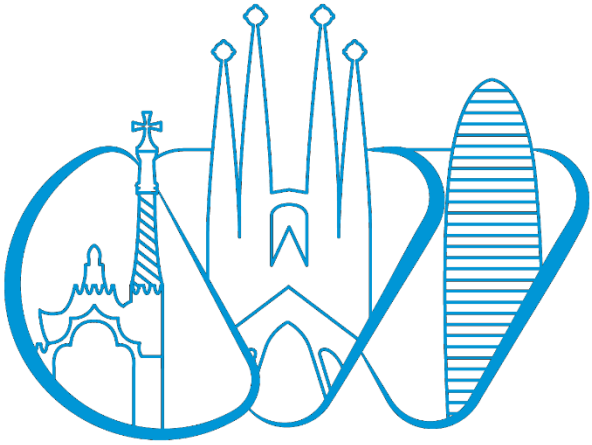


CodeWars

BCN 2023

Problems
and
Solutions

#	Problem	Points
1	Ohm's Law	1
2	Pentagonal Numbers	2
3	DNI Letter	3
4	Oh! Yeah!	3
5	Disemvowelling	4
6	Table Officials	4
7	Blood Test	4
8	Collatz Conjecture	5
9	Doppler Effect	5
10	Everything But Me	5
11	Training R3-AD	5
12	Automatic Hangman	6
13	A Beautiful Mind	6
14	La Casa de Papel	6
15	Mesoamerican Pyramids	6
16	Acronymizer	7
17	Fixing Sentences	7
18	Framing	7
19	Golf Ball	7
20	Sudoku Friends	7
21	Close Encounters Of The String Kind	8
22	Martes Y Trece	8
23	Minesweeper	9
24	Synthetic Division	10
25	The Sheldon Prime	11
26	Chain Reaction	12
27	3D Box Drawing	13
28	Flags	14
29	Top Pizza	15
30	MotoHP	16
31	Time Is Gold	25
32	Mutant Mushrooms	30
33	Voxels	35

1 Ohm's Law

1 point

Introduction

One of the basic laws of electrical circuits is Ohm's law which states that the current passing through a resistor is proportional to the voltage over the resistance. That is, I = V / R.

Where the units are I = current in Amps, V = voltage in Volts, and R = resistance in Ohms.

While Roger is setting up his electrical circuit, he must find out the voltage needed to produce a certain current for a given resistor. Can you write a simple program to help Roger to automate this job?

Input

The input consists of two lines. Each one has a single positive integer where the first line represents the current in Amps and the second line is the resistance in Ohms.

Output

The output will print the voltage needed to produce the desired current.

Example

Input

```
2

200
```

Output

```
400
```

Python

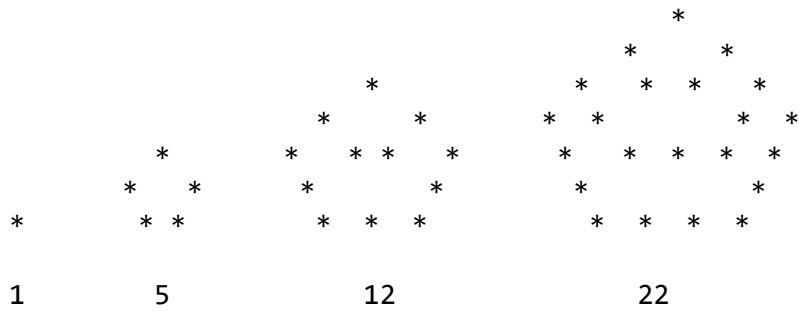
```python
# Read the current value
i = int(input())
# Read the resistor value
r = int(input())
# Calculate the voltage
v = i * r
# Print the voltage
print(v)
```

2 Pentagonal Numbers

2 points

Introduction

Have you ever heard of pentagonal numbers? These numbers are defined by the sequence 1, 5, 12, 22, 35, 51, ... It is also worthy to note that such pentagonal numbers can be represented following a regular geometrical arrangement of equally spaced dots.



The generalized pentagonal numbers are those of the form:

$$P_n = \frac{n * (3 * n - 1)}{2}$$

Given this formula can you write a program to find out the number of dots for a given n[th] pentagonal number?

Input

A single line with a positive number.

Output

A single line with the corresponding n[th] pentagonal number.

Example

Input

```
3
```

Output

```
12
```

C++

```cpp
#include <iostream>

using namespace std;


int main() {
    int input;
    cin >> input;

    int result = ((input * (3 * input - 1))/ (2));
    cout << result;

    return 0;
}
```

3 DNI Letter

3 points

Introduction

A DNI is composed of a number (8 digits) and a letter at the end.

