BCN • LEO • MAD • VAL

#HPCodeWars2025
Barcelona

Problems
and solutions

#	Problem	Points
1	Age-o-matic	1
2	Word By Word Printer	3
3	Drone Navigation	3
4	HP CodeWars BCN 10th Edition	3
5	Captain Tsubasa	4
6	Rounding Grades	5
7	Loan Monthly Payment	5
8	Newton Raphson Square Root	6
9	Double Double	6
10	Pangram	6
11	Nostromo's Armstrong Number Detector	6
12	SOS Morse	7
13	Automatic Text Corrector	7
14	Shifting Initial Letter	7
15	Public-Private Key Cryptosystem	8
16	String Slimmer	8
17	Gravitational Solver	9
18	Quest For The Nth Day	10
19	Coding Id	11
20	Olympic Medal Table	11
21	Hourglass	11
22	Bracket Notation	13
23	Enchanted Vines	13
24	Ecosystem Simulator	13
25	Stem And Leaf	14
26	Hanged Man (Interactive)	14
27	The Lost Treasure Map	17
28	Magic Square Makeover	18
29	Mastering The Matrix Determinant	19
30	Four Up	21
31	Randomized Factorio Run	27
32	Summon The Moon Lord (Interactive)	30

1 Age-o-matic

1 points

Introduction

As science assignment in high school, you are creating a new machine called Age-o-Matic. It can take a person's age in years and convert it into the number of days they've been alive! Can you write a program that helps the Age-o-Matic do its job? To make things easier do not consider leap years.

Input

The input consists of a positive integer representing the age of a person in years.

Output

The output should be a positive integer representing how many days the person has been alive.

Example

Input

```
10
```

Output

```
3650
```

Python

```python
ageInYears = int(input())
ageInDays = ageInYears * 365
print(ageInDays)
```

2 Word By Word Printer

3 points

Introduction

You have been gifted an old-school printer that can only print one word per line. With such a printer, to print a given sentence composed of several words, the sentence must be split word by word. Can you write a program to split a sentence into several words, allowing you to use the printer?

Input

The input consists of a single line containing a sentence. The sentence may contain letters, digits, spaces, and punctuation marks.

Output

Print each word of the sentence on separate lines, as if the printer can only handle one word at a time.

Example

Input

```
Hello, world! This is a sample sentence with 1 line.
```

Output

```
Hello,
world!
This
is
a
sample
sentence
with
1
line.
```

Python

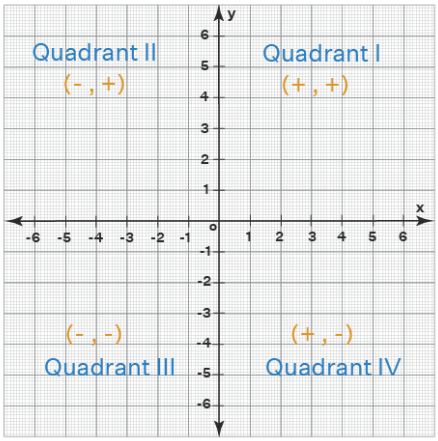
```python
sentence = input()
for i in sentence.split():
    print(i)
```

3 Drone Navigation

3 points

Introduction

In drone navigation, it's crucial to determine the quadrant in which a drone is located based on its coordinates. The airspace is divided into four quadrants, numbered from 1 to 4, as shown in the diagram below:



For example, if a drone is at coordinates (12, 5), it lies in quadrant 1 since both its *x* and *y* values are positive. Conversely, if a drone is at coordinates (-7, 8), it lies in quadrant 2 since its *x* value is negative and its *y* value is positive. Your task is to determine the quadrant in which a drone is located based on its coordinates. You can assume that neither of the two coordinates will be 0.

Input

The first line of input contains the integer x, representing the *x*-coordinate of the drone. The second line of input contains the integer y, representing the *y*-coordinate of the drone.

Output

Output the quadrant number (1, 2, 3, or 4) for the drone's coordinates (*x*, *y*).

Example 1

Input

```
12
5
```

Output

```
1
```

Example 2

Input

```
-7
8
```

Output

```
2
```

Python

```python
# Read the coordinates from the input
x = int(input())
y = int(input())

# Determine the quadrant of the point
if x > 0 and y > 0:
    print(1)
elif x < 0 and y > 0:
    print(2)
elif x < 0 and y < 0:
    print(3)
else:
    print(4)
```

4 HP CodeWars BCN 10th Edition

3 points

Introduction

The year 2025 marks the 10th edition of the HP CodeWars Barcelona. The competition began in 2015 and has been held annually, except for the year 2020, when it was suspended due to the COVID-19 pandemic. Since one year was skipped, determining the correct edition number for each year can be confusing. Can you help the organizers by writing a program to compute the edition of a given year?

Input

A positive integer representing a given year.

Output

The program should output one of the following based on the year:

- A positive integer representing the edition number if the competition took place that year.
- " Coming Soon" if the competition had not yet started in that year (i.e. after, 2025).
- "CANCELLED" if the competition was canceled in that year (i.e., 2020).
- "Did not exist" if the competition did not exist in that year (i.e., before 2015).

Example 1

Input

```
2025
```

Output

```
10
```

Example 2

Input

```
2020
```

Output

```
CANCELLED
```

Example 3

Input

```
2030
```

Output

```
Coming Soon
```

Example 4

Input

```
2000
```

Output

```
Did not exist
```

Python

```python
year = int(input())
if year > 2025:
    print("Coming soon")
elif year<2015:
    print("Did not exist")
elif year == 2020:
    print("CANCELLED")
elif year>=2015 and year<2020:
    print(year-2014)
else:
    print(year-2015)
```

5 Captain Tsubasa

4 points

Introduction

In a Japanese small town, there are two young soccer players, Oliver Atom and Mark Lenders, who are known for their incredible goal-scoring abilities. Every month, they compete to see who can score the most goals. Their coach keeps track of the number of goals each player scores every month. Now, the coach wants to know how many months Oliver scored more goals than Mark. Can you help the coach find out?

Input

The first line contains a list of integers representing the number of goals Oliver scored each month over the course of a year.

The second line contains a list of integers representing the number of goals Mark scored each month over the course of a year.

Output

An integer representing the number of months Oliver scored more goals than Mark.

Example

Input

```
3 7 1 0 5 6 12 0 0 1 2 5
1 2 3 4 5 6 7 8 9 10 11 12
```

Output

```
3
```

Python

```python
oliver = input().split()
mark = input().split()
months = 0
for i in range(len(oliver)):
    if int(oliver[i]) > int(mark[i]):
        months += 1
print(months)
```

6 Rounding Grades

5 points

Introduction

Alex has just received their semester grades. Their teacher has asked them to round these grades to a specified number of decimal places for the final report. To make this task easier, Alex decides to write a program that will do the rounding for them.

Input

The input consists of a line with grades separated by spaces and another line containing an integer that specifies the precision.

Output

The output should be a single line with the rounded grades separated by spaces.

Example 1

Input

```
8.5678 9.2364 7.6789
1
```

Output

```
8.6 9.2 7.7
```

Example 2

Input

```
8.5678 9.2364 7.6789
3
```

Output

```
8.568 9.236 7.679
```

Python

```python
grades = input().split()
grades = [float(x) for x in grades]
precision = int(input())
rounded_grades = [round(grade, precision) for grade in grades]
print(" ".join(map(str, rounded_grades)))
```

7 Loan monthly payment

5 points

Introduction

Aitana wants to buy a house. It costs 300000 €. She does not have all the money needed so she has asked for a loan (L) from the bank for 80% of the cost, that is 240000 €. Aitana will have to make a monthly payment (P) to the bank until she returns all the money.

Of course, the loan is not for free. The bank will also charge Aitana each month with a percentage of the remaining loan. This is called an interest rate. The yearly interest rate percentage (Y) will be 4, but the interest will be charged each month, so you will have to calculate the monthly interest rate (r) with the following formula:

$$r = \frac{Y}{100 \times 12}$$

Aitana wants to calculate the monthly payment (P) to check if she will be able to pay it back with her salary or not. The bank allows her to choose in how many months (m) she wants to pay the loan back. If she chooses a long period, the monthly payment will be lower (good), but she will pay more interests in total (bad). She must try different periods to find the optimum balance between paying less each month and paying less interest.

Help Aitana by writing a program that calculates the monthly payment (P) depending on the number of months of the loan (m). Use the following formula:

$$P = \frac{L * r}{1 - (1 + r)^{-m}}$$

Input

The number of months of the loan (m). This will be an integer number greater than zero.

Output

The output should be the monthly payment (P) printed out with 2 decimals rounded to the nearest hundredth.

Example 1

Input

```
300
```

Output

```
1266.81
```

Example 2

Input

```
1
```

Output

```
240800.00
```

Python

```python
import math

# Constants
loan_amount = 240000.0  # Loan amount
yearly_interest_rate = 4.0  # Yearly interest rate in percentage
monthly_interest_rate = yearly_interest_rate / (100 * 12)

# Get the number of months
months = int(input())

# Calculate monthly payment using the annuity formula
monthly_payment = loan_amount * monthly_interest_rate / (1 - math.pow(1 +
monthly_interest_rate, -months))

# Output the result with 2 decimal places
print(f"{monthly_payment:.2f}")
```

8 Newton-Raphson Square Root

6 points

Introduction

Jimmy is a curious and enthusiastic boy who is always eager to learn new things. One day, in his math class, his teacher asked him to solve the square root of a number. Jimmy tried using his calculator, but he had no battery left. Frustrated, he decided to ask his friend, the programmer, for help.

The programmer explained to Jimmy that there is a way to calculate the square root without a calculator, using a method called Newton-Raphson. This method is a bit more advanced, but with a little patience and practice, Jimmy could learn to use it and solve square roots on his own.

