University Interscholastic League

Computer Science Competition

2015 Regional Programming Problem Set

<u>DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!</u>

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Table of Contents

1. Codebreaker

Program Name: Codebreaker.java Input File: codebreaker.dat

This encryption algorithm is a variation of the substitution cipher. Each letter in the plain text is replaced by some letter of the alphabet in the encrypted text. A letter could be replaced by itself. You can think of the encrypted message as a simple permutation of the letters of the alphabet. Assume that the encrypted messages are only composed of lower case letters of the English alphabet. There are no upper case letters, digits, or punctuation marks in either the plain text or encrypted text. Words are separated by single spaces. You are given a dictionary of known words that could be in the encrypted message. Given a message that appears to be encrypted, determine if you can decrypt it using the words in the dictionary. If you can decrypt the message, print the decrypted message. Otherwise print that the message cannot be decrypted. Assume that a message can be decrypted if there exists some permutation such that all of the decrypted words are in the known words list.

Input

The first line of input contains two integers N and M. N is the number of words in your dictionary and M is the number of encrypted messages you must decrypt. The next N lines are the words in your dictionary, one word per line, in lowercase. Following the dictionary are M lines of messages, each encrypted in the manner described above. Assume that each message is encoded using a different permutation of letters and that if the message can be decrypted there is only one valid decryption.

Output

For each encrypted message, print out its decrypted form, or print that it cannot be decrypted.

Constraints

```
1 <= N <= 80
1 <= M <= 20
```

Example Input File

```
6 3
and
dice
jane
puff
spot
yertle
xxxx yyy zzzz www yyyy aaa bbbb ccc dddddd
hxsn xsb qymm xsb rqat bjvn pnetfn
jane and puff and spot and yertle diced
```

Example Output to Screen

```
Unable to decrypt message.
jane and puff and spot dice yertle
Unable to decrypt message.
```

2. Code Quality

Program Name: CodeQuality.java   Input File: codequality.dat

It is your first day on the job at Facebizzle, and you cannot wait to sit down and write some code. However, Facebizzle uses the esoteric Flowjure language, which is a variant of LISP, and is heavy on the use of parentheses, brackets, and braces. Since you are new to writing Flowjure, you want to write a helper script to check if your Flowjure is valid and compute some statistics on our code.

The most important characters in Flowjure are `(`, `[`, `{`, `}`, `]`, and `)`. In order to be a valid Flowjure program, each the different types of parentheses must be balanced. Every opening parenthesis must be closed with the appropriate character in reverse order. For example, `([])` is balanced, but `{[}]` is not balanced, since the `[` is closed with the `}` character.

Input

The file begins with an integer `T (1 ≤ T ≤ 10)`. After that, `T` test cases follow. Each test case begins with an integer `N`, `(1 ≤ N ≤ 30)`. `N` is the number of lines of code. After that follows `N` lines of Flowjure code. Flowjure code can contain any characters, but the script that you are writing ignores non-parenthesis characters.

Output

If the code block is valid Flowjure, print "`YES A () B [] C {}`", where A, B, and C are the number of parenthesis, square brackets, and curly braces respectively. If the code block is not valid Flowjure, print "`NO X`" where X is the 0-based index of the first character that makes the code invalid. Ignore newlines when calculating this value. If it is impossible to detect that the code is invalid until the end of the program, print "`NO -1`".

Example Input File

```
3
1
(map #(str "Hello " % "!" ) ["Ford" "Arthur" "Tricia"])
3
(apply map vector [[:a :b :c]
                   [:d :e :f]
                   [:g :h :i]])
2
def fib (cons 1 (cons 1
    (lazy-seq (map + fib (rest  fib))))))
```

Example Output to Screen

```
YES 4 () 2 [] 0 {}
YES 2 () 8 [] 0 {}
NO 63
```

Explanation of Example Output

The last example input is missing an opening parenthesis, but we can't detect this until character 63. The other two inputs are valid Flowjure programs.