To establish the letter of a DNI you need to calculate the remainder when the number is divided by 23 and then convert that value to a letter using this table:

REMAINDER	0	1	2	3	4	5	6	7	8	9	10	11
LETTER	T	R	W	A	G	M	Y	F	P	D	X	B
REMAINDER	12	13	14	15	16	17	18	19	20	21	22	
LETTER	N	J	Z	S	Q	V	H	L	C	K	E	

Given a number calculate the corresponding letter.

Input

The numerical part of a DNI (an 8 digit number)

Output

The corresponding letter

Example 1

Input

```
12345678
```

Output

```
Letter: Z
```

Example 2

Input

```
26841269
```

Output

```
Letter: Q
```

Java

```java
import java.lang.Math;
import java.util.Scanner;

public class DNIletter {

    public static void main(String[] args)
    {

        char[] letters = {'T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X',
'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E'};

        Scanner myInput = new Scanner( System.in );

        int number  = myInput.nextInt();

        int remainder = number % 23;

        System.out.println("Letter: " + Character.toString(letters[remainder]));
    }
}
```

4 Oh! Yeah!

3 points

Introduction

Have you ever seen the movie "Ferris Bueller's Day Off" (1986)? It's a teen comedy about a high school senior, Ferris Bueller, who fakes illness to stay home from school. This movie popularized the song "Oh! Yeah!" by Swiss electronic music band Yello that lately was also used in some TV advertising campaigns. Maybe you seen one of them... This song repeats the expression "Oh! Yeah! several times but extending the pronunciation so that it could be transcribed as "Oooh! Yeeeaaah!"

Since listening to song lyrics repeating same expression can be quite boring, so why not code a program to write out the expression with each vowel repeated as many times as indicated by the input number.

Input

A positive integer number defining the number of vowels to be repeated.

Output

In a single line the expression "OH! YEAH!" but repeating each vowel the same number of times as the input number.

Example

Input

```
3
```

Output

```
OOOH! YEEEAAAH!
```

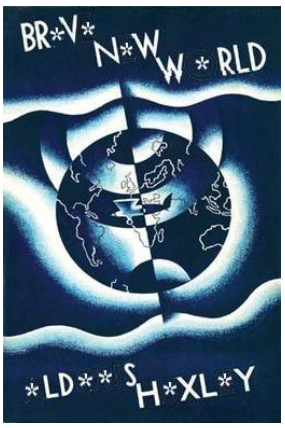
Python

```python
number = int(input());

for i in range(number):
  print("O", end='')

print("H! Y", end='')

for i in range(number):
  print("E", end='')

for i in range(number):
  print("A", end='')

print("H!")
```

5 Disemvoweling

4 points

Introduction



In a dystopian society of the future, vowels have been forbidden. Now humans have learnt to talk without the need of vowel speech sounds. Only consonants are used. Such an "advance" has some drawbacks. For example, people get angry as they cannot read old books because there are plenty of vowels. To keep them happy you have been requested to create a program that removes vowels from a given text. You just need to substitute the vowels with *. This way people will be able to read old books, keep the ancient knowledge and be happy again. This is the art of disemvowelling.

Input

A text line containing words, numbers, white spaces, punctuation marks, question, and exclamation marks, ...

Output

Same line as the input but replacing any vowel with an asterisk (*) symbol.

Example 1

Input

```
Hello World!
```

Output

```
H*ll* W*rld!
```

Example 2

Input

```
2001: A Space Odyssey
```

Output

```
2001: * Sp*c* *dyss*y
```

Python

```python
text = input()
disemvoweledText = ""

def isVowel(letter):
  letter = letter.upper()
  if (letter == "A" or letter == "E" or
      letter == "I" or letter == "O" or
      letter == "U"):
    return True
  else:
    return False

for i in text:
  if isVowel(i):
    disemvoweledText = disemvoweledText + "*"
  else:
    disemvoweledText = disemvoweledText + i

print(disemvoweledText)
```

6 Table Officials

4 points

Introduction

The table officials in a basketball match are responsible for keeping track of each team's scoring, timekeeping, individual and team fouls, player substitutions, team possession and the shot clock. By the end of the match, you will help them to provide some data and statistics like the total amount of points per player and their overall shooting effectiveness. You decide to make a simple program to make the task easier.

Input

The input refers to the data of a single player and will be always four positive integer numbers split into four lines where:

- The first number is the total of free throws scored with a value of one point.
- The second number is the total of field goals of two points.
- The third number is the total of field goals scored behind the three-point line.
- The fourth number is the total of shots performed during the match.

Output

A single line representing the total amount of points achieved by the player and their effectiveness percentage (rounding down and without decimals).

