<CODEWARS2026/>

BCN · LEO · VAL

Valencia 2026

Problems

and solutions

#	Problem	Points
1	Frame Counter	2
2	Galactic Number Formatting	4
3	Swimming	5
4	Core Computation	5
5	Clone Replicator	6
6	The Stuttering Spell	6
7	Pentavocalic	6
8	Echoes Of Camel Case	6
9	Frequency-Hopping Spread Spectrum	6
10	First-Class Lever Simulator	7
11	Automorphic Numbers	7
12	The Mirror Game	7
13	Compression Algorithm	7
14	Pintan Bastos	7
15	Pace Calculator	8
16	Scrabble Scoring	8
17	Radio Chatter	9
18	Transpose Chords	11
19	Expected Goals	13
20	Orbital Trajectory Verification	13
21	GamerTag	13
22	The Ancient Library	13
23	Village Pathfinding	18
24	The Storm And The Supply Drop	18
25	Langton's Ant Escape	23
26	The Ancestor Oracle (Interactive)	25

1 Frame Counter

2 points

Introduction

Thelma is a video editor working on a documentary film. Thelma is editing footage captured at different frame rates and needs to calculate how many frames will be processed for various segments of the documentary.

FPS (frames per second) is the number of frames a computer screen shows every second. To calculate the total number of frames in a video segment, you need to multiply the FPS by the total number of seconds in that segment.

Help Thelma calculate the total number of frames for different video segments!

Input

The input contains two integers on separate lines. First line contains the duration of the video segment in minutes ($1 \le minutes \le 1000$). Second line represents the frames per second of the video ($1 \le fps \le 120$).

Output

Print a single integer with the total number of frames in the video segment.

Example

Input

```
10
25
```

Output

```
15000
```

Python

```python
# Read input
minutes = int(input())
fps = int(input())

# Calculate total frames
total_frames = fps * minutes * 60

# Output result
print(total_frames)
```

2 Galactic Number Formatting

4 points

Introduction

In a distant future, the Cosmic Accountants manage the vast sums of interstellar trade. To improve readability, they need a system to format large numbers using a single point (.) as a thousand separators. Your task is to implement this system with a program for integer numbers.

Input

An integer inside range [-1.000.000.000.000.000.000 , 1.000.000.000.000.000.000].

Output

Print the integer with thousand separators.

Example 1

Input

```
42
```

Output

```
42
```

Example 2

Input

```
12345
```

Output

```
12.345
```

Example 3

Input

```
1234567890
```

Output

```
1.234.567.890
```

Example 4

Input

```
-4567890
```

Output

```
-4.567.890
```

Python

```python
number = input()
restoreNegative = False
length = len(number)

if number[0] == "-":
    length -= 1
    number = number[1:]
    restoreNegative = True

res = ""

for i in range(length, 0, -1):
    res += number[length - i]
    if i % 3 == 1 and i != 1:
        res += "."

if  restoreNegative:
    res = "-" + res

print(res)
```

3 Swimming

5 points

Introduction

You are organizing a swimming pool competition where swimmers must track time using a special watch on the pool. The watch has four colored needles that always form a cross pattern (perpendicular to each other) and rotate together at the same speed.

The competition starts with the black needle at position 0° (12 o'clock). All four needles move at the same speed and always maintain a cross formation. Your task is to calculate where each of the four colored needles points after *m* minutes and *s* seconds have elapsed:

- **Black needle**: Starts at second 0 (12 o'clock position)
- **Red needle**: Starts 3 o'clock relative position
- **Green needle**: Starts 6 o'clock relative position
- **Blue needle**: Starts 9 o'clock relative position

All needles complete one full rotation in exactly 60 seconds. The watch face is marked with second positions from 0 to 59, where 0 is at 12 o'clock, 15 is at 3 o'clock, 30 is at 6 o'clock, and 45 is at 9 o'clock.

Input

Two positive integers on separate lines:
First line: m - the number of minutes elapsed
Second line: s - the number of seconds elapsed

Output

Four lines, each containing a single integer representing which second mark (0-59) each needle is pointing to:

First line: Black needle position (as a second mark 0-59)
Second line: Red needle position (as a second mark 0-59)
Third line: Green needle position (as a second mark 0-59)
Fourth line: Blue needle position (as a second mark 0-59)

Example 1

Input

```
0
15
```

Output

```
15
30
45
0
```

Example 2

Input

```
0
30
```

Output

```
30
45
0
15
```

Example 3

Input

```
1
0
```

Output

```
0
15
30
45
```

C++

```cpp
#include <iostream>
using namespace std;

int main() {
    int minutes, seconds;
    cin >> minutes >> seconds;

    // Calculate total seconds and find black needle position
    int totalSeconds = minutes * 60 + seconds;
    int black = totalSeconds % 60;

    // All needles form a cross (15 seconds apart)
    int red = (black + 15) % 60;
    int green = (black + 30) % 60;
    int blue = (black + 45) % 60;

    cout << black << endl;
    cout << red << endl;
    cout << green << endl;
    cout << blue << endl;

    return 0;
}
```

4 Core Computation

5 points

Introduction

In the year 3026, humanity has discovered an ancient alien data core containing encrypted numerical sequences. Scientists believe that each sequence holds a hidden key at its center, critical for unlocking advanced technology. As a top cybernetic analyst, your task is to extract this core value from given numbers. Given an integer N, can you write a program that finds out its core value: if N has an odd number of digits, return the middle digit, but if N has an even number of digits, return the rounded average of the two middle digits as an integer.

Input

A single N integer (positive or negative, up to 18 digits long).

Output

A single positive integer representing the core value.

Example 1

Input

```
12345
```

Output

```
3
```

Example 2

Input

```
-98765
```

Output

```
7
```

Example 3

Input

```
2468
```

Output

```
5
```

Python

```python
N = input()

# Remove the negative sign if it exists
if N[0] == "-":
    N = N[1:]

# Check if the number of digits is even or odd
# If even, get the middle two digits and calculate the average
if len(N)%2 == 0:
    average = (int(N[len(N)//2 - 1]) + int(N[len(N)//2])) / 2
    print(int(average))

else:
    middle_digit = N[(len(N)//2)]
    print(middle_digit)
```

5 Clone Replicator

6 points

Introduction

You're working at the galaxy's leading cloning facility. The company has a revolutionary replication system where clones can create more clones - but there's a mathematical pattern to it.

When you create a Generation 1 clone, it can produce 1 offspring ($1^3 = 1$). A Generation 2 clone is more stable and can produce 8 offspring ($2^3 = 8$). Generation 3 clones are even better, producing 27 offspring ($3^3 = 27$), and so on. The replication capacity grows cubically with each generation!

Your mission: Write a program for the facility's computer system that displays the replication capacity for each generation up to a given number. The board of directors needs this report for their presentation, and they want it formatted exactly right. After all, precision matters when you're cloning the future of humanity.

Input

A single positive integer N ($1 \leq N \leq 100$) representing the number of generations.

Output

For each generation from 1 to N, print the following line:

```
Generation G = C offspring
```

Example

Input

```
5
```

Output

```
Generation 1 = 1 offspring
Generation 2 = 8 offspring
Generation 3 = 27 offspring
Generation 4 = 64 offspring
Generation 5 = 125 offspring
```

Python

```python
n = int(input())

for i in range(1, n + 1):
    cube = i ** 3
    print(f"Generation {i} = {cube} offspring")
```

6 The Stuttering Spell

6 points

Introduction

You're an apprentice wizard at the Arcane Academy of Mystical Arts, and something went terribly wrong during today's Enchantment class. Professor Mumblebore was demonstrating a new spell called "Echo Incantation" when his wand misfired, causing every magical text in the library to become cursed!

Now all the ancient spellbooks, potion recipes, and magical scrolls have their letters doubled. "Fireball" became "FFiirreebbaalll", "Dragon" turned into "DDrraaggoonn", and even simple words like "Wand" are now "WWaanndd". The librarian is panicking because students can't read their homework assignments!

Write a program to reverse the curse and restore the magical texts to their original form by removing the duplicate letters. The spell only affected letters - numbers, spaces, and punctuation marks (including commas, periods, colons, semicolons, question and exclamation marks) remained unaffected. You'll need to process multiple lines of cursed text until you encounter the word "STOP" on its own line, which signals the end of the damaged section.

Input

There are multiple lines of text, up to 125 characters per line. The input ends when the word `STOP` appears alone on its own line.

Output

For each line of input (except "STOP"), output the corrected text with duplicate consecutive letters removed.