The programmer taught Jimmy that, to solve the problem, he could use this iterative formula:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{x_o}{x_n}\right)$$

Where n refers to the number of iterations, so x_{n+1} and x_n represent respectively the value of x to be found at step $n+1$ and its value at step n. Finally x_0 is the number you want to find the square root of.

Input

The input consists of a positive integer representing the number to find the square root of, and another positive integer representing the number of iterations for the Newton-Raphson formula.

Output

The output should be the approximation rounded of the square root of the number with six decimals.

Example

Input

```
2
3
```

Output

```
1.414216
```

Python

```python
num = float(input())
iterations = int(input())
estimation = num

for _ in range(iterations):
    next = 0.5 * (estimation + num / estimation)
    estimation = next

print(f"{estimation:.6f}")
```

9 Double-double

6 points

Introduction

In basketball, a double-double is a performance in which a player accumulates a double-digit total in two of five statistical categories — points, rebounds, assists, steals, and blocked shots — in a game. Given the player's statistics, write a program to check if the player has achieved a double-double.

Input

The input consists of five lines where each line corresponds to an statistic:

- points (integer): The number of points scored by the player.
- rebounds (integer): The number of rebounds made by the player.
- assists (integer): The number of assists made by the player.
- steals (integer): The number of steals made by the player.
- blocks (integer): The number of blocks made by the player.

Output

The output is a boolean value (True or False) indicating whether the player has achieved a double-double. If the value is True, also print the first two statistics that are double-digit in the order they appear.

Example 1

Input

```
2
1
0
10
5
```

Output

```
False
```

Example 2

Input

```
20
10
7
9
8
```

Output

```
True
points
rebounds
```

Example 3

Input

```
9
12
8
11
10
```

Output

```
True
rebounds
steals
```

Python

```python
# Read the statistics from the input
points = int(input())
rebounds = int(input())
assists = int(input())
steals = int(input())
blocks = int(input())

# Store the statistics in a list of tuples
categories = [("points", points), ("rebounds", rebounds), ("assists", assists),
("steals", steals), ("blocks", blocks)]

# Initialize variables
count = 0
first_two_keys = []

# Check if the player has at least two categories with 10 or more
for category, value in categories:
    if value >= 10:
        count += 1
        if count <= 2:
            first_two_keys.append(category)

# Output the result
if count >= 2:
    print("True")
    print(first_two_keys[0])
    print(first_two_keys[1])
else:
    print("False")
```

10 Oh! I found a pangram

6 points

Introduction

Did you hear about the alphabet's party? They invited all the letters, but the 'lazy programmer' sent his regrets because he couldn't find a pangram! A pangram is a sentence, phrase, or piece of text that contains every letter of the alphabet at least once. Your challenge is to help him out and write a program that checks if a sentence contains all the letters of the alphabet. Don't leave any letter behind!

Input

A single line containing the sentence to be analyzed.

Output

A single line stating if a pangram was found or not.

Example 1

Input

```
The quick brown fox jumps over the lazy dog.
```

Output

```
Pangram found
```

Example 2

Input

```
The quick brown fox jumps over the lazy cat.
```

Output

```
Pangram not found
```

Python

```python
text = input()

text = text.lower()

alphabet= "abcdefghijklmnopqrstuvwxyz"

for i in text:
    alphabet =  alphabet.replace(i,"")

if len(alphabet) == 0:
    print ("Pangram found")
else:
    print ("Pangram not found")
```

11 Nostromo's Armstrong Number Detector

6 points

Introduction

In the year 2122, the commercial towing spaceship Nostromo is on its return trip to Earth. The ship's onboard computer, MU-TH-UR 6000, has detected an unknown signal from a nearby planetoid. The crew is awakened from stasis to investigate. As they prepare to land on the planetoid, MU-TH-UR runs a diagnostic to ensure all systems are operational. One of the checks involves validating the integrity of the ship's power cells, which are labeled with unique numerical codes.

Your task is to help MU-TH-UR create an Armstrong Number Detector. An Armstrong number is a number that is the sum of its own digits each raised to the power of the number of digits. For example, 153 is an Armstrong number because ($1^3 + 5^3 + 3^3 = 153$).

Input

A single positive integer number.

Output

Return True if n is an Armstrong number, or False otherwise.

Example 1

Input

```
371
```

Output

```
True
```

Example 2

Input

```
100
```

Output

```
False
```

Python

```python
def no_of_digits(num):
    i = 0
    while num > 0:
        num //= 10
        i+=1

    return i

def required_sum(num):
    i = no_of_digits(num)
    s = 0

    while num > 0:
        digit = num % 10
        num //= 10
        s += pow(digit, i)

    return s

num = int(input())
s = required_sum(num)

if s == num:
    print("True")
else:
    print("False")
```

12 SOS Morse

7 points

Introduction

Imagine you are stranded on a deserted island with access to rudimentary technology for sending messages in Morse code. You cannot transmit a voice or text message, but you can send Morse sequences using a device that emits and receives intermittent signals. In an emergency, you want to send the universal distress signal, SOS (...---...), in Morse code to increase your chances of being rescued. Sending the pattern ...---... at least three times will allow a nearby emergency boat to detect your presence. Sending SOSOS (...---...---...) will count as two times sent, saving the transmission of an S.

Input

The input consists of a line with dots and dashes.

Output

The output will print "Saved" if the SOS Morse pattern appears 3 times or more in the message. Otherwise, the output will be "Not saved".

Example 1

Input

```
...---...-.--.--.....-.....---...
```

Output

```
Saved
```

Example 2

Input

```
..---...-.---.....---.....---...
```

Output

```
Not saved
```

Python

```python
data = input()
count = 0
index = 0

while True:
    index = data.find("...---...", index)

    # Verify if there are more "...---..." matches
    if index == -1:
        print("Not saved")
        break

    count += 1
    if count == 3:
        print("Saved")
        break

    # Move the search index to the next possible "..."
    index += 6
```

13 Automatic Text Corrector

7 points

Introduction

Paige is a professional editor working on an urgent project, but her word processor does not automatically correct common formatting issues, such as double spaces or lowercase letters following periods. To address this, she has asked you to create a program that quickly resolves these issues:

- Replace all multiple spaces with a single space.
- Ensure that each sentence starts with a capital letter by adding a space after periods (if missing) and capitalizing the first letter of the next word.

Input

The input consists of a single line of text.

Output

The output should be the modified line of text with the following corrections applied:

- The first letter is capitalized.
- Each sentence begins with a capital letter after a period exclusively.
- Multiple spaces are reduced to a single space.
- A space follows each period, if missing.

Example

Input

```
this is a great competition.have a    nice   day!
```

Output

```
This is a great competition. Have a nice day!
```

Python

```python
# The program should not correct accents nor spaces before commas to avoid using
libraries
def remove_double_spaces(text):
    # Replace all double spaces with a single space
    while "  " in text:
        text = text.replace("  ", " ")
    return text
def fix_capital_letters(text):
    # Transform first letter into capital
    text = text.capitalize()
    index = 0

    while True:
        # Search for next point in the text
        index = text.find('.', index)

        # End if no more points
        if index == -1:
            break

        # We check if the period is followed by a letter without a space
        if index + 1 < len(text) and text[index + 1].isalpha():
            # We add a space after the period and convert the letter to
uppercase
            text = text[:index + 1] + ' ' + text[index + 1].upper() + text[index
+ 2:]
        elif index + 2 < len(text) and text[index + 1] == ' ' and text[index +
2].isalpha():
            # We convert the letter after the space to uppercase if it already
exists
            text = text[:index + 2] + text[index + 2].upper() + text[index + 3:]

        index += 1

    return text

text = input()
text = remove_double_spaces(text)
text = fix_capital_letters(text)
print(text)
```

14 Shifting Initial Letter

7 points

Introduction

The letters decided to have some fun by playing a similar game to musical chairs. They wanted to see what would happen if they swapped its initial letter with their neighbors. They need your help to achieve it. Given a sentence, create a program that shifts the initial letter of each word to the beginning of the next word in the sentence (shifting right). The last word shifts its initial letter to the first word in the sentence.

Input

A single line containing a sentence formed by one or more words.

Output

The resulting sentence of shifting the initial letter of each word to the next word in the sentence (shifting right).

Example 1

Input

```
Hello world!
```

Output

```
wello Horld!
```

Example 2

Input

```
Single
```

Output

```
Single
```

Example 3

Input

```
this is a test about simple shifting of the initial letter
```

Output

```
lhis ts i aest tbout aimple shifting sf ohe tnitial ietter
```

Python

```python
# Read the sentence
sentence = input()

# Split the sentence into words
words = sentence.split()

# If there is only one word, print it
if len(words) == 1:
    print(*words)
else:
    # Initialize a list to store the shifted words
    shifted_words = []
    first_letters = []

    # Iterate through each word to collect the first letter
    for word in words:
        # Get the first letter of the word
        first_letters.append(word[0])

    # First word replaces its first letter with the first letter of the last
word
    shifted_words.append(first_letters[-1] + words[0][1:])

    # Iterate through rest of words to shift the first letter
    cnt = 0
    for word in words[1:]:
        shifted_words.append(first_letters[cnt] + word[1:])
        cnt += 1

    print(*shifted_words)
```

15 Public-Private Key Cryptosystem

8 points

Introduction

The RSA (Rivest–Shamir–Adleman) is a cryptosystem widely used for secure data transmission. This system uses a public-private key pair, created by a pair of prime numbers. The messages can be encrypted by anyone who knows the public key, but only decrypted by someone who knows both prime numbers. The security of this method relies on the difficulty to produce two large prime numbers. So, for generating this large prime numbers, we can use a computer. Can you code a program to generate the first N prime numbers to be used later in a public-private key cryptosystem?

Input

A number greater than zero, N.

Output

Print out the first N^{th} prime numbers having only one prime number per line.