Example 1

Input

```
1
6
2
12
```

Output

```
19 75%
```

Example 2

Input

```
1
0
1
9
```

Output

```
4 22%
```

Example 3

Input

```
5
3
2
13
```

Output

```
17 76%
```

C++

```cpp
#include <iostream>

using namespace std;

int main() {
    float freeThrows, twoPoints, threePoints, shoots;

    cin >> freeThrows >> twoPoints >> threePoints >> shoots;

    float points = (freeThrows + (2 * twoPoints) + (3 * threePoints));
    float effectiveness = ((freeThrows + twoPoints + threePoints) / shoots) *
100;


    cout << points << " " << static_cast<int>(effectiveness) << "%";
    return 0;
}
```

7 Blood Test

4 points

Introduction

A blood test consists of an examination of a blood sample used in health care to determine physiological and biochemical states to assess an individual's general health. Once the sample is analyzed, the laboratory compiles the results in a blood test report. This report details different components in the blood and their levels. To easily read the results, each of the values is written next to a healthy range. Some of the ranges depends on the gender of the individual.

Here's a typical range of results related to complete blood count:

Component	Definition	Normal range
Red blood cells	Cells responsible for carrying oxygen throughout the body	**male:** 4.3–5.9 million/mm^3 **female:** 3.5–5.5 million/mm^3
White blood cells	Immune system cells in the blood	4500–11000/mm^3
Platelets	The substances that control the clotting of the blood	150000–400000/mm^3
Hemoglobin	Protein within the red blood cells that carries oxygen to organs and tissues, and carbon dioxide back to the lungs	**male:** 13.5–17.5 g/dL **female:** 12.0–16.0 g/dL
Hematocrit	Percentage of blood made of red blood cells	**male:** 41–53% **female:** 36–46%

Given the patient's gender and their five component levels can you code a program to check whether any of the parameters is out of range?

Note: The limits of normal range are not included as normal values. For example, 4.3 million/mm3 red blood cells for male IS NOT a normal value.

Input

The input consists of six lines:
The first line defines the gender of the patient: Male or Female.
The second line refers to the red blood cells expressed in a decimal value in million/mm3
The third line indicates the total of white blood cells per mm3
The fourth line reports the number of platelets per mm3
The fifth line is a decimal value defining the hemoglobin in grams/deciliter
The last line is for hematocrit, expressing percentage of blood made of red blood cells.

Output

A readable report stating whether the blood test is normal or if the patient needs to visit the doctor:
Red blood cells: NORMAL or VISIT THE DOCTOR
White blood cells: NORMAL or VISIT THE DOCTOR
Platelets: NORMAL or VISIT THE DOCTOR
Hemoglobin: NORMAL or VISIT THE DOCTOR
Hematocrit: NORMAL or VISIT THE DOCTOR

Example 1

Input

```
Male
4.2
5900
150001
14.2
50
```

Output

```
Red blood cells: VISIT THE DOCTOR
White blood cells: NORMAL
Platelets: NORMAL
Hemoglobin: NORMAL
Hematocrit: NORMAL
```

Example 2

Input

```
Female
4.6
4400
300000
14.5
51
```

Output

```
Red blood cells: NORMAL
White blood cells: VISIT THE DOCTOR
Platelets: NORMAL
Hemoglobin: NORMAL
Hematocrit: VISIT THE DOCTOR
```

Python

```python
sex = input()
redBloodCells = float(input())
whiteBloodCells = int(input())
platelets = int(input())
hemoglobin = float(input())
hematocrit = int(input())

if sex == "Male":
    if redBloodCells > 4.3 and redBloodCells < 5.9:
        res = "NORMAL"
    else:
        res = "VISIT THE DOCTOR"
else:
```

```
    if redBloodCells > 3.5 and redBloodCells < 5.5:
        res = "NORMAL"
    else:
        res = "VISIT THE DOCTOR"

print("Red blood cells: " + res)

if whiteBloodCells > 4500 and whiteBloodCells < 11000:
    res = "NORMAL"
else:
    res = "VISIT THE DOCTOR"

print("White blood cells: " + res)

if platelets > 150000 and platelets < 400000:
    res = "NORMAL"
else:
    res = "VISIT THE DOCTOR"

print("Platelets: " + res)

if sex == "Male":
    if hemoglobin > 13.5 and hemoglobin < 17.5:
        res = "NORMAL"
    else:
        res = "VISIT THE DOCTOR"
else:
    if hemoglobin > 12 and hemoglobin < 16:
        res = "NORMAL"
    else:
        res = "VISIT THE DOCTOR"


print("Hemoglobin: " + res)

if sex == "Male":
    if hematocrit > 41 and hematocrit < 53:
        res = "NORMAL"
    else:
        res = "VISIT THE DOCTOR"
else:
    if hematocrit > 36 and hematocrit < 46:
        res = "NORMAL"
    else:
        res = "VISIT THE DOCTOR"

print("Hematocrit: " + res)
```

8 Collatz Conjecture

5 points

Introduction

The Collatz conjecture is one of the easiest to state but difficult to prove mathematical problems. Consider a positive number greater than 1. If it's odd, multiply it by 3 and add 1. If it's even, simply divide it by 2. Then apply the same rules to the new number you got. The conjecture is about what happens as you keep repeating the process.