Example

Input

```
TThhee aanncciieennntt ssppeellll rreeqquuiirreess 3 iinnggrreeddiieenntts:
1 ccuupp ooff ddrraaggoonn sssccaalleess,
2 ddrrooppss ooff pphhooeenniixx tteeaarrss,
aanndd aa ppiinncchh ooff ssttaarrdduusstt ffrroomm
tthhee NNoorrtthheerrnn SSkkiieess.
WWaarrnniinngg: DDoo nnoott ssuubbssttiittuuttee iinnggrreeddiieenttss!
CCaann yyoouu ffiixx tthhiiss?
STOP
```

Output

```
The ancient spell requires 3 ingredients:
1 cup of dragon scales,
2 drops of phoenix tears,
and a pinch of stardust from
the Northern Skies.
Warning: Do not substitute ingredients!
Can you fix this?
```

Python

```python
line = input()

while line != "STOP":
    result = []
    i = 0
    while i < len(line):
        char = line[i]
        result.append(char)

        # If it's a letter and the next character is the same letter, skip the
duplicate
        if char.isalpha() and i + 1 < len(line) and line[i + 1] == char:
            i += 2  # Skip both the current and next character
        else:
            i += 1  # Move to next character

    print(''.join(result))

    # Read next line
    line = input()
```

7 Pentavocalic

6 points

Introduction

You've been hired as a Vowel Detective by the Secret Word Agency! Your job is to catch words that have all five vowels (a, e, i, o, u) exactly once each. These special words are called "pentavocalic".

Can you create a simple program that, given a word, finds out whether it is pentavocalic?

Input

A single word in lowercase letters.

Output

If the word is pentavocalic, print "Pentavocalic". Otherwise, print "Not pentavocalic"

Example 1

Input

```
murcielago
```

Output

```
Pentavocalic
```

Example 2

Input

```
ferrocarril
```

Output

```
Not pentavocalic
```

Python

```python
def is_pentavocalic(word):
    """
    Check if a word is pentavocalic (contains each vowel exactly once).

    Args:
        word (str): The word to check

    Returns:
        bool: True if the word contains each vowel (a, e, i, o, u) exactly once
    """
    vowels = 'aeiou'

    # Count occurrences of each vowel
    vowel_counts = {}
    for vowel in vowels:
        vowel_counts[vowel] = word.count(vowel)

    # Check if each vowel appears exactly once
    return all(count == 1 for count in vowel_counts.values())

# Example usage:
if __name__ == "__main__":
    word = input()
    if is_pentavocalic(word):
        print("Pentavocalic")
    else:
        print("Not pentavocalic")
```

8 Echoes Of Camel Case

6 points

Introduction

After a school literary fair, some students argue that for AI to feel more "human," variable names should read like character names from a novel. Your task is to convert snake_case identifiers (like hero_with_headphones) into CamelCase (like HeroWithHeadphones), making the code feel more natural.

Hint: Focus on trimming underscores and capitalizing the first letter of every word fragment, even the very first one.

Input

The first line contains an integer n (1 ≤ n ≤ 1000) - the number of identifiers.
The next n lines each contain a non-empty snake_case identifier composed only of lowercase English letters and underscores. There cannot be two consecutives '_'. The maximum length of a snake word is 100 characters

Output

For each identifier, print its CamelCase reconstruction on its own line.

Example 1

Input

```
3
hero_with_headphones
art_club
late_night_reader
```

Output

```
HeroWithHeadphones
ArtClub
LateNightReader
```

Example 2

Input

```
2
river_song
second_breakfast
```

Output

```
RiverSong
SecondBreakfast
```

Python

```python
def snake_to_camel(snake_str):
    """Convert snake_case to CamelCase (PascalCase)"""
    words = snake_str.split('_')
    return ''.join(word.capitalize() for word in words)

def main():
    n = int(input())
    for _ in range(n):
        identifier = input().strip()
        print(snake_to_camel(identifier))

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

9 Frequency-Hopping Spread Spectrum

6 points

Introduction

When you connect Bluetooth headphones, when a phone searches for Wi-Fi, or when a drone communicates with its controller, the signal travels through an environment full of interference: other devices, walls, electromagnetic noise and many users sharing the same channels. To avoid losing the connection, many systems use FHSS (Frequency-Hopping Spread Spectrum). This technique rapidly switches between different frequencies following a mathematical sequence known only to the transmitter and the receiver. If both follow the same pattern, they can stay synchronized even when some frequencies are blocked or noisy.

The idea traces back to Hedy Lamarr, an actress and inventor who, during World War II, co-designed an early frequency-hopping system to prevent radio-guided torpedoes from being jammed. Today, the same principle is at the core of technologies like Bluetooth and some Wi-Fi standards.

In this challenge, you will model a mini FHSS system. You need to generate the expected frequency hopping sequence and compare it with the received sequence to calculate the synchronization quality.

Input

The input consists of two lines. First line contains the *seed*, an integer representing the starting frequency. Second line is the *received sequence*, that is, a list of integers representing the frequencies detected by the receiver.

Frequency Generation Rule

The transmitter generates the next frequency based on the current one using the following formula:

$$next\ freq = (current\ freq \cdot 17 + 5)\ \%\ 256$$

where % stands for modulo operation.

The sequence starts with the *seed* as the first element. The length of the generated sequence should match the length of the *received sequence*.

Given a *seed* of 10 and a *received sequence* of 10, 175, 164, 233 this is of how it works:

$$Gen[0] = 10\ (seed)$$
$$Gen[1] = (10 * 17 + 5)\ \%\ 256 = 175$$
$$Gen[2] = (175 * 17 + 5)\ \%\ 256 = 2980\ \%\ 256 = 164$$
$$Gen[3] = (164 * 17 + 5)\ \%\ 256 = 2793\ \%\ 256 = 233$$

$Generated$: $[10, 175, 164, 233]$
$Received$: $[10, 175, 164, 233]$
$Matches$: $4/4 = 1.0$ since generated and received frequencies match.

Given a *seed* of 10 and a *received sequence* of 10, 0, 164, 0 this is of how it works:

$$Gen[0] = 10\ (seed)$$
$$Gen[1] = (10 * 17 + 5)\ \%\ 256 = 175$$
$$Gen[2] = (175 * 17 + 5)\%\ 256 = 2980\ \%\ 256 = 164$$
$$Gen[3] = (164 * 17 + 5)\ \%\ 256 = 2793\ \%\ 256 = 233$$

$Generated$: $[10, 0, 164, 0]$
$Received$: $[10, 175, 164, 233]$
$Matches$: $2/4 = 0.5$ because frequencies match only 10 and 164 values.

Output

Return a decimal value representing the synchronization score, which is the ratio of matching frequencies to the total number of frequencies. The result should be rounded to 2 decimal places.

Example 1

Input

```
10
10 175 164 233
```

Output

```
1.00
```

Example 2

Input

```
10
10 0 164 0
```

Output

```
0.50
```

Example 3

Input

```
10
10 176
```

Output

```
0.50
```

Example 4

Input

```
0
0 5 0
```

Output

```
0.67
```

Python

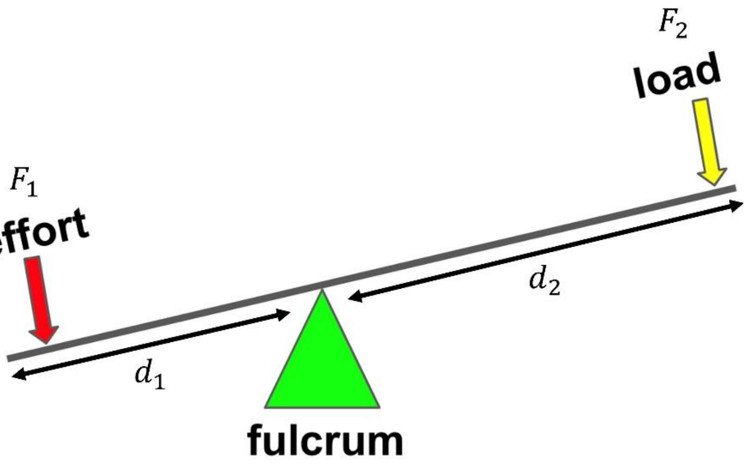
```python
def frequency_hopping_spread_spectrum(seed, received_sequence):
    """
    Calculates the synchronization score between the generated FHSS sequence
    and the received sequence.

