ACM@FSU
Fall 2016 Programing Contest
October 22, 2016

Do not open until contest starts
Instructions for Participants

- Contest URL: `http://52.32.226.59/domjudge/`
- You have 5 hours to answer questions.
- You may submit solutions in the following languages:
  - C/C++ (1999, 2011)
  - Java 7
  - C# (3.2.8.0)
  - Python (2.7 or 3.x)
  - Perl
  - Javascript
- You only allowed access to official language documentation and Melina Myers' COP3014 reference material. You are restricted to:
  - C/C++: http://www.cplusplus.com/reference/
  - Java: http://docs.oracle.com/javase/6/docs/api/
  - C#: https://msdn.microsoft.com/en-us/library/618ayhy6.aspx
  - Python 2.7: https://docs.python.org/2/
  - Python 3.x: https://docs.python.org/3/
  - Perl: http://perldoc.perl.org/
  - Javascript: http://developer.mozilla.org/en-US/docs/Web/Javascript/Reference
  - COP3014 Reference: http://www.cs.fsu.edu/ vastola/cop3014/

- You are also allowed one textbook or material no larger than 8.5" x 11" x 2" volume.
- No other resources (e.g., Stack Overflow, Google, Wikipedia) are permitted. Using non-permitted materials will lead to disqualification.
- Teams are restricted to using one workstation (computer) each.
- Use of a cell phone to circumvent these restrictions will lead to disqualification. Use of cell phones in contest rooms is not permitted.
- All input is redirected via STDIN.
- **Scoring:**
  - Teams are ranked according to score. A higher score is rewarded by answering more questions while acquiring fewer penalties.
  - The team that solves the greatest number of questions in the quickest time wins.
  - Teams which solve the same number of problems are ranked by least total time.
  - Teams may resubmit solutions as many times as needed, but incorrect submission attempts will result in time penalties (and thus a lower score.)
  - The scoreboard may be accessed during the first four hours of the contest. The scoreboard will freeze during the final hour.
  - The Clarifications tab on Domjudge may be used to submit questions pertaining to each problem. Do not use this feature to request troubleshooting help.

Announcement - HackFlorida

HackFlorida is a brand new, flagship hackathon here at Florida State. Hosted by students from various student groups around campus and partnered with ACM, HackFlorida aims to pioneer a new type of hackathon which rewards technical skill and heavyweight hacks. HackFlorida aims to be the first of many hackathons built by technology, for technology.

HackFlorida is hosting an announcement & interest meeting on Wednesday, October 26th in Love 151. We encourage anyone interested in building the technologies used to host the hackathon to attend.

1 Finding Factorials

Our friendly graduate student, Preston Hamlin, has gotten a rare case of amnesia where he has forgotten all his math and can only learn things backwards. To help him recover, everyone in the department is developing a program to teach him a basic topic, but the method done so in reverse.

Your job is to teach him factorials. Factorials are number expansions such that

5! = 5 * 4 * 3 * 2 * 1 = 120

To teach Preston factorials, write a program that takes in a number and calculates if it is a factorial or not. If it is, the result should give the number as a factorial. If not, print NONE.

1.1 Input

The first line of the input will be a single integer representing the n number of test cases for the program's execution. There will be n following lines each containing a single integer.

1.2 Output

Your output should be either a number followed by an exclamation point (e.g. "8!"), or "NONE".

1.3 Sample Input/Output

Sample Input	Sample Output
2	
40320	8!
56789	NONE

2 Lexicographical Anagrams

An anagram is direct word switch or word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once; for example, the word *cinema* can be rearranged into *iceman*.

The lexicographic or lexicographical order (also known as dictionary order or alphabetical order) is a generalization of the way the alphabetical order of words is based on the alphabetical order of their component letters.

In this problem, you must determine from a given input string an anagram that is first in lexicographical order among all possible anagrams.