Example

Input

5

Output

2

3

5

7

11

Python

```python
# Public-Private Key Cryptosystem
n = int(input())
primes = []
p = 2
while len(primes) < n:
    is_prime = True
    for i in primes:
        if p % i == 0:
            is_prime = False
            break
    if is_prime:
        primes.append(p)
        print(p)
    p += 1
```

C++

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int input;
    cin >> input;
    bool is_prime;
    vector<int> primes;
    int num = 2;
    while (primes.size() < input) {
        is_prime = true;
        for (int i = 0; i < primes.size(); i++) {
            if (num % primes[i] == 0) {
                is_prime = false;
                break;
            }
        }
        if (is_prime) {
            //If the number is prime, we add it to the vector and print it.
            primes.push_back(num);
            cout << num << endl;
        }
        num += 1;
    }
    return 0;
}
```

16 String slimmer

8 points

Introduction

One day, while experimenting with an alphabet soup, Ada discovered that certain letters had a peculiar property: when two identical letters stood next to each other, they would vanish into thin air! Intrigued by this phenomenon, Ada asked for your help to simulate slimming strings of letters.

Given a string consisting of lowercase letters from 'a' to 'z', perform a series of operations to reduce the string. In each operation, select a pair of adjacent matching letters and remove them. Continue this process to delete as many characters as possible and return the resulting string. If the string becomes empty, print "Empty String". Can you write down a program to do it?

Input

A single string with lowercase letters

Output

The reduced string or "Empty String"

Example 1

Input

```
aa
```

Output

```
Empty String
```

Example 2

Input

```
abb
```

Output

```
a
```

Example 3

Input

```
helloabba
```

Output

```
heo
```

Python

```python
word = input()
slim = True
while slim:
    slim = False
    for i in range(len(word) - 1):
        if word[i] == word[i + 1]:
            word = word[:i] + word[i + 2:]
            slim = True
            break
if word != "":
    print(word)
else:
    print("Empty String")
```

17 Gravitational Solver

9 points

Introduction

Jimmy is learning how to solve gravitational physics problems for his upcoming exam, but he is unsure if his answers are correct. All the gravitational problems that Jimmy is trying to solve are based on the following formula:

$$\frac{GMm}{r^2} = \frac{mV^2}{r}$$

Where the variables have the following meaning:

- G is the constant of universal gravitation which is equal to $6.67 \cdot 10^{-11}\ Nm^2/kg^2$.
- M is the mass of the main planet and m is the mass of the satellite that orbits around that planet.
- r is the sum of the radius of the main planet (R_m) and the distance from the surface of the planet (h), so r is $R_m + h$.
- V is the velocity of the satellite orbiting.

The problems involve finding the unknown value of one of the variables: V, M, R_m or h, given the values of the other variables.

Input

The first value corresponds to the mass of the main planet (M), and it is a positive float.

The second value corresponds to the radius of the main planet (R_m), and it is a positive float.

The third value corresponds to the distance from the surface of the main planet to the satellite (h), and it is a positive float.

The fourth value corresponds to the velocity (V) of the satellite, and it is a positive float.

One of these values will be unknown and represented as #.

Output

The resulting value of the unknown parameter.

Example 1

Input

```
600000000000000000000000
42200000
1
#
```

Output

```
307.95149127012644
```

Example 2

Input

```
600000000000000000000000
#
1
307.95149127012644
```

Output

```
42200000.0
```

Python

```python
def solve_gravitational_problem(G, M, Rm, h, V):
    if V == '#':
        r = Rm + h
        V = (G * M / r) ** 0.5
        print(""+str(V))
    elif M == '#':
        r = Rm + h
        M = V**2 * r / G
        print(""+str(M))
    elif Rm == '#':
        r = (G * M)/V**2
        Rm = r - h
        print(""+ str(Rm))
    elif h == '#':
        r = (G * M)/V**2
        h = r - Rm
        print(""+str(h))
    else:
        print("All values provided")


g = 0.0000000000667
m = input()
rm = input()
h = input()
v = input()

if m != '#':
    m = float(m)
if rm != '#':
    rm = float(rm)
if h != '#':
    h = float(h)
if v != '#':
    v = float(v)

solve_gravitational_problem(g, m, rm, h, v)
```

18 Quest For the Nth Day

10 points

Introduction

In a quirky town filled with eccentric characters, there was a bumbling detective named Frank Drebin. One day, Frank stumbled upon a mysterious case involving a planned robbery. The only clue was a cryptic note stating that the robbery would occur on the nth day of a given year.

Determined to solve the case, Frank unsuccessfully tried to find out the specific date from the note. Can you help detective Drebin by coding a program that prints out the exact date?

Input

The input consists of two lines:

- The first line contains an integer representing the nth day of the year ($1 \leq day \leq 366$).
- The second line has an integer representing the year ($1 \leq year \leq 9999$).

Output

Print the date in the format: month and date.

Example 1

Input

```
105
2024
```

Output

```
April 14
```

Example 2

Input

```
365
1999
```

Output

```
December 31
```

Example 3

Input

```
104
2023
```

Output

```
April 14
```

Example 4

Input

```
32
2023
```

Output

```
February 1
```

Python

```python
MONTH_DAYS = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False

def day_of_year_to_date(day_of_year, year):
    if is_leap_year(year):
        MONTH_DAYS[1] = 29
    else:
        MONTH_DAYS[1] = 28

    month = 0
    while day_of_year > MONTH_DAYS[month]:
        day_of_year -= MONTH_DAYS[month]
        month += 1

    return month + 1, day_of_year

MONTH_NAMES = ["January", "February", "March", "April", "May", "June",
               "July", "August", "September", "October", "November", "December"]

day_of_year = int(input())
year = int(input())

month, day = day_of_year_to_date(day_of_year, year)

print(MONTH_NAMES[month - 1], day)
```

C++

```cpp
#include <iostream>
#include <iomanip>
#include <ctime>

std::tm day_of_year_to_date(int day_of_year, int year) {
    std::tm first_day_of_year = {};
    first_day_of_year.tm_year = year - 1900; // tm_year is years since 1900
    first_day_of_year.tm_mon = 0; // January
    first_day_of_year.tm_mday = 1; // First day of the month

    std::time_t first_day_time = std::mktime(&first_day_of_year);
    std::time_t target_time = first_day_time + (day_of_year - 1) * 24 * 60 * 60;

    std::tm target_date = *std::localtime(&target_time);
    return target_date;
}

int main() {
    int day_of_year, year;
    std::cin >> day_of_year >> year;

    std::tm target_date = day_of_year_to_date(day_of_year, year);

    char buffer[100];
    std::strftime(buffer, sizeof(buffer), "%B", &target_date);
    std::cout << buffer << " " << target_date.tm_mday << std::endl;

    return 0;
}
```

19 Coding id

11 points

Introduction

During summer you got an internship in the secret service. To protect their identity, each secret agent has an id number that is generated from their first name. You have been assigned to develop a program that calculates the id number of an agent given these rules:

1. The agent's first name is a single word composed of upper and lowercase letters from the English alphabet.

2. The name will be coded using the 26 first prime numbers, starting by 2, 3, 5, ..., 97, 101.

3. Each letter of the name will be assigned a prime number. This value does not change if the letter is upper or lower case.

4. The id is obtained by adding all the prime numbers corresponding to the letters.

5. The assignment order between letters and prime numbers in a given name is determined by the first letter of that name. If the first letter is 'A', there is a direct correspondence between letters and prime numbers, such that A→2, B→3, C→5, ..., Y→97, Z→101. However, if the first letter of the name is 'B', the baseline correspondence is circularly shifted by one position, so that: A→101, B→2, C→3, ... Z→97.

Input

The input is a single word representing the agent's name to code.

Output

Print out the coding id corresponding to the input name.

Example 1

Input

```
Alan
```

Output

```
84
```

Example 2

Input

```
Colin
```

Output

```
126
```

Python

```python
# Getting the name to code
name = input()

# Getting the first letter of the name
firstLetter = name[0]

# Initialize the variable to compute the coding id
id = 0
number = 2

numPrimes = 0

# List containing the first 26 prime numbers
primeNumbers=[]


while numPrimes < 26:
  isPrime = True
  for i in range(2, int(number/2) + 1):
    if(number % i == 0):
      isPrime = False

  if(isPrime):
    primeNumbers.append(number)
    numPrimes += 1

  number += 1



# Traverse the name to code
for i in name:
  # Find out the index corresponding to current letter i
  index = (ord(i.upper()) - ord(firstLetter.upper())) % 26
  # Add the corresponding primer number to find out the coding id
  id = id + primeNumbers[index]

# Print out the resulting coding id
print(id)
```

C++

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
int primeNumbers[26];
bool isPrime = true;
int number = 2;
int numPrimes = 0;
while (numPrimes < 26) {
  isPrime = true;
  for (int i = 2; i <= number/2; i++) {
    if(number % i == 0) {
      isPrime = false;
    }
  }
  if(isPrime) {
    primeNumbers[numPrimes] = number;
    numPrimes++;
  }
  number++;
}
string name;
cin >> name;
int result = 0;
int position = 0;
int currentPosition;
for(int i = 0; i < name.length(); i++) {
    int letter = toupper((char)name[i]);
    if(position == 0) {
        position = letter - 64;
        result += primeNumbers[0];
    }
    else {
        if(letter-(64+position) < 0) {
            currentPosition = 26 - (-1 * (letter - (64+position)));
            result += primeNumbers[currentPosition];
        }
        else{
            currentPosition = letter - (64 + position);
            result += primeNumbers[currentPosition];
        }
    }
}
cout << result;
return 0;}
```

20 Olympic Medal Table

11 points

Introduction

The Olympic medal table lists the number of medals each country wins in the Olympics. The convention used by the International Olympic Committee is to sort countries by the number of gold medals their athletes have earned. In the event of a tie in gold medals, the number of silver medals is considered, followed by the number of bronze medals. If two countries have the same number of gold, silver, and bronze medals, they are listed alphabetically.

Input

The input data includes several countries, with the following format for each country:

- The first line contains the country's name.
- The next three lines provide the number of medals: gold, silver, and bronze, respectively.

The input ends with the character '#'.