    Args:
    seed: int, starting frequency
    received_sequence: List of int, frequencies received

    Returns:
    float: Synchronization score (matches / total), rounded to 2 decimals
    """

    if not received_sequence:
        return 0.0

    matches = 0
    current_freq = seed

    # Iterate through the received sequence
    for received_freq in received_sequence:
        # Check if the current generated frequency matches the received one
        if current_freq == received_freq:
            matches += 1

        # Generate the next frequency for the next iteration
        # Rule: next_freq = (current_freq * 17 + 5) % 256
        current_freq = (current_freq * 17 + 5) % 256

    # Calculate ratio
    score = matches / len(received_sequence)

    return round(score, 2)

if __name__ == "__main__":
    # Example usage
    seed = int(input().strip())
    received_sequence = list(map(int, input().strip().split()))
    result = frequency_hopping_spread_spectrum(seed, received_sequence)
    print(f"{result:.2f}")
```

10 First-Class Lever Simulator

7 points

Introduction

A first-class lever consists of a rigid bar that pivots around a fulcrum placed between the effort force and the load.



The balance of forces follows the equation $F_1 \times d_1 = F_2 \times d_2$ where:

- F_1 is the applied effort force,
- d_1 is the distance of the effort force from the fulcrum,
- F_2 is the load force,
- d_2 is the distance of the load from the fulcrum.

Your task is to write a program that determines whether a given lever system is in equilibrium. If not, the program should calculate the missing force required to balance the system. To keep things simple, the lever is assumed to be weightless.

Input

Four lines with a single value in this order:

- F_1 is the effort force is a positive integer between 1 and 1000 Newtons,
- d_1 is the effort arm length is a positive integer between 1 and 10 meters,
- F_2 is the load force is an integer between 1 and 1000 Newtons, but could be -1 if unknown
- d_2 is the load arm length is a positive integer between 1 and 10 meters.

Output

- If the system is balanced, simply print: `"The lever is in equilibrium."`
- If the system is unbalanced and F_2 is unknown, calculate and print the required load force rounding to two decimal places: `"To balance the lever, the load force should be X.YY Newtons."`
- If the system is unbalanced and F_1 needs to be adjusted, print: “`The system is unbalanced. Adjust the effort force.`”

Example 1

Input

```
50
2
-1
5
```

Output

```
To balance the lever, the load force should be 20.00 Newtons.
```

Example 2

Input

```
30
3
45
2
```

Output

```
The lever is in equilibrium.
```

Example 3

Input

```
40
4
20
3
```

Output

```
The system is unbalanced. Adjust the effort force.
```

C++

```
#include <iostream>
#include <iomanip>

int main() {
    int F1, d1, F2, d2;
    std::cin >> F1 >> d1 >> F2 >> d2;

    // Calculate the moments
    int moment1 = F1 * d1;
    int moment2 = F2 * d2;

    if (F2 == -1) {
        // Calculate the required load force to balance the lever
        double required_F2 = static_cast<double>(moment1) / d2;
        std::cout << "To balance the lever, the load force should be " <<
std::fixed << std::setprecision(2) << required_F2 << " Newtons." << std::endl;
    } else if (moment1 == moment2) {
        // The lever is in equilibrium
        std::cout << "The lever is in equilibrium." << std::endl;
    } else {
        // The system is unbalanced and F1 needs to be adjusted
        std::cout << "The system is unbalanced. Adjust the effort force." <<
std::endl;
    }

    return 0;
}
```

11 Automorphic Numbers

7 points

Introduction

Your friend Katherine, a math enthusiast like the character from the movie "Hidden Figures", just discovered a fascinating mathematical phenomenon while playing with a calculator! She noticed that some numbers have a special property: when you square them, the result ends with the original number itself. For example:

- $1^2 = 1$ (ends in **1**)
- $5^2 = 25$ (ends in **5**)
- $6^2 = 36$ (ends in **6**)
- $76^2 = 5776$ (ends in **76**)

These are called Automorphic Numbers - numbers that appear at the end of their own square! However, not all numbers have this magical property:

- $2^2 = 4$ (doesn't end in **2**)
- $3^2 = 9$ (doesn't end in **3**)
- $10^2 = 100$ (doesn't end in **10**)

Katherine challenges you to write a program that can quickly determine if any given number is automorphic or not.

Input

A single positive integer N ($1 \leq N \leq 10000$)

Output

Print a single line with **Automorphic** if N is an automorphic number otherwise print **Not Automorphic**

Example 1

Input

```
1
```

Output

```
Automorphic
```

Example 2

Input

```
3
```

Output

```
Not Automorphic
```

Python

```python
n = int(input())

# Calculate the square
square = n * n

# Convert to strings for easy comparison
n_str = str(n)
square_str = str(square)

# Check if the last digits of square equal n
# Get the last len(n_str) characters from square_str
if square_str.endswith(n_str):
    print("Automorphic")
else:
    print("Not Automorphic")
```

12 The Mirror Game

7 points

Introduction

A mirror number is a number that looks the same when viewed in a mirror. Certain digits have special mirror properties:

- **Symmetric digits** (mirror to themselves):

  $0 \leftrightarrow 0$ $1 \leftrightarrow 1$ $8 \leftrightarrow 8$

- **Mirror pairs** (mirror to each other):

  $2 \leftrightarrow 5$ $6 \leftrightarrow 9$

- **Invalid digits** (have no mirror): 3, 4 and 7

For a number to be a mirror number all its digits must be valid (0, 1, 2, 5, 6, 8, 9) and when you mirror each digit AND reverse the number, you get back the original number. Can you write a program that determines if a number is mirror?

Input

A single line containing a positive integer N ($1 \leq N \leq 10^9$).

Output

Print **MIRROR** if the number is a mirror number otherwise print **NOT MIRROR**.

Example 1	Example 2	Example 3	Example 4	Example 5
Input	**Input**	**Input**	**Input**	**Input**
`101`	`69`	`123`	`25`	`1`
Output	**Output**	**Output**	**Output**	**Output**
`MIRROR`	`MIRROR`	`NOT MIRROR`	`MIRROR`	`MIRROR`

Example 6

Input

```
80
```

Output

```
NOT MIRROR
```

Python

```python
def is_mirror_number(n):
    """
    Check if a number is a mirror number.

    A mirror number is one where:
    1. All digits are valid (0,1,2,5,6,8,9)
    2. When mirrored and reversed, it equals the original

    Args:
        n (str): The number as a string

    Returns:
        bool: True if it's a mirror number, False otherwise
    """
    # Define mirror mappings
    mirror = {
        '0': '0',
        '1': '1',
        '2': '5',
        '5': '2',
        '6': '9',
        '8': '8',
        '9': '6'
    }

    # Check if all digits are valid (can be mirrored)
    for digit in n:
        if digit not in mirror:
            return False

    # Mirror each digit and reverse
    mirrored = ''.join(mirror[digit] for digit in n)
    mirrored_reversed = mirrored[::-1]

    # Check if mirrored & reversed equals original
    return mirrored_reversed == n

# Read input
n = input().strip()
# Check if it's a mirror number
if is_mirror_number(n):
    print("MIRROR")
else:
    print("NOT MIRROR")
```

13 Compression Algorithm

7 points

Introduction

Your phone's storage is almost full! You've got thousands of messages, and many of them have looooong sequences of repeated characters (like when your friend types "yaaaaaay" or "noooooo").

You decide to write a compression algorithm to save space. The idea is simple: instead of writing the same character multiple times, just write the character followed by how many times it appears!

But here's the clever part: only compress sequences of 3 or more repeated characters. Why? Because compressing aa to a2 doesn't save any space - it's the same length! Your algorithm should be smart about when compression actually helps.

For example:
- aaaaa becomes a5 (the letter a appears 5 times)
- hello stays hello (no repeated consecutive characters)

The Rule:
- If a character repeats 3+ times consecutively: write it once followed by the count
- If a character repeats 1-2 times: just write it normally

So aaaaabbbbccdddddefgg becomes a5b4ccd5efgg (notice that cc and gg stay as-is because they're only 2 characters).

Input

A single line containing a string s consisting of lowercase letters only. The string length will be between 1 and 10,000 characters.

Output

Output a single line with the compressed version of the string following the rules above.

Example 1

Input

```
aaaaabbbbccdddddefgg
```

Output

```
a5b4ccd5efgg
```

Example 2

Input

```
programming
```

Output

```
programming
```

Python

```python
s = input()

if len(s) == 0:
    print()
    exit()

result = []
i = 0

while i < len(s):
    current_char = s[i]
    count = 1

    # Count consecutive occurrences
    while i + count < len(s) and s[i + count] == current_char:
        count += 1

    # Compress only if 3 or more
    if count >= 3:
        result.append(current_char + str(count))
    else:
        result.append(current_char * count)

    i += count

print(''.join(result))
```

14 Pintan Bastos

7 points

Introduction

"Pintan bastos" is a common Spanish idiom meaning "things are looking bad," "the situation is getting tough," or "the going's getting rough," originating from the Spanish deck of cards where 'bastos' (clubs) are the lowest-ranking suit, signaling a difficult or unfavorable turn of events.

You're creating a card game app and need to display Spanish card names. Instead of storing all 40 card names, you'll use numbers 0 to 39 and convert them to card names when needed. The Spanish deck has 40 cards, that is, 4 suits × 10 ranks each.

Why store 40 card names when you can be lazy like a programmer and just remember 14? That's the spirit of coding: work smarter, not harder! Your task is to write a program that converts card numbers to their Spanish names.

Suits	
0	Oros
1	Copas
2	Espadas
3	Bastos

Ranks	
0	As
1	2
2	3
3	4
4	5
5	6
6	7
7	Sota
8	Caballo
9	Rey

Think of the 40 cards as being organized in 4 groups of 10 like these:

- From card 0 to card 9 the suits are Oros.
- From card 10 to card 19 the suits are Copas.
- From card 20 to card 29 the suits are Espadas.
- From card 30 to card 39 the suits are Bastos.

For card 14, since it belongs to Copas suit and 4 represents rank 5, it is the 5-de-Copas.

Input

The input is composed of several lines where:
- first line is the number of cards (n) to convert to name
- next n lines are one card number per line (each from 0 to 39)

Output

Print the card names in format "Rank-de-Suit" one per line.

Example

Input

```
5
25
32
39
20
6
```

Output

```
6-de-Espadas
3-de-Bastos
Rey-de-Bastos
As-de-Espadas
7-de-Oros
```

Python

```python
def card_name(value):
    """
    Convert a card value (0-39) to its Spanish name.