3. Espionage

Program Name: Espionage.java   Input File: espionage.dat

Universal Intercontinental Laminates (UIL) is in the process of developing new polymers, with the different offices across the globe collaborating on the research. Their competitor, your employer, would love to get their hands on the data, so UIL is being very secretive. One important piece of data is the molecular weight of the large polymer molecules that are being developed. The molecular weight is expressed as an integer. Whenever one research branch of UIL wants to send the molecular weight of a polymer to another branch, they encode that information as a piece of text, i.e. a string of printable ASCII characters. Your employers have intercepted this data. They have also figured out how the molecular weight data is encoded in the text. Your job is to write a program that will do the decoding.

Here is the encryption algorithm. Let `s` be the string that represents the encoded message. Call `si` the representation of the string obtained by replacing each character with the decimal representation of its ASCII value. For example, if `s` = `"ab"`, then `si` = `"9798"`. Let `i` be the integer value of `si`. In this example, `i = 9798`. You are given another integer `d`, and the molecular weight, is `i modulo d` (`i%d`). Only printable ASCII characters are used in the encoding, and there are no newlines or tab characters.

Input
The first line of input is an integer `t`, the number of test cases.
For each test case, there are two lines -- the first is a text line `s`, and the next is an integer `d`.

Constraints
`t` < 20
The length of 0 < `s` < 1000 and characters
0 < `d` < 100000

Output
Print one line for each test case with the molecular weight.

Example Input File
```
3
ab
9798
ab
12194
34508393
9087
```

Example Output to Screen
```
0
9798
8492
```

4. Fibonacci

Program Name: Fibonacci.java   Input File: fibonacci.dat

The standard textbook example for recursion is the code that computes the terms in the Fibonacci sequence. This is possibly one of the worst examples for recursion. In fact, it should be given as an example of when not to use recursion.

Here is the code as given in most textbooks:

```
public static int fibonacci(int n)
{
    if (n == 0 || n == 1)
      return n;
    else
      return fibonacci(n - 1) + fibonacci(n - 2);
}
```

The nth fibonacci number is defined as the sum of the (n – 1)th fibonacci number and the (n - 2)th fibonacci number. For the base case the 0th fibonacci number is 0 and the 1st fibonacci number is 1.

If we were to expand all of the calls made to the method fibonacci(), you would see that there are a lot of duplicate function calls that end up wasting time and space on the function call stack. For small values of n this is not a problem. For example, for n = 4 there are only 4 duplicate calls. However for bigger values of n the number of duplicate calls escalates.

Given some integer N, determine how many duplicate calls will be made to the fibonacci() method.

Input
The first line of input will contain a single integer T, the number of test cases.
The only line of each test case contains a single integer N.

Output
For each test case, print out the number of duplicated function calls.

Constraints
```
1 <= T <= 10
1 <= N <= 50
```

Example Input File
```
3
4
9
1
```

Example Output to Screen
```
4
99
0
```

5. Fractal

Program Name: Fractal.java   Input File: fractal.dat

You recently learned about fractals in geometry class, and after deciding they are the coolest things ever, you decide to implement one. But having one pattern is boring, so you decide to define multiple rules of expansion for creating a fractal. You want to see what the fractal looks like after multiple iterations as defined by the expansion rules.

You will be given multiple fractals to create. Each fractal consists of the rules that define its expansion, as well as a list of tasks to be performed. Each task consists of a start state, and the number of expansions to run. Each rule defines what each string in the current state becomes in the next iteration of expansion.

To run an expansion of a given state, go through each string in the current state, and if it has a rule to expand it, replace it with the specified expansion in the next state; otherwise, just copy it to the next state.

Input
The first line of input contains F, the number of fractals that follow.

The first line of each fractal contains two integers R and T, the number of rules and tasks in the fractal.
The next R lines each contain one rule in the format S → S', where S is a string and S' is a space separated list of strings. Each S is unique, so there will not be multiple possible expansions for a single string.
The next T lines each contain a task. Each task consists of two lines. The first line contains two integers N and X, the number of strings in the start state, and X is the number of expansions to run. The next line contains N space-separated strings, the strings in the start state.

Output
For each fractal, output the fractal number, followed by the final results of each task, each on a separate line.

