Colombian Collegiate Programming League

CCPL 2024

Round 9 -- October 5

Problems

This set contains 12 problems; pages 1 to 21.

(Borrowed from several sources online.)

A - Angry Programmer 1

B - Bingo 3

C - Let's Yum Cha! 4

D - Dominator 6

E - The Bridges of Kolsberg 8

F - Curious Fleas 10

G - Seven Seas 12

H - Stopping Doom's Day 14

I - Decode the tape 15

J - String Factoring 17

K - Keys 18

L - The Mailbox Manufacturers Problem 20

Official site `http://programmingleague.org`

Follow us on 𝕏 `@CCPL2003`

11506

A - Angry Programmer

Source file name: `angry.c`, `angry.cpp`, `angry.java`, *or* `angry.py`

You, a programmer of an important software house, have been fired because you didn't solve an important problem that was assigned to you. You are very furious and want to take revenge on your boss,breaking the communication between his computer and the central server.

The computer of your boss and the central server are in the same network, which is composed of many machines (computers) and wires linking pairs of those machines. There is at most one wire between any pair of machines and there can be pairs of machines without a wire between them.

To accomplish your objective, you can destroy machines and wires, but you can't destroy neither the computer of your boss nor the central server, because those machines are monitored by security cameras. You have estimated the cost of blowing up each machine and the cost of cutting each wire in the network.

You want to determine the minimum cost of interrupting the communication between your boss' computer and the central server. Two computers A and B can communicate if there is a sequence of undestroyed machines $x_1, \ldots, x_n$ such that $x_1 = A$, $x_n = B$ and x_i is linked with x_{i+1} with an uncut wire (for each $1 \leq i \leq n-1$).

Input

The input consists of several test cases. Each test case is represented as follows:

- A line with two integers M and W ($2 \leq M \leq 50, 0 \leq W \leq 1000$), representing (respectively) the number of machines and the number of wires in the network.
- $M - 2$ lines, one per machine (different from the boss' machine and the central server), containing the following information separated by spaces:
  - An integer i ($2 \leq i \leq M - 1$) with the identifier of the machine. Assume that the boss' machine has id 1 and that the central server has id M.
  - An integer c ($0 \leq c \leq 100000$) specifying the cost of destroying the machine.
- W lines, one per wire, containing the following information separated by spaces:
  - Two integers j and k ($1 \leq j < k \leq M$) specifying the identifiers of the machines linked by the wire. Remember that the wire is bidirectional.
  - An integer d ($0 \leq d \leq 100000$) specifying the cost of cutting the wire.

The end of the input is specified by a line with the string '0 0'.
Suppose that the machines have distinct identifiers.

The input must be read from standard input.

Output

For each test case, print a line with the minimum cost of interrupting the communication between the computer of your boss and the central server.

The output must be written to standard output.

Sample Input	Sample Output

```
4 4
3 5
2 2
1 2 3
1 3 3
2 4 1
3 4 3
4 4
3 2
2 2
1 2 3
1 3 3
2 4 1
3 4 3
0 0
```
	```
	4
	3
	```

B - Bingo

Source file name: `bingo.c`, `bingo.cpp`, `bingo.java`, *or* `bingo.py`

Albert, Charles and Mary invented a new version of the classical game Bingo. In traditional Bingo the game is presided over by a non-player known as the caller. At the beginning of the game each player is given a card containing a unique combination of numbers from 0 to N arranged in columns and rows. The caller has a bag containing $N + 1$ balls, numbered from 0 to N. On each turn, the caller randomly selects a ball from the bag, announces the number of the drawn ball to the players, and sets the ball aside so that it cannot be selected again. Each player searches his card for the called number and marks it if he finds it. The first player who marks a complete pre-announced pattern on the card (for example, a full horizontal line) wins a prize.