    Args:
        value: Integer from 0 to 39

    Returns:
        String in format "Rank-de-Suit"
    """
    suits = ['Oros', 'Copas', 'Espadas', 'Bastos']
    ranks = ['As', '2', '3', '4', '5', '6', '7', 'Sota', 'Caballo', 'Rey']

    suit_index = value // 10
    rank_index = value % 10

    suit = suits[suit_index]
    rank = ranks[rank_index]

    return f"{rank}-de-{suit}"

"""Main function to process card values."""
n = int(input().strip())

# Read each card value on a separate line
values = []
for _ in range(n):
    values.append(int(input().strip()))

# Convert each card value to its name
card_names = [card_name(value) for value in values]

# Print all card names in separate lines
for card in card_names:
    print(card)
```

15 Pace Calculator

8 points

Introduction

🏃 You've just downloaded a fancy new running app, but it's showing your speed in km/h and you're confused! "How fast am I really going?" you wonder. All the cool runners talk about their "pace" - that mysterious mm:ss.s/km number that somehow makes more sense when you're huffing and puffing up a hill.

When tracking running performance, athletes often use "pace" instead of speed. Pace is measured in minutes per kilometer (min/km) and represents how long it takes to run one kilometer.

The tricky part is that pace uses a time format (mm:ss.s) rather than decimal numbers. For example, if you run 5 kilometers in 23 minutes and 20 seconds, your pace is 4:40.0 per kilometer (4 minutes and 40.0 seconds per kilometer).

Your task is to write a program that calculates the pace given the distance run and the total time taken. Help yourself and fellow runners understand their true performance!

Formula:
- Total time in seconds = minutes × 60 + seconds
- Pace in seconds per km = total time in seconds ÷ distance in km
- Convert back to mm:ss.s format

Input

The first line contains a single floating-point number distance representing the distance run in kilometers.

The second line contains the total time in the format hh:mm:ss, where hh is hours, mm is minutes and ss is seconds.

Output

Output a single line with the pace in the format mm:ss.s/km, representing minutes and seconds per kilometer. Always use exactly 2 digits for seconds (with leading zero if needed) and exactly 1 decimal place.

Example 1

Input

```
5.0
00:23:20
```

Output

```
4:40.0/km
```

Example 2

Input

```
10.5
00:52:30
```

Output

```
5:00.0/km
```

Python

```python
# Read input
distance = float(input())
time_str = input()

# Parse time (hh:mm:ss)
time_parts = time_str.split(':')
hours = int(time_parts[0])
minutes = int(time_parts[1])
seconds = int(time_parts[2])

# Convert total time to seconds
total_seconds = hours * 3600 + minutes * 60 + seconds

# Calculate pace in seconds per km
pace_seconds = total_seconds / distance

# Convert to mm:ss.s format
pace_minutes = int(pace_seconds // 60)
pace_secs = pace_seconds % 60

# Output with proper formatting
print(f"{pace_minutes}:{pace_secs:04.1f}/km")
```

16 Scrabble Scoring

8 points

Introduction

Scrabble is a classic word game where players use letter tiles to form words on a game board. Each letter has a point value, and words placed strategically can earn bonus points. Players take turns building words, connecting them to existing ones on the board. The goal is to score the highest points by forming meaningful words while using special spaces like double or triple word scores.

These are the point values for each letter in Scrabble:

Points	Letters
0	The blank tile as character _
1	A, E, I, L, N, O, R, S, T and U
2	D and G
3	B, C, M and P
4	F, H, V, W and Y
5	K
8	J and X
10	Q and Z

Can you create a program that takes two words –one for Player 1 and one for Player 2– and determines which word has the higher score? The program should report the winning player
and the point difference. Additionally, it should account for the possibility of a tie.

Input

It consists of two words, each written in capital letters and presented on separate lines. First line contains the word submitted by Player 1 while second line is for the word submitted by Player 2

Output

If Player 1's word has a higher score: "Player 1 wins by X points". If Player 2's word has a higher score: "Player 2 wins by X points". In case of a tie: “Tie by 7 points”

Example 1

Input

```
CODE
WARS
```

Output

```
Tie by 7 points
```

Example 2

Input

```
HEL_O
RADAR
```

Output

```
Player 1 wins by 1 points
```

Example 3

Input

```
MAGAZINE
TRAVELLER
```

Output

```
Player 1 wins by 8 points
```

Python

```python
"""
These are the point values for each letter used in Scrabble game:

0 Points - Blank tile "_"
1 Point - A, E, I, L, N, O, R, S, T and U
2 Points - D and G
3 Points - B, C, M and P
4 Points - F, H, V, W and Y
5 Points - K
8 Points - J and X
10 Points - Q and Z
"""

def getScrabbleScore(word):
    letter_values = {
        "_": 0,
        "A": 1, "E": 1, "I": 1, "L": 1, "N": 1, "O": 1, "R": 1, "S": 1, "T": 1,
"U": 1,
        "D": 2, "G": 2,
        "B": 3, "C": 3, "M": 3, "P": 3,
        "F": 4, "H": 4, "V": 4, "W": 4, "Y": 4,
        "K": 5,
        "J": 8, "X": 8,
        "Q": 10, "Z": 10
    }
    score = 0
    for letter in word.upper():
        score += letter_values.get(letter, 0)
    return score

# Read the two words from input
word1 = input()
word2 = input()

# Calculate the scores for both words
score1 = getScrabbleScore(word1)
score2 = getScrabbleScore(word2)

# Compare the scores and print the result
if score1 > score2:
    print("Player 1 wins by " + str(score1 - score2) + " points")
elif score2 > score1:
    print("Player 2 wins by " + str(score2 - score1) + " points")
else:
    print("Tie by " + str(score1) + " points")
```

17 Radio Chatter

9 points

Introduction

The static crackles. A secret agent leans closer to the receiver. Suddenly, a burst of radio chatter comes through—no ordinary words, but a long stream of the radiotelephony alphabet. Every letter of the English alphabet is disguised as its corresponding codeword:

A → ALPHA	J → JULIETT	S → SIERRA
B → BRAVO	K → KILO	T → TANGO
C → CHARLIE	L → LIMA	U → UNIFORM
D → DELTA	M → MIKE	V → VICTOR
E → ECHO	N → NOVEMBER	W → WHISKEY
F → FOXTROT	O → OSCAR	X → XRAY
G → GOLF	P → PAPA	Y → YANKEE
H → HOTEL	Q → QUEBEC	Z → ZULU
I → INDIA	R → ROMEO	

Your mission is clear: decode the incoming radio chatter. The transmission is one continuous string of radiotelephony codewords glued together. Your job is to break it apart and recover the hidden message in plain English letters.

Input

A single line with a string of characters.

Output

Print the message

Example

Input

```
HOTELECHOLIMALIMAOSCAR
```

Output

```
HELLO
```

Python

```python
# Read the ciphered message from input
ciphered_message = input()

# Define the phonetic alphabet mapping
reverse_phonetic = {'ALPHA': 'A', 'BRAVO': 'B', 'CHARLIE': 'C', 'DELTA': 'D',
'ECHO': 'E', 'FOXTROT': 'F', 'GOLF': 'G', 'HOTEL': 'H', 'INDIA': 'I', 'JULIET':
'J', 'KILO': 'K', 'LIMA': 'L', 'MIKE': 'M', 'NOVEMBER': 'N', 'OSCAR': 'O',
'PAPA': 'P', 'QUEBEC': 'Q', 'ROMEO': 'R', 'SIERRA': 'S', 'TANGO': 'T',
'UNIFORM': 'U', 'VICTOR': 'V', 'WHISKEY': 'W', 'XRAY': 'X', 'YANKEE': 'Y',
'ZULU': 'Z'}

# Decode the message
decoded_message = ""
i = 0
while i < len(ciphered_message):
    found = False
    # Try each codeword to see if it matches at current position
    for codeword, letter in reverse_phonetic.items():
        if ciphered_message[i:].upper().startswith(codeword):
            decoded_message += letter
            i += len(codeword)
            found = True
            break
    if not found:
        i += 1  # Skip unrecognized character

