Colombian Collegiate Programming League

CCPL 2015

Round 9 – August 22

Problems

This set contains 12 problems; pages 1 to 18.
All problems in the set are original.

Official site `http://programmingleague.org`

Follow us on Twitter `@CCPL2003`

A - Prove Them All

Source file name: `all.c`, `all.cpp`, *or* `all.java`
Author(s): Camilo Rocha

Alex is a developer at the *Formal Methods Inc.* central office. Everyday Alex is challenged with new practical problems related to automated reasoning. Along with her team, Alex is currently working on new features for a computational theorem prover called "Prove Them All" (or PTA for short). The PTA inference engine is based, mainly, on the *modus ponens* inference rule:

$$\frac{\psi, \quad (\psi \rightarrow \phi)}{\therefore \phi}$$

This rule is commonly used in the following way: for any pair of formulae ϕ and ψ, if there is a proof of the formula ψ and a proof of the logical implication $(\psi \rightarrow \phi)$, then there is a proof of ϕ. In other words, if ψ and $(\psi \rightarrow \phi)$ are theorems, then ϕ is a theorem too.

Today's challenge for Alex and her team is as follows: given a collection of formulae Γ and some relationships among them in the form of logical implication, what is the minimum number of formulae in Γ that need to be proven (outside of PTA) so that the rest of formulae in Γ can be proven automatically using only modus ponens?

Input

The input consists of several test cases. The first line of the input contains a non-negative integer indicating the number of test cases. Each test case begins with a line containing two blank-separated integers m and n ($1 \leq m \leq 10000$ and $0 \leq n \leq 100000$), where m is the number of formulae in Γ of the form ϕ_a and n the number of logical implications which have been proven between some of these formulae. The next n lines contain each two blank-separated integers a and b ($1 \leq a, b \leq n$), indicating that $(\phi_a \rightarrow \phi_b)$ is a proven logical implication. Each test case in the input is followed by a blank line.

The input must be read from standard input.

Output

For each test case, output one line with the format "`Case k: c`" where k is the case number starting with 1 and c is the minimum number of formulae in Γ that need to be proven outside of PTA so that the rest of the formulae in Γ can be proven automatically using only modus ponens.

The output must be written to standard output.

Sample Input	**Sample Output**
1 4 4 1 2 1 3 4 2 4 3	Case 1: 2

B - Baking Cakes with Grandma

Source file name: `baking.c`, `baking.cpp`, *or* `baking.java`
Author(s): Camilo Rocha

After five years, Eloi is visiting grandma again. She is very proud because he has, by now, mastered the craft of sewing buttons. Eloi is about to finish his degree in math at the Academy of Colombian Mathematicians (ACM); as a matter of fact, he just defended his dissertation about arrangements of colored buttons.

Grandma is currently into baking cakes of all sorts and flavors. "C is for cake; that's good enough for me... fresh from the oven!" Well, for Eloi, C means programming contests, sleepless nights, and humongous cups of Colombian coffee. Anyway, he's here to take a break from that and relax baking cakes with granny.

Grandma usually bakes her cakes in baking pans which were once round, but due to carelessness and time, now have several dents all along their border. Eloi just can't stop thinking about mathematics, so he has realized that there is an interesting geometrical puzzle related to the cakes. Given a circular baking pan with n dents on its border, what is the largest m such that there are m dents amongst the n which form a regular m-gon?

Input

The input consists of several test cases. Each test case consists of two lines. The first line of a test case contains an integer n ($3 \leq n < 10^3$) indicating the number of dents in the border of the baking pan. The second line contains n blank-separated integers $a_1, \ldots, a_n$ (with $1 \leq a_i \leq 10^5$): a_i indicates the length of the arc between the ith dent and the next.

The input must be read from standard input.

Output

For each test case output a line with the largest m such that there are m dents amongst the n that form a regular m-gon. If such an m does not exist, then output "`-1`".

The output must be written to standard output.

Sample Input	Sample Output
3 1 1 1 3 1 2 1 4 2 1 1 2 5 2 1 1 2 2 5 1 1 3 1 1 5 1 1 2 1 1	3 -1 3 4 -1 3

C - Tennis Championship

Source file name: `champion.c`, `champion.cpp`, *or* `champion.java`
Author(s): Rafael García

A certain tennis championship with P players has a particular set of rules:

1. Before every round, players are paired randomly.
2. Each pair so defined establishes a match that will be played.
3. The winner of a match advances to the next round in the tournament and the loser is eliminated from competition.
4. If the number of players before a round is odd, then one player (chosen at random) is automatically promoted to the next round.

