections from planet to planet they are still bound by the limitations of hyperspace travel.

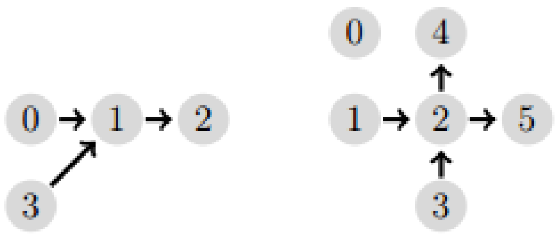


Figure B.1: Illustration of the Sample Inputs.

Input

The input begins with an integer N specifying the number of planets ($0 < N \leq 1000$). The planets are numbered from 0 to $N - 1$. The following N lines specify the hyperspace connections. The i-th of those lines first contains the count of connections K ($0 \leq K \leq N - 1$) from planet i followed by K integers specifying the destination planets.

Output

Output the minimum number of persons needed to visit every planet.

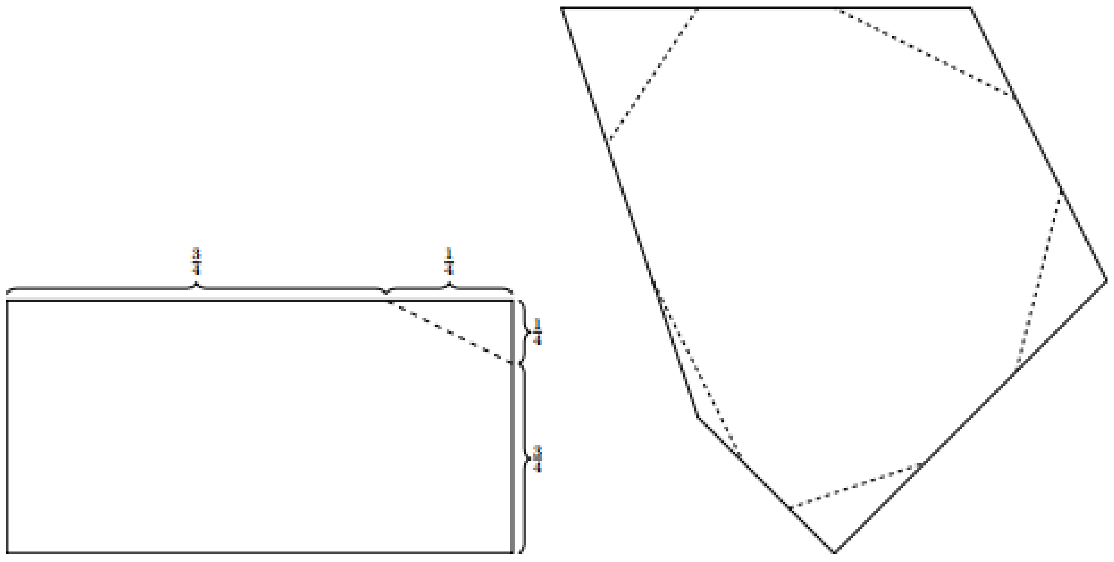
Example

Input	Output
4	2
1 1	
1 2	
0	
1 1	
6	4
0	
1 2	
2 4 5	
1 2	
0	
0	

Problem C. Cake

Source file name:	cake.c, cake.cpp, cake.java
Input:	`Standard`
Output:	`Standard`

Sophie loves to bake cakes and share them with friends. For the wedding of her best friend Bea she made a very special cake using only the best ingredients she could get and added a picture of the engaged couple on top of the cake. To make it even more special she did not make it round or square, but made a custom convex shape for the cake. Sophie decided to send the cake by a specialized carrier to the party. Unfortunately, the cake is a little too heavy for their default cake package and the overweight fees are excessive. Therefore, Sophie decides to remove some parts of the cake to make it a little lighter.
Sophie wants to cut the cake the following way: First, she chooses a real number $s \geq 2$. For each vertex and each incident edge of the cake she marks where $1/s$ of the edge's length is. Afterwards, she makes a direct cut between the two markings for each vertex and removes the vertex that way.