# Output the decoded message
print(decoded_message)
```

18 Transpose Chords

11 points

Introduction

Your band just got an amazing gig, but there's a problem! The singer has a cold and can't hit high notes. "Can we play everything a bit lower?" they plead. No worries - you've got this!

Musical chords are built from notes in a chromatic scale. The chromatic scale consists of 12 semitones that repeat in octaves:

C, C#, D, D#, E, F, F#, G, G#, A, A#, B

Think of it like a circular clock with 12 hours, but instead of hours, we have notes!

Note that there are two ways to name some notes:
- Sharp notation: C# (C sharp) means one semitone above C
- Flat notation: Db (D flat) means one semitone below D

For this problem, we'll use sharp notation exclusively in our output (because musicians like to keep things sharp 😎). But beware that flats may appear in the input and must be converted to sharps in the output.

Transposing a chord means shifting it up or down by a certain number of semitones. For example, transposing C up by 2 semitones gives D, and transposing E down by 3 semitones gives C#.

Chords can have modifiers that don't change during transposition:
- m for minor (e.g., Am, Dm)
- maj7 for major seventh (e.g., Cmaj7)
- 7 for dominant seventh (e.g., G7)
- And combinations like m7, maj9, etc.

Only the root note changes; modifiers stay the same. It's like keeping your chord's personality but changing its pitch!

Remember that the chromatic scale is circular: after B comes C again. When transposing, if you go past B, wrap around to C. Similarly, going below C wraps to B.

Input

The first line contains a single integer n representing the number of semitones to transpose (can be negative).

The second line contains a string with chord names separated by spaces. Each chord consists of:
- A note name (A, B, C, D, E, F, or G)
- Optionally followed by # (sharp) or b (flat)
- Optionally followed by modifiers (any combination of letters and numbers like m, 7, maj7, m7, etc.)

Output

Output a single line with the transposed chords separated by spaces. Always use sharp notation (`#`) in the output, never flat notation (`b`). Preserve all modifiers exactly as they appear in the input.

Example 1

Input

```
2
C G Am F
```

Output

```
D A Bm G
```

Example 2

Input

```
-3
D# Fm7 Bb G#maj7
```

Output

```
C Dm7 G Fmaj7
```

Python

```python
# Chromatic scale using sharp notation
CHROMATIC = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']

# Map flat notes to their sharp equivalents
FLAT_TO_SHARP = {
    'Db': 'C#',
    'Eb': 'D#',
    'Gb': 'F#',
    'Ab': 'G#',
    'Bb': 'A#'
}

def parse_chord(chord):
    """Parse a chord into root note and modifiers."""
    # Check for flat notation first (two characters)
    if len(chord) >= 2 and chord[1] == 'b':
        root = chord[:2]
        modifiers = chord[2:]
    # Check for sharp notation
    elif len(chord) >= 2 and chord[1] == '#':
        root = chord[:2]
        modifiers = chord[2:]
    # Natural note
    else:
        root = chord[0]
        modifiers = chord[1:]

    return root, modifiers

def transpose_note(note, semitones):
    """Transpose a single note by the given number of semitones."""
    # Convert flat to sharp if needed
    if note in FLAT_TO_SHARP:
        note = FLAT_TO_SHARP[note]

    # Find current position in chromatic scale
    current_index = CHROMATIC.index(note)

    # Calculate new position (modulo 12 for wraparound)
    new_index = (current_index + semitones) % 12

    return CHROMATIC[new_index]

def transpose_chord(chord, semitones):
    """Transpose a chord by the given number of semitones."""
    root, modifiers = parse_chord(chord)
    new_root = transpose_note(root, semitones)
    return new_root + modifiers

# Read input
n = int(input())
chords = input().split()

# Transpose each chord and output
transposed = [transpose_chord(chord, n) for chord in chords]
print(' '.join(transposed))
```

19 Expected Goals

13 points

Introduction

Expected Goals (xG) is a statistical metric that estimates the probability of a shot resulting in a goal based on various factors. In this simplified model, we consider two key factors:

- Distance: How far the shot was taken from the goal (in meters)
- Angle: The shooting angle relative to the goal center (in degrees, being 0° = straight on and 90° = extreme angle)

The xG for each shot is calculated using the formula:

$$xG = A \cdot e^{(-B \cdot distance)} \cdot e^{(-C \cdot angle)}$$

where:

- A = 0.95 (maximum probability scaling factor)
- B = 0.12 (distance decay rate)
- C = 0.01 (angle decay rate)
- e is Euler's number (~2.718)

The total xG for a team is the sum of xG values for all their shots. Based on the xG difference, we predict the match outcome:

- If |home_xG - away_xG| ≤ 0.5: Expected result is a draw
- If home_xG > away_xG (by more than 0.5): Expected result is home win
- If away_xG > home_xG (by more than 0.5): Expected result is away win

Can you write a program that calculates the total xG for each team, predicts the expected result, and compares it with the actual match outcome?

Hint: When no shots are provided for a team, their total xG is 0. Distance must be between 0-105 meters (length of a football pitch), and angle must be between 0-90 degrees.

Input

The input consists of three lines:

Line 1: Home team shots as comma-separated distance-angle pairs

- Format: distance1,angle1,distance2,angle2,...
- Distance and angle values are floats (decimal numbers)
- Example: 10,15,20,30 means two shots: one from 10m at 15°, another from 20m at 30°
- Empty input means no shots (total xG = 0)

Line 2: Away team shots in the same format as Line 1

Line 3: Actual match result in format X-Y (no spaces)

- X = home goals (non-negative integer)
- Y = away goals (non-negative integer)
- Example: 2-1 means home team scored 2, away team scored 1

Output

The output should display the match xG summary with exactly the following format:

```
--- Match xG summary ---
Expected result: <home/away/draw>
Actual result: <home/away/draw>
Match expected?: <YES/NO>
```

Note: The actual result is determined by comparing goals scored:

- If home goals > away goals: actual result is home
- If away goals > home goals: actual result is away
- If goals are equal: actual result is draw

Example 1

Input

```
6,0,10,10,15,20
25,25,30,30
2-1
```

Output

```
--- Match xG summary ---
Expected result: home
Actual result: home
Match expected?: YES
```

Example 2

Input

```
25,35,30,40
10,5,9,8
0-2
```

Output

```
--- Match xG summary ---
Expected result: away
Actual result: away
Match expected?: YES
```

Example 3

Input

```
15.5,20.3
18.2,25.7
1-1
```

Output

```
--- Match xG summary ---
Expected result: draw
Actual result: draw
Match expected?: YES
```

Example 4

Input

```
15,20
18,22
0-2
```

Output

```
--- Match xG summary ---
Expected result: draw
Actual result: away
Match expected?: NO
```

Example 5

Input

```
18,21,20,30.1

1-0
```

Output

```
--- Match xG summary ---
Expected result: draw
Actual result: home
Match expected?: NO
```

Python

```python
import math

# Constants for xG calculation
A = 0.95  # Maximum probability scaling factor
B = 0.12  # Distance decay rate
C = 0.01  # Angle decay rate

def calculate_xg(distance: float, angle: float) -> float:
    """Calculate xG for a single shot."""
    return A * math.exp(-B * distance) * math.exp(-C * angle)

def calculate_total_xg(shots_line: str) -> float:
    """Calculate total xG for a team from their shots input."""
    if not shots_line.strip():
        return 0.0

    values = [float(x) for x in shots_line.strip().split(',')]
    total_xg = 0.0

    # Process pairs of (distance, angle)
    for i in range(0, len(values), 2):
        distance = values[i]
        angle = values[i + 1]
        total_xg += calculate_xg(distance, angle)

    return total_xg

def get_expected_result(home_xg: float, away_xg: float) -> str:
    """Determine expected result based on xG difference."""
    diff = home_xg - away_xg
    if abs(diff) <= 0.5:
        return "draw"
    elif diff > 0:
        return "home"
    else:
        return "away"

def get_actual_result(home_goals: int, away_goals: int) -> str:
    """Determine actual result based on goals scored."""
    if home_goals > away_goals:
        return "home"
    elif away_goals > home_goals:
        return "away"
    else:
        return "draw"

def main():
    # Read input
    home_shots = input()
    away_shots = input()
    score = input().strip()

    # Calculate xG for each team
    home_xg = calculate_total_xg(home_shots)
    away_xg = calculate_total_xg(away_shots)

    # Parse actual score
    home_goals, away_goals = map(int, score.split('-'))

    # Determine results
    expected_result = get_expected_result(home_xg, away_xg)
    actual_result = get_actual_result(home_goals, away_goals)

    # Check if match expected
    match_expected = "YES" if expected_result == actual_result else "NO"

    # Output
    print("--- Match xG summary ---")
    print(f"Expected result: {expected_result}")
    print(f"Actual result: {actual_result}")
    print(f"Match expected?: {match_expected}")

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

20 Orbital Trajectory Verification

13 points

Introduction

In the 1960s, during the Space Race, NASA faced an immense challenge: sending astronauts into space and bringing them back safely. In 1962, as NASA prepared for the Friendship 7 mission with John Glenn becoming the first American to orbit Earth, critical calculations were performed by human mathematicians called “computers.”

Katherine Johnson was one of the most brilliant of these mathematicians. She specialized in calculating and verifying orbital trajectories, reentry points, and splashdown zones. Her task was to ensure the calculated trajectory matched the spacecraft’s actual path within a safe margin. Johnson’s contributions, along with those of colleagues Dorothy Vaughan and Mary Jackson, were later chronicled in the 2016 film "Hidden Figures".

Your mission is to step into Katherine Johnson’s role for a moment. Your task is to write a program that compares the calculated ideal trajectory against the actual telemetry data from the spacecraft. You must determine how many data points in the actual path deviate from the calculated path by more than a specified safety margin.