What will it happen? Does the number you start with affect the number you end up with? Should it end or continue to infinity? Collatz conjectured that if you run this process long enough, all starting values will lead to 1. Nowadays the conjecture has been verified up to 2^{68}... Can you write a program to find out the resulting sequence when you apply collatz conjecture to a number?

Input

The input will be a single line containing a positive number greater than 1.

Output

The output will print the sequence of numbers obtained following the Collatz conjecture rules. The numbers should be separated by characters " -> ". At the end print out the total number of steps to reach 1.

Example 1

Input

```
6
```

Output

```
6 -> 3 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1 [8]
```

Example 2

Input

```
7
```

Output

```
7 -> 22 -> 11 -> 34 -> 17 -> 52 -> 26 -> 13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8
-> 4 -> 2 -> 1 [16]
```

C++

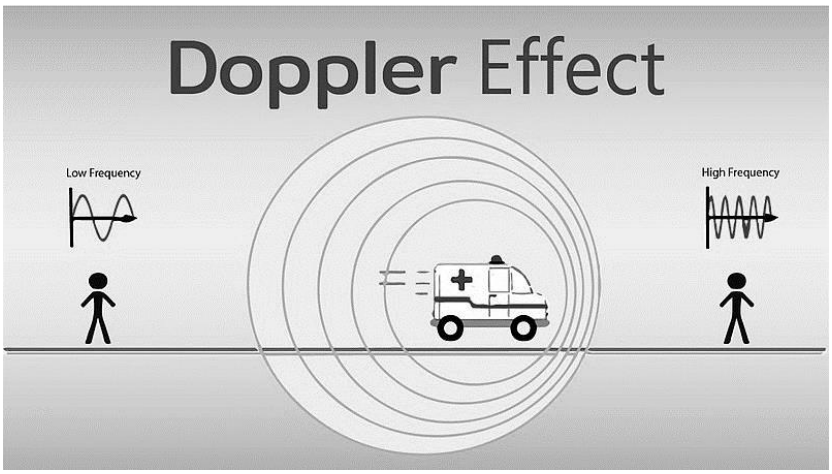
```cpp
#include <iostream>

int main(void)
{
    uint32_t n;
    uint32_t steps = 0;
    std::cin >> n;

    while (n != 1) {

        std::cout << n << " -> ";

        if (n % 2 == 0)
        {
            n = n / 2;
        }
        else
        {
           n = 3 * n + 1;
        }

        steps = steps + 1;
    }
    std::cout << n;
    std::cout << " [" << steps << "]" << std::endl;
}
```

9 Doppler Effect

5 points

Introduction

Have you ever stood on the side of a road when a fast car drove past you? If you have then you might have noticed that as the car approaches you, the sound of the car's engine gets louder and sounds different. The car doesn't change its engine noise while it is moving, so what's happening? The change in the way you hear a noisy object as it moves toward or away from you is called the Doppler effect.



It happens because sound moves in waves, known as sound waves, and its frequency gets higher as the noisy object comes toward you. The frequency refers to the rate at which the waves reach you. The sound waves in front of the noisy object bunch up with less space between them. So, the waves that reach you are at a higher frequency, changing the way you hear the sound. This is known as the apparent frequency (f') and it can be calculated with this formula:

$$f' = f * \frac{c + vr}{c + vs}$$

where f is the actual frequency,
c is the speed of the sound waves in the medium,
vr is the velocity of the observer with respect to the medium and
vs is the velocity of the noisy object with respect to the medium. This velocity can be positive, negative or zero depending its direction of travel relative to the observer.

Input

The input is composed by four integer values in this order:

- actual frequency in Hz
- speed of the sound waves in the medium in meters/second
- velocity of the observer in kilometres per hour
- velocity of the noisy object in kilometres per hour

Output

The apparent frequency observed with two decimals resolution in Hz.