Constraints
```
1 <= F <= 5
1 <= R, T <= 10
1 <= number of strings in S' <= 3
1 <= N <= 4
1 <= X <= 5
```

Example Input File
```
2
1 3
X -> X X
1 1
X
1 2
X
2 1
X X
1 1
A -> A B
1 5
A
```

Example Output to Screen
```
Fractal #1:
X X
X X X X
X X X X
Fractal #2:
A B B B B B
```

Explanation of Example Output
For the first fractal, every X turns into two X's. Therefore, for the first task, we expand X once, getting X X. For the second task, we expand X twice, getting X X and then X X X X. For the third case, we expand X X once, getting X X X X.

For the second fractal, A turns into A B, and B stays a B since there is no rule for B.

We have to expand A five times. After once time it is A B, after two it is A B B, after three it is A B B B, after four it is A B B B B, and after 5 it is A B B B B B.

6. Fraction Addition

Program Name: FractionAdd.java   Input File: fractionadd.dat

It is easy to evaluate arithmetic expressions that involve integers. But to evaluate fractions exactly is slightly more difficult. In this problem you will be given fractions that you will have to add and compare your result with another fraction.

Input
The input contains multiple test cases. The first line contains **T** the number of test cases. **T** lines follow, one for each test case. Each test case has a series of **F** fractions with '+' symbols between them, followed by the '?' symbol, followed by another series of **K** fractions. Each fraction is represented by its integer numerator, then a '/' symbol, and then its integer denominator. Every integer is separated from the preceding and following symbols by a space. You have to sum up the fractions before the '?' symbol and check to see if the resulting fraction **a/b** is greater than, equal to, or less than the sum of the fractions after the '?' symbol (**c/d**).

Constraints
1 <= **T** <= 10
1 <= **F, K** <= 5
1 <= numerator <= 30
1 <= denominator <= 30

Output
For every test case, print **a/b > c/d** or **a/b < c/d** or **a/b = c/d** where **a** and **b** are the integer numerator and denominator of the left-hand-side sum, respectively, and **c** and **d** are the integer numerator and denominator of the right-hand-side sum, respectively. The only spaces are before and after the equality or inequality symbols. Make sure you present the fraction in reduced form, with the numerator and denominator having no common divisors except 1. The fractions could be improper fractions where the numerator is greater than the denominator.

Example Input File
```
4
1 / 5 + 6 / 5 ? 1 / 1
4 / 5 + 2 / 1 + 1 / 3 ? 1 / 4 + 3 / 4 + 1 / 3
3 / 2 + 5 / 4 ? 5 / 3 + 10 / 8
1 / 10 + 2 / 10 ? 6 / 20
```

Example Output to Screen
```
7/5 > 1/1
47/15 > 4/3
11/4 < 35/12
3/10 = 3/10
```

7. Lineup

Program Name: Lineup.java   Input File: lineup.dat

When the drill sergeant blows his whistle, all of the privates must line up. To make it easy to check roll, the privates line up in the order on the list that the sergeant has. Unfortunately he has lost that list, and wants you to reconstruct it before 0800 today.

The order of the list is as follows:
If two privates are different height, the shorter one goes first.
If two privates are the same height, the one with the lexicographically smallest last name goes first.
If they have the same height and last name, the one with the lexicographically smallest first name goes first.
Lexicographical order is the order Java compares strings in by default.

Input
The first line of input will contain a single integer T, the number of test cases.
The first line of each test case contains a single integer N, the number of soldiers.
The following N lines each contain three integers F, L, and H, where F is the first name of the soldier, L is the last name of the soldier, and H is the height of the soldier in inches.

Output
For each test case, print the test case number on its own line, followed by the names of the soldiers in the order they are on the roll call sheet.

Constraints
```
1 <= T <= 10
1 <= N <= 1000
1 <= H <= 100
```

Example Input File
```
2
2
David Smith 66
Andrew John 65
5
Josh Smith 67
John Smith 67
Jake Smithman 67
Jack Smith 65
Jacob Smith 75
```

Example Output to Screen
```
Test Case #1:
Andrew John
David Smith
Test Case #2:
Jack Smith
John Smith
Josh Smith
Jake Smithman
Jacob Smith
```

Explanation of Example Output
In the first test case, Andrew John is shorter than David Smith, so Andrew goes first and David goes second.