(a) Cutting the upper-right corner of a rectangle with $s = 4$

(b) Cutting a cake with $s = 3$

Figure C.1: Illustration of the first two Sample Inputs.

Sophie does not want to cut more from the cake than necessary for obvious reasons. Can you tell her how to choose s?

Input

The first line contains a floating point number a and an integer N, where a denotes the ratio of the cake's weight allowed by the carrier and N the number of vertices of the cake ($0.25 \leq a < 1$, $3 \leq N \leq 100$). a will be specified with at most 7 digits after the decimal point.
Then follow N lines, each describing one of the cake's vertices with two integers x_i and y_i , the coordinates of the vertex ($0 \leq x_i, y_i \leq 10^8$ for all $1 \leq i \leq N$). The vertices are given in the order in which they form a strictly convex shape.
You may safely assume that the weight is uniformly distributed over the area of the cake. Furthermore, it will always be possible to cut the cake with some $2 \leq s \leq 1000$ such that the proportion of the remaining

cake is a of the original weight.

Output

Print a line containing s, the biggest value as specified above such that the remaining cake's weight is at most the proportion a of its original weight. Your answer will be considered correct if the absolute error is at most 10^{-4} .

Example

Input	Output

0.875 4
0 0
8 0
8 4
0 4
	4.000000

0.85 5
6 0
12 6
9 12
0 12
3 3
	3.000000

0.999998 4
20008 10000
15004 15005
10001 20009
15005 15004
	1000.000000

Problem D. Carpets

Source file name:	carpets.c, carpets.cpp, carpets.java
Input:	`Standard`
Output:	`Standard`

The computer science Professor Toving Liles loves the floor tiles in his office so much that he wants to protect them from damage by careless students. Therefore, he would like to buy cheap small rectangular carpets from the supermarket and cover the floor such that:

1. The entire floor is covered.
2. The carpets do not overlap.
3. The carpets are rotated arbitrarily.
4. No carpet is cut into pieces.

But when checking the supermarket's stock he begins to wonder whether he can accomplish his plan at all. Can you help him?

Input

The first line contains two integers W and H describing the size of his room ($1 \leq W$, $H \leq 100$). The second line contains an integer c, denoting the number of different carpet colors the supermarket has in stock ($1 \leq c \leq 7$). Each of the following c lines consists of three integers a_i , w_i , and h_i , which means: the supermarket's stock contains a_i carpets of size w_i , h_i and color i ($1 \leq a_i \leq 7$, $1 \leq w_i \leq 100$, $1 \leq h_i \leq 100$). The supermarket has at most 7 carpets, i.e. $\sum_i a_i \leq 7$.

Output

For the given room dimensions and the supermarket's stock of carpets, print "yes" if it is possible to cover the room with carpets as specified above and "no" otherwise.

Example

Input	Output

`2 4`
`2`
`3 1 3`
`2 2 1`
	`yes`

`100 100`
`3`
`4 42 42`
`1 100 16`
`1 32 42`
	`no`

Problem E. Change of Scenery

Source file name:	change.c, change.cpp, change.java
Input:	`Standard`
Output:	`Standard`

Every day you drive to work using the same roads as it is the shortest way. This is efficient, but over time you have grown increasingly bored of seeing the same buildings and junctions every day. So you decide to look for different routes. Of course you do not want to sacrifice time, so the new way should be as short as the old one. Is there another way that differs from the old one in at least one street?