Output

The sorted Olympic Medal Table where each line will consist of

#Position #Country #GoldMedals #SilverMedals #BronzeMedals

Example 1

Input

```
France
8
5
16
Canada
6
5
7
South Africa
0
2
0
United States of America
37
34
37
#
```

Output

```
1 United States of America 37 34 37
2 France 8 5 16
3 Canada 6 5 7
4 South Africa 0 2 0
```

Example 2

Input

```
Lithuania
0
1
1
Estonia
0
1
1
Spain
0
1
2
Jamaica
1
1
1
Belgium
0
2
3
#
```

Output

```
1 Jamaica 1 1 1
2 Belgium 0 2 3
3 Spain 0 1 2
4 Estonia 0 1 1
5 Lithuania 0 1 1
```

Python

```python
def read_input():
    countries = []
    while True:
        country_name = input().strip()
        if country_name == '#':
            break
        gold_medals = int(input().strip())
        silver_medals = int(input().strip())
        bronze_medals = int(input().strip())

        countries.append((country_name, gold_medals, silver_medals,
bronze_medals))
    return countries

def sort_countries(countries):
    # Sort primarily by gold, then by silver, continue with bronze and finally
alphabetically (all descending)
    return sorted(countries, key=lambda x: (-x[1], -x[2], -x[3], x[0]))

def print_medal_table(sorted_countries):
    for i, (country, gold, silver, bronze) in enumerate(sorted_countries,
start=1):
        print(f"{i} {country} {gold} {silver} {bronze}")

def main():
    countries = read_input()
    sorted_countries = sort_countries(countries)
    print_medal_table(sorted_countries)

# Call the main function
main()
```

21 Hourglass

11 points

Introduction

An hourglass is an ancient device used to measure time. It has two glass bulbs connected by a narrow tube, and sand flows from the top bulb to the bottom one in a set amount of time, usually an hour. Can you write a program that takes an even integer *n* as input and draws an ASCII art hourglass of height *n*?

Input

The input consists of a single positive even integer *n*.

Output

The output is an ASCII art hourglass of height *n* as it is show in the examples.

Example 1

Input

2

Output

```
|====|
|\**/|
|/**\|
|====|
```

Example 2

Input

4

Output

```
|======|
|\****/|
| \**/ |
| /**\ |
|/****\|
|======|
```

Example 3

Input

8

Output

```
|==========|
|\********/|
| \******/ |
|  \****/  |
|   \**/   |
|   /**\   |
|  /****\  |
| /******\ |
|/********\|
|==========|
```

Python

```python
num = int(input())

# Print top half of the hourglass
print('|'+'='*(num + 2)+'|')
step = 0
for i in range(num,0,-2):
    print('|'+' ' * step + '\\' + '*' * i + '/'+' ' * step+'|')
    step += 1

# Print bottom half of the hourglass
step = (num//2) - 1
for i in range(2,num+1,2):
    print('|'+' ' * step + '/' + '*'*i + '\\'+ ' ' * step+'|')
    step -= 1
print('|'+'='*(num + 2)+'|')
```

22 Bracket Notation

13 points

Introduction

Given two positive integers, a numerator and a denominator, your task is to determine the decimal representation of the fraction formed by these two integers.

Input

The input consists of two lines:

- The first line contains a positive integer representing the numerator of the fraction.
- The second line contains a positive integer representing the denominator of the fraction.

Output

The output is a single line containing the decimal representation of the fraction. If the decimal representation has repeating decimals, they should be enclosed within parentheses. The maximum period length for repeating decimals considered is 6.

Example 1

Input

```
1
3
```

Output

```
0.(3)
```

Example 2

Input

```
3
8
```

Output

```
0.375
```

Example 3

Input

```
8
4
```

Output

```
2
```

Example 4

Input

```
7
11
```

Output

```
0.(63)
```

Example 5

Input

```
29
12
```

Output

```
2.41(6)
```

Python

```python
# Read numerator and denominator from input
numerator = int(input())
denominator = int(input())

def find_repeating_decimal(numerator, denominator):
    remainder_map = {}
    remainder = numerator % denominator
    decimal_part = ""

    while remainder != 0 and remainder not in remainder_map:
        remainder_map[remainder] = len(decimal_part)
        remainder *= 10
        decimal_part += str(remainder // denominator)
        remainder %= denominator

    if remainder == 0:
        return decimal_part
    else:
        start = remainder_map[remainder]
        return decimal_part[:start] + "(" + decimal_part[start:] + ")"

# Find the integer part
integer_representation = str(numerator // denominator)

# And concatenate the decimal part
decimal_representation = find_repeating_decimal(numerator, denominator)

# Check if the decimal representation is empty
if decimal_representation == "":
    # Print the result
    print(integer_representation)
else:
    decimal_representation = integer_representation + "." +
decimal_representation
    # Print the result
    print(decimal_representation)
```

23 Enchanted Vines

13 points

Introduction

In a magical forest, the elves need to connect different pieces of enchanted vines to create a single, powerful vine for their annual festival. The vines are connected through a spell, and the cost of the spell depends on the lengths of the vines being connected. So, connecting two vines is equal to the sum of their lengths. Your task is to help the elves by creating a program to connect a series of vines with minimum cost.

Given four enchanted vines with lengths of 4, 3, 2 and 6, it is possible to connect the vines in different ways. The optimal in this case is:

1. First, connect the vines of lengths 2 and 3, which makes the available vines of lengths 4, 5 and 6, with a spell cost of 2 + 3 = 5.
2. Now, connect the vines of lengths 4 and 5, which makes the available vines of lengths 9 and 6, with a spell cost of 4 + 5 = 9.
3. Finally, connect the two remaining vines, with a spell cost of 9 + 6 = 15

The optimized total cost for connecting all vines is 5 + 9 + 15 = 29.

Other ways of connecting vines would always have same or higher cost. For example, if we connect 4 and 6 first (we get three vines of lengths 3, 2 and 10), then connect 10 and 3 (we get two vines of lengths 13 and 2). Finally connecting vines of lengths 13 and 2, the total cost in this way is 10 + 13 + 15 = 38.

Input

The input consists of a single line of positive integers containing the lengths of the different enchanted vines, we need to connect these vines to form one vine.

Output

The minimum spell cost to connect the enchanted vines.

Example

Input

```
4 3 2 6
```

Output

```
29
```

Python

```python
# Read the enchanted vines from the input
vines = input().split()

# Initialize the spell cost
spellCost = 0

# While there are more than 2 vines
while len(vines) >= 2:
    # If there are exactly 2 vines, add their cost to the spell cost
    if len(vines) == 2:
        spellCost += sum(map(int, vines))
        vines = []
    else:
        # Find the two vines with the minimum cost
        min_cost_vines = sorted(vines, key=int)[:2]
        # Remove the two vines with the minimum cost
        vines.remove(min_cost_vines[0])
        vines.remove(min_cost_vines[1])
        # Find the cost of the new vine from joining the two vines
        newVine = sum(map(int, min_cost_vines))
        # Add the cost of the new vine to the spell cost
        spellCost += newVine
        # Add the new vine to the list of vines
        vines.append(newVine)

print(spellCost)
```

24 Ecosystem simulator

13 points

Introduction

You are responsible for simulating an ecosystem composed of prey and predators. In this ecosystem, prey have a birth rate and a natural death rate, while predators depend on hunting prey to survive. Predators die if they do not manage to hunt for three consecutive days.

The ecosystem follows these rules:

Prey Population Growth:

- Each day, the prey population increases based on a birth rate. For example, if there are 50 prey and the birth rate is 0.1, 5 prey are born every day, if its 0.01, 0 preys would be born every day.

Natural Death of Prey:

- Each day, a proportion of the prey population dies naturally.

Predation:

- Each day, each predator hunts a specific number of prey (predation rate).
- The number of prey hunted cannot exceed the available prey.

Survival of Predators:

- Predators that do not hunt for three consecutive days die.

Input

The input consists of five lines:

- Initial prey population (positive integer).
- Initial predator population (positive integer).
- Prey birth rate (float between 0 and 1).
- Prey natural death rate (float between 0 and 1).
- Predation rate (positive integer indicating the number of prey hunted by each predator per day).
- Simulation days (positive integer)

Output

The output should display the prey and predator populations at the end of each day.

Example 1

Input

```
50
10
0.04
0.02
1
10
```

Output

```
Day 1: Prey = 41, Predators = 10
Day 2: Prey = 32, Predators = 10
Day 3: Prey = 23, Predators = 10
Day 4: Prey = 13, Predators = 10
Day 5: Prey = 3, Predators = 10
Day 6: Prey = 0, Predators = 10
Day 7: Prey = 0, Predators = 10
Day 8: Prey = 0, Predators = 10
Day 9: Prey = 0, Predators = 3
Day 10: Prey = 0, Predators = 0
```

Example 2

Input

```
30
5
0.2
0.05
3
8
```

Output

```
Day 1: Prey = 20, Predators = 5
Day 2: Prey = 8, Predators = 5
Day 3: Prey = 1, Predators = 5
Day 4: Prey = 0, Predators = 5
Day 5: Prey = 0, Predators = 5
Day 6: Prey = 0, Predators = 5
Day 7: Prey = 0, Predators = 1
Day 8: Prey = 0, Predators = 0
```

Python

```python
# Example input
prey_population = int(input())
predator_population = int(input())
prey_birth_rate = float(input())
prey_death_rate = float(input())
predation_rate = int(input())  # Each predator hunts one prey per day
days= int(input())
# Run simulation
predators_days_without_food = [0] * predator_population

for day in range(1, days + 1):
    # Birth of prey
    new_prey = int(prey_population * prey_birth_rate)
    # Natural death of prey
    natural_prey_deaths = int(prey_population * prey_death_rate)

    # Prey hunted by predators
    prey_hunted = min(int(predator_population * predation_rate),
prey_population)

    # Update prey population
    prey_population += new_prey - natural_prey_deaths - prey_hunted

    # Handle predators dying from not eating for 3 days
    for i in range(len(predators_days_without_food)):
        if predators_days_without_food[i] >= 3:
            predators_days_without_food[i] = -1  # Mark predators that died

    predators_days_without_food = [d for d in predators_days_without_food if
d != -1]
    predator_population = len(predators_days_without_food)
    for i in range(len(predators_days_without_food)):
        if prey_hunted > 0:
            predators_days_without_food[i] = 0  # Predator fed
            prey_hunted -= 1
        else:
            predators_days_without_food[i] += 1  # Did not eat
    if prey_population < 0:
        prey_population = 0
    if predator_population < 0:
        predator_population = 0
    print(f"Day {day}: Prey = {prey_population}, Predators =
{predator_population}")
```

25 Stem And Leaf

14 points

Introduction

In statistics, data is often displayed using tables, charts and graphs. However, these methods can sometimes fail to preserve the actual data values. To maintain the integrity of the data values, a stem and leaf plot can be used. This plot organizes and sorts data simultaneously by dividing each value into a "stem" (the leading digit or digits) and a "leaf" (the remaining digit). This method effectively groups and displays sorted data, making it particularly useful when dealing with large datasets.

