Kattis

Some More Introductions

2025-03-07

Problems

A. Luhn's Checksum Algorithm
B. Seven Wonders
C. The Mailbox Manufacturers Problem
D. Parsing Hex
E. Cent Savings
F. IsItHalloween.com
G. Fridge

Advice, hints, and general information

- Your solution programs should read input from standard input (e.g. System.in in Java or std::cin in C++) and produce output on standard output (e.g. System.out in Java or std::cout in C++). Anything written on standard error will be ignored. For further details and examples, please refer to the documentation in the help pages for your favorite language on Kattis.
- If you think some problem is ambiguous or underspecified, you may ask the judges for a clarification request through the Kattis system. The most likely response is “No comment, read problem statement”, indicating that the answer can be deduced by carefully reading the problem statement or by checking the sample test cases given in the problem.

Luhn's Checksum Algorithm

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

In 1954, Hans Peter Luhn, a researcher at IBM, filed a patent describing a simple checksum algorithm for numbers written as strings of base-10 digits. If a number is chosen according to Luhn's technique, the algorithm provides a basic integrity check. This means that with reasonably high probability it can detect whether one or more digits have been accidentally modified. (On the other hand, it provides essentially no protection against intentional modifications.) Most credit card and bank card numbers can be validated using Luhn's checksum algorithm, as can the national identification numbers of several countries (including Canada).

Image by simpson33 (iStock), Used unc licer

Given a number $n = d_k d_{k-1} \ldots d_2 d_1$, where each d_i is a base-10 digit, here is how to app Luhn's checksum test:

1. Starting at the *right* end of n, transform every *second* digit d_i (i.e., $d_2, d_4, d_6, \ldots$) a follows:
    1. multiply d_i by 2
    2. if $2 \cdot d_i$ consists of more than one digit, i.e., is greater than 9, add these digi together; this will always produce a single-digit number
2. Add up all the digits of n after the transformation step. If the resulting sum is divisib by 10, n *passes* the Luhn checksum test. Otherwise, n *fails* the Luhn checksum test.

For example, consider the number $n = 1234567890123411$ from Sample Input 1. The fir row of Figure 1 gives the original digits of n, and the second row contains the digits of n aft

the transformation step, with transformed digits shown in bold. The sum of the digits in th second row is

$$2+2+6+4+1+6+5+8+9+0+2+2+6+4+2+1=60$$

and since 60 is divisible by 10, n passes the Luhn checksum test.

1	2	3	4	5	6	7	8	9	0	1	2	3	4	1	1
2	2	**6**	4	**1**	6	**5**	8	**9**	0	**2**	2	**6**	4	**2**	1

Figure 1: Application of Luhn's algorithm to $n = 1234567890123411$

Input

The first line of input contains a single integer T $(1 \leq T \leq 100)$, the number of test case Each of the following T lines contains a single test case consisting of a number given as string of base-10 digits (`0`–`9`). The length of each string is between 2 and 50, inclusive, ar numbers may have leading (leftmost) zeros.

Output

For each test case, output a single line containing "`PASS`" if the number passes the Luh checksum test, or "`FAIL`" if the number fails the Luhn checksum test.

Sample Input 1

```
3
00554
999
1234567890123411
```

Sample Output 1

```
PASS
FAIL
PASS
```

Seven Wonders

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

Seven Wonders is a card drafting game in which players build structures to earn points. The player who ends with the most points wins. One winning strategy is to focus on building scientific structures. There are three types of scientific structure cards: Tablet (‘T’), Compass (‘C’), and Gear (‘G’). For each type of cards, a player earns a number of points that is equal to the squared number of that type of cards played. Additionally, for each set of three different scientific cards, a player scores 7 points.
For example, if a player plays 3 Tablet cards, 2 Compass cards and 1 Gear card, she gets $3^2 + 2^2 + 1^2 + 7 = 21$ points.

It might be tedious to calculate how many scientific points a player gets by the end of each game. Therefore, you are here to help write a program for the calculation to save everyone’s time.

Input

The input has a single string with no more than 50 characters. The string contains only letters ‘T’, ‘C’ or ‘G’, which denote the scientific cards a player has played in a Seven Wonders game.

Output

Output the number of scientific points the player earns.

Note

Seven Wonders was created by Antoine Bauza, and published by Repos Production. Antoine Bauza and Repos Production do not endorse and have no involvement with the ProgNova contest.

Sample Input 1

```
TCGTTC
```

Sample Output 1

```
21
```

Sample Input 2

```
CCC
```

Sample Output 2

```
9
```

Sample Input 3

```
TTCCGG
```

Sample Output 3

```
26
```

The Mailbox Manufacturers Problem

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

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. Now, 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 $1 + 2 + 3 + \ldots + m = \frac{m(m+1)}{2}$ 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 fewer fire 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 $x - 1$ fire-crackers.

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: 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

The input starts with a single integer N ($1 \leq N \leq 100$) 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.

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.

Sample Input 1

```
4
1 10
1 100
3 73
5 100
```

Sample Output 1

```
55
5050
382
495
```

Parsing Hex

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

This problem is simple. Just search the input for hexadecimal numbers, and print any numbers you find in both hexadecimal and decimal format.