Input

The input is composed by three lines:

- First line has the calculated path. A list of tuples (x, y) representing the planned coordinates (integers or decimals).
- Second line refers to the actual path. A list of tuples (x, y) representing the spacecraft's actual position (integers or decimals).
  First and second lines are guaranteed to have the same number of coordinates.
- Third line contains the margin. A non-negative decimal representing the maximum allowed Euclidean distance deviation.

The Euclidean distance

The Euclidean distance between two points (x_1, y_1) and (x_2, y_2) is given by the formula:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

It is easy to calculate the square root by simply raising to the power of 0.5.

Output

Return a tuple like this: (*deviations*, *status*) where

- *deviations* is an integer as the number of points where the deviation exceeds input margin
- *status* is a string, either "Stable" if deviations is zero, otherwise "Unstable"

Explanation

Given three points (0, 0), (1, 1), (2, 2) of calculated path and three points (0, 0.1), (1, 1.2), (2, 2.05) of actual path with a 0.15 margin, these are the calculations applied:

- Point 0: Distance between (0,0) and (0,0.1) is 0.1. (0.1 <= 0.15) **OK**
- Point 1: Distance between (1,1) and (1,1.2) is 0.2. (0.2 > 0.15) **DEVIATION**
- Point 2: Distance between (2,2) and (2,2.05) is 0.05. (0.05 <= 0.15) **OK**

Total deviations: 1. Since deviations > 0 then status is **Unstable**.

Example 1

Input

```
(0, 0), (1, 1), (2, 2)
(0, 0), (1, 1), (2, 2)
0.15
```

Output

```
(0, Stable)
```

Example 2

Input

```
(0, 0), (1, 1), (2, 2)
(0, 0.1), (1, 1.2), (2, 2.05)
0.15
```

Output

```
(1, Unstable)
```

Python

```python
import re

def parse_path(line):
    """Parse a line of tuples like '(0, 0), (1, 1), (2, 2)' into a list of
tuples."""
    # Find all tuples using regex
    pattern = r'\(([^)]+)\)'
    matches = re.findall(pattern, line)
    path = []
    for match in matches:
        x, y = match.split(',')
        path.append((float(x.strip()), float(y.strip())))
    return path


def orbital_trajectory_verification(calculated_path, actual_path, margin):
    """
    Calculates the number of points where the actual path deviates from the
    calculated path by more than the specified margin.

    Args:
    calculated_path: List of tuples (x, y)
    actual_path: List of tuples (x, y)
    margin: Float, maximum allowed distance

    Returns:
    tuple: (int: Number of deviations, str: "Stable" or "Unstable")
    """

    deviations = 0

    # Iterate through both paths simultaneously
    for calc, act in zip(calculated_path, actual_path):
        # Calculate Euclidean distance
        distance = ((calc[0] - act[0])**2 + (calc[1] - act[1])**2)**0.5

        # Check if distance exceeds margin
        if distance > margin:
            deviations += 1

    status = "Stable" if deviations == 0 else "Unstable"
    return (deviations, status)
```

```
if __name__ == "__main__":
    # Read input
    calculated_path = parse_path(input())
    actual_path = parse_path(input())
    margin = float(input())

    # Calculate and print result
    deviations, status = orbital_trajectory_verification(calculated_path, 
actual_path, margin)
    print(f"({deviations}, {status})")
```

21 Gamer Tag

13 points

Introduction

A popular online gaming platform has developed a new system for encoding special achievement messages and secret gamer tags. The code is based on a simple substitution system using only 3 different symbols: X (Cross), O (Circle) and I (Line). Gamer tags are encoded using pairs of symbols (digraphs) that represent different letters or signals.

For instance: the sequence XO gives the letter M and the sequence OI encodes for letter R. There are special digraphs for signals like starting a tag (XI) and for ending a tag (OX or XX). Note that the special sequence IO makes the decoder discard (delete) the last letter decoded (if any) and there are no multiple START signals in each sequence.

DIGRAPH	LETTER	DIGRAPH	LETTER	DIGRAPH	SIGNAL
XO	M	OI	R	XI	START
XX	N	II	S	OX	END
OO	P	IX	T	XX	END
				IO	DELETE

Write a program that given a string containing a symbol sequence, returns the decoded gamer tag (letters concatenated without spaces) or "None" if no valid sequence is found. To be valid, a sequence must have a START at the beginning, at least one letter and an END signal at the end.

Input

A string with a symbol sequence using single capital letters.

Output

The program must return the decoded gamer tag or "None" if no valid tag is found.

Example 1

Input

```
XIXOOOOIOX
```

Output

```
MPR
```

Example 2

Input

```
XIXOOOOIIOIIOX
```

Output

```
MPS
```

Example 3

Input

```
XIXOOIOIIO
```

Output

```
None
```

Example 4

Input

```
XIIOXOXXXX
```

Output

```
MN
```

Python

```python
def decode_cipher(sequence):
    """
    Decodes a secret cipher message from a symbol sequence.

    :param sequence: String containing X, O, I symbols
    :return: Decoded message or "None" if invalid
    """
    # Define the cipher mapping
    cipher_map = {
        'XO': 'M',
        'XX': 'N',
        'OO': 'P',
        'OI': 'R',
        'II': 'S',
        'IX': 'T'
    }

    # Special sequences
    START = 'XI'
    END_1 = 'OX'
    END_2 = 'XX'
    DELETE = 'IO'

    # Check if sequence has valid length (must be even for pairs)
    if len(sequence) % 2 != 0:
        return "None"

    # Split into pairs
    pairs = []
    for i in range(0, len(sequence), 2):
        pairs.append(sequence[i:i+2])

    # Check for START
    if not pairs or pairs[0] != START:
        return "None"

    # Find END position
    end_pos = -1
    for i in range(len(pairs) - 1, 0, -1):
        if pairs[i] == END_1 or pairs[i] == END_2:
            end_pos = i
            break

    if end_pos == -1:
        return "None"

    # Process the sequence between START and END
    message = []
    for i in range(1, end_pos):
        pair = pairs[i]
        if pair == DELETE:
            # Delete last character if exists
            if message:
                message.pop()
        elif pair in cipher_map:
            message.append(cipher_map[pair])
        else:
            # Invalid pair found
            return "None"

    # Must have at least one letter
    if not message:
        return "None"

    return ''.join(message)

if __name__ == "__main__":
    sequence = input().strip()
    result = decode_cipher(sequence)
    print(result)
```

22 The Ancient Library

13 points

Introduction

Deep in the archives of an ancient library, you discover a mysterious scroll containing encoded messages left by a secret society of scholars. After studying their notes, you learn they used a clever letter-scrambling technique.

Each letter is first converted to a number (A = 0, B = 1, ..., Z = 25) as x, then transformed using a secret formula:

$$E(x) = (a \cdot x + b) \; mod \; 26$$

where:

- a and b are society's secret keys chosen to scramble the message and
- mod 26 the result stays between 0 and 25 (that is, within the alphabet letters)

Constraints and Important Considerations

1. $1 \leq a \leq 25$ and a must be coprime with 26, that is, $\gcd(a, 26) = 1$. If a shares common factors with 26 (like $a = 2,13,26$), it won't have a modular inverse, making decode impossible.
2. $1 \leq b \leq 25$
3. The scrambled message consists only of uppercase and lowercase letters.

You've managed to recover the keys from a separate document. Can you develop a program to decode their messages?

Input

The input is composed of three lines. The first line contains number key a. The second line contains number key b. The third line contains the Affine ciphered message.

Output

A single line with the decoded message.

Example

Input

```
7
4
Cgdsykg hy SyzgCeta Bevg nor
```

Output

```
Welcome to CodeWars Have fun
```

Python

```python
def modular_inverse(a, m):
    """Find the modular inverse of a under modulo m."""
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def affine_decipher(a, b, ciphered_text):
    """Decipher the text using the Affine cipher."""
    m = 26  # Number of letters in the English alphabet
    a_inv = modular_inverse(a, m)
    if a_inv is None:
        raise ValueError("a and 26 are not coprime, decryption is not 
possible.")

    deciphered_text = ""
    for char in ciphered_text:
        if char.isalpha():
            is_upper = char.isupper()
            char = char.lower()
            y = ord(char) - ord('a')
            x = (a_inv * (y - b)) % m
            deciphered_char = chr(x + ord('a'))
            deciphered_text += deciphered_char.upper() if is_upper else 
deciphered_char
        else:
            deciphered_text += char  # Non-alphabetic characters remain 
unchanged
    return deciphered_text

# Read input a and b key parameters  and the ciphered message
a = int(input())
b = int(input())
ciphered_message = input()

# Decipher the message
deciphered_message = affine_decipher(a, b, ciphered_message)

# Print the deciphered message
print(deciphered_message)
```

23 Village Pathfinding

18 points

Introduction

You are exploring a Minecraft world represented as a 2D grid. Your player starts at P and you want to reach the village at V. Each cell may be:

- . empty block (walkable)
- # obstacle (not walkable)
- P player start (exactly one)
- V village (exactly one)

Why find the village? Early-game villages provide food (crops), beds, an Iron Golem guardian, and (most importantly) villagers for trading enchanted books, tools, and resources. Securing the shortest safe path before nightfall reduces risk from mobs and lets you begin trading quickly. Obstacles (#) symbolize ravines, walls, or dense forests you must route around.