Consider the following example: the number of meals sold by a restaurant each day over a period of 20 days was: 28, 34, 23, 35, 16, 17, 47, 5, 60, 26, 39, 35, 47, 35, 38, 35, 55, 47, 54, 48. To build a stem and leaf plot manually do the following steps:

1. Sort your data in ascending order.
2. Divide your data into stem and leaf values. You must separate each value in two parts: the stem, equal to all number digits but last and the leaf, equal to the last digit. For numbers in range 0-9 you must add a "0" at the start.
3. Write down your stem values to set up the groups (stems without leaves should not be printed).

```
0|
1|
2|
3|
4|
5|
6|
```

4. Finally add the leaf values in numerical order to create the depths for each stem value group.

```
0|5
1|67
2|368
3|4555589
4|7778
5|45
6|0
```

To make it easier to build the stem and leaf plot, all the input values will have one or two digits.

Input

The input consists of variable number of lines, one positive integer in range [0..99] per line. The input ends with a line containing a single character '#'.

Output

The steam and leaf plot.

Example 1

Input

```
28
34
23
35
16
17
47
5
60
26
39
35
47
35
38
35
55
47
54
48
#
```

Output

```
0 | 5
1 | 6 7
2 | 3 6 8
3 | 4 5 5 5 5 8 9
4 | 7 7 7 8
5 | 4 5
6 | 0
```

Example 2

Input

```
0
2
5
10
12
5
11
99
2
23
12
99
46
23
68
73
35
54
66
73
81
1
2
22
34
89
11
15
#
```

Output

```
0 | 0 1 2 2 2 5 5
1 | 0 1 1 2 2 5
2 | 2 3 3
3 | 4 5
4 | 6
5 | 4
6 | 6 8
7 | 3 3
8 | 1 9
9 | 9 9
```

Example 3

Input

```
25
30
40
41
42
43
44
45
46
47
48
49
71
73
75
77
#
```

Output

```
2 | 5
3 | 0
4 | 0 1 2 3 4 5 6 7 8 9
5 |
6 |
7 | 1 3 5 7
```

Python

```python
def stem_and_leaf_plot(data):
    # Create an empty dictionary to store the stems and leaves
    stems = {}
    # Find the min an max values in the data
    min_value = min(data)
    max_value = max(data)
    # Create a list of all possible stems
    for i in range(min_value // 10, max_value // 10 + 1):
        stems[i] = []

    # Find the stem and leaf for each number in the data
    for number in data:
        # Extract the stem and leaf from the number
        stem = number // 10
        leaf = number % 10
        stems[stem].append(leaf)

    # Print the stem and leaf plot in ascending order
    for stem in sorted(stems.keys()):
        leaves = ' '.join(str(leaf) for leaf in sorted(stems[stem]))
        print(f"{stem} | {leaves}")

# Main program

# Read the data from standard input
readData = input()
input_data = ""
while readData != "#":
    input_data = input_data + readData + " "
    readData = input()

# Convert the input data into a list of integers
data = list(map(int, input_data.split()))

# Finally, call the stem_and_leaf_plot function with the data
stem_and_leaf_plot(data)
```

26 Interactive Hanged Man (Interactive problem)

14 points

Introduction

You are playing a game of hanged man against your online friend Zoro, but you are too lazy to keep writing letters and trying to guess the word. So, you decide to create a program that asks the letter by you and sends the complete word once you have found all the letters. However, the version of the game you are playing is different, you must complete the word by yourself, your friend will only send you the position of the letter you have asked, not with the previous ones you already asked. You can ask an unlimited number of single letters, but you can only guess the word once.

Input

If the outputted letter is in the word, the input you receive is a string with the positions where the letter is in.
If the outputted letter is not in the word, the input you receive is a string with as many dashes as letters are in the hidden word.

Output

A single letter every time you want to check where the letter is.
A word with at least two letters to guess the hidden word.
After every question you need to flush the standard output to ensure that it is sent to Zoro. For example, you can use fflush(stdout) in C++, System.out.flush() in Java and sys.stdout.flush() in Python.

Example 1

Communication example 1

You	Zoro
d	
	d__
o	
	_o_
g	
	__g
dog	

Example 2

You	Communication example 2	Zoro
k		
		k____
a		
		_a_a_a
n		
		___n_
o		
		_____
t		
		__t__
katana		

Python

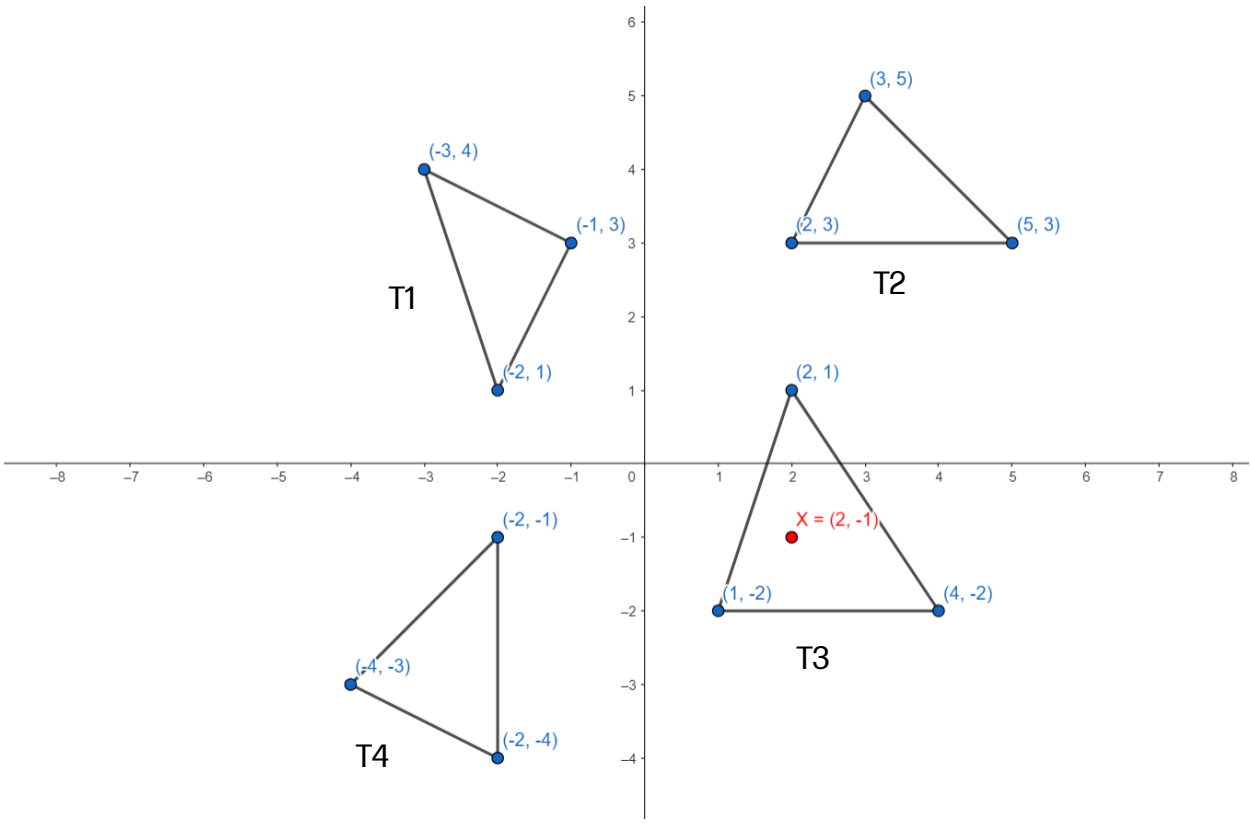
```python
import sys

n = 1

letters="eioubcdfghjklmnpqrstvwxyz"

print("a")
sys.stdout.flush()
answer = input()

for i in range(len(letters)):
    pos = 0
    print(letters[i])
    sys.stdout.flush()
    response = input()
    for c in response:
        if c != '_':
            answer = answer[:pos] + c + answer[pos+1:]
        pos += 1
    if answer.find('_')==-1:
        print(answer)
        sys.stdout.flush()
        break
```

27 The Lost Treasure Map

17 points

Introduction

In the mystical land of Trianglia, a treasure map has been discovered! The map shows several not overlapping triangles, and X marks the spot where the treasure might be hidden. Your task is to write a program to determine if any of the triangles on the map contains the treasure point.



To find the area of a triangle given its three vertices (x_1, y_1), (x_2, y_2), and (x_3, y_3), you can use the formula:

$$Area = \frac{1}{2} \mid x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \mid$$

Input

One or more lines defining for each line a triangle name and its three pairs of coordinates (x_1, y_1), (x_2, y_2), and (x_3, y_3). Finally, a last line defining point X and its pair of coordinates (px, py). Notice that all coordinates for triangles and point are integer numbers.