Input

The first line of the input starts with three integers N, M and K, where N is the number of junctions and M is the number of streets in your city, and K is the number of junctions you pass every day ($1 \leq K \leq N \leq 10000$, $0 \leq M \leq 1000000$).
The next line contains K integers, the (1-based) indices of the junctions you pass every day. The first integer in this line will always be 1, the last integer will always be N. There is a shortest path from 1 to N along the K junctions given.
M lines follow. The i-th of those lines contains three integers a_i, b_i, c_i and describes a street from junction a_i to junction b_i of length c_i ($1 \leq a_i, b_i \leq N$, $1 \leq c_i \leq 10000$). Streets are always undirected.
Note that there may be multiple streets connecting the same pair of junctions. The shortest path given uses for every pair of successive junctions a and b a street of minimal length between a and b.

Output

Print one line of output containing "yes" if there is another way you can take without losing time, "no" otherwise.

Example

Input	Output

`3 3 3`
`1 2 3`
`1 2 1`
`2 3 2`
`1 3 3`
	`yes`

`4 5 2`
`1 4`
`1 2 2`
`2 4 1`
`1 3 1`
`3 4 2`
`1 4 2`
	`no`

Problem F. Divisions

Source file name:	divisions.c, divisions.cpp, divisions.java
Input:	`Standard`
Output:	`Standard`

David is a young boy and he loves numbers. Recently he learned how to divide two numbers. David divides the whole day. He is happy if the result of the division is an integer, but he is not very amused if this is not the case. After quite a while he decided to use only a single dividend each day.
The parents of David are very careful and they would like to ensure that David experiences enough happiness. Therefore they decide which number David will use as the dividend for this day.
There is still a problem: The parents are not very good at math and don't know how to calculate the number of positive integral divisors for a given dividend N, which lead to an integral result. Now it's up to you to help David's parents.

Input

The single input line contains the single integer N, where N is chosen as a dividend $(1 \leq N \leq 10^{18})$.

Output

Print the number of positive integral divisors of N that lead to an integral result of the division.

Example

Input	Output
12	6
999999999999999989	2
100000007700000049	4

Problem G. Extreme Sort

Source file name:	extreme.c, extreme.cpp, extreme.java
Input:	`Standard`
Output:	`Standard`

John likes sorting algorithms very much. He has studied quicksort, merge sort, radix sort, and many more.
A long time ago he has written a lock-free parallel string sorting program. It was a combination of burstsort and multi-key quicksort. To implement burstsort you need to build a tree of buckets. For each input string you walk through the tree and insert part of the string into the right bucket. When a bucket fills up, it "bursts" and becomes a new subtree (with new buckets). Well, enough about the past.