2.1 Input

The first line of the input will be a single integer representing the n number of test cases for the program's execution. There will be n following lines each containing a string. All strings will consist only of lowercase characters.

2.2 Output

The each line of the output should be the lexicographically first anagram for the corresponding input.

2.3 Sample Input/Output

Sample Input	Sample Output
3	
aabaa	aaaab
abbaa	aaabb
acbac	aabcc

3 Traveling DJ Problem

Deciding Graduate School just wasn't his thing, David chose to try his hand at being a DJ. Under the name DJ Pedro, he tours the country playing for only the best venues. However there are only so many venues to choose from, so he has to plan his route very carefully.

In order to keep things interesting, he makes sure he never ends his tour by playing the same venues multiple times in a row. As long as it's not a perfect loop, he's in the clear. Help DJ Pedro catch the loops in his tour route!

Each venue is represented by a single character. Given a character string input, determine if the string terminates in a repeating loop of venues (characters) appearing at least twice. If so, print the character loop; otherwise print NONE.

3.1 Input

The first line of the input will be a single integer representing the n number of test cases ($n < 50$) for the program's execution. Each following n lines will consist of both uppercase and lowercase characters, where each line is fewer than 512 characters in length. Uppercase and lowercase characters are treated separately.

3.2 Output

Each line of output should either show the character loop for its corresponding input line, or in the case of no character loop, display the word none in all lowercase letters.

3.3 Sample Input/Output

Sample Input	Sample Output
3	
abcxyzxyz	xyz
abcxyabcxy	abcxy
abcdxyxyxya	none

4 Hipster Matchmaking

Jon, a great paragon of hipster-dom, has enlisted the services of Hipster Matchmaking Inc. in order to find his perfect soulmate. As the newest matchmaker in the organization, you will have to develop an algorithm and find Jon's ideal match using his ideal hipster criteria. Be careful though, some candidates might be too hipster to handle!

In order to help you along, Jon has provided the following information:

- A possible date's hipster level is defined by two hipster values:
  - How many super cool indie rock bands they know that you probably haven't heard of.
  - How often they tell everyone that they aren't a hipster in a day.
- If a person doesn't know any cool indie rock bands that you probably haven't heard of, then their hipster level is how often they tell everyone that they aren't hipster in a day plus one.
- If a person does know of at least one band that you probably haven't heard of, but they do not tell anyone that they aren't a hipster each day, then their hipster level is equivalent to that of a person who knows of one less indie band that you probably haven't heard of, and tells people they aren't hipster only once a day.
- If they know both of at least one band that you probably haven't heard of, and tell at least one person that they aren't hipster in a day, then their hipster level is equivalent to that of a person who knows of one less band that you probably haven't heard of, and tells people that they aren't a hipster the amount of times of a hipster level of someone who knows the same amount of bands that you probably haven't heard of but tells people that they aren't hipster one less time a day.

4.1 Input

The input file contains several test cases, each of them as described below.

The very first line represents the number of test cases that will be run. Each test case consists of the following: A single line with two integers x and y (0 <= x, y <= 1,000,000) that represent the hipster values for Jon himself.

This is followed by an integer n (1 <= n <= 10,000), which denotes the number of possible candidates that exist.

Finally, the last n lines represent the hipster values for the set of candidates in a test case sorted in ascending order of hipster-ness. **No negative numbers will show up in any test case for this question.**

4.2 Output

For each test case, print, on a single line, the values for the candidate that is the perfect hipster match for Jon. That is, the hipster value for Jon must equal the hipster value for a candidate, otherwise print -1.

4.3 Sample Input/Output

Sample Input	Sample Output
3 2 26 5 2 14 1 53 2 39 3 8 4 6 0 100 8 3 1 1 20 1 46 3 3 1 69 2 49 2 81 10 15 3 0 3 0 34 1 35 3 10	1 53 2 49 -1

5 Parsing Passwords

Tim has discovered a new way to store his passwords: he hides them inside longer strings with other words. In doing so, he makes sure he follows two rules: the password is the longest sequence of characters that appears multiple times. However, Tim recently lost his decryption software, and now he just has a big list of encrypted passwords! Help Tim out by decrypting some of his passwords for him.

