COMPUTER SCIENCES

George Fox University
H.S. Programming Contest
Division – I
April 10, 2021

General Notes

1. Do the problems in any order you like. They do not have to be done in order
(hint: the easiest problem may not be the first problem)

2. Scoring: The team who solves the most problems in the least amount of time with the least submissions wins. Each wrong submission will receive a 20 min time penalty that will only be added to the time score once the problem has been successfully solved. Time is calculated for each problem as the total time from the start of the contest to the time it was solved.

3. There is no extraneous input. All input is exactly as specified in the problem. Integer inputs will not have leading zeros.

4. Your program should not print extraneous output. Do not welcome the user. Do not prompt for input. Follow the form exactly as given in the problem.
(hint: spaces? No spaces? What does spec say!)

5. All solutions must be a single source code file.

1. Odd Palindrome

We say that a string is *odd* if and only if all palindromic substrings of the string have odd length. Given a string *s*, determine if it is *odd* or not.

A substring of a string s is a nonempty sequence of consecutive characters from *s*. A palindromic substring is a substring that reads the same forwards and backwards.

Input
The first input will be a single integer indicating the number of following inputs to test. Each input consists of a single line containing the string s (1 <= length of *s* <= 100).
It is guaranteed that *s* consists of lowercase ASCII letters ('a'-'z') only.

Output
If s is odd, then print "Odd." on a single line (without quotation marks). Otherwise, print "Or not." on a single line (without quotation marks).

Example Input:
```
4
amanaplanacanalpanama
madamimadam
annamyfriend
nopalindromes
```

Output to screen:
```
Odd.
Odd.
Or not.
Odd.
```

This page intentionally left blank

2. Fear Factoring

The Slivians are afraid of factoring; it's just, well, difficult.

Really, they don't even care about the factors themselves, just how much they sum to.

We can define F(n) as the sum of all of the factors of n; so F(6) = 12 and F(12) = 28. Your task is, given two integers a and b with a <= b, to calculate

$$S = \sum F(n), \text{ where } a \leq n \leq b$$

In other words, you are to calculate the sum of the sum of factors for all integers between a and b, inclusive.

Input
The first input will be an integer indicating the number of data sets to evaluate. Each data set will be two integers *a b* separated by a space. Constraints for a and b: $(1 \leq a \leq b \leq 10^{12}; b - a \leq 10^{6})$

Output
Print *S* on a single line.

Example Input:
```
6
101 101
28 28
1 10
987654456799 987654456799
963761198400 963761198400
5260013877 5260489265
```

Output to screen:
```
102
56
87
987654456800
5531765944320
4113430571304040
```

This page intentionally left blank

3. Avoiding Airports

David is looking to book some travel all over the world. There are *n* countries that he can visit, and *m* flights that are available. The *i*th flight goes from country *ai* to country *bi*. It departs at time *si*, and lands at time *ei*.

David is currently in the airport in country 1, and the current time is 0, and he would like to travel to country *n*. He does not care about the total amount of time needed to travel, but he really hates waiting in the airport. If he waits *t* time units in an airport, he gains t^2 units of frustration. Help him find an itinerary that minimizes the total frustration.

Input
The first input will be a single integer indicating how many scenarios to be solved.
For each travel scenario, the first line of input contains two space-separated integers *n* and *m*:
($2 \leq n \leq 200{,}000$; $1 \leq m \leq 200{,}000$).

Each of the next *m* lines contains four space-separated integers *ai, bi, si,* and *ei* ($1 \leq ai, bi \leq n$; $0 \leq si \leq ei \leq 106$). This describes a flight from country *ai* to country *bi* that departs at time *si* and arrives at time *ei*.

A flight might have the same departure and arrival country.

No two flights will have the same arrival time or have the same departure time. In addition, no flight will have the same arrival time as the departure time of another flight. Finally, it is guaranteed that there will always be a way for David to arrive at his destination.

Output
Print, on a single line, the minimum total frustration.

In the first sample, it is optimal to take this sequence of flights:

- Flight 5. Goes from airport 1 to airport 2, departing at time 3, arriving at time 8.
- Flight 3. Goes from airport 2 to airport 1, departing at time 9, arriving at time 12.
- Flight 7. Goes from airport 1 to airport 3, departing at time 13, arriving at time 27.
- Flight 8. Goes from airport 3 to airport 5, departing at time 28, arriving at time 100.

The frustration for each wait is 3^2, 1^2, 1^2, and 1^2, respectively. Thus, the total frustration is 12.

Note that there is an itinerary that gets David to his destination faster. However, that itineraryhas a higher total frustration.