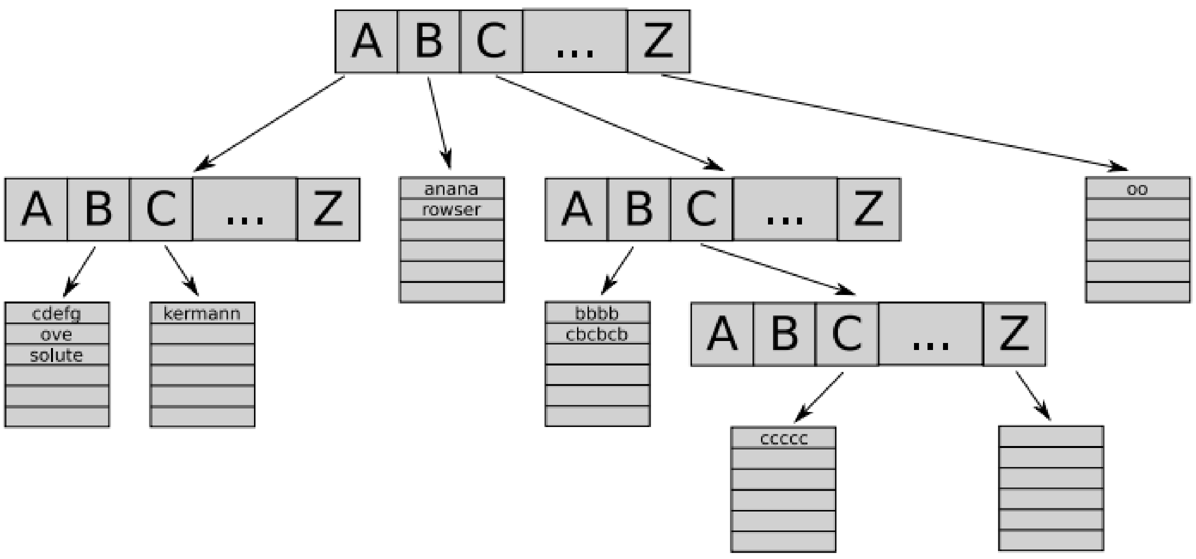


Figure G.1: Burstsort data structure

Today John is playing with sorting algorithms again. This time it's numbers. He has an idea for a new algorithm, "extreme sort". It's extremely fast, performance levels are OVER NINETHOUSAND. Before he tells anyone any details, he wants to make sure that it works correctly.
Your task is to help him and verify that the so-called *extreme property* holds after the first phase of the algorithm. The extreme property is defined as min $(x_{i,j}) \leq 0$, where

$$x_{i,j} = \begin{cases} a_j - a_i & \text{for} \quad 1 \leq i < j \leq N \\ 9001 & \text{otherwise} \end{cases}$$

Input

The first line contains a single integer N ($1 \leq N \leq 1024$). The second line contains N integers $a_1\ a_2 \ldots a_N$ ($1 \leq a_i \leq 1024$).

Output

Print one line of output containing "yes" if the extreme property holds for the given input, "no" otherwise.

Example

Input	Output
2 1 2	yes
4 2 1 3 4	no

Problem H. Legacy Code

Source file name:	legacy.c, legacy.cpp, legacy.java
Input:	`Standard`
Output:	`Standard`

Once again you lost days refactoring code, which never runs in the first place. Enough is enough - your time is better spent writing a tool that finds unused code!
Your software is divided into packages and executables. A package is a collection of methods. Executables are packages defining among other methods exactly one method with name `PROGRAM`. This method is executed on the start of the corresponding executable. Ordinary packages have no method named `PROGRAM`.
Each method is uniquely identified by the combination of package and method names. E.g. the method with the identifier `SuperGame::PROGRAM` would be the main method of the executable `SuperGame`.
For every method in your software you are given a list of methods directly invoking it. Thus you can easily identify methods, that are never called from any method. However, your task is more challenging: you have to find unused methods. These are methods that are never reached by the control flow of any executable in your software.

Input

The first line of the input contains an integer N, the number of methods in your software ($1 \leq N \leq 400$).
Each method is described by two lines, totaling in $2 \cdot N$ lines. The first line consists of the unique identifier of the method and k_i, the number of methods directly invoking this one ($0 \leq k_i \leq N$). The second line consists of a set of k_i identifiers of these calling methods or is empty if there are no such methods, i.e. $k_i = 0$.
Method identifiers consist of a package name followed by two colons and a method name like `Packagename::Methodname`. Both strings, the package and the method name, each consist of up to 20 lowercase, uppercase characters or digits (a-z, A-Z, 0-9).
There will be exactly N different method identifiers mentioned in the input.

Output

A line containing the number of unused methods in your software.

Example

Input	Output
2 SuperGame::PROGRAM 0 HelpPackage::HelpFunction 2 HelpPackage::HelpFunction SuperGame::PROGRAM	0
2 Loop::CallA 1 Loop::CallB Loop::CallB 1 Loop::CallA	2
2 SuperGame::PROGRAM 1 SuperServer42::PROGRAM SuperServer42::PROGRAM 1 SuperServer42::PROGRAM	0

Problem I. Milling machines

Source file name:	milling.c, milling.cpp, milling.java
Input:	`Standard`
Output:	`Standard`

A fab lab is an open, small-scale workshop where you can create or fabricate almost anything you want mostly by using computer controlled tools like a laser cutter or a 3D printer. The FAU fab lab recently got a CNC milling machine. Using the milling machine you can cut or remove material with different tools from the surface of a workpiece. It is controlled via a computer program.
I sometimes wondered what happens if multiple different shaped workpieces are sent through the same milling program. For simplification assume that we have only two dimensional workpieces without holes. A milling program consists of multiple steps; each step describes where the milling machine has to remove material (using different tools) from the top of the surface.