This process should be repeated over and over again until there is exactly one player left. Such a player will be the champion.

The Tennis Championship Organization wants to calculate the total number of matches needed to determine the champion.

Input

The input consists of several test cases, each one consisting of a single line containing a positive integer P, the number of players.

The input must be read from standard input.

Output

For each test case, output a line with one integer indicating the number of matches needed to determine the champion.

The output must be written to standard output.

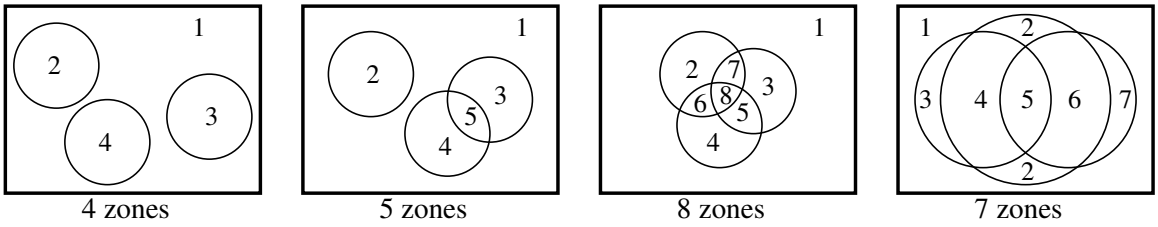
Sample Input	Sample Output
3 2	2 1

D - Euler Diagrams

Source file name: `diagrams.c`, `diagrams.cpp`, *or* `diagrams.java`
Author(s): Federico Arboleda, Rafael García, and Alejandro Sotelo

An *Euler diagram* (named after Leonhard Euler) consists of simple closed curves in the plane, usually circles, that depict sets. The spatial relationships between the regions bounded by each curve (overlap, containment or neither) corresponds to set-theoretic relationships (intersection, subset and disjointness, respectively); depending on the relative location and size of the curves, the plane (or, as is usually the case, a paper sheet) is divided in a certain number of *zones*, each one of which represents an intersection of the original sets or their complements. A more restrictive form of Euler diagrams are *Venn diagrams*, which must include all logically possible zones of overlap between its curves.

Formally, given circular regions $S_1, S_2, \ldots, S_n$ in the plane, we shall define a *zone* as a nonempty set of the form $f_1(S_1) \cap f_2(S_2) \cap \cdots \cap f_n(S_n)$, where, for each i, either $f_i(S_i) = S_i$ or $f_i(S_i) = {S_i}^c$ (the complement of S_i with respect to the drawing surface).



Given a rectangular drawing surface and a collection of circles, find the number of zones in which the surface is split. Note that, in the last example, zone 2 is labeled twice even though both labels are in the same set.

Input

The input consists of several test cases. Each case begins with three blank-separated positive integers, W, H and n, which represent, respectively, the width of the drawing surface, the height of the drawing surface, and the number of circles in the diagram ($1 \leq W \leq 1000$, $1 \leq H \leq 1000$ and $0 \leq n \leq 100$). Each one of the next n lines consists of three blank-separated positive integers, x, y and r, specifying the center (x, y) and radius r of a circle ($0 \leq x \leq W$, $0 \leq y \leq H$, and $1 \leq r \leq W + H$).

You may assume every circle is fully contained within the drawing surface, that no two circles intersect at a single point, that every two circles are different, and that the sides of the surface are not tangent to any circle.

The end of the input is given by $W = H = n = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For every test case print a line with the number of zones in which the drawing surface was split by the circles.

The output must be written to standard output.

Sample Input	Sample Output
60 44 3 12 14 10 24 32 10 48 26 10 60 44 3 16 16 10 34 30 10 44 22 10 60 44 3 24 16 10 28 28 10 36 20 10 60 44 3 30 22 20 20 22 16 40 22 16 50 50 4 25 25 5 25 25 10 25 25 15 25 25 20 50 50 3 25 25 5 25 25 10 25 25 15 50 50 2 25 25 5 25 25 10 50 50 1 25 25 5 50 50 0 50 50 5 15 25 6 20 25 6 25 25 6 30 25 6 35 25 6 50 50 3 25 35 10 15 25 9 35 25 9 0 0 0	4 5 8 7 5 4 3 2 1 13 6

E - Going Shopping with Grandma (I)