In the second test case, Jack Smith is the shortest, so he goes first. Then, we have 3 people who are 5'7", so we need to sort them by name. Smith comes before Smithman lexicographically, and the tie between John and Josh is broken since John comes lexicographically before Josh. Thus, the next three are John, Josh, and Jake. Finally, Jacob Smith is the tallest, so he comes last.

8. Pattern

Program Name: Pattern.java   Input File: pattern.dat

There are patterns that appear in poetry and music. Songs for example often utilize the AABA form, while narrative motifs often employ a Chiastic structure, such as ABBA, to compare and contrast details of particular importance. As a first step in developing the software to determine whether a particular passage matches a specific pattern, you are asked to determine if a short phrase matches a particular pattern.

Input

The first line of input contains an integer T, the number of test cases in the input.
Each of the following T test cases consists of two lines. The first line contains an uppercase alphabetical character pattern, like ABBA, and the second line contains a phrase consisting of space delimited lowercased words.

Output

For each pattern and phrase pair, determine if the words match the pattern, as in there exists some distinct mapping of each unique character in the pattern to a unique word.

Constraints

```
1 <= T <= 100
1 <= Length of character pattern <= 32
```

Example Input File

```
3
ABBA
hello world world hello
RAD
happy happy day
NEVER
foo bar baz qux waldo
```

Example Output to Screen

```
Matches
Does Not Match
Does Not Match
```

Explanation of Example Output

In the first case, A matches with hello and B matches with world.
In the second case, happy is not distinct as it would have to match both R and D.
In the third case, E is not distinct as it would have to match both bar and qux.

9. Railroad

Program Name: Railroad.java   Input File: railroad.dat

You are in charge of constructing a high speed railroad across a desert. The desert is very flat, except for a number of rectangular plateaus that would impede the construction. The train goes very fast, so it must travel in a straight line the whole time. It would be very expensive to tunnel through the plateaus, so your boss wants to know how feasible it is to avoid them.

The train is 6 meters wide, and the engineers tell you it should have at least 2 meters of buffer on each side for safety. Thus, you have decided to break the desert up into a 10x10 meter grid. Your boss says you can either go across the desert vertically or horizontally, so long as there is a straight line of 10 x10 grid cells from one edge to the other that are not part of any plateau. How many ways can you make the train get across the desert?

The row number and column number start at zero. To cross the desert means either starting at the first row and going to the last row in a straight vertical path without going through a plateau or starting at the first column and going to the last column in a straight horizontal path without going through a plateau.

Input

The first line of input contains T, the number of test cases that follow.
The first line of each test case contains three integers N, M, and K. N and M are the number of rows and columns of the desert grid, and K is the number of plateaus in the desert.
The next K lines each contain a single plateau in 4 integers: R, C, W, L. R and C are the row and column of the minimum row/column corner of the plateau, and W and L are the width and length of the plateau. Thus, the plateau covers the intersection of rows R, R+1, R+2, …, R+W-1, and columns C, C+1, C+2, …, C+L-1.

Output

For each test case, print out the number of possible ways to construct a valid track for the train.

Constraints

```
1 <= T <= 10
1 <= N, M <= 10^5
0 <= K <= 50
0 <= R < N, 1 <= W <= N, 1 <= R + W <= N
0 <= C < M, 1 <= L <= M, 1 <= C + L <= M
```

Example Input File

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

Example Output to Screen

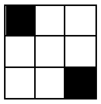
```
0
3
2
1
```

Explanation of Example Output

In the first case, the entire 1x1 desert was covered, so there is no valid path.

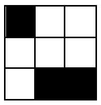
In the second case, the whole desert is open. Thus, you could go all the way down the 2 columns or 1 row.

In the third case, the desert looks like this:



So the only way to get across is down column 1 or across row 1.