Given a character string input, find a subsequence that is longer and appears more often than other subsequences. When choosing the best candidate subsequence, priority is given to the length of a sequence over its frequency. To break ties between subsequences of equal length and frequency, the sequence which appears later alphabetically is chosen.

For example, the sequence "abcd" appearing once is selected over "xyz" appearing twice. Likewise, the sequence "abc" appearing twice is selected over "xyz" appearing once. For "abc" and "xyz" each appearing once, the latter sequence is chosen.

There are some constraints about the sequences:

- Any given character may only appear in a sequence once.
- A sequence must appear at least twice to be selected.
- A sequence must be at least 3 characters in length.

As such, the input "abcda" has no acceptable sequence as it is constrained by the first two rules. Likewise, the input "zyababzy" also does not contain any acceptable sequences.

That being said, there will be an acceptable sequence for each input test case. Additionally, all input characters will be in lowercase.

5.1 Input

The first line of the input will be a single integer representing the n number of test cases for the program's execution. There will be n following lines each containing a string. All strings will consist only of lowercase characters.

5.2 Output

Each line of output should be the longest frequent subsequence from the corresponding input line, as described in the first section.

5.3 Sample Input/Output

Sample Input	Sample Output
3	
aabbcc	abc
abcabcxyzxyz	abcxyz
xyzabcabcxyz	xyz

6 What If: No Rules

Arithmetic and Logic equations use a set of operator precedence rules to disambiguate expression representation. For arithmetic, we all know that multiplication has higher precedence than addition, so the expression:

5 + 8 × 3, results in a value of **29**, not **39**.

What if there were no rules for evaluating binary logical operations?

Expressions like:

true or false xor true

could result in a value of **true** if interpreted as **true or (false xor true)**, or result in the value of **false** if interpreted as **(true or false) xor true**.

Write a program that evaluates logical expressions and for each expression outputs the number of ways in which the expression could be evaluated as **true** and the number of ways in which the expression could be evaluated as **false**.

6.1 Input

The first line of text input contains an integer that specifies the number of expressions to evaluate. Each successive text line contains a logical expression using values true and false and binary operators and, or, and/or xor.

6.2 Output

Output consists of a line of text for each expression specifying the number of ways in which the expression evaluates to true and false. You may assume that each equation has fewer than 20 operators.

6.3 Sample Input/Output

Sample Input	Sample Output
5	
true	1 true and 0 false
true and false	0 true and 1 false
true or false xor true	1 true and 1 false
true xor false or true xor false	2 true and 3 false
true or true or true or true or true	14 true and 0 false

7 Nightly Tours

There is a problem in the land of chess. A Knight is secretly meeting with the Queen each night. The King is concerned and has stationed guards throughout the kingdom to catch the Knight. The knight is aware of this and carefully plans his route to meet the Queen each night. The King has asked you to determine whether the Knight was blocked from seeing the queen over the past N days. The Bishops cannot see the Queen, nor can they see the starting location of the Knight.

You are given the location of the Queen and Bishops guarding her each night. You have been able to infer the starting location of the Knight each night. What nights could the Knight have met with the Queen?

7.1 Input

The first line contains the number N, following N lines each containing the coordinates of: Queen, Knight, Bishop1, and Bishop2 The coordinates are 0,0 for the top-left square and 7,7 for the bottom-right square.

7.2 Output

There should be N lines of output stating whether the Knight could meet the Queen on each of the seven nights.

7.3 Sample Input/Output

Sample Input	Sample Output
3	
0 0 0 4 1 1 0 7	Knight can meet Queen
0 0 0 4 1 1 1 2	Knight is blocked
6 1 0 7 3 7 0 0	Knight can meet Queen