In the Albert-Charles-Mary version, on each turn, the caller draws a first ball, returns it to the bag, draws a second ball, returns it to the bag, and then calls out the absolute difference between the two ball numbers. To generate even more excitement, before the game started a possibly empty subset of balls is removed from the bag, in such a way that at least two balls remain there. They would like to know if every number from 0 to N may still be called out with the new drawing method considering the balls that were left in the bag.

Input

Each test case is given using exactly two lines. The first line contains two integers N and B. The meaning of N was described above ($1 \leq N \leq 90$), while B represents the number of balls which remained in the bag ($2 \leq B \leq N + 1$). The second line contains B distinct integers b_i, indicating the balls which remained in the bag ($0 \leq bi \leq N$).

The last test case is followed by a line containing two zeros.

The input must be read from standard input.

Output

For each test case output a single line containing a single uppercase 'Y' if is possible to call out every number from 0 to N, inclusive, or a single uppercase 'N' otherwise.

The output must be written to standard output.

Sample Input	Sample Output

6 7
2 1 3 4 0 6 5
5 4
5 3 0 1
5 3
1 5 0
0 0
	Y
	Y
	N

11566

C - Let's Yum Cha!

Source file name: `cha.c`, `cha.cpp`, `cha.java`, *or* `cha.py`

Yum cha, a term in Cantonese, literally meaning "drinking tea", refers to the custom of eating small servings of different foods (dim sum) while sipping Chinese tea. It is an integral part of the culinary culture of Guangdong and Hong Kong. For Cantonese people, to yum cha is a tradition on weekend mornings, and whole families gather to chat and eat dim sum and drink Chinese tea. The tea is important, for it helps digest the foods. In the past, people went to a teahouse to yum cha, but dim sum restaurants have been gaining overwhelming popularity recently.

Dim Sum literally means "touch your heart", which consists of a wide spectrum of choices, including combinations of meat, seafood, vegetables, as well as desserts and fruit. Dim sum can be cooked, inter alia, by steaming and deep-frying. The various items are usually served in a small steamer basket or on a small plate. The serving sizes are usually small and normally served as three or four pieces in one dish. Because of the small portions, people can try a wide variety of food.

Some well-known dim sums are:

- **Har gow**: A delicate steamed dumpling with shrimp filling and thin (almost translucent) wheat starch skin. It is one of my favourite dim sum.
- **Siu mai**: A small steamed dumpling with pork inside a thin wheat flour wrapper. It is usually topped off with crab roe and mushroom.
- **Char siu bau** A bun with Cantonese barbeque-flavoured pork and onions inside. It is probably the most famous dim sum around the world.
- **Sweet cream bun**: A steamed bun with milk custard filling. It is sweet and spongy.
- **Spring roll**: Consists of sliced carrot, wood-ear fungus, and sometimes meat, rolled inside a thin flour skin and deep-fried. It is crispy and delicious.

The picture on the right shows some of the dim sums mentioned above. Can you name them?

Today you go to yum cha with N friends. You and your friends have agreed that everyone will pay the same amount of money (to within one dollar), and each and every one of you will pay at most $$x$. In the restaurant there are K kinds of dim sums to choose from. Every one of you has assigned an integer "favour index" to each dim sum, ranging from 0 to 10.

Now you are responsible for choosing what dim sums to order. You wanted to **maximize your total "favour value"** from dim sums you choose, but you will certainly get beaten by all your friends if you ignore their interest. Therefore, you shall **maximize the mean of the total favour value everyone gets**, computed using the formula.

$$\frac{Total\ favour\ value\ of\ all\ dishes\ ordered}{N+1}$$

instead. Note that even though we are considering the "mean favour value", this does not imply that everyone can actually get a piece of every ordered item!! Anyway, since you are very good friends, you will certainly find some ways to share the food so that everyone is happy, right? :-)

Since you would like to try more different kinds of dim sums, you shall NOT order more than 2 dishes of a same type of dim sum. Also, since you do not want to waste food, you shall NOT order more than $2(N + 1)$ dishes in total (i.e. 2 dishes for each of you).