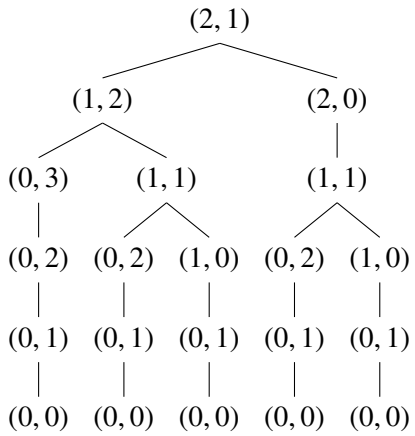
Source file name: `eloi.c`, `eloi.cpp`, *or* `eloi.java`
Author(s): Camilo Rocha

Sometimes, going shopping with grandma can be a very exciting and fun adventure! Eloi is going shopping with grandma this evening because of the holidays; just perfect for his saying: "Sewing, baking, and shopping with grandma, it all goes together... a grandmother, at holiday time, is worth gold." They also are stopping at the pharmacy: granny is losing her memory and her bottle of memory pills is running low ... how sad!

The memory pills come in two sizes: *large* and *small*. The dose in each large pill is equivalent to that in two small ones. Eloi observes granny picks a pill at random from the bottle every day: if it's a small one, she takes it; otherwise, she splits it and takes a half, replacing the other which is from then on considered a small pill.

Given a certain bottle with l large pills and s small pills, we say that the pair (l, s) is the *bottle configuration*. Eloi is interested in the *pill tree* associated with bottle configuration (l, s), in which left or right branching represents a large or small pill being picked, respectively. Formally it's the labeled binary tree with root (l, s) in which a node (u, v) has a *left child* $(u - 1, v + 1)$ if $u > 0$ and a *right child* $(u, v - 1)$ if $v > 0$.

For example, the pill tree associated with bottle configuration $(2, 1)$ (2 large, 1 small) is depicted below:



Eloi then asks himself: how many nodes does the pill tree associated with bottle configuration (l, s) have?

Input

The input consists of several test cases. Each test case consists of a line with two blank-separated integers l and s ($0 \leq l \leq 1000$ and $0 \leq s \leq 1000$).

The end of the input is given by $l = s = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For each l and s, output a line with the number of nodes in the pill tree associated to (l, s). Since this number can be very large, print it modulo 9 999 959 999.

The output must be written to standard output.

Sample Input	**Sample Output**
2 1 6 5 100 2 19 78 1000 1000 0 0	21 31654 5306431377 1942584859 4124225148

F - Going Shopping with Grandma (II)

Source file name: `pharmacy.c`, `pharmacy.cpp`, *or* `pharmacy.java`
Author(s): Camilo Rocha

After leaving the pharmacy with grandma, Eloi has realized there are still some interesting mathematical puzzles regarding granny's pill taking routine.

Granny's memory pills come in two sizes: *large* and *small*. The dose in each large pill is equivalent to that in two small ones. Eloi observes granny picks a pill at random from the bottle every day: if it's a small one, she takes it; otherwise she splits it and takes a half, replacing the other which is from then on considered a small pill.

Eloi would like to solve the following puzzles regarding a given bottle with l large pills and s small pills:

1. What is the expected number of small pills remaining when the last large pill is picked?
2. What is the expected day in which the last large pill is picked?

Your task is to help Eloi solve those puzzles.

Input

The input consists of several test cases. Each test case consists of a line with two blank separated numbers l and s ($0 \leq l \leq 100$ and $0 \leq s \leq 100$).

The end of the input is given by $l = s = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For each test case, output a line with two blank-separated numbers a_1 and a_2: a_1 is the answer to question 1 and a_2 to question 2 above. Each a_i must approximate the correct answer to within 10^{-6}.

The output must be written to standard output.

Sample Input	Sample Output
2 1 6 5 100 2 19 78 0 0	1.833333333333 3.166666666667 3.164285714286 13.835714285714 5.207179497838 196.792820502162 7.447739657144 108.552260342856

G - Trading Card Game

Source file name: `game.c`, `game.cpp`, *or* `game.java`
Author(s): Federico Arboleda and Alejandro Sotelo

Little Ricky is obsessed over the new trading card game, *Sorcery: The Meeting*. He just cannot stop talking about it! He is so obsessed, in fact, that he spends most of his monthly allowance in buying *Sorcery: The Meeting* trading cards, in the hopes of getting all of them.

