Préparation au concours ACM – TP 1

Christoph Dürr Jill-Jênn Vie

September 19, 2014

Problems

All written contest materials will be in English.

Solutions to problems submitted for judging are called runs. Each run is judged as accepted or rejected, and the team is notified of the results. Rejected runs will be marked as follows:

- Compilation Error
- Wrong Answer
- Time Limit Exceeded
- Presentation Error (lack of newline, extra spaces, etc.)
- Runtime Error (most possibly a segmentation fault)

Scoring

Teams will be ranked by number of solved problems. In case of a tie, teams with less total submission time are ranked first. For each solved problem, the time (in minutes) since contest start will be added to the team's total time. For each wrong submission prior to the successful one, 20 penalty minutes are added.

Material

Each team will be permitted to provide a PDF of up to 25 pages of notes within the limits described during Team Certification. Three copies will be printed and placed at the team's workstation for use during the World Finals.

Each team member may bring one printed, unannotated natural language dictionary. You may bring mascots such as stuffed toy animals or party hats.

Computer

The keyboards will have Portuguese layout, but teams may bring their own USB-keyboard.

The command line for compiling used in the automated judge system is the following:

```
g++ -w -O3 -pipe $SOURCE
```

Links

You need to create an account on `http://uva.onlinejudge.org` to submit your codes.

For a few problems, you can get the output of any input on `uvatoolkit.com`.

Your Favorite Number

Write a program that prints your favorite (integer) number. It can be any number: your age, the problems your team solved at UVa Online Judge, your TopCoder score, etc. Just print the one you like the most.

Input

There is no input for this problem.

Output

The program must write to the standard output your favorite number.

Sample Input 1

Sample Output 2

```
42
```

Sample Input 1

Sample Output 2

```
168
```

Mean Value

Write a program that computes the mean value of a list of integers.

Input

An integer N, followed by a line containing N integers.

Output

The mean value of these N integers. Write out the result to two decimal places.

Sample Input

```
3
2 1 1
```

Sample Output

```
1.33
```

Find the Twin

You are given a list of words. Find out if there's a double.

Input

An integer T, followed by T cases of the following form:

- An integer N;
- On the next N lines, a word.

Output

For each instance you must output one line of the following form:

```
Case #d: ANSWER
```

Where `d` stands for the instance number (starting from 1) and `ANSWER` is the lowest double in lexicographical order if it exists, `None` otherwise.

Sample Input

```
3
1
Alice
2
Bob
Bob
3
Charles
Damien
Etienne
4
Frederic
Fanja
Fanja
Frederic
```

Sample Output

```
Case #1: None
Case #2: Bob
Case #3: None
Case #4: Fanja
```

Pascal's Triangle of Death

UVa 485

In this problem, you are asked to generate Pascal's Triangle. Pascal's Triangle is useful in many areas from probability to polynomials to programming contests. It is a triangle of integers with "1" on top and down the sides. Any number in the interior equals the sum of the two numbers above it. For example, here are the first 5 rows of the triangle.

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

In "Pascal's Triangle of Death," you are to generate a left justified Pascal's Triangle. When any number in the triangle is exceeds or equals 10^{60}, your program should finish printing the current row and exit. The output should have each row of the triangle on a separate line with one space between each element.

The final element of each line should be directly followed by a newline. There is no space after the last number on each line.

Sample Input

There is no input for this problem.

Sample Output

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
.
.
.
```

etc.

UVa 10189

The Problem

Have you ever played Minesweeper? It's a cute little game which comes within a certain Operating System which name we can't really remember. Well, the goal of the game is to find where are all the mines within a $M \times N$ field. To help you, the game shows a number in a square which tells you how many mines there are adjacent to that square. For instance, suppose the following 4×4 field with 2 mines (which are represented by an * character):

```
*...
....
.*..
....
```

If we would represent the same field placing the hint numbers described above, we would end up with:

```
*100
2210
1*10
1110
```

As you may have already noticed, each square may have at most 8 adjacent squares.

The Input

The input will consist of an arbitrary number of fields. The first line of each field contains two integers n and m ($0 < n, m \leqslant 100$) which stands for the number of lines and columns of the field respectively. The next n lines contain exactly m characters and represent the field. Each safe square is represented by an "." character (without the quotes) and each mine square is represented by an "*" character (also without the quotes). The first field line where $n = m = 0$ represents the end of input and should not be processed.

The Output

For each field, you must print the following message in a line alone:

```
Field #x:
```

Where `x` stands for the number of the field (starting from 1). The next n lines should contain the field with the "." characters replaced by the number of adjacent mines to that square. There must be an empty line between field outputs.

Sample Input

```
4 4
*...
....
.*..
....
3 5
**...
.....
.*...
0 0
```

Sample Output

```
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100
```

Power Strings

UVa 10298

Given two strings a and b we define $a * b$ to be their concatenation. For example, if $a =$ "abc" and $b =$ "def" then $a * b =$ "$abcdef$". If we think of concatenation as multiplication, exponentiation by a non-negative integer is defined in the normal way: $a^0 =$ "" (the empty string) and $a^{n+1} = a * (a^n)$. Each test case is a line of input representing s, a string of printable characters. For each s you should print the largest n such that $s = a^n$ for some string a. The length of s will be at least 1 and will not exceed 1 million characters. A line containing a period follows the last test case.

Sample Input

```
abcd
aaaa
ababab
.
```

Output for Sample Input

```
1
4
3
```

LCM Pair Sum

UVa 12546 (SWERC 2012)

One of your friends desperately needs your help. He is working with a secret agency and doing some encoding stuffs. As the mission is confidential he does not tell you much about that, he just want you to help him with a special property of a number. This property can be expressed as a function $f(n)$ for a positive integer n. It is defined as:

$$f(n) = \sum_{\substack{1 \leqslant p \leqslant q \leqslant n \\ lcm(p,q)=n}} (p+q)$$

In other words, he needs the sum of all possible pairs whose least common multiple is n. (The least common multiple (LCM) of two numbers p and q is the lowest positive integer which can be perfectly divided by both p and q). For example, there are 5 different pairs having their LCM equal to 6 as $(1,6),(2,6),(2,3),(3,6),(6,6)$. So $f(6)$ is calculated as $f(6) = (1+6)+(2+6)+(2+3)+(3+6)+(6+6) = 7+8+5+9+12 = 41$. Your friend knows you are good at solving this kind of problems, so he asked you to lend a hand. He also does not want to disturb you much, so to assist you he has factorized the number. He thinks it may help you.

Input

The first line of input will contain the number of test cases T ($T \leqslant 500$). After that there will be T test cases. Each of the test cases will start with a positive number C ($C \leqslant 15$) denoting the number of prime factors of n. Then there will be C lines each containing two numbers P_i and a_i denoting the prime factor and its power (P_i is a prime between 2 and 1000) and ($1 \leqslant a_i \leqslant 50$). All the primes for an input case will be distinct.

Output

For each of the test cases produce one line of output denoting the case number and $f(n)$ modulo 1000000007. See the output for sample input for exact formatting.

Sample Input 1

```
3
2
2 1
3 1
2
2 2
3 1
1
5 1
```

Sample Output

```
Case 1: 41
Case 2: 117
Case 3: 16
```