Example 1

Input

```
200
340
60
-50
```

Output

```
218.74 Hz
```

Example 2

Input

```
25
10
108
36
```

Output

```
50.00 Hz
```

Python

```python
actualFreq = int(input())
soundWavesSpeed = int(input())
vr = int(input()) # Speed in km/h
# Convert it to m/s
vr = vr * 1000 / 3600

vs = int(input()) # Speed in km/h
# Convert it to m/s
vs = vs * 1000 / 3600

observerFreq = actualFreq * (soundWavesSpeed +vr) /(soundWavesSpeed+vs)

print("{:.2f} Hz".format(round(observerFreq,2)))
```

10 Everything But Me

5 points

Introduction

Write a program that calculates a set of the sums and products of a given series of integers but with each calculation excluding the integer that corresponds to its position in the order of the calculations being undertaken (so for the first sum and product calculations the first integer is excluded, and so on).

Input

Several lines, each containing an integer value until the character '#' marks the end of the input sequence.

Output

The first line contains per each read integer position the sum of all the input values except itself.

The second line contains per each read integer position the product of all the input values except itself.

Example 1

Input

```
5
6
7
-8
11
#
```

HINT:
16 = 6 + 7 + (-8) + 11
-3696 = 6 · 7 · (-8) · 11

Output

```
16 15 14 29 10
-3696 -3080 -2640 2310 -1680
```

Python

```python
num = input()
numbers = []

while num != "#":
  numbers.append(int(num))
  num = input()

totalProduct = 1
totalSum = 0
for i in numbers:
  totalProduct = totalProduct * int(i)
  totalSum = totalSum + int(i)

res1 = ""
res2 = ""
for i in numbers:
    res1 = res1 + str(totalSum-int(i)) + " "
    res2 = res2 + str(totalProduct//int(i)) + " "


print(res1.rstrip())
print(res2.rstrip())
```

11 Training R3-AD

5 points

Introduction

In a galaxy far, far away, you are training the last model droid R3-AD to read regular English texts. R3-AD already knows a subset of the alphabet and it is not able to distinguish uppercase and lowercase. To consolidate its learning the droid must read a book. But as you can imagine R3-AD can read a word only if it is formed exclusively by letters it already knows. To evaluate how it is improving its reading capability you are curious about which words can be read and which cannot. Can you quickly code a program that reports which words can be read by R3-AD?

Input

The first line contains the letters R3-AD can read. Every letter will appear only once.

The second line contains a positive number referring to the number of words in the book.

Then there will be a line per each word of the book.

Output

For each of the words, print a single line stating Yes in case R3-AD can read it, and No otherwise.

Example

Input

```
abcdefimnop
5
mine
done
far
End
May
```

Output

```
Yes
Yes
No
Yes
No
```

C++

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <cctype>
using namespace std;


int main() {
    string letters;
    cin >> letters;
    int lettersSize = letters.size();
    int nWords;
    cin >> nWords;
    string word;
    vector<string> words;
    for(int i = 0; i < nWords; i++) {
        cin >> word;
        words.push_back(word);
    }

    bool found;
    vector<bool> result;
    int size;

    for(int j = 0; j < nWords; j++) { //Words
        size = words[j].length(); //Word size

        for(int k = 0; k < size; k++) {
            found = false;
            for(int l = 0; l < lettersSize; l++) {
                if(letters[l] == tolower(words[j][k])) {
                    found = true;
                }
            }
            if(!found) {
                result.push_back(false);
                break;
            }
        }
        if(found) {
            result.push_back(true);
        }
    }

    int resultSize = result.size();
```

```
    for(int m = 0; m < resultSize; m++) {
        if(result[m] == true) {
            cout << "Yes";
        }
        else{
            cout << "No";
        }
        if(m != resultSize-1) {
            cout << endl;
        }
    }