Output

The output will report if a given triangle contains point X in a sentence like this:

Triangle *triangle_id* contains point X.

Otherwise, the output will be like this:

No triangle contains point X.

Example 1

Input

```
T1: (-3,4),(-1,3),(-2,1)
T2: (3,5),(2,3),(5,3)
T3: (2,1),(4,-2),(1,-2)
T4: (-2,1),(-2,-4),(-4,-3)
X: (2,-1)
```

Output

```
Triangle T3 contains point X.
```

Example 2

Input

```
T1: (2,1),(4,-2),(1,-2)
T2: (0,1),(1,0),(-1,-1)
X: (0,2)
```

Output

```
No triangle contains point X.
```

Example 3

Input

```
Bermudas: (-3,4),(-1,3),(-2,1)
X: (-2,2)
```

Output

```
Triangle Bermudas contains point X.
```

Python

```python
# Function to calculate the area of a triangle given its three coordinates (x1, y1, x2, y2, x3, y3)
def calculateTriangleArea(x1, y1, x2, y2, x3, y3):
     return abs((x1 * (y2 - y3) + x2 * (y3 - y1)
                 + x3 * (y1 - y2)) / 2.0)

# Function to check if a point (x, y) is inside a triangle given its three coordinates (x1, y1, x2, y2, x3, y3)
# A point is inside the triangle if the sum of the areas of the three triangles formed by the point and
# the three vertices of the triangle is equal to the area of the original triangle
def is_point_inside_triangle(x, y, triangle):
    # Calculate the area of the triangle
    triangleArea = calculateTriangleArea(triangle[0], triangle[1], triangle[2], triangle[3], triangle[4], triangle[5])

    # Calculate area of triangle PAC
    A1 = calculateTriangleArea(x, y, triangle[0], triangle[1], triangle[4], triangle[5])

    # Calculate area of triangle PAB
    A2 = calculateTriangleArea(x, y, triangle[0], triangle[1], triangle[2], triangle[3])

    # Calculate area of triangle PBC
    A3 = calculateTriangleArea(x, y, triangle[2], triangle[3], triangle[4], triangle[5])

    # Check if sum of A1, A2 and A3 is same as triangleArea
    if(triangleArea == A1 + A2 + A3):
        return True
    else:
        return False

# Function to read triangles and point from standard input
def read_input():
    import sys
    input = sys.stdin.read
    data = input().splitlines()

    # Read triangles
    triangles = []
    for line in data[:-1]:
        parts = line.split(": ")
```

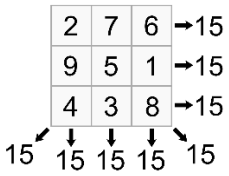
```
        # Extract triangle id and vertices
        triangle_id = parts[0]
        vertices = parts[1].split(",")
        # Removing parentheses and converting to integers
        vertices = [v.replace("(","").replace(")","") for v in vertices]
        vertices = [int(coord) for v in vertices for coord in
v.split(",")]
        # Append triangle to the list
        triangles.append((triangle_id, vertices))

    # Read point X
    point_line = data[-1]
    point_parts = point_line.split(": ")
    point_label = point_parts[0]
    point_coords = tuple(map(int, point_parts[1].strip("()").split(',')))

    return triangles, (point_label, point_coords)

# Main program
triangles, point = read_input()

triangleX = None
for triangle in triangles:
    if is_point_inside_triangle(point[1][0], point[1][1], triangle[1]):
        triangleX = triangle[0]
        break

if triangleX != None:
    print("Triangle", triangle[0], "contains point X.")
else:
    print("No triangle contains point X.")
```

28 Magic Square Makeover

18 points

Introduction

A magic square is a n x n matrix of different positive integers from 1 to n^2, where the sum of any row, column, or diagonal of length n is always equal to the same number. For sake of simplicity in this challenge, let's consider only 3 x 3 magic squares.



Given a 3 x 3 square, your task is to transform it into a magic square with the least possible cost. The cost of replacing a digit x in the square by any other digit y is computed as $|x-y|$.

Input

The input consists of *3*x *3*square formed by three lines, where each line contains three digits.

Output

The output consists of the *3*x *3*magic square with the least possible cost in the same form as the input, followed by a fourth line containing the cost of the changes made.

Example 1

Input

```
5 3 4
1 5 8
6 4 2
```

Output

```
8 3 4
1 5 9
6 7 2
Cost: 7
```

In this example notice that minimum cost was achieved with three replacements done at a cost of: $|5-8| + |8-9| + |4-7| = 7$

Python

```python
#!/bin/python3

import math
import os
import random
import re
import sys

magicSquares = [
                [2,7,6],[9,5,1],[4,3,8],
                [2,9,4],[7,5,3],[6,1,8],
                [4,3,8],[9,5,1],[2,7,6],
                [4,9,2],[3,5,7],[8,1,6],
                [6,1,8],[7,5,3],[2,9,4],
                [6,7,2],[1,5,9],[8,3,4],
                [8,1,6],[3,5,7],[4,9,2],
                [8,3,4],[1,5,9],[6,7,2]
               ]
def formingMagicSquare(s):
    min_cost = float('inf')

    # Iterate over all possible magic squares
    for i in range(0, len(magicSquares), 3):
        current_magic_square = magicSquares[i:i+3]
        cost = 0

        # Calculate the cost to convert s to the current magic square
        for row in range(3):
            for col in range(3):
                cost += abs(s[row][col] - current_magic_square[row][col])
        if cost < min_cost:
            min_cost = cost
            min_magic_square = current_magic_square

    return min_magic_square, min_cost

# Main program
s = []
for _ in range(3):
    s.append(list(map(int, input().rstrip().split())))
magic_square, res = formingMagicSquare(s)
for row in magic_square:
    print(" ".join(map(str, row)))
print("Cost: " + str(res))
```

29 Mastering The Matrix Determinant

19 points

Introduction

The determinant is a special number that can be calculated from a square matrix. It provides important information about the matrix, such as whether it has an inverse, and it can also be used in solving systems of linear equations, among other applications.

For *2*x *2*matrix,

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

the determinant is calculated using the formula:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = (-1)^{i+j}ad + (-1)^{i+j}bc = (-1)^{2}ad + (-1)^{3}bc = ad - bc$$

where indexes *i*and *j*refer to the row and column in the matrix of elements *a*and *b*.

For a *3*x *3*matrix,

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

the determinant is calculated using this formula:

$$\begin{aligned} \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} &= (-1)^{1+1}a\begin{vmatrix} e & f \\ h & i \end{vmatrix} + (-1)^{2+1}d\begin{vmatrix} b & c \\ h & i \end{vmatrix} + (-1)^{3+1}g\begin{vmatrix} b & c \\ e & f \end{vmatrix} \\ &= (-1)^{2}a\begin{vmatrix} e & f \\ h & i \end{vmatrix} + (-1)^{3}d\begin{vmatrix} b & c \\ h & i \end{vmatrix} + (-1)^{4}g\begin{vmatrix} b & c \\ e & f \end{vmatrix} \\ &= a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - d\begin{vmatrix} b & c \\ h & i \end{vmatrix} + g\begin{vmatrix} b & c \\ e & f \end{vmatrix} \end{aligned}$$

where the indexes *i*and *j*refer to the row and column of the elements in the matrix. Applying the formula for determinant of a *2*x *2*matrix, it can be easily calculated as:

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a(ei - fh) - d(bi - ch) + g(bf - ce)$$

This procedure is known as Laplace expansion, an algorithm for finding the determinant of a matrix.

It can be easily expanded to *4*x *4*matrix, as this example shows:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 7 & 3 \\ 4 & 10 & 14 & 6 \\ 3 & 4 & 2 & 7 \end{pmatrix}$$

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 7 & 3 \\ 4 & 10 & 14 & 6 \\ 3 & 4 & 2 & 7 \end{vmatrix} = (-1)^{1+1} \cdot 1 \begin{vmatrix} 5 & 7 & 3 \\ 10 & 14 & 6 \\ 4 & 2 & 7 \end{vmatrix} + (-1)^{2+1} \cdot 2 \begin{vmatrix} 2 & 3 & 4 \\ 10 & 14 & 6 \\ 4 & 2 & 7 \end{vmatrix} +$$

$$(-1)^{3+1} \cdot 4 \begin{vmatrix} 2 & 3 & 4 \\ 5 & 7 & 3 \\ 4 & 2 & 7 \end{vmatrix} + (-1)^{4+1} \cdot 3 \begin{vmatrix} 2 & 3 & 4 \\ 5 & 7 & 3 \\ 10 & 14 & 6 \end{vmatrix} =$$

$$5 \begin{vmatrix} 14 & 6 \\ 2 & 7 \end{vmatrix} + (-10) \begin{vmatrix} 7 & 3 \\ 2 & 7 \end{vmatrix} + 4 \begin{vmatrix} 7 & 3 \\ 14 & 6 \end{vmatrix} + (-4) \begin{vmatrix} 14 & 6 \\ 2 & 7 \end{vmatrix} + 20 \begin{vmatrix} 3 & 4 \\ 2 & 7 \end{vmatrix} + (-8) \begin{vmatrix} 3 & 4 \\ 14 & 6 \end{vmatrix} +$$

$$8 \begin{vmatrix} 7 & 3 \\ 2 & 7 \end{vmatrix} + (-20) \begin{vmatrix} 3 & 4 \\ 2 & 7 \end{vmatrix} + 16 \begin{vmatrix} 3 & 4 \\ 2 & 7 \end{vmatrix} + (-6) \begin{vmatrix} 7 & 3 \\ 14 & 6 \end{vmatrix} + 15 \begin{vmatrix} 3 & 4 \\ 14 & 6 \end{vmatrix} + (-30) \begin{vmatrix} 3 & 4 \\ 7 & 3 \end{vmatrix} = 0$$

Given this rule of thumb, can you write a program that finds the determinants of any *2 x 2,3 x 3* or *4 x 4* matrix?