(continued on the next page)

Example Input:

```
2
5 8
1 2 1 10
2 4 11 16
2 1 9 12
3 5 28 100
1 2 3 8
4 3 20 21
1 3 13 27
3 5 23 24
3 5
1 1 10 20
1 2 30 40
1 2 50 60
1 2 70 80
2 3 90 95
```

Output to screen:

```
12
1900
```

4. Grid Coloring

You have an m-by-n grid of squares that you wish to color. You may color each square either red or blue, subject to the following constraints:

- Every square must be colored.
- Colors of some squares are already decided (red or blue), and cannot be changed.
- For each blue square, all squares in the rectangle from the top left of the grid to that square must also be blue.

Given these constraints, how many distinct colorings of the grid are there? The grid cannot be rotated.

Input

The first input will be an integer indicating the number of data sets to evaluate.
Each data set starts with two space-separated integers *m* and *n* ($1 \leq m, n \leq 30$).

Each of the next *m* lines contains *n* characters, representing the grid. Character 'B' indicates squares that are already colored blue. Similarly, 'R' indicates red squares. Character '.' indicates squares that are not colored yet.

Output
Print, on a single line, the number of distinct colorings possible. The example below shows the 6 coloring possibilities for the first sample set of data.

```
BB      BB      BR      BR      BB      BB
BB      BR      BR      BR      BR      BB
BR      BR      BR      RR      RR      RR
```

Example Input:

```
3
3 2
..
B.
.R
7 6
......
.....B
.B..R.
......
...B..
.R....
...R..
2 2
R.
.B
```

Output to Screen:

```
6
3
0
```

This page intentionally left blank

5. Spinning Up Palindromes

“Sabotage!”, exclaimed J. R. Diddly, president and founder of Diddly Widgets Inc.

“Vandalism, perhaps. Nothing's actually been damaged.” responded Robert Lackey, the chief accountant.

Both were staring up at the large counter suspended above the factory floor, a counter that had faithfully recorded the number of widgets that had come off the assembly line since the factory was opened. But someone had changed the number being displayed so that it formed...

“It's a palindrome.” said Lackey. “It reads the same forwards as backwards.”

“What I don't understand,” said Diddly, “is why our security guards didn't catch the vandals during their regular sweeps. It must have taken them hours to click forward to this new number, one step at a time.”

“No.” replied Lackey. “Although we only advance the rightmost digit each time a new widget is built, it's possible to spin any of the digits. With a little planning, this might have taken only a few seconds.”

Consider a digital counter consisting of *k* wheels, each showing a digit from 0 to 9. Each wheel is mounted so that it can advance to the next digit in a single step, e.g., from 3 to 4, or from 8 to 9.

It is also possible to advance from digit 9 to digit 0. However, when this happens, the wheel on its immediate left will also advance to the next digit automatically. This can have a cascade effect on multiple wheels to the left, but they all happen in a single step.

Given the current setting of the counter, find the smallest number of steps until one can reach a palindrome. The palindrome must respect leading zeros, e.g., 0011 is not a palindrome.
For example, for input 610, it takes four steps. This can be done by incrementing the 6 wheel four times, resulting in 010.

Input
The first input will be a single integer n indicating how many lock settings to evaluate. Each lock setting input contains a string of k digits (1 <= k <= 40), representing the current setting of the counter.

Note: lock settings may contain leading zeros

Output
Print, on a single line, the minimum number of wheel advances necessary to produce a palindrome.

(continued on the next page)

Example Input:

```
6
0
009990001
29998
610
981
90841947009409037971917182478011970192 68
```

Output to Screen:

```
0
3
5
4
2
54
```

6. Delayed Work

You own a company that hires painters to paint houses. You can hire as many painters as you want, but for every painter you hire, you have to pay X dollars (independent of how long the painter works on the house). In addition, you have to pay a penalty of D x P dollars overall if it takes D days to finish painting the house. This penalty does not depend on the number of painters you hire; furthermore, even if the time taken is not a whole number of days, you are only penalized for the exact time taken.

All painters paint at the same rate. One painter can finish painting the house in K days. The painters cooperate so well that it will only take K/M days for M painters to finish painting the house.

What is the minimum amount of money you will have to pay, including what you pay to the painters and the cost penalty, to get a house painted?

Input
The first input will be a single integer indicating how many houses we need to paint. Each houses input consists of a single line containing three space-separated integers K, P, and X
(1 <= K, P, X <= 10,000).

Output
For each house, print on a single line, the minimum cost to have the house painted, rounded and displayed to exactly three decimal places.