You can move up, down, left, or right to adjacent walkable cells. Find out the minimum steps to reach V from P, or Impossible if no path exists. A step is a move from one cell to an adjacent (non-diagonal) cell.

Input

The first line contains two integers (first rows and second columns) representing the grid dimensions.
Next rows lines: each a string of length cols composed of characters from {'.', #, P, V}.

Output

A single integer: the minimum number of steps from P to V, or the word Impossible if no path exists.

Example 1

Input

```
4 4
P...
.#..
..#V
....
```

Output

```
5
```

Example 2

Input

```
3 3
P.#
.#.
#V#
```

Output

```
Impossible
```

Python

```python
from collections import deque

def shortest_path_steps(grid):
    rows = len(grid)
    cols = len(grid[0]) if rows else 0
    start = end = None
    for i in range(rows):
        for j in range(cols):
            cell = grid[i][j]
            if cell == 'P':
                start = (i, j)
            elif cell == 'V':
                end = (i, j)
    if start is None or end is None:
        return None  # indicates Impossible

    directions = [(1,0), (-1,0), (0,1), (0,-1)]
    visited = [[False]*cols for _ in range(rows)]
    q = deque([(start[0], start[1], 0)])
    visited[start[0]][start[1]] = True

    while q:
        x, y, d = q.popleft()
        if (x, y) == end:
            return d
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < rows and 0 <= ny < cols and not visited[nx][ny] and
grid[nx][ny] != '#':
                visited[nx][ny] = True
                q.append((nx, ny, d + 1))
    return None

def main():
    rows, cols = map(int, input().split())
    raw_grid = [list(input().strip()) for _ in range(rows)]
    result = shortest_path_steps(raw_grid)
    if result is None:
        print("Impossible")
    else:
        print(result)

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

24 The Storm And The Supply Drop

18 points

Introduction

You are in the final circle of a Battle Royale match. The Storm is shrinking rapidly! You have just opened a legendary Supply Drop, but your backpack has limited space.

You see a list of items, each with a specific Inventory Cost (size) and a Tactical Value (utility). You need to fill your backpack with the combination of items that gives you the highest possible Tactical Value without exceeding your backpack's capacity.

Given the capacity of your backpack and a list of available items (with their sizes and values), calculate the maximum Tactical Value you can achieve.

Input

- The first line contains two positive integers: N (# of items) and C (backpack capacity) where $1 \le N \le 100$ and $1 \le C \le 1000$
- The next N lines each contain two positive integers: w (size) and v (value) for each item where $1 \le w \le C$ and $1 \le v \le 1000$.

Output

Print a single positive integer representing the maximum Tactical Value.

Example

Input

```
4 10
4 40
3 30
5 50
2 10
```

Output

```
90
```

Explanation: Taking items 1 (size 4, val 40) and 3 (size 5, val 50) = Size 9, Value 90 **or** taking items 2 (size 3, val 30), 3 (size 5, val 50) and 4 (size 2, val 10) = Size 10, Value 90.

Python

```python
import sys

def solve():
    # Reading input from standard input
    try:
        input_data = sys.stdin.read().split()
    except Exception:
        return

    if not input_data:
        return

    iterator = iter(input_data)

    try:
        n = int(next(iterator))
        capacity = int(next(iterator))

        weights = []
        values = []

        for _ in range(n):
            weights.append(int(next(iterator)))
            values.append(int(next(iterator)))

    except StopIteration:
        return

    # DP Solution (0/1 Knapsack)
    # dp[w] will store the max value for capacity w
    dp = [0] * (capacity + 1)

    for i in range(n):
        w = weights[i]
        v = values[i]
        # Iterate backwards to avoid using the same item multiple times for one
capacity
        for j in range(capacity, w - 1, -1):
            if dp[j - w] + v > dp[j]:
                dp[j] = dp[j - w] + v
    print(dp[capacity])

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

25 Langton's Ant Escape

23 points

Introduction

Professor Langton has created a peculiar robotic ant for his laboratory experiments. This ant follows very simple rules as it walks on a grid of white squares. The Ant's Rules are:

1. If the ant is on a WHITE square:
    a. Turn 90° to the RIGHT
    b. Flip the square's color to BLACK
    c. Move forward one square

2. If the ant is on a BLACK square:
    a. Turn 90° to the LEFT
    b. Flip the square's color to WHITE
    c. Move forward one square

The ant starts on a white square (all squares are initially white) and begins facing a given direction (North, South, East, or West).

Professor Langton wants to know: Will the ant escape from the grid, or will it stay contained? If it escapes, he wants to know exactly when and from which edge!

Your task is to simulate the ant's movement and determine if it leaves the grid boundaries within a maximum number of steps.

Considerations about the grid coordinate system where the ants move:

- (0, 0) is the TOP-LEFT corner
- X increases to the RIGHT
- Y increases DOWNWARD
- North means moving UP (Y decreases)
- South means moving DOWN (Y increases)
- East means moving RIGHT (X increases)
- West means moving LEFT (X decreases)

Input

Line 1: Two integers W and H - the width and height of the grid (1 ≤ W, H ≤ 100)
Line 2: An integer N - the maximum number of steps to simulate (1 ≤ N ≤ 10000)
Line 3: Two integers X and Y - the ant's starting position (0 ≤ X < W, 0 ≤ Y < H)
Line 4: A character D - the ant's starting direction: N (North), S (South), E (East), or W (West)

Output

The program outputs a visual representation of the grid at each step, showing the ant's movement and the evolving pattern of black and white squares.

For each step (0 to N):
Step <n>:
<grid visualization>
where grid visualization means:
- 0 represents a WHITE square
- 1 represents a BLACK square
- X represents the ant's current position
- When the ant escapes, X appears outside the grid boundaries

Final Result:
If the ant escapes the grid (tries to move outside boundaries):
```
=== ESCAPED <step> <edge> ===
```
where <step> is the step number when it escaped (0-indexed) and <edge> is one of: TOP, BOTTOM, LEFT, RIGHT

If the ant stays within the grid for all N steps:
```
=== CONTAINED <x> <y> <direction> ===
```
where <x> <y> is the final position and <direction> is the final direction (N/S/E/W)

WARNING: Do not add any trailing spaces beyond the grid width or the ant (X) position. Each line should end immediately after the last character of the grid or the X marker.

Example 1

Input

```
4 3
30
0 1
W
```

Output

```
Step 0:
0 0 0 0
X 0 0 0
0 0 0 0
Step 1:
X 0 0 0
1 0 0 0
0 0 0 0
Step 2:
1 X 0 0
1 0 0 0
0 0 0 0
Step 3:
1 1 0 0
1 X 0 0
0 0 0 0
Step 4:
1 1 0 0
X 1 0 0
0 0 0 0
Step 5:
1 1 0 0
0 1 0 0
X 0 0 0
```

Example 2

Input

```
7 7
5
5 5
S
```

Output

```
Step 0:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 X 0
0 0 0 0 0 0 0
Step 1:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 X 1 0
0 0 0 0 0 0 0
Step 2:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 X 0 0
0 0 0 0 1 1 0
0 0 0 0 0 0 0
```

```
Step 6:
  1 1 0 0
  0 1 0 0
X 1 0 0 0
=== ESCAPED 6 LEFT ===
```

```
Step 3:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 1 X 0
0 0 0 0 1 1 0
0 0 0 0 0 0 0
Step 4:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 1 1 0
0 0 0 0 1 X 0
0 0 0 0 0 0 0
Step 5:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 1 1 0
0 0 0 0 1 0 X
0 0 0 0 0 0 0
=== CONTAINED 6 5 E ===
```