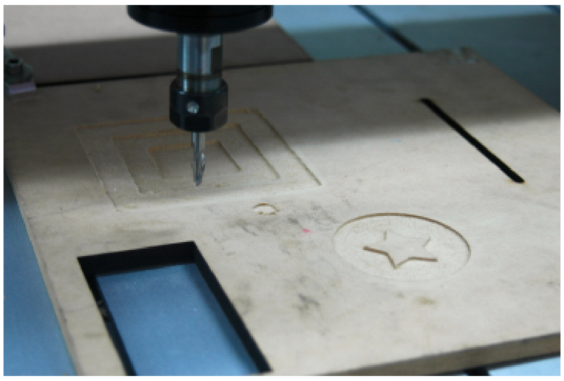


Photo by aurelie ghalim on Flickr

Input

The first line consists of two integers W and S, where W gives the number of workpieces and S the number of steps in the milling program ($1 \leq W, S \leq 10^4$). The next line consists of two integers X and Y , where X gives the width and Y gives the maximal possible height of workpieces ($1 \leq X, Y \leq 100$).
Then follow W lines, each describing one workpiece. Each workpiece description consists of X non-negative integers specifying the surface height in that column.
Then follow S lines, each describing one milling step of the milling program. Each milling step description consists of X non-negative integers s_i ($0 \leq s_i \leq Y$) specifying the amount of surface to cut off in each column (relative to the height of the milling area, i.e. Y , not relative to the top of the workpiece). See Fig. I.1 for details.

Output

For each workpiece, output one line containing X integers specifying the remaining surface heights (in the same order as in the input).

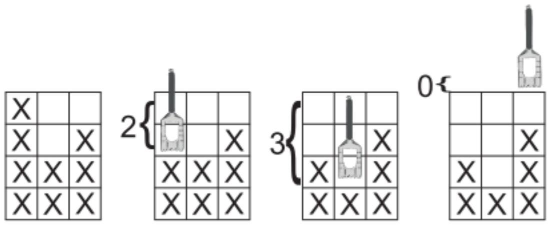


Figure I.1: Second workpiece in first sample: initial workpiece followed by milling in each column – the value in the milling program determines the vertical position of the cutter head.

Example

Input	Output

2 1
3 4
4 4 4
4 2 3
2 3 0
	2 1 4
	2 1 3

1 3
10 100
11 22 33 44 55 66 77 88 99 100
1 100 1 100 1 100 1 100 1 100
58 58 58 58 58 58 58 58 58 58
42 42 42 42 42 42 42 42 66 42
	11 0 33 0 42 0 42 0 34 0

Problem J. Souvenirs

Source file name:	souvenirs.c, souvenirs.cpp, souvenirs.java
Input:	`Standard`
Output:	`Standard`