When computing the amount of money to be paid we shall NOT just add up the prices of the dim sums. We also need to take care of the following two charges:

- **Tea charge**: everyone is charged $T for the tea provided
- **10% service charge**: you need to pay 10% of (Dim sum prices + Tea charge) as service charge. This value is to be rounded **up** to the nearest dollar.

Constraints:

- $1 \leq N \leq 10$
- $1 \leq x \leq 100$
- $0 \leq T \leq 20$
- $1 \leq K \leq 100$
- $\$1 \leq \ price\ of\ a\ dim\ sum\ \leq \$100$.

Input

Input consists of no more than 25 test cases. Each case begins with four integers N, x, T and K, whose meanings have been explained above. Then comes K lines, each giving the information of one particular dim sum. The first integer is the price of the dim sum, followed by your favour index, then N integers which are the favour indices of your N friends.

Input ends with a line with four zeros.

The input must be read from standard input.

Output

For each case, your program should give the optimal mean favour value, correct to 2 decimal places. This value is always positive.

The output must be written to standard output.

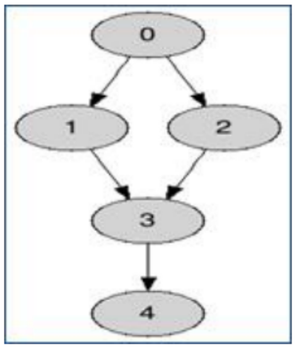
Sample Input	Sample Output

`3 10 5 2`
`6 7 5 6 9`
`10 9 10 10 8`
`0 0 0 0`
	`16.00`

D - Dominator

Source file name: `dominator.c`, `dominator.cpp`, `dominator.java`, *or* `dominator.py`

In graph theory, a node X dominates a node Y if every path from the predefined start node to Y must go through X. If Y is not reachable from the start node then node Y does not have any dominator. By definition, every node reachable from the start node dominates itself. In this problem, you will be given a directed graph and you have to find the dominators of every node where the 0-th node is the start node.



As an example, for the graph shown right, 3 dominates 4 since all the paths from 0 to 4 must pass through 3. 1 doesn't dominate 3 since there is a path 0-2-3 that doesn't include 1.

Input

The first line of input will contain T (≤ 100) denoting the number of cases.

Each case starts with an integer N ($0 < N < 100$) that represents the number of nodes in the graph. The next N lines contain N integers each. If the j-th (0 based) integer of i-th (0 based) line is '1', it means that there is an edge from node i to node j and similarly a '0' means there is no edge.

The input must be read from standard input.

Output

For each case, output the case number first. Then output $2N + 1$ lines that summarizes the dominator relationship between every pair of nodes. If node A dominates node B, output 'Y' in cell (A, B), otherwise output 'N'. Cell (A, B) means cell at A-th row and B-th column. Surround the output with '—', '+' and '-' to make it more legible. Look at the samples for exact format.

The output must be written to standard output.

Sample Input	**Sample Output**

```
2
5
0 1 1 0 0
0 0 0 1 0
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0
1
1
```
	```
	Case 1:
	+---------+
	|Y|Y|Y|Y|Y|
	+---------+
	|N|Y|N|N|N|
	+---------+
	|N|N|Y|N|N|
	+---------+
	|N|N|N|Y|Y|
	+---------+
	|N|N|N|N|Y|
	+---------+
	Case 2:
	+-+
	|Y|
	+-+
	```

1172

E - The Bridges of Kolsberg

Source file name: `bridges .c`, `bridges .cpp`, `bridges .java`, *or* `bridges .py`

King Beer has a very hard region to rule, consisting of lots of cities with very sectarian operating system beliefs and high levels of trade. These cities are placed along a river, the Klsberg, along its Northern and Southern banks. The cities are economically separated from each other, since the river is wide and dangerous.