Python

```python
# Read input
w, h = map(int, input().split())
n = int(input().strip())
x, y = map(int, input().split())
direction_char = input().strip()

# Direction mapping: N=0, E=1, S=2, W=3
dir_map = {'N': 0, 'E': 1, 'S': 2, 'W': 3}
dir_names = ['N', 'E', 'S', 'W']
edge_names = ['TOP', 'RIGHT', 'BOTTOM', 'LEFT']

# Movement deltas: [dx, dy] for each direction
# North: y-1, East: x+1, South: y+1, West: x-1
deltas = [(0, -1), (1, 0), (0, 1), (-1, 0)]

direction = dir_map[direction_char]

# Track black squares (empty set means all white initially)
black_squares = set()

def print_grid(step_num, ant_x, ant_y, ant_escaped=False):
    """Print the current grid state"""
    print(f"Step {step_num}:")

    # Determine grid display range based on escape
    start_row = -1 if ant_escaped and ant_y == -1 else 0
    end_row = h + 1 if ant_escaped and ant_y == h else h
    start_col = -1 if ant_escaped and ant_x == -1 else 0
    end_col = w + 1 if ant_escaped and ant_x == w else w

    for row in range(start_row, end_row):
        row_str = ""

        # Add left column (for LEFT escape or spacing)
        if start_col == -1:
            if row == ant_y and ant_x == -1:
                row_str += "X "
            elif 0 <= row < h:
                row_str += "  "
            else:
                row_str += "  "

        # Print main grid content
        if 0 <= row < h:
            for col in range(w):
                # Check if ant is at this position
                if col == ant_x and row == ant_y and not ant_escaped:
                    row_str += "X"
                elif (col, row) in black_squares:
                    row_str += "1"
                else:
                    row_str += "0"
                if col < w - 1:
                    row_str += " "
        else:
            # Top or bottom row outside grid
            for col in range(w):
                if row == ant_y and col == ant_x:
                    row_str += "X"
                else:
                    row_str += " "
                if col < w - 1:
                    row_str += " "

        # Add right column (for RIGHT escape)
        if end_col > w:
            if row == ant_y and ant_x == w:
                if 0 <= row < h:
                    row_str += " X"
                else:
                    row_str += "  "
            elif 0 <= row < h:
                row_str += "  "
            else:
                row_str += "  "

        print(row_str)

# Print initial state
print_grid(0, x, y)

# Simulate N steps
escaped = False
for step in range(1, n + 1):
    # Check current square color
    is_black = (x, y) in black_squares
    current_color = "BLACK" if is_black else "WHITE"

    if is_black:
        # On black: turn left, flip to white
        direction = (direction - 1) % 4
        black_squares.remove((x, y))
    else:
        # On white: turn right, flip to black
        direction = (direction + 1) % 4
        black_squares.add((x, y))

    # Move forward in current direction
    dx, dy = deltas[direction]
    next_x = x + dx
    next_y = y + dy

    # Check if escaped
    if next_x < 0:
        x, y = next_x, next_y  # Save escape position
        escaped = True
        print_grid(step, x, y, escaped)
        print(f"=== ESCAPED {step} LEFT ===")
        break
    elif next_x >= w:
        x, y = next_x, next_y  # Save escape position
        escaped = True
        print_grid(step, x, y, escaped)
        print(f"=== ESCAPED {step} RIGHT ===")
        break
    elif next_y < 0:
        x, y = next_x, next_y  # Save escape position
        escaped = True
        print_grid(step, x, y, escaped)
        print(f"=== ESCAPED {step} TOP ===")
        break
    elif next_y >= h:
        x, y = next_x, next_y  # Save escape position
        escaped = True
        print_grid(step, x, y, escaped)
        print(f"=== ESCAPED {step} BOTTOM ===")
        break

    # Move to next position
    x, y = next_x, next_y

    # Print grid after this step
    print_grid(step, x, y, escaped)

# Print final result message for contained case
if not escaped:
    print(f"=== CONTAINED {x} {y} {dir_names[direction]} ===")
```

26 The Ancestor Oracle (Interactive problem)

25 points

Introduction

Deep in the Enchanted Kingdom, there exists a magical crystal ball known as The Ancestor Oracle. This mystical artifact holds the complete family tree of an ancient royal dynasty, but the written records were lost centuries ago.

The Oracle has a peculiar power: when you speak the names of any two family members, it reveals their closest common ancestor – the nearest relative that both share in their lineage.

You are a young historian who has been given a rare opportunity to consult the Oracle. Your mission is to reconstruct the entire family tree by asking clever questions. But beware! The Oracle's magic is limited – you can only ask a certain number of questions before it falls silent for another hundred years.

The royal family tree has N members, numbered from 1 to N. Member 1 is the original founder of the dynasty (the root of the family tree). Every other member has exactly one parent in the tree.

Can you figure out all the parent-child relationships before your questions run out?

Input

The first input the judge sends a single line containing two integers `N Q`, where `N` is a positive integer representing the number of family members (nodes in the tree) and `Q` is a positive integer as the maximum number of questions you can ask.

The rest of the inputs will come after your outputs. The judge will respond with a single positive integer: the Lowest Common Ancestor of members a and b – that is, the closest family member who is an ancestor of both a and b.

In case you exceed the number of questions, the judge will respond with **`LIMIT_EXCEEDED`** and you lose.

When your program outputs an `ANSWER e1_a e1_b e2_a e2_b ... e(N-1)_a e(N-1)_b` the judge will respond with `CORRECT` when you successfully reconstructed the family tree or `WRONG` when it does not match the secret tree.

Output

Once your program receives the two integers N and Q, you may ask questions by printing a line in the format:

```
LCA a b
```

where `a` and `b` are distinct integers between 1 and N.

Rules:

- Every member is considered an ancestor of themselves.
- Note that if `a` is an ancestor of `b`, then you will have that `LCA(a, b) = a`
- The founder (member 1) is an ancestor of everyone.

When you have determined the family tree structure, print the solution as:

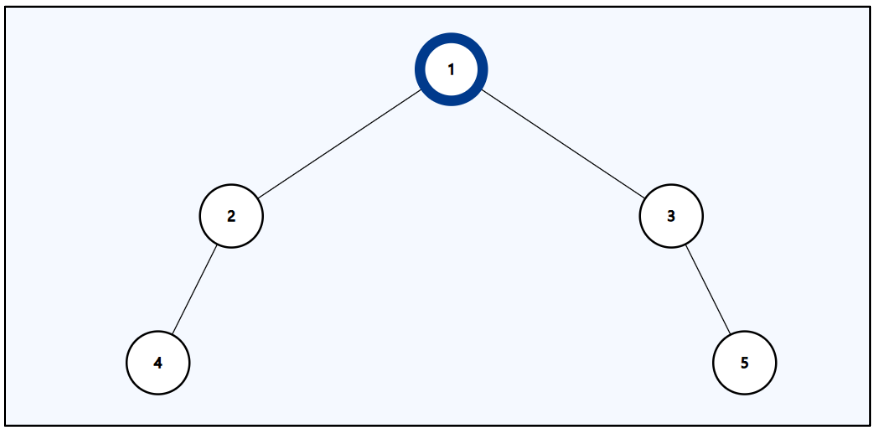
```
ANSWER e1_a e1_b e2_a e2_b ... e(N-1)_a e(N-1)_b
```

This represents the N-1 edges of the tree. Each pair `(ei_a, ei_b)` indicates that members `ei_a` and `ei_b` are directly related (parent-child). The edges can be listed in any order, and for each edge, either member can be listed first.

Every time you want to send a command to the judge, you should do a flush. You can use fflush(stdout) in C, cout.flush() in C++ and sys.stdout.flush() in Python.

Example 1

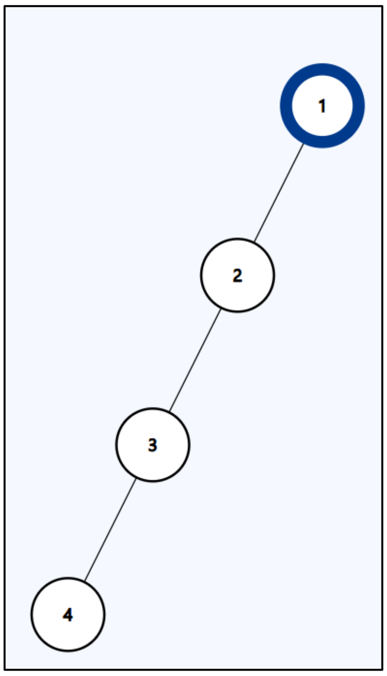
Judge	Communication	Your
5 45		
		LCA 2 3
1		
		LCA 2 4
2		
		LCA 2 5
1		
		LCA 3 5
3		
		ANSWER 1 2 1 3 2 4 3 5
CORRECT		



Example 2

Communication

Judge	Your
4 24	
	LCA 2 3
2	
	LCA 2 4
2	
	LCA 3 4
3	
	ANSWER 1 2 2 3 3 4
CORRECT	



Python

```python
import sys

def query_lca(a, b):
    """Query the LCA of nodes a and b"""
    print(f"LCA {a} {b}", flush=True)
    response = input().strip()
    return int(response)

def solve():
    # Read N and Q
    line = input().split()
    n = int(line[0])
    q = int(line[1])

    if n == 1:
        # Edge case: single node, no edges
        print("ANSWER", flush=True)
        return

    # Build tree incrementally
    # children[i] = list of children of node i (in the tree we've built so far)
    children = [[] for _ in range(n + 1)]
    parent = [0] * (n + 1)
    tree_nodes = [1]  # Nodes already in our tree

    edges = []

    for x in range(2, n + 1):
        # Find parent of node x by walking down the tree
        current = 1  # Start from root

        while True:
            # Check if x belongs to any child's subtree
            found_child = False
            for c in children[current]:
                lca = query_lca(c, x)
                if lca == c:
                    # x is in the subtree rooted at c
                    current = c
                    found_child = True
                    break

            if not found_child:
                # x's parent is current
                parent[x] = current
                children[current].append(x)
                edges.append((current, x))
                tree_nodes.append(x)
                break

    # Output the answer
    answer_parts = []
    for a, b in edges:
        answer_parts.append(str(a))
        answer_parts.append(str(b))

    print("ANSWER " + " ".join(answer_parts), flush=True)

    # Read response
    response = input().strip()
    # Program should terminate after receiving response

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