Input

The input is a matrix of *n*rows and *n*columns, with each row on a separate line. The input ends with a line containing a single character '#'.

Output

The output consists of a line returning the corresponding determinant of the input matrix.

Example 1

Input

```
1 2
2 5
#
```

Output

```
Determinant: 1
```

Example 2

Input

```
1 4 -1
-1 3 2
2 2 0
#
```

Output

```
Determinant: 20
```

Example 3

Input

```
1 2 3 4
2 5 7 3
4 10 14 6
3 4 2 7
#
```

Output

```
Determinant: 0
```

Python

```python
def get_matrix_minor(matrix, i, j):
    return [row[:j] + row[j+1:] for row in (matrix[:i] + matrix[i+1:])]

def get_determinant(matrix):
    # Base case for 2x2 matrix
    if len(matrix) == 2:
        return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]

    # Otherwise, calculate the determinant using the formula
    determinant = 0
    for c in range(len(matrix)):
        determinant += ((-1)**c) * matrix[0][c] *
get_determinant(get_matrix_minor(matrix, 0, c))
    return determinant

def read_matrix():
    matrix = []
    while True:
        row = input().strip()
        if row == '#':
            break
        matrix.append(list(map(int, row.split())))
    return matrix

# Main program
matrix = read_matrix()

# Calculate the determinant
determinant = get_determinant(matrix)
print(f"Determinant: {determinant}")
```

30 Four Up!

21 points

Introduction

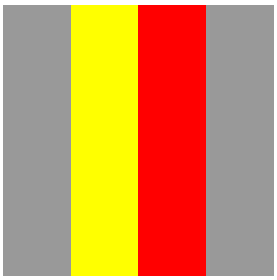
Little Timmy wants to learn how to play the famous game of Four Up. The reason he wants to learn how to play is that he intends to use this game as a way to encrypt and decrypt his ideas so no one can copy them. For that, he asks you to teach him how to solve the game.

As you may know, Four Up is a game where two players take turns. They are usually called player 'Red' and player 'Yellow'. First, one player starts by inserting his/her disk, and then the next player inserts his/her disk. Once a disk is inserted into a column, it goes all the way down until it reaches the bottom of the board or another disk (in this case, the inserted disk stops at that position). They keep playing this way until one of them connects four disks of the same color. The connections can be horizontal, vertical, or diagonal.

To get an idea of how the board and disks look, take a look at the following picture:

The player 'Red' has four connected disks diagonally of the same color. Therefore, he/she wins the game against player 'Yellow'.

Another example, as seen in the picture below, shows both players drawing as they connect four disks each. In this case, Timmy wants them to draw instead of declaring one of them the winner.



Input

You will be given the dimensions of the board (Rows and Columns), followed by a series of moves for each player, starting with a letter and followed by a number indicating the column where they want to place the disk. So:

- The first line indicates the number of rows of the board.
- The second line indicates the number of columns of the board.
- Then, as many lines as possible indicating a player and the column number (separated by a whitespace) he/she wants to play. For instance:
  - r 3 means player 'Red' wants to place the disk in column 3.
  - y 4 means player 'Yellow' wants to place the disk in column 4.
- The last line contains the character '#' indicating the end of the problem.

Note 1: You should keep reading the moves until there are no more entries.
Note 2: The minimum board size is 4x4.
Note 3: The given number of rows and columns start at 0.

Output

Timmy wants you to give him the drawing and the winner of the game. The drawing should be in the following format:

- The first line is a special code for Timmy that is always the same: **P3**
- The second line holds the **columns** and **rows** of the board (Note: columns go first).
- The third line is a special code for Timmy that is always the same: **255**
- Then, all the rows and columns of the board should be listed. Each cell of the board is an (Red Green Blue) **RGB** triplet separated by spaces. The rows should be separated by a new line.
  - For the 'Red' player, the RGB is **255 0 0**
  - For the 'Yellow' player, the RGB is **255 255 0**
  - For the empty cells, the RGB should be **153 153 153**
- Then, a text indicating the winner or the draw:
  - If 'Red' wins, it should say: **Red wins**
  - If 'Yellow' wins, it should say: **Yellow wins**
  - If no player wins, it should say: **Draw!**

Note: If you save this output to a file with a ".ppm" extension, you can open this special code with your favorite image reader and see it converted to an image. That's Timmy's magic.

Example 1

Input

```
6
7
r 6
y 3
r 5
y 4
r 5
y 3
r 3
y 1
r 4
y 1
r 4
y 5
r 3
y 1
#
```

Output

```
P3
7 6
255
153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153
153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153 153
153 153 153 153 153 153 153 153 153 255 0 0 153 153 153 153 153 153 153 153 153
153 153 153 255 255 0 153 153 153 255 0 0 255 0 0 255 255 0 153 153 153
153 153 153 255 255 0 153 153 153 255 255 0 255 0 0 255 0 0 153 153 153
153 153 153 255 255 0 153 153 153 255 255 0 255 255 0 255 0 0 255 0 0
Red wins
```

Example 2

Input

```
4
4
y 1
r 2
y 1
r 2
y 1
r 2
y 1
r 2
#
```

Output

```
P3
4 4
255
153 153 153 255 255 0 255 0 0 153 153 153
153 153 153 255 255 0 255 0 0 153 153 153
153 153 153 255 255 0 255 0 0 153 153 153
153 153 153 255 255 0 255 0 0 153 153 153
Draw!
```

Python

```
import sys

# Leer el número de filas (R) y columnas (C)
R = int(input())
C = int(input())
print("P3")
print(f"{C} {R}")  # ancho y alto en píxeles
print("255")  # color máximo

# f: libre; r: jugador rojo; y: jugador amarillo
table = [['f' for _ in range(C)] for _ in range(R)]
while True:
    line = input().strip()
    if line == "#":
        break  # Detener la lectura si se encuentra '#'

    if line[0] == 'r' or line[0] == 'y':
        find = 0
        col = int(line[2])
        for k in range(R-1, -1, -1):
            if table[k][col] == 'f':
                table[k][col] = line[0]
                break
for i in range(R):
    for j in range(C):
        if table[i][j] == 'r':
            print("255 0 0", end="")
        elif table[i][j] == 'y':
            print("255 255 0", end="")
        else:
            print("153 153 153", end="")
        if j != C - 1:
            print(" ", end="")
    print()
y = 0
r = 0
for i in range(R):
    for j in range(C):
        v = table[i][j]
        if v == 'f':
            continue
        w = 1
        for l in range(1, 4):
```

```
            if j + l < C and table[i][j + l] == v:
                w += 1
        if w == 4:
            if v == 'r':
                r = 1
            elif v == 'y':
                y = 1
            continue
        w = 1
        for l in range(1, 4):
            if i + l < R and table[i + l][j] == v:
                w += 1
        if w == 4:
            if v == 'r':
                r = 1
            elif v == 'y':
                y = 1
            continue
        w = 1
        for l in range(1, 4):
            if i - l >= 0 and j + l < C and table[i - l][j + l] == v:
                w += 1
        if w == 4:
            if v == 'r':
                r = 1
            elif v == 'y':
                y = 1
            continue
        w = 1
        for l in range(1, 4):
            if i - l >= 0 and j - l >= 0 and table[i - l][j - l] == v:
                w += 1
        if w == 4:
            if v == 'r':
                r = 1
            elif v == 'y':
                y = 1
            continue

if (r == 1 and y == 1) or (r == 0 and y == 0):
    print("Draw!")
elif r == 1:
    print("Red wins")
elif y == 1:
    print("Yellow wins")
```

31 Randomized Factorio run

27 points

Introduction

Hi there fellow engineer! Welcome to the world of Factorio, a game where you can build a factory as big as your imagination can reach.

Factorio has a really simple mechanic: turn things into other things. For example we can grab iron ore and turn it into iron plates, and then we can turn those plates into gears, then those become transportation belts... All those thanks to your companion: the assemblers.

The assemblers take some fixed amount of materials, consumes them, and finally generate another fixed amount of a more complex materials. For example, we can build 2 small electric poles providing 2 copper cables and 1 wood. Note that recipes needs to be made by whole, so you can't provide "one copper cable" and "half wood" to provide only one small electric pole.

Not everything can be crafted from the assemblers; there's some materials (we'll refer to them as "base materials") that needs to be extracted from the ground, or manually collected. Those are: "Iron ore", "Copper ore", "Coal", "Stone, "Uranium ore", "Water", "Crude oil", "Wood", and "Fish".

To celebrate the release of Factorio 2.0 we'll request you to write a program that calculates how much of each base materials we need in order to craft some amount of a final item, but there's a catch! The habitants of the Factorio world don't like pollution, so following HP's sustainability commitment we request you to minimize the number of built items, so you consume as less amount of base materials as possible and that way reducing waste and pollution.

To re-use materials we'll use a common Factorio concept: the "main bus". A main bus is a place where all the crafted materials go in, so other sections of the factory can re-use them instead of building them from scratch. To minimize waste, our main bus will contain any previously crafted item.

To reduce complexity, consider that one material can only be obtained from one and only one recipe; the "base materials" are unobtainable from any recipe.

Also, don't expect that the recipes are always the same. We're playing randomized mode, so now we may need 1 copper plate to build 2 copper cables, but next time could be 2 copper plates and 1 fish to build 1 copper cable!

Input

The input consists of the number of the desired item we want, followed by the material itself.
Next we have a number that specifies the number of recipes we're providing you.
For each recipe, you'll have:

- The recipe name (it usually is <output material> followed by "recipe", but don't assume that's always the case!), the number of different materials that go in, and the number of built materials
- For each input material: the quantity requested, and the material type
- For each output material: the quantity requested, and the material type

Output

The output should be the number of each base material (and what base material) to craft the required number of items, in the most optimal way. The output should always be in the provided order.