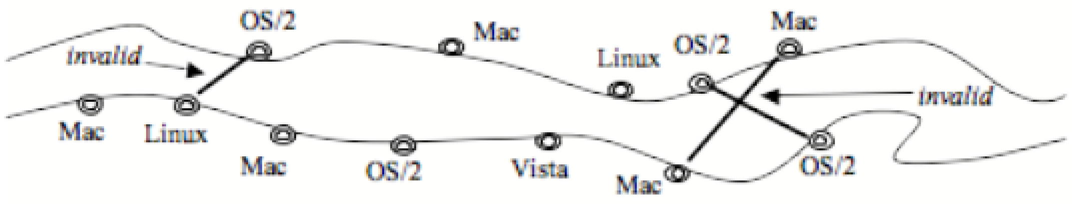


Figure 1: *
A section of the Klsberg showing some invalid bridges

King Beer would like to build some bridges connecting opposite banks of the river. He was strongly advised against making bridges between cities with different operating systems beliefs (those guys really hate each other). So, he is just going to build bridges between cities sharing the same operating system belief (even if the resulting bridges are quite long and strangely shaped). However, it is technical impossible to build bridges that cross other bridges.

The economical value of a bridge is the sum of the trade values the two cities it connects. The King wants to maximize the sum of all possible bridge values while minimizing the number of bridges to build.

Given two sets of cities, return the maximum possible sum of all bridge values and the smallest number of valid bridges necessary to achieve it.

Input

The first line is an integer with the number of samples. For each sample, the next line has a nonnegative integer, not greater than 1,000, indicating the number of cities on the Northern riverbank. Then, on each line, comes the city information with the form

cityname ostype tradevalue

where, separated by empty spaces, there are two strings, *cityname* and *ostype*, with no more than 10 characters each, and *tradevalue* which is a non-negative integer not greater than 10^6. The sequence of lines represents the cities from left to right along the riverbank. Next, there is the same kind of information to describe the Southern riverbank.

The input must be read from standard input.

Output

For each sample, a line consisting of the maximum possible sum of all bridge values, one empty space, the number of bridges.

The output must be written to standard output.

Sample Input	Sample Output

```
1
3
mordor Vista 1000000
xanadu Mac 1000
shangrila OS2 400
4
atlantis Mac 5000
hell Vista 1200
rivendell OS2 100
appleTree Mac 50
```
	`1002250 2`