Of course, buying every single opened card would be too expensive for poor little Ricky, who has to buy everything on his mother's allowance. Instead, he decides to save up during some months and then buy as many unopened cards as he can, hoping he can get them all. Fortunately for Ricky, unopened cards are sold individually!

Being as obsessed as he is, he knows exactly how many cards are in circulation and that, unlike in some other trading card games, there is exactly the same chance to find each card in any single buy. Sadly, Ricky is not very good at math and so he cannot even begin to comprehend what he's going to find in such a big buy. He has, though, put aside some of his allowance this month to ask for your help in calculating the odds of making a good buy (in exchange for the price of a couple of *Sorcery: The Meeting* cards, of course).

If Ricky tells you there are N *Sorcery: The Meeting* trading cards in circulation, all of them equally likely, and he has saved enough to buy m of them at the same time, what is the probability that he will get exactly k different cards?

Being as obsessed as he is, he knows that floating-point numbers would necessarily incur a loss of precision, which he will not tolerate. Therefore, he wants this information as a fraction in lowest terms.

Input

The input consists of several test cases. Each case is a line with three blank-separated integers, N, m and k, which represent, respectively, the total number of cards in circulation, the number of cards Ricky is going to buy, and the number of different cards he expects to get ($1 \leq N \leq 100$, $0 \leq m \leq 100$ and $0 \leq k \leq 100$).

The end of the input is given by $N = m = k = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For every test case print a line of the form "`p/q`", where $\frac{p}{q}$ is a fraction in lowest terms representing the probability that Ricky will get exactly k different cards under the described conditions. A probability of 0 should be represented as "`0/1`" and a probability of 1 should be represented as "`1/1`".

The output must be written to standard output.

Sample Input	Sample Output
10 1 1 10 2 2 10 12 11 10 4 2 10 4 1 10 4 0 0 0 0	1/1 9/10 0/1 63/1000 1/1000 0/1

H - Harvest Moon

Source file name: `harvest.c`, `harvest.cpp`, *or* `harvest.java`
Author(s): Federico Arboleda, Rafael García, and Alejandro Sotelo

The farmer Yasuhiro has recently bought a rectangular plot. Prior to sowing time, the day before a full moon, he traced a rectangular grid on his plot and separated his seeds in two categories: medicinal plants and fruit trees.

The land quality is, of course, not uniform. Some cells of the grid are better than others depending on certain factors such as soil permeability, coarseness, irrigation type, and ground slope. To quantify all those variables, Yasuhiro has defined a *productivity factor* for each cell: a number between 0 and 100 indicating how much he would benefit from sowing in that particular cell. Specifically, a factor of 0 means he would not benefit at all from that cell, while a factor of 100 means he would benefit the most.

Yasuhiro has also made a table with information about the plant species he is going to sow. This table describes the category of each species (medicinal plant or fruit tree), its cost per cell, and the minimum and maximum possible number of cells occupied by that species. The total benefit from sowing one particular species in any particular cell is equal to the productivity factor of that cell times the cost per cell of that species.

Given the number of rows and columns in the grid, the productivity factor of each cell, and the table with the plant information, you must calculate the maximum possible benefit which can be obtained by sowing according to the following rules:

- In each cell, at most one plant species can be sown.
- For every species, the number of cells where it is sown must be between the minimum and the maximum specified in the table.
- No two cells in the same row can contain the same species of medicinal plant. Likewise, no two cells in the same column can contain the same species of fruit tree.
- The total benefit is the sum of the individual benefits from each sown cell.

Input

The input consists of several test cases. The specification of each test case follows:

- First, there is a line with three integers R, C, and E, which specify, respectively, the number of rows in the grid, the number of columns in the grid, and the number of species in the table ($1 \leq R \leq 4$, $1 \leq C \leq 4$ and $1 \leq E \leq 10$).
- Then follow R lines, each one of them with C blank-separated integers between 0 and 100. The jth number of line i is the productivity factor of the cell in row i and column j.
- Finally, there are E lines, one for each species in the table, each comprising the following blank-separated data:
  - The character '`M`' if it's a medicinal plant or '`F`' if it's a fruit tree.
  - An integer d which indicates the current species' cost per cell ($1 \leq d \leq 10^4$).
  - Two integers n and m which are, respectively, the minimum and maximum number of cells for the current species ($0 \leq n \leq m \leq 4$).

The end of the input is given by $R = C = E = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For each test case, print a line with the maximum possible benefit which Yasuhiro can obtain from sowing the plot according to the rules. If it is not possible to sow the plot as specified, then print "0".