Example 1

Input

```
20;Iron plate
1
Iron plate recipe;1;1
1;Iron ore
1;Iron plate
```

Output

```
20;Iron ore
0;Copper ore
0;Coal
0;Stone
0;Uranium ore
0;Water
0;Crude oil
0;Wood
0;Fish
```

Example 2

Input

```
5;Inserter
6
Inserter recipe;3;1
1;Iron plate
1;Iron gear wheel
1;Electronic circuit
1;Inserter
Copper cable recipe;1;1
1;Copper plate
2;Copper cable
Electronic circuit recipe;2;1
3;Copper cable
1;Iron plate
1;Electronic circuit
Iron gear wheel recipe;1;1
2;Iron plate
1;Iron gear wheel
Copper smelting;1;1
1;Copper ore
1;Copper plate
Iron smelting;1;1
1;Iron ore
1;Iron plate
```

Output

```
20;Iron ore
8;Copper ore
0;Coal
0;Stone
0;Uranium ore
0;Water
0;Crude oil
0;Wood
0;Fish
```

Python

```python
from collections import defaultdict
from dataclasses import dataclass
from typing import Dict, List, Set

@dataclass
class Recipe:
    name: str
    inputs: Dict[str, int]
    outputs: Dict[str, int]

class FactorioCalculator:
    def __init__(self):
        self.recipes = defaultdict(str) # Maps material to its recipe
        self.base_materials = [
            "Iron ore", "Copper ore", "Coal", "Stone",
            "Uranium ore", "Water", "Crude oil", "Wood", "Fish"
        ]
        self.main_bus = defaultdict(int)  # Tracks crafted materials

    def add_recipe(self, recipe_str: str) -> None:
        # Parse recipe string
        first_line, *rest = recipe_str.strip().split('\n')
        recipe_name, input_count, output_count = first_line.split(';')
        input_count, output_count = int(input_count), int(output_count)

        inputs = {}
        outputs = {}

        # Parse inputs
        for i in range(input_count):
            qty, material = rest[i].split(';')
            inputs[material] = int(qty)

        # Parse outputs
        for i in range(input_count, input_count + output_count):
            qty, material = rest[i].split(';')
            outputs[material] = int(qty)
            self.recipes[material] = Recipe(recipe_name, inputs, outputs)

    def calculate_requirements(self, target_qty: int, target_material: str) ->
Dict[str, int]:
        requirements = defaultdict(int)
        if target_material in self.base_materials:
            requirements[target_material] = target_qty
            return requirements
```

```
        needed = defaultdict(int)
        needed[target_material] = target_qty

        while any(amt > 0 for material, amt in needed.items()
                  if material not in set(self.base_materials)):  # Convert to set
for membership test
            for material, amount in list(needed.items()):
                if amount <= 0 or material in self.base_materials:
                    continue

                # Check main bus first
                used_from_bus = min(amount, self.main_bus[material])
                needed[material] -= used_from_bus
                self.main_bus[material] -= used_from_bus

                if needed[material] <= 0:
                    continue

                # Need to craft more
                recipe = self.recipes[material]
                output_qty = recipe.outputs[material]  # Get output quantity for
target material
                batches = (needed[material] + output_qty - 1) // output_qty

                # Add recipe inputs to needed materials
                for input_material, input_qty in recipe.inputs.items():
                    needed[input_material] += batches * input_qty

                # Add all outputs to main bus
                for output_material, output_amt in recipe.outputs.items():
                    crafted = batches * output_amt
                    self.main_bus[output_material] += crafted

                # Take what's needed from main bus
                used_from_bus = min(needed[material], self.main_bus[material])
                needed[material] -= used_from_bus
                self.main_bus[material] -= used_from_bus

        # Collect base material requirements
        for material, amount in needed.items():
            if material in self.base_materials and amount > 0:
                requirements[material] = amount

        return requirements

    def solve(self, input_str: str) -> str:
```

```
        # Parse input
        lines = input_str.strip().split('\n')
        target_qty, target_material = lines[0].split(';')
        target_qty = int(target_qty)

        recipe_count = int(lines[1])
        current_line = 2

        # Process recipes
        for _ in range(recipe_count):
            recipe_lines = []
            first_line = lines[current_line]
            _, input_count, output_count = first_line.split(';')
            recipe_lines.append(first_line)

            total_lines = 1 + int(input_count) + int(output_count)
            for i in range(1, total_lines):
                recipe_lines.append(lines[current_line + i])

            self.add_recipe('\n'.join(recipe_lines))
            current_line += total_lines

        # Calculate requirements
        requirements = self.calculate_requirements(target_qty, target_material)

        # Format output
        output_lines = []
        for material in self.base_materials:
            output_lines.append(f"{requirements.get(material, 0)};{material}")

        return '\n'.join(output_lines)

import sys

def main():
    calculator = FactorioCalculator()
    input_str=sys.stdin.read()
    print(calculator.solve(input_str))


if __name__ == "__main__":
    main()
```

32 Summon The Moon Lord (Interactive problem)

30 points

Introduction

You are a sorcerer living in an apocalyptic world. To resolve the chaos, you must find four hidden altars to summon the Moon Lord and eradicate all evil.

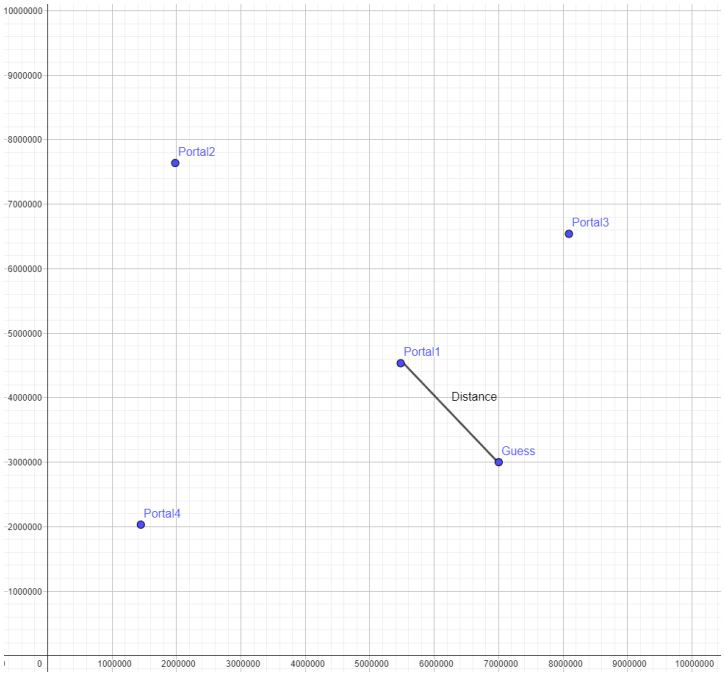
However, this world is extremely large (10^7 km x 10^7 km), so you cannot search for the altars randomly. After hundreds of years of magical investigations, you have obtained a spell (the judge's program) that allows you to determine the distance from a specified position to the nearest altar. However, the received distance is squared and multiplied by your mana cost, so you will need to perform some transformations.

$$Spell\ distance = (real\ distance)^2 \times mana\_cost$$

Your goal is to create a program that processes the information received by your spell (the judge's program) to find the positions of the altars (x,y). Once you find the position of all the altars, you will summon the moon lord.

The first line you will receive from the spell (the judge's program) will be the mana cost, after that you will have to guess one position in order to receive the distance from the position you sent to the nearest altar, when you send the position of an altar to the spell (the judge's program), you will receive a 0, and that altar will not be taken into account as nearest in next guesses.

Map Example



Input

The first input contains the mana cost, a positive integer sent by the judge's program.
The rest of the inputs will come after your outputs, containing the square of the distance from the position you sent to the closest altar, multiplied by the mana cost of your spell (distance^2 · mana cost).
If you have found the position of an altar, you will receive a 0.

Output

Two coordinates (x, y) corresponding to the position you want to calculate the distance from to reach the altar ($1 \leq x, y \leq 10^7$). You can ask for positions 2500 times before you ran out of mana. Every time you want to send a coordinate, you should do a flush. You can use fflush(stdout) in C++, System.out.flush() in Java and sys.stdout.flush() in Python.

Example

Spell (Judge's program)	Communication example 1	Your program
300		
		0 0
12300		
		0 1
9600		
		4 5
0		
		0 0

----------REPEAT FOR THE OTHER 3 ALTARS----------

0		
You summoned the MoonLord!		

Python

```python
import sys
from math import sqrt
mana_cost=int(input())
n = 4
L, H = 0, 10**7
def query(x, y):
    global n
    if x < L or x > H or y < L or y > H: return -1
    print(x, y)
    sys.stdout.flush()
    d = int(input())//mana_cost
    if d == 0: n -= 1
    if n == 0: exit(0)
    return d
# Circle-Circle intersection. Rounds all intersections to integers.
def intersect(p1, d1, p2, d2):
    assert p1[1] in [L, H] and p1[1] in [L, H]
    if (d1 - d2 + 1) % 2 != 0: return None
    dx = (d1 - d2 + 1) // 2

    dy = round(sqrt(d1 - dx * dx))
    if dx * dx + dy * dy != d1: return None
    if p1[1] != p2[1]: dx, dy = dy, dx
    # Move (dx, dy) towards the center.
    x, y = (dx if p1[0] == 0 else H - dx, dy if p1[1] == 0 else H - dy)
    if (x-p2[0])**2 + (y-p2[1])**2 != d2: return None
    return (x, y)
ps = [
    ((0, 0), (0, 1)),
    ((0, 0), (1, 0)),
    ((0, H), (0, H - 1)),
    ((0, H), (1, H)),
    ((H, 0), (H - 1, 0)),
    ((H, 0), (H, 1)),
    ((H, H), (H - 1, H)),
    ((H, H), (H, H - 1)),
]
while n > 0:
    for p, q in ps:
        d1 = query(*p)
        if d1 == 0: continue
        d2 = query(*q)
        r = intersect(p, d1, q, d2)
        if d1 > 0 and d2 > 0 and r:
            query(*r)
```