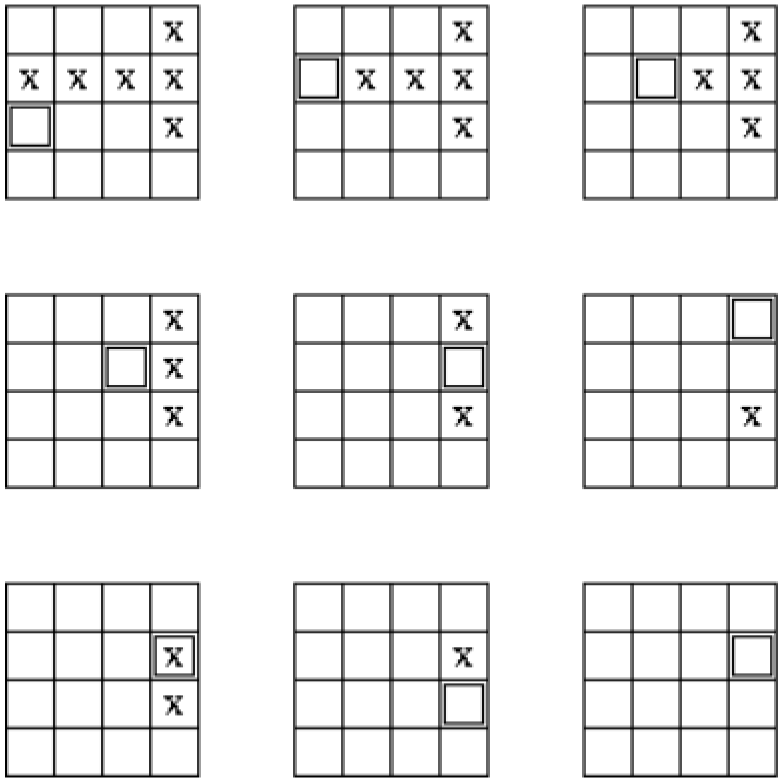
F - Curious Fleas

Source file name: `fleas.c`, `fleas.cpp`, `fleas.java`, *or* `fleas.py`

Your most prized possession is your flea circus. The main attraction of this circus is the act performed by six acrobatic fleas. One day you accidentally knock over the container holding these fleas and they all spill out onto the 4×4 grid of square tiles that make up your kitchen floor. Surprisingly, no two fleas land on the same tile.

Your son Andy has a (normal, six-sided) die that is exactly the same size as a grid square. You decide to play a game. First, you place the die precisely on a single tile with no fleas. Then you start rolling the die over an edge onto one of the adjacent tiles. The fleas are very curious; they like to explore. Whenever you move a die onto a square containing a flea, it jumps to the bottom of the die. The fleas are small enough to hide in the recesses formed by the dots on the die, so they wont be killed in the process. Similarly, if a flea is on the side of a die that is moved to the floor, it jumps from the die to the tile.

The goal of your game is to move the die in such a manner that all fleas end up sitting on the die. The die can not leave the 4×4 grid that is your kitchen. An example of a solution for the first sample input follows:



Input

The first line of input indicates the number of test cases to follow. Each test case consists of a blank line followed by 4 lines of 4 characters which describes the initial grid configuraion. A '.' denotes an empty square, an 'X' denotes a flea, and a 'D' indicates the initial position of the die. There will always be exactly one 'D' and exactly six 'X' characters in each test case.

The input must be read from standard input.

Output

For each test case, output a line containing a single number: the minimum number of moves that are required to get a flea on every face of the die at the same time. If a case has no solution, output the word 'impossible' instead.

The output must be written to standard output.

Sample Input	Sample Output

```
2

...X
XXXX
D..X
....

DX.X
.X..
..X.
X..X
```
	```
	8
	14
	```

10923

G - Seven Seas

Source file name: `seven.c`, `seven.cpp`, `seven.java`, *or* `seven.py`

Seven Seas is a nice game where you are the captain of a battle ship. You are in the middle of the ocean and there are some enemy ships trying to catch you. Your mission is to run away from them. Destroy the enemies if you can!

The game is played on a 9×8 board and you can move in all the 8 directions, one step each time. You move first, then the enemy ships move. The enemy ships are pretty dumb, so that they will always move to the closest position they can get to you, even if that position will destroy their ship. A ship is destroyed if it moves to a cell that contais a rock. If two enemy ships move to the same cell they are also destroyed and their remains will stay in that cell so that if another ship try to move there it will be destroyed too.

If an enemy ship reaches your ship, you are dead and the game is over.

Input

The first line of the input contains the number of scenarios. Each scenarion consists of a 9×8 board of charachters. Your ship is represented by a 'S' on the board and the enemy ships are represented by a 'E'. The rocks are represented by a '#' and the '.' represents the sea. See the sample input to see the specific input format.

There will be at least one and at most nine enemy ships. There will be an empty line between two scenarios.

The input must be read from standard input.

Output

For each scenario you must find out if it is possible to destroy all the enemy ships in less than 10 steps. If that is the case, you must print one line containing the string: 'I'm the king of the Seven Seas!'. Otherwise, you must print one line with the string: 'Oh no! I'm a dead man!'.

The output must be written to standard output.