    return 0;
}
```

12 Automatic Hangman

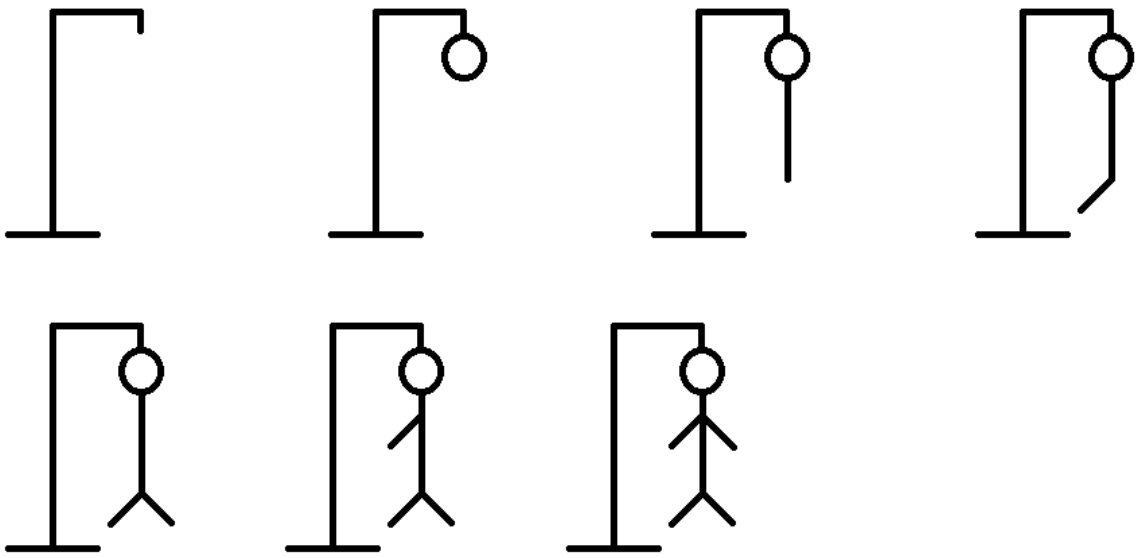
6 points

Introduction

For sure you have played the classic Hangman game. It's an ideal game to play with low resources, you can play with just a paper and a pencil. It's easy to learn how to play, every game doesn´t last more than 3 minutes, and it can be very funny!

So, what we want to do is an Automatic hangman application that simulates games.

The rules are simple: There is a hidden word to guess, and the player has to say letters that they think would be on the hidden word. If they miss a letter, they lose a life. For this simulation, the player has 7 lives, corresponding to the states of the game. When the 7 lives are lost, the game ends.



Input

The input consists of:

- The first line defines the word to guess in capital letters. This word must contain at least one letter to guess.
- The second line is the sequence of letters that represents the player tries. Letters can be repeated, so, if player repeats a wrong letter, they will lose another life. If they say a correct letter twice or more, nothing changes.

Output

The output consists of:

- The initial hidden word, expressed in "_".
- The final hidden word, where correct letters are shown.
- A final game status message:

STATUS	MESSAGE
Word completely guessed	`Player wins!`
Word not guessed completely, but player has lives	`Word not completed and player is still alive.`
The player lost all lives	`Player loses.`

- Number of lives when the game ends, expressed like `Lives: numberOfLives`

Example 1

Input

```
HELLO
HELO
```

Output

```
_____
HELLO
Player wins!
Lives: 7
```

Example 2

Input

```
HELLO
OXEXX
```

Output

```
_____
_E__O
Word not completed and player is still alive.
Lives: 4
```

Example 3

Input

```
HELLO
ABCDEFGHI
```

Output

```
_____
HE___
Player loses.
Lives: 0
```

C++

```cpp
#include <iostream>
#include <string>

int main()
{
    std::string guessWord;
    std::cin >> guessWord;

    std::string guessLetter;
    std::cin >> guessLetter;

    int lives = 7;
    int lettersSolved = 0;
    int guessWordSize = guessWord.size();

    std::string hiddenWord = std::string(guessWordSize, '_');
    std::string solvedWord = std::string(guessWordSize, '_');

    for (int i = 0; i < guessLetter.size(); i++)
    {
        bool found = false;
        for (int j = 0; j < guessWord.size(); j++)
        {
            if (guessWord[j] == guessLetter[i])
            {
                found = true;
                solvedWord[j] = guessLetter[i];
                lettersSolved++;
            }
        }
        if (!found)
            lives--;
        if ((lives == 0) || (lettersSolved == guessWordSize))
            break;
    }

    std::cout << hiddenWord << std::endl;
    std::cout << solvedWord << std::endl;

    if (lettersSolved == guessWordSize)
        std::cout<< "Player wins!";
    else if (lives > 0)
        std::cout<< "Word not completed and player is still alive.";
    else
        std::cout<< "Player loses.";
```

```
    std::cout << std::endl << "Lives: " << lives << std::endl;

    return 0;
}
```

13 A Beautiful Mind

6 points

Introduction

The movie "A beautiful mind" (2001) is a biography of mathematician John Nash. In 1994 he won the Nobel prize in Economics. He also made contributions to game theory, differential geometry and partial differential equations.

As part of his obsession with numerology he focused on bijective base-26 system where latin alphabet letters "A" to "Z" are used to represent the 26-digit values one to twenty-six as A=1, B=2, C=3, ..., Z=26. And what is next to Z? Quite simple, just AA=27, AB=28, and so on. In short, each digit position represents a power of twenty-six. Accordingly, we have that ABCD represents $1 \cdot 26^3 + 2 \cdot 26^2 + 3 \cdot 26^1 + 4 \cdot 26^0 = 19010$.

This maybe sounds strange , but it is the same system that many spreadsheets use to assign labels to their columns.

Can you write code to convert inputs between positive numbers and base-26 system strings...or the reverse?

Input

The input can be either a positive number or a base-26 system string.

Output

When the input is a positive number convert it to its corresponding base-26 system string. In case the input is a base-26 system string then convert the output to the positive number.

Example 1

Input