Input

Input is a sequence of at most 100 text lines, ending at end of file. Each line has at most 100 characters, and may contain one or more hexadecimal numbers. No hexadecimal number spans multiple lines. A hexidecimal number begins with '`0x`' or '`0X`' (that's a number zero followed by a letter x), followed by a string of hexadecimal digits (0–9, a–f, or A–F). A hexadecimal number should be as long as possible, but no hexadecimal number in the input is greater than `0xffffffff`. No two hexadecimal numbers are adjacent on any line.

Output

For each hexadecimal number (in the order they appear in the input), print a line containing the hexadecimal number as it appeared in the input and its non-negative decimal equivalent.

Sample Input 1

```
uyzrr0x5206aBCtrrwm0Xa8aD4poqwqr
pqovx0x6d3e6-+ 230xB6fcgmmm
```

Sample Output 1

```
0x5206aBC 86010556
0Xa8aD4 690900
0x6d3e6 447462
0xB6fc 46844
```

Cent Savings

CPU TIME LIMIT	MEMORY LIMIT
3 seconds	1024 MB

Picture by Tijmen Stam via Wikimedia Commons, cc by-sa

To host a regional contest like NWERC a lot of preparation is necessary: organizing rooms and computers, making a good problem set, inviting contestants, designing T-shirts, booking hotel rooms and so on. I am responsible for going shopping in the supermarket.
When I get to the cash register, I put all my n items on the conveyor belt and wait until all the other customers in the queue in front of me are served. While waiting, I realize that this supermarket recently started to round the total price of a purchase to the nearest multiple of 10 cents (with 5 cents being rounded upwards). For example, 94 cents are rounded to 90 cents, while 95 are rounded to 100.

It is possible to divide my purchase into groups and to pay for the parts separately. I managed to find d dividers to divide my purchase in up to $d + 1$ groups. I wonder where to place the dividers to minimize the total cost of my purchase. As I am running out of time, I do not want to rearrange items on the belt.

Input

The input consists of:

- one line with two integers n ($1 \leq n \leq 2\,000$) and d ($1 \leq d \leq 20$), the number of items and the number of available dividers;

- one line with n integers $p_1, \ldots p_n$ ($1 \leq p_i \leq 10\,000$ for $1 \leq i \leq n$), the prices of the items in cents. The prices are given in the same order as the items appear on the belt.

Output

Output the minimum amount of money needed to buy all the items, using up to d dividers.

Sample Input 1

```
5 1
13 21 55 60 42
```

Sample Output 1

```
190
```

Sample Input 2

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

Sample Output 2

```
0
```

IsItHalloween.com

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

Happy Halloween! Author: Petar Milošević, cc by-sa

HiQ recently got an assignment from a client to create a clone of the immensely popular website https://IsItHalloween.com. The website is a very simple one. People will visit the site occasionally to see if it is Halloween. Whenever it is, the website should print out `yup`, otherwise it should print out `nope` on the screen.

Since HiQ is such a popular firm, they don't have time to complete this assignment right now. Their frontend engineers have already programmed the frontend of the website that prints out `yup` or `nope`, but not the backend microservice that determines whether it is indeed Halloween or not. Do you have time to help them?

The behaviour of the server should be as follows: it gets as input the current date in the format `FEB 9`, where `FEB` is the month given in three letters (`JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC`) and 9 is the day of the month starting at 1. It should then determine if this date represents October 31 or December 25 (since $31_8 = 25_{10}$).

Input

The input consists of a single line containing a date of the format `FEB 9`, with the month and date separated by a single space.

Output

If the date is October 31 or December 25, output `yup`. Otherwise, output `nope`.

Sample Input 1

```
OCT 31
```

Sample Output 1

```
yup
```

Sample Input 2

```
JUN 24
```

Sample Output 2

```
nope
```

Fridge

CPU TIME LIMIT	MEMORY LIMIT
1 second	1024 MB

The technology behind the fridge has changed little over the years. Even so, many of the original owners of the Fred W. Wolf domestic refrigerator of 1913 would be amazed by the size and features of the modern appliances. However, since the 1960s one thing has been common for all fridge owners around the world: fridge magnets.

An effective, albeit lazy, way to keep a small child entertained is to supply them with a set of magnetic numbers and a large magnetic surface, such as said fridge, to provide the playing field upon which to apply these digits.

Far from a time-wasting exercise, this provides valuable training in the mathematical field of *counting*: moving the digits around to form “1”, “2”, and so on up to such heights as “10”, “11”, “12”, and even beyond.

The possibilities are endless! ...Or at least, they would be, if the supply of digits was not limited. Given the full list of what numbers we are in possession of, what is the smallest positive number that *cannot* be made using each of digits at most once?

Input

- One string of at most 1000 digits, containing the available digits in no particular order.

Output

- One line containing one positive integer: the smallest natural number that it is not possible to assemble from the supplied digits.

Sample Input 1

```
7129045863
```

Sample Output 1

```
11
```

Sample Input 2

```
55
```

Sample Output 2

```
1
```

Sample Input 3

```
123456789
```

Sample Output 3

```
10
```