Sample Input

```
3
........
.E.#....
...E....
..#.....
........
........
..S.....
........
........

........
.E.E....
...S....
.E..E...
........
........
........
........
........

E......#
........
........
........
........
........
........
.......S
#.......
```

Sample Output

```
I’m the king of the Seven Seas!
Oh no! I’m a dead man!
Oh no! I’m a dead man!
```

12143

H - Stopping Doom's Day

Source file name: `doom.c`, `doom.cpp`, `doom.java`, *or* `doom.py`

So! The time of the universe is up and it is the dooms day after five hours :-P, and you must stop it. But to do so you have to know the value of the following expression T:

$$T = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} \sum_{m=1}^{n} |(|i-j| * |j-k| * |k-l| * |l-m| * |m-i|)| \right) \% 10007$$

Because the secret code that will save the universe from being doomed have something to do with the value of the above expression for some value of n.

Input

The input file contains 1000 lines of inputs. Each line contains a single integer n ($0 < n \leq 2000000000$). A line containing a single zero terminates input.

The input must be read from standard input.

Output

For each line of input produce one line of output. This line contains the value of T

The output must be written to standard output.

Sample Input	Sample Output
12 20 1001 0	2199 803 2390

I - Decode the tape

Source file name: `decode.c`, `decode.cpp`, `decode.java`, *or* `decode.py`

"Machines take me by surprise with great frequency."
Alan Turing

Your boss has just unearthed a roll of old computer tapes. The tapes have holes in them and might contain some sort of useful information. It falls to you to figure out what is written on them.

Input

The input will contain one tape.

The input must be read from standard input.

Output

Output the message that is written on the tape.

The output must be written to standard output.

Sample Input

```
___________
| o   .  o|
|  o  .   |
| ooo .  o|
| ooo .o o|
| oo o.  o|
| oo  . oo|
| oo o. oo|
| o   .   |
| oo  . o |
| ooo . o |
| oo o.ooo|
| ooo .ooo|
| oo o.oo |
| o   .   |
| oo  .oo |
| oo o.ooo|
| oooo.   |
| o   .   |
| oo o. o |
| ooo .o o|
| oo o.o o|
| ooo .   |
| ooo . oo|
| o   .   |
| oo o.ooo|
| ooo .oo |
| oo  .o o|
| ooo . o |
| o   .   |
| ooo .o  |
| oo o.   |
| oo  .o o|
|  o  .   |
| oo o.o  |
| oo  .  o|
| oooo. o |
| oooo.  o|
|  o  .   |
| oo  .o  |
| oo o.ooo|
| oo  .ooo|
| o o .oo |
|    o. o |
___________
```

Sample Output

```
A quick brown fox jumps over the lazy dog.
```

J - String Factoring

Source file name: factoring.c, factoring.cpp, factoring.java, *or* factoring.py

Spotting patterns in seemingly random strings is a problem with many applications. E.g. in our efforts to understand the genome we investigate the structure of DNA strings. In data compression we are interested in finding repetitions, so the data can be represented more efficiently with pointers. Another plausible example arises from the area of artificial intelligence, as how to interpret information given to you in a language you do not know. The natural thing to do in order to decode the information message would be to look for recurrences in it. So if the SETI project (the Search for Extra Terrestrial Intelligence) ever get a signal in the H21-spectra, we need to know how to decompose it.