The output must be written to standard output.

Sample Input	Sample Output
2 2 1 10 10 10 10 M 3000 0 4 3 3 2 10 50 90 50 90 10 90 10 50 M 1500 3 3 F 500 3 3 2 2 1 100 100 100 100 F 2000 3 4 2 3 1 100 100 100 100 100 100 M 2000 0 3 0 0 0	60000 480000 0 400000

I - Accelleratii Incredibus

Source file name: `incredibus.c`, `incredibus.cpp`, *or* `incredibus.java`
Author(s): Federico Arboleda, Rafael García, and Alejandro Sotelo

If you're on a highway and Road Runner goes "beep beep"
just step aside or you might end up in a heap.
Road Runner, Road Runner runs on the road all day.
Even the Coyote can't make him change his ways.

— *The Road Runner Show* theme song —

The Road Runner *(Accelleratii incredibus)* is a very fast-running ground bird which can be found in the roads of the hot and lonely Southwestern United States. Wile E. Coyote *(Carnivorous vulgaris)*, a clever canine, has repeatedly and unsuccessfully tried to catch it using every kind of trap imaginable.

The Interstate is the Road Runner's favourite road, since it's a straight, L miles long path along which it can run at constant speed without stopping or turning. One particular summer day, the Road Runner is sunbathing on the Interstate and wants to run home (also on the Interstate) in exactly m minutes, never exceeding v miles per minute. The Coyote meanwhile has installed bombs at certain positions along the road and programmed them to go off at certain times in the hopes of catching the Road Runner in one of the explosions.

Every minute, the Road Runner can move an integer amount of miles which must be less than or equal to v, avoiding the spots in the road which have a bomb programmed to go off during that minute. Given the positions of the bombs which will go off each minute, you must calculate the minimum amout of miles the Road Runner must move to go from its sunbathing spot to its home in exactly m minutes, avoiding all explosions.

Input

The input consists of several test cases. Each case begins with a line with a positive integer L which is the length of the Interstate in miles ($1 \leq L \leq 1000$). Then follows a line with two integers x_i and x_f which are, respectively, the initial position of the Road Runner and the location of its home ($0 \leq x_i \leq L$, $0 \leq x_f \leq L$). The next line contains two integers m and v which represent, respectively, the amount of minutes in which the Road Runner wants to reach its home and the maximum allowed velocity in miles per minute ($1 \leq m \leq 100$, $1 \leq v \leq L$). Each one of the next m lines contains a string of $L + 1$ characters; character i of line j is '`X`' if there is a bomb at position i which will go off during the jth minute, or '`.`' otherwise.

You may assume that no bomb will go off at position x_i during the first minute. The end of the input is given by $L = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a line with the minimum amount of miles the Road Runner must move to go home in exactly m minutes and avoiding all explosions. If it is not possible for the Road Runner to go home avoiding all explosions, then output -1.

The output must be written to standard output.

Sample Input	Sample Output
6 0 6 5 6 ...X... XX..XX. ...X... .XX.... .... 6 0 6 5 5 ...X... XX..XX. ...X... .XX.... .... 6 1 6 3 3 ....X... .... 6 1 6 3 2 ....X... 0	10 -1 5 -1

J - Ant-Man's Sugar Journey

Source file name: `journey.c`, `journey.cpp`, *or* `journey.java`
Author(s): Federico Arboleda and Diego Satoba

Ant-Man is the latest superhero on the stage. Like most superheroes, he has got unique powers – besides shrinking to the size of an insect, he can control ants through his suit.

As usual, Ant-Man should use his powers only for the greater good (e.g., world peace, saving mankind, or supervising programming contests), but more often than not, he uses them to impress his girlfriend, Wasp.

This time he has invited Wasp over for some Colombian coffee. Not being used to the strong taste, she wants to sweeten it, but this being a programming contest, there's one little problem: the ants have taken all the sugar to their nest while Ant-Man wasn't looking.

The ants' nest is unlike any other: it has got only one entrance and one (different) exit, and comprises a network of tunnels connecting them, with several intersections and branching. Since there are many ants living in the tunnels, every tunnel may be run only in a predetermined direction, and there is no path of tunnels from any intersection to itself, no dead ends, and no inaccessible intersections. The ants are keeping a sugar cube at every intersection.

Wanting to impress Wasp some more, he will use his ants to bring his sugar back instead of going in himself, and he will do it with the minimum possible number of ants. Each ant is strong enough to carry an unlimited amount of sugar cubes at the same time, but Ant-Man doesn't want them to feel like tools, so he will not order any ant to re-enter the nest after its sugar run is done.