You are on vacation in a country, where only two types of coins are used: silver and gold. The relative value of one gold coin compared to the number of silver coins has to be big, because there are only these two coin types. Although this is quite unhandy for the merchants, they do not want to deal with a third coin type.
Their idea is that the value of the silver coins is changed in a way that a gold coin is worth less silver coins. To make a statement they tie up silver coins in packages and will give change only in gold coins and packaged silver coins but no single silver coins. However, they do not change their prices to multiples of the silver package values.
So the merchants have to round the change. Different merchants use different rounding methods. Honest merchants round to the nearest value and up if equal. Greedy merchants always round downwards. Generous merchants round always upwards. The merchants also did not negotiate on one package size, so they might all be different. But they are always an integral factor of the current gold coin value in silver coins.
You have some gold coins left and want to buy as many souvenirs as possible. You already know the different prices of all merchants at the market (the price is lower than what one gold coin is worth) and which merchant is greedy, generous and honest as well as their silver coin package sizes. You only have time to go over the market once, so you need to visit the merchants in the given order. Furthermore, you do not want to exploit generous merchants. Thus, if you can pay the exact price, you do so; otherwise you pay with exactly one gold coin. You pay the other merchants either the exact prize or with exactly one gold coin. You buy at most one souvenir at each merchant. Then, you unpack the change, i.e. the silver coin packages, so you can use the coins separately.
For example in the third test case you skip the first merchant and buy at the second one with a gold coin. Because you cannot pay the third merchant exactly, you use a gold coin again. With the change of the two purchases (six and eight silver coins) you can pay the souvenirs at the remaining three merchants.

Input

The first line contains three integers g, c, and n, where g denotes the value of one gold coin in silver coins, c denotes the number of gold coins you have and n the number of merchants on the market ($1 < g \leq 100$, $1 \leq c$, $n \leq 100$).
Then follow n lines each describing one merchant. Each of these lines starts with a string ``greedy'', ``honest'' or ``generous'' describing the rounding habit of the merchant. The string is followed by two integers p_i and s_i , where p_i denotes the package size of silver coins the merchant uses and s_i specifies the prices of the souvenir at that merchant in silver coins ($1 \leq p_i \leq g$, $1 \leq s_i < g$).

Output

One line containing the maximal number of souvenirs you can buy.

Example

Input	Output
42 1 2	2
generous 21 41	
honest 6 21	
42 1 2	1
honest 21 11	
generous 6 23	
12 2 6	5
greedy 1 5	
greedy 1 6	
generous 4 7	
greedy 4 6	
greedy 6 6	
honest 2 2	

Problem K. Upside down primes

Source file name:	primes.c, primes.cpp, primes.java
Input:	`Standard`
Output:	`Standard`

Last night, I must have dropped my alarm clock. When the alarm went off in the morning, it showed 51:80 instead of 08:15. This made me realize that if you rotate a seven segment display like it is used in digital clocks by 180 degrees, some numbers still are numbers after turning them upside down.

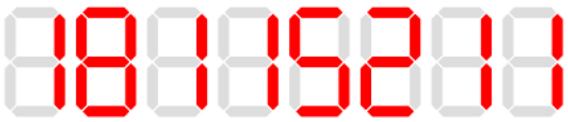


Figure K.1: Prime number 18115211 on a seven segment display (see third sample).

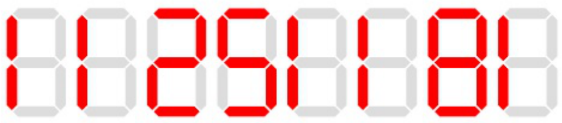
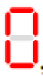
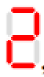
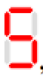
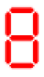
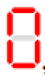
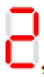
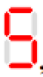
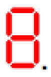
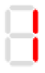
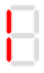
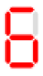
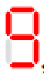
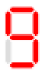
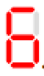
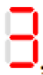
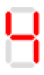
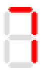
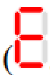
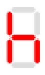
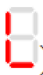


Figure K.2: 18115211 turned upside down (i.e. rotated by 180 degrees) gives 11251181, whichis not prime.

As you can see,

- , , , and  still are , , , and .
-  is still readable as  (only moved left).
-  turns into , while  turns into .
- ,  and  are no longer valid numbers (,  and )

My favourite numbers are primes, of course. Your job is to check whether a number is a prime and still a prime when turned upside down.

Input

One line with the integer N in question ($1 \le N \le 10^{16}$). N will not have leading zeros.

Output

Print one line of output containing ‘‘`yes`’’ if the number is a prime and still a prime if turned upside down, ‘‘`no`’’ otherwise.

Example

Input	Output
151	yes
23	no
18115211	no