One way of capturing the redundancy of a string is to find its factoring. If two or more identical substrings A follow each other in a string S, we can represent this part of S as the substring A, embraced by parentheses, raised to the power of the number of its recurrences. E.g. the string DOODOO can be factored as $(\text{DOO})^2$, but also as $(\text{D}(\text{O})^2)^2$. Naturally, the latter factoring is considered better since it cannot be factored any further. We say that a factoring is irreducible if it does not contain any consecutive repetition of a substring. A string may have several irreducible factorings, as seen by the example string POPPOP. It can be factored as $(\text{POP})^2$, as well as $\text{PO}(\text{P})^2\text{OP}$. The first factoring has a shorter representation and motivates the following definition. The weight of a factoring, equals the number of characters in it, excluding the parentheses and the exponents. Thus the weight of $(\text{POP})^2$ is 3, whereas $\text{PO}(\text{P})^2\text{OP}$ has weight 5. A maximal factoring is a factoring with the smallest possible weight. It should be clear that a maximal factoring is always an irreducible one, but there may still be several maximal factorings. E.g. the string ABABA has two maximal factorings $(\text{AB})^2\text{A}$ and $\text{A}(\text{BA})^2$.

Input

The input consists of several rows. The rows each hold one string of at least one, but less than 80 characters from the capital alphabet A-Z. The input is terminated by a row containing the character '*' only. There will be no white space characters in the input.

The input must be read from standard input.

Output

For each string in the input, output one line containing the weight of a maximal factoring of the string.

The output must be written to standard output.

Sample Input	Sample Output
PRATTATTATTIC	6
GGGGGGGGG	1
PRIME	5
BABBABABBABBA	6
ARPARPARPARPAR	5
*	

K - Keys

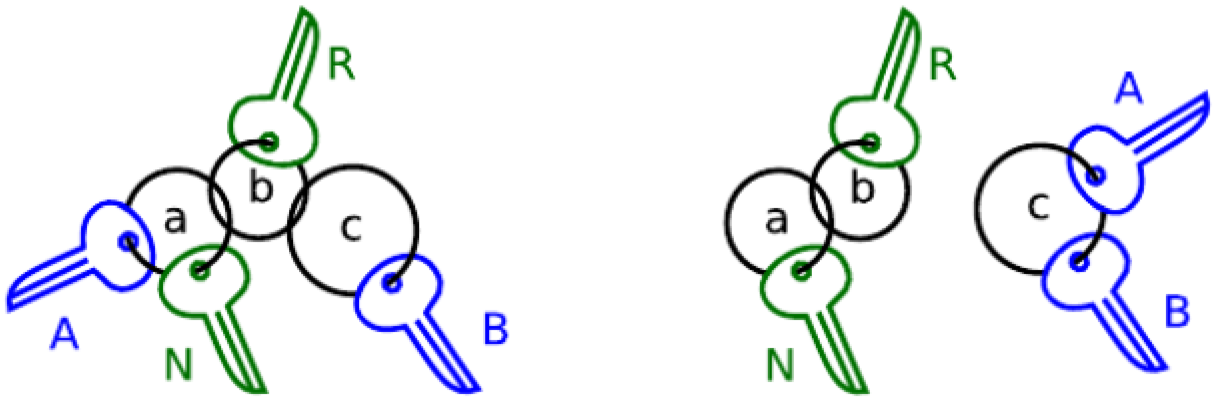
Source file name: `keys.c`, `keys.cpp`, `keys.java`, *or* `keys.py`

Adam carries a bunch of keys attached to key rings, some of which may be connected to each other. The rings are common key rings, so a key can be attached to or detached from a ring by sliding along the spiral. In the same way, two rings can be connected or disconnected. Adam wants to give some of the keys to Brenda. Since manipulating the keys and rings is often an annoying task (and also dangerous to one's fingernails), Adam is looking for a way to minimize the number of key and ring operations.

Every key attachment, key detachment, ring connection, or ring disconnection is considered one operation. Since manipulating two rings is significantly easier than sliding a key, we first want to minimize the number of keys being detached and attached. Among solutions with the same minimal number of key operations, you need to find the one with the minimal number of ring connections and disconnections.

When all the operations are complete, Adam and Brenda must each carry one connected group of rings and keys. The only exception is when either of them would have no keys at all—in such a case, no ring is needed. Each key must be attached to exactly one ring. Some rings (but not keys) may be considered leftovers and may remain disconnected from the two groups.