In the fourth test case, the desert looks like this:



And thus, the only way to get to the other side is across row 1.

10. Sur

Program Name: Sur.java Input File: sur.dat

In researching Russian surnames, you have learned a little about their naming tradition, in the fact that quite often you can tell the gender of the person by their surname. For example, Andropov, the name of a Soviet politician back in the Cold War days, indicates a man. On the other hand, Andropova would be the name used for a woman with that surname, simply by adding an "a" to the "ov". Tennis stars Maria Sharapova and Anna Kournikova are two well-known celebrities with such female surnames. You have not yet discovered any other surname characteristics, although there very well may be other such traditions in the Russian culture.

For now, based on your research so far (as described above) you simply need to peruse through a list of names and determine how many surnames clearly indicate a man, how many indicate a woman, and how many are not clear as to the gender.

Input
A series of Russian surnames, each on one line.

Output
Three integers on one line, the first indicating how many Russian males, the second the number Russian females, and the third the number of names with an undetermined gender.

Example Input File
```
Andropov
Sharapova
Kournikova
Bobrov
Pakhomova
Yefimov
Gachev
Voskoboynikov
Nikitin
Severova
```

Example Output to Screen
```
4 4 2
```

Explanation of Output
There are 4 names that end in "ov", 4 ending in "ova", and 2 that are neither.

11. Toffee

Program Name: Toffee.java Input File: toffee.dat

Your little sister has been invited to an Easter Egg hunt. Each egg contains a variable number of miniature chocolate covered butter toffee that absolutely melts in your mouth. She has been given a basket and she can collect as many eggs that she wants subject to the following constraints. The Easter eggs have been arranged in the form of a square grid. She can start anywhere on the first row in that square grid and she can only move to the square directly below or to the square diagonally below and to the right. She can only pick the eggs in the squares that she visits. She wants your help in getting the maximum number of chocolate covered butter toffee.

Input

The first line has a single integer T that is the number of test cases to follow. Each test case has the first line giving N the dimension of the square grid, where $1 \leq N \leq 10$. There are N lines of data that follow. Each line of data has N integers that represent the number of toffees in each egg. The number of toffees in each egg is in the range 1 through 50 inclusive. The integers are separated by one or more spaces.

Output

Your output will be a single integer giving the maximum number of toffees that you can collect for each of the test cases. Since we are not interested in knowing the path that she takes through the grid, it is quite possible that there may be more than one path that yields the maximum number of toffees for a given test case.

Example Input File

```
3
3
4 9 2
3 5 7
8 1 6
4
1 2 3 4
8 7 6 5
9 1 2 3
7 6 5 4
5
14 22 43 27  8
33 19  7 13 41
37 17 37  3 31
 5  9 16 18  2
 6 24 20 23 29
```

Example Output to Screen

```
22
25
134
```

Explanation of Example Output

For the first test case the path starts at 9, then 7 and 6 giving 22.
For the second test case the path starts at 1, then 8, 9, and 7 giving 25.
In the third test case, the path taken starts at 43 then 7, 37, 18, and 29 giving 134.

12. Tri

Program Name: Tri.java   Input File: tri.dat

One of the fundamental principles in geometry is that the sum of the lengths of any two sides of any triangle is always greater than the length of the third side. For example, 1, 2, and 3 will not work as sides of a triangle since 1 + 2 is not greater than 3, however, 2, 2, and 3 will work. Given three integer values, determine whether or not they can be the sides of a triangle. The values given are guaranteed to be greater than zero, but could be as large as one yottabyte (a very big number).

Input
Several sets of integers, each set on one line.

Output
The three values in ascending order, followed by YES or NO, (all uppercase letters) indicating whether or not the three given sides can represent the sides of a triangle.

Example Input File
```
1 2 3
2 2 3
3 3 6
3 7 3
3 2 2
14 12 17
8 8 5
10 7 2
```

Example Output to Screen
```
1 2 3 NO
2 2 3 YES
3 3 6 NO
3 3 7 NO
2 2 3 YES
12 14 17 YES
5 8 8 YES
2 7 10 NO
```