```
ABC
```

Output

```
731
```

Example 2

Input

```
54
```

Output

```
BB
```

Python

```python
# Read input as string
value = str(input())

if (value.isdigit()):
    # Converting from number to base-26
    number = int(value)
    res = number
    string = ""
    while number > 0:
      res = number % 26
      number = number // 26
      # Beware of the case when number is multiple of 26, then the remainder and
quotient must shifted since
      # our set begins with 1 instead of 0. Remember that 1 matches with A, 2
with B, ... and 26 with Z.
      if res == 0:
          res = res + 26
          number = number - 1
      string = chr(res+64) + string

    print(string)

else:
    # Converting from base-26 to number
    string = value
    value = 0
    power = 0

    # String is reversed to simplify how the traverse is done increasing the
power digit after digit
    for i in reversed(string):
      value = (ord(i) - 64) * pow(26, power) + value
      power = power + 1

    print(value)
```

14 La Casa de Papel

6 points

Introduction

The main safe-deposit of the Royal Mint is extremely secured against robbery. The famous group led by the Professor that wears Salvador Dalí masks has attempted several money heists.

To avoid these kinds of assaults, the main door allowing access to new banknotes is protected by a list of keywords that must be entered in proper order. To test the current level of security, you receive the sequence of keywords used but knowing that the words were shifted a certain number of positions. Can you write a program that prints out the list of keywords in the correct order to open the door?

Input

Several lines, each with a single keyword
Followed by a positive number referring to the shift to be applied. Such number is lesser than the number of keywords received.

A single character '#' marks the end of the input lines

Output

The list of the keywords properly shifted.

Example

Input

```
Berlin
Madrid
Paris
Rome
London
Tokyo
2
#
```

Output

```
Paris
Rome
London
Tokyo
Berlin
Madrid
```

C++

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

using namespace std;


int main() {
    string input;

    vector<string> keywords;
    int shifting;
    while(cin >> input && input != "#"){
        if(isdigit(input[0])) {
            shifting = stoi(input);
        }
        else{
            keywords.push_back(input);
        }
```

```
    }

    int size = keywords.size();
    int position;

    for (int j = 0; j < size; j++) {
        if(j+shifting < size) {
            position = j+shifting;
            cout << keywords[position];
            if(j != size-1) cout << endl;
        }
        else{
            position = (j+shifting)-size;
            cout << keywords[position];
            if(j != size-1) cout << endl;
        }
    }

    return 0;
}
```

15 Mesoamerican Pyramids

6 points

Introduction

Dr. Jones, the world-renowned archeology professor, is investigating the pre-Columbian cultures and civilizations located in Central America. Their architecture produced the Mesoamerican pyramids. The most popular were built by Aztecs and Mayans. These structures, although similar to the Egyptian pyramids, are distinguished by having flat tops and stairs ascending their faces. It is important to note that the pyramids have layers and the number of steps per layer is the same on all of its faces.

During the investigation, a book was found that contained the plans of hundreds of Mesoamerican pyramids. Dr. Jones believes that most of the plans are real, but some of them are fake. He realized in the fake plans that the number of stairs per layer is not the same on all of the faces. Because Dr. Jones is very busy and given the clue he has pointed out, can you help him to detect fake Mesoamerican pyramids?

Input

The first line gives the number of pyramids to analyse. Then for each pyramid a line with the following information is provided: the name of the village in a single word, for the north face of the pyramid the number of steps per each layer in ascending order separated with a white space, then a separator "#", and the same information in descending order for the south face.

Output

For each pyramid return a line stating if the number of steps per layer is the same in both faces.