The left side of the following figure shows an initial configuration consisting of four keys on three rings. Adam wishes to give Brenda the two keys labeled N and R. This can be accomplished by two key operations and one ring operation, resulting in the configuration shown on the right side of the figure.



Input

Each test case contains one or more lines, each containing a two letter string. Lowercase letters `(a - z)` represent key rings and uppercase letters `(A - Z)` represent keys. The two letters on a line specify either a key attached to a ring or two rings connected together. The end of each test case is denoted by a line containing the digit zero.

Keys denoted by letters A through M remain with Adam, and keys denoted by letters N through Z are given to Brenda.

No line contains two uppercase letters. No pair of letters are specified more than once in the same test case. Each key is connected to exactly one ring. There are no "circles" in the ring configurations (disconnecting any two rings will increase the number of connected groups). All existing keys and rings are mentioned at least once.

The input must be read from standard input.

Output

For each test case, display the case number followed by the minimal number of key attach/detach operations and the minimal number of ring connect/disconnect operations. If there is no way to split the keys as requested, display the case number and the word ‘impossible’ instead of the two integers.

The output must be written to standard output.

Sample Input	Sample Output

```
ab
bc
aA
aN
Rb
cB
0
aA
bB
Cc
0
aA
aZ
0
aA
bB
cC
xX
yY
ax
xb
by
yc
0
```
	```
	Case 1: 2 1
	Case 2: 0 2
	Case 3: impossible
	Case 4: 0 7
	```

L - The Mailbox Manufacturers Problem

Source file name: `mailbox.c`, `mailbox.cpp`, `mailbox.java`, *or* `mailbox.py`

In the good old days when Swedish children were still allowed to blow up their fingers with fire-crackers, gangs of excited kids would plague certain smaller cities during Easter time, with only one thing in mind: to blow things up. Small boxes were easy to blow up, and thus mailboxes became a popular target.

A small mailbox manufacturer is interested in how many fire-crackers his new mailbox prototype can withstand without exploding and has hired you to help him. He will provide you with k ($1 \leq k \leq 10$) identical mailbox prototypes each fitting up to m ($1 \leq m \leq 100$) crackers.

However, he is not sure of how many fire-crackers he needs to provide you with in order for you to be able to solve his problem, so he asks you. You think for a while and then say: ‘‘Well, if I blow up a mailbox I can't use it again, so if you would provide me with only $k = 1$ mailboxes, I would have to start testing with 1 cracker, then 2 crackers, and so on until it finally exploded. In the worst case, that is if it does not blow up even when filled with m crackers, I would need $1 + 2 + 3 + \cdots + m = \frac{m(m+1)}{2}$ crackers. If $m = 100$ that would mean more than 5000 fire-crackers!’’. ‘‘That's too many’’, he replies. ‘‘What if I give you more than $k = 1$ mailboxes? Can you find a strategy that requires less crackers?’’

Can you? And what is the minimum number of crackers that you should ask him to provide you with?

You may assume the following:

1. If a mailbox can withstand x fire-crackers, it can also withstand $x - 1$ firecrackers.
2. Upon an explosion, a mailbox is either totally destroyed (blown up) or unharmed, which means that it can be reused in another test explosion.

Note that if the mailbox can withstand a full load of m fire-crackers, then the manufacturer will of course be satisfied with that answer. But otherwise he is looking for the maximum number of crackers that his mailboxes can withstand.

Input

he input starts with a single integer N ($N \geq 0$) indicating the number of test cases to follow. Each test case is described by a line containing two integers: k and m, separated by a single space.

The input must be read from standard input.

Output

For each test case print one line with a single integer indicating the minimum number of fire-crackers that is needed, in the worst case, in order to figure out how many crackers the mailbox prototype can withstand.

The output must be written to standard output.

Sample Input	**Sample Output**

4
1 10
1 100
3 73
5 100
	55
	5050
	382
	495