Example Input:
```
2
31 41 59
3 4 5
```

Output to Screen:
```
549.200
16.000
```

This page intentionally left blank

7. Latin Squares

A Latin Square is an n-by-n array filled with n different digits, each digit occurring exactly once in each row and once in each column. (The name “Latin Square” was inspired by the work of Leonhard Euler, who used Latin characters in his papers on the topic.)

A Latin Square is said to be in reduced form if both its top row and leftmost column are in their natural order. The natural order of a set of digits is by increasing value.

Your team is to write a program that will read an n-by-n array, and determine whether it is a Latin Square, and if so, whether it is in reduced form.

Input
The first input will be a single integer indicating how many Latin Squares are to be evaluated. The first line of each Latin Square contains a single integer n (2 <= n <= 36). Each of the next n lines contains n digits in base n, with the normal digits ‘0’ through ‘9’ for digit values below 10 and uppercase letters ‘A’ through ‘Z’ representing digit values 10 through 35. All digits will be legal for base n; for instance, if n is 3, the only legal characters in the n input lines describing the square will be ‘0’, ‘1’, and ‘2’.

Output
If the given array is not a Latin Square, print “No” on a single line (without quotation marks). If it is a Latin Square, but not in reduced form, print “Not Reduced” on a single line (without quotation marks). If it is a Latin Square in reduced form, print “Reduced” on a single line (without quotation marks).

(continued on next page)

Example Input:

```
3
3
012
120
201
4
3210
0123
2301
1032
11
0123458372A
A9287346283
0285475A834
84738299A02
1947584037A
65848430002
038955873A8
947530200A8
93484721084
95539A92828
04553883568
```

Output to Screen:

```
Reduced
Not Reduced
No
```

8. Unloaded Dice

Consider a six-sided die, with sides labeled 1 through 6. We say the die is fair if each of its sides is equally likely to be face up after a roll. We say the die is loaded if it isn't fair. For example, if the side marked 6 is twice as likely to come up as than any other side, we are dealing with a loaded die.

For any die, define the expected result of rolling the die to be equal to the average of the values of the sides, weighted by the probability of those sides coming up. For example, all six sides of a fair die are equally likely to come up, and thus the expected result of rolling it is (1+2+3+4+5+6)=6 = 3:5.

You are given a loaded die, and you would like to unload it to make it more closely resemble a fair die. To do so, you can erase the number on one of the sides, and replace it with a new number which does not need to be an integer or even positive. You want to do so in such a way that:

- The expected result of rolling the die is 3.5, just like a fair die.
- The difference between the old label and the new label on the side you change is as small as possible.

Input

The first input will be a single integer indicating how many data sets are to evaluated. Each data set consists of a single line containing six space-separated nonnegative real numbers $v_1 \ldots v_6$, where v_i represents the probability that side i (currently labeled by the number i) is rolled.

It is guaranteed that the given numbers will sum to 1.

Output

Print, on a single line, the absolute value of the difference between the new label and old label, rounded and displayed to exactly three decimal places.

Example Input

```
2
0.16666 0.16667 0.16667 0.16666 0.16667 0.16667
0.2 0.2 0.1 0.2 0.2 0.1
```

Example Output to Screen

```
0.000
1.000
```

This page intentionally left blank

9. Purple Rain

Purple rain falls in the magic kingdom of Linearland which is a straight, thin peninsula.

On close observation however, Professor Nelson Rogers finds that the purple rain is actually a mix of red and blue raindrops.

In his zeal, he records the location and color of the raindrops in different locations along the peninsula. Looking at the data, Professor Rogers wants to know which part of Linearland had the "least" purple rain.

After some thought, he decides to model this problem as follows. Divide the peninsula into n sections and number them west to east from 1 to n. Then, describe the raindrops as a sequence of R and B, depending on whether the rainfall in each section is primarily red or blue. Finally, find a subsequence of contiguous sections where the difference between the number of R and the number of B is maximized.

Input

The first input will be a single integer indicating the number of data sets to evaluate. Each data set consists of a single line containing a string of n characters ($1 <= n <= 10^5$), describing the color of the raindrops in sections 1 to n.

It is guaranteed that the string consists of uppercase ASCII letters `R' and `B' only.

Output

Print, on a single line, two space-separated integers that describe the starting and ending positions of the part of Linearland that had the least purple rain. These two numbers should describe an inclusive range; both numbers you print describe sections included in the range.

If there are multiple possible answers, print the one that has the westernmost starting section. If there are multiple answers with the same westernmost starting section, print the one with the westernmost ending section.