Ant-Man has asked his old cellmate for help in performing this task, and as usual, he "knew someone who knows someone" who has relayed this problem to you. Now you must calculate the minimum possible number of ants needed to bring back all the sugar.

Input

The input consists of several test cases. Each case begins with a line containing two blank-separated integers N and M ($2 \leq N \leq 100$ and $1 \leq M \leq 5000$), which represent the number of intersections and the number of tunnels in the nest, respectively; the entrance and exit points are counted as intersections. Next come M lines with two blank-separated integers u and v ($0 \leq u \leq N-1$, $0 \leq v \leq N-1$, and $u \neq v$), meaning that there is a tunnel from intersection u to intersection v (running in that direction).

The end of the input is given by $N = M = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For each test case print a line containing the minimum number of ants needed to recover all the sugar.

The output must be written to standard output.

Sample Input	Sample Output
1 9 12 0 1 0 2 1 3 1 4 2 4 2 5 3 6 4 6 4 7 5 7 6 8 7 8 0 0	3

K - Prime Kebab Menu

Source file name: `kebab.c`, `kebab.cpp`, *or* `kebab.java`
Author(s): Rafael García

Prime is an unusual but efficient Kebab restaurant. *Prime* has many small tables, which allow it to serve groups of various sizes without problems. We went to *Prime* last weekend and saw one small table with 3 people, another one with 5 people, and another with 14 people – the servers simply rearrange the tables as they see fit to cater to those groups.

One unusual feature of *Prime* is that each client may choose only one dish from their large menu. Even more strangely, we noticed that the waiters of *Prime* relay a whole order to the kitchen with just one number! I saw the group of 3 people ask for the first, third, and fifth dishes in the menu, and their waiter relayed this to the kitchen as 110. When the group of five asked for dishes number 1, 2, 3, 4, and 5, their waiter said only 2310. Finally, when the group of 14 requested the first dish 10 times and the second one 4 times, I heard their waiter say 82 944. Amazingly, when the respective waiters came back with the meals, all orders were correctly fulfilled!

This strange numerical code is still incomprehensible to me. I have found, through some research, that the waiters never say 1. Of course this doesn't help me much and I still have lots of questions, such as: if a waiter relays just one number to the kitchen, how do the chefs calculate the number of clients in the table? I am confident that, if you can answer this for me, then I can fully crack the restaurant code.

Your task is to write a program that outputs the number of clients in the table given the number relayed by a waiter.

Input

The input consists of several test cases. Each test case is a line containing an integer $1 < n < 10^{14}$, which is a number relayed by a waiter.

The end of the input is given by $n = 1$, which should not be processed as a test case.

The input must be read from standard input.

Output

For each test case, print a line with the number of clients in the group.

The output must be written to standard output.

Sample Input	Sample Output
110 82944 1	3 14

L - The Weakest Link

Source file name: `link.c`, `link.cpp`, *or* `link.java`
Author(s): Camilo Rocha

It is clearly a literal fact that a chain is only as strong as its weakest link. The conversion of that notion into a figurative phrase was established in the language by the 18th century. Thomas Reid's *Essays on the Intellectual Powers of Man* (1786), included this line:

> In every chain of reasoning, the evidence of the last conclusion can be no greater than that of the weakest link of the chain, whatever may be the strength of the rest.

In this problem a *chain of length n* is a string $C = c_1 c_2 \ldots c_n$ of n lowercase characters where c_n is considered to be followed by c_1 in a cyclical fashion. Character i is said to be *weaker* than character j in a chain C if the string $c_i c_{i+1} \ldots c_n c_1 \ldots c_{i-1}$ comes before the string $c_j c_{j+1} \ldots c_n c_1 \ldots c_{j-1}$ in lexicographical order.

Given a chain C, your task is to find the weakest character in C.

Input

The first line of the input contains a non-negative integer N indicating the number of test cases. Each test case comprises a single line with a nonempty string C of at most 50 000 lowercase characters of the English alphabet '`a`'-'`z`'. You may assume that `a` < `b` < $\cdots$ < `z` as usual.

The input must be read from standard input.

Output

For each test case, output a line containing an integer w such that c_w is the weakest character in the chain C. If there is more than one such value, then output the smallest one.

The output must be written to standard output.

Sample Input	Sample Output
4 ccpl abracadabra hocuspocus aa	1 11 8 1