Example Input

```
2
BBRRBRRBRB
BBRBBRRB
```

Example Output to Screen

```
3 7
1 5
```

This page intentionally left blank

10. Star Arrangements

The recent vote in Puerto Rico favoring United States statehood has made flag makers very excited. An updated flag with 51 stars rather than the current one with 50 would cause a huge jump in U.S. flag sales. The current pattern for 50 stars is five rows of 6 stars, interlaced with four offset rows of 5 stars. The rows alternate until all stars are represented.

```
* * * * * *
 * * * * *
* * * * * *
 * * * * *
* * * * * *
 * * * * *
* * * * * *
 * * * * *
* * * * * *
```

This pattern has the property that adjacent rows differ by no more than one star. We represent this star arrangement compactly by the number of stars in the first two rows: 6,5.

A 51-star flag that has the same property can have three rows of 9 stars, interlaced with three rows of 8 stars (with a compact representation of 9,8). Conversely, if a state were to leave the union, one appealing representation would be seven rows of seven stars (7,7).

A flag pattern is visually appealing if it satisfies the following conditions:

- Every other row has the same number of stars.
- Adjacent rows differ by no more than one star.
- The first row cannot have fewer stars than the second row.

Your team sees beyond the short-term change to 51 for the U.S. flag. You want to corner the market on flags for any union of three or more states. Given the number S of stars to draw on a flag, find all possible visually appealing flag patterns.

Input

The first line will contain a single integer n that indicates the number of data sets that follow. Each data set will consists of a single line containing the integer S (3 <= S <= 32,767).

Output

For each flag, on the first line, print S, followed by a colon. Then, for each visually appealing flag of S stars, print its compact representation, one per line.

This list of compact representations should be printed in increasing order of the number of stars in the first row; if there are ties, print them in order of the number of stars in the second row. The cases 1-by-S and S-by-1 are trivial, so do not print those arrangements.

The compact representations must be printed in the form “x,y”, with exactly one comma between x and y and no other characters.

(continued on the next page)

Example Input

```
3
3
50
51
```

Example Output to Screen

```
3:
2,1
50:
2,1
2,2
3,2
5,4
5,5
6,5
10,10
13,12
17,16
25,25
51:
2,1
3,3
9,8
17,17
26,25
```

11. Halfway

A friend of yours has written a program that compares every pair of a list of items. With n items, it works as follows. First, it prints a 1, and it compares item 1 to items 2, 3, 4, . . , n. It then prints a 2, and compares item 2 to items 3, 4, 5, . . . , n. It continues like that until every pair of items has been compared exactly once. If it compares item x to item y, it will not later compare item y to item x. Also, it does not compare any item to itself.

Your friend wants to know when his program is halfway done. For a program that makes an odd number of total comparisons, this is when it is doing the middle comparison. For a program that makes an even number of total comparisons, this is when it is doing the first of the two middle comparisons.

What will the last number printed be when the program is halfway done?

Note that since the earlier items have more comparisons than the later items, the answer is not simply n=2.

Input
The first line will contain a single integer n that indicates the number of data sets that follow. Each data set consists of a single line containing the integer n ($2 <= n <= 10^9$).

Output
For each data set, print on a single line, the last number your friend's program prints when it is halfway done.

Example Input:
```
5
4
7
10
1919
290976843
```

Output to screen:
```
1
2
3
562
85225144
```

This page intentionally left blank

12. Complexity

Define the *complexity* of a string to be the number of distinct letters in it. For example, the string `string` has complexity 6 and the string `letter` has complexity 4. You like strings which have complexity either 1 or 2. Your friend has given you a string and you want to turn it into a string that you like. You have a magic eraser which will delete one letter from any string. Compute the minimum number of times you will need to use the eraser to turn the string into a string with complexity at most 2.

Input
The first input will be a single integer indicating how many data sets are to follow. Each data set consists of a single line that contains a single string of at most 100 lowercase ASCII letters ('a'–'z').

Output

Print, on a single line, the minimum number of times you need to use the eraser.

Example Input
```
7
string
letter
aaaaaa
uncopyrightable
ambidextrously
assesses
assassins
```

Output to screen:
```
4
2
0
13
12
1
2
```

This page intentionally left blank

13. Forbidden Zero

You're writing the positive integers in increasing order starting from one. But you've never learned the digit zero, and thus omit any number that contains a zero in any position. The first ten integers you write are: 1, 2, 3, 4, 5, 6, 7, 8, 9, and 11.

You have just written down the integer n (which is guaranteed to not contain the digit zero). What will be the next integer that you write down?

Input
The first input will be a single integer indicating how many data sets are to follow. Each data set consists of a single line containing the integer n (1 <= n <= 999,999).
It is guaranteed that n does not contain the digit zero.

Output
For each data set, print on a single line, the next integer you will be writing down.

Example Input:
```
2
99
1234
```

Output to screen:
```
111
1235
```

This page intentionally left blank