Example

Input

```
2
Tenochtitlan 2 3 4 5 4 # 4 5 4 3 2
Teotihuacan 1 2 2 3 4 4 6 7 # 7 6 4 3 3 2 2 1
```

Output

```
Tenochtitlan has same number of steps for both faces
Teotihuacan has NOT same number of steps for both faces
```

Python

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

for i in range(num):
  ok = True
  line = input().split()
  name = line[0]
  length = len(line)
  counter = 1
  j = line[counter]
  while j != "#":
    if j != line[length - counter]:
      ok = False
      break
    counter = counter + 1
    j = line[counter]

  if ok:
    print( name + " has same number of steps for both faces")
  else:
    print( name + " has NOT same number of steps for both faces")
```

16 Acronymizer

7 points

Introduction

You have a summer internship at a local newspaper. Your job consists of reviewing the news to identify potential acronyms in the text. An acronym is an abbreviation formed from the initial letters of other words and written as a single word. To make your life easier, you have decided to write a program to do the job.

HINT: To make things easier consider that there will not be two consecutive acronyms.

Input

A sentence with words separated by a single space, without commas and ended with a full stop.

Output

A sentence replacing the input sentence with the consecutive words that contain the first character capitalized by its corresponding acronym.

Example 1

Input

```
Welcome to Code Wars held by Hewlett Packard in Barcelona site.
```

Output

```
Welcome to CW held by HP in Barcelona site.
```

Example 2

Input

```
They will travel to United States and visit the National Aeronautics Space
Administration.
```

Output

```
They will travel to US and visit the NASA.
```

Python

```
text = input()

outputText = ""
words = []

for i in text.split():
  # Store the words that have a capital letter that are acronym candidates
  if i[0].isupper():
    words.append(i)
  else:
    # Add to the output string the word read when there are no acronym
candidates stored
    if len(words) == 0:
      outputText += i  + " "
    # Add to the output string the single word with a capital letter
    elif len(words) == 1:
      outputText += words[0] + " " + i + " "
    # Process the acronym candidates
    else:
      for w in words:
        outputText += w[0]
      outputText += " " + i  + " "
    words = []

if len(words) > 0:
  for w in words:
    outputText += w[0]
  outputText += "."

print(outputText.rstrip())
```

17 Fixing Sentences

7 points

Introduction

A virus has infected your high school computer server, and it's messing up the e-mail service. It is attacking e-mail messages by randomly reversing some words. This is affecting students as they cannot understand their homework. You decided to help your friends by providing them with a mobile app that reverts the effects of the virus attack. Your app uses an internet dictionary to check word by word their correctness in a sentence, and when a potential reversed word is detected, you get its position within the sentence. With this job done, you just need to reverse those words to resolve the trouble created by the virus.

Can you write a program that, given a sentence with reversed words and their positions, prints out the fixed sentence?

Input

A line containing a single sentence with some words reversed.
Several lines with a single positive number describing the position of a reversed word.
A single character '#' marks the end of the input lines.

Output

A single line with the original sentence fixed.

Example 1

Input

```
Notice that this drow is reversed and this rehtona oot.
4
9
10
#
```

Output

```
Notice that this word is reversed and this another too.
```

Example 2

Input

```
Here we do not have a full pots
8
#
```

Output

```
Here we do not have a full stop
```

Python

```python
sentence = input().split(" ")

wordToReverse = input()

while wordToReverse !="#":

  # Get the word
  pos = int(wordToReverse) - 1
  word = sentence[pos]

  # Reverse the given word
  reversedWord = word[::-1]

  # Consider the case of having a full stop at the end of a sentence
  if reversedWord[0] == ".":
      reversedWord = reversedWord[1:] + "."

  # Replace original word with the reversed word
  sentence[pos] = reversedWord

  # Get next word
  wordToReverse = input()

print(" ".join(sentence))
```

18 Framing

7 points

Introduction

A new app for messaging Internet Frames is being developed. In this app every single message is printed having a word per line inside a frame like in this example.

```
##########
# Hello  #
# World! #
##########
```

To make things easier you think about coding a program to build several Internet Frames one after the other. The hight of the frame is defined by the number of words inside the Internet Frame. And the width of a frame depends on the longest word in the message. When concatenating Internet Frames of different width, print a single horizontal line between the frames with the wider size of previous and next frame.

Input

A line or more with one or more words per line.
A single character '#' marks the end of the input lines.

Output

One or more rectangular frames containing the words of each line printed one per line in a rectangular frame.

Example 1

Input

```
Hi! This is frame number one
This is the 2nd
Finally, the fantastic third frame
#
```

Output

```
##########
# Hi!    #
# This   #
# is     #
# frame  #
# number #
# one    #
##########
# This #
# is   #
# the  #
# 2nd  #
############
# Finally,  #
# the       #
# fantastic #
# third     #
# frame     #
############
```

Python

```
footer=""

# Read lines from standard input
line = input()
while line != "#":

# Get the single words
  words = line.split()
  longestWord = 0

# For each line find the longest word to find the size of the frame
  for i in words:
    if len(i) > longestWord:
      longestWord = len(i)

  # Compose header
  header = "#"*(longestWord+4)

  # Print longest header or footer
  if len(header) > len(footer):
    print(header)
  else:
    print(footer)

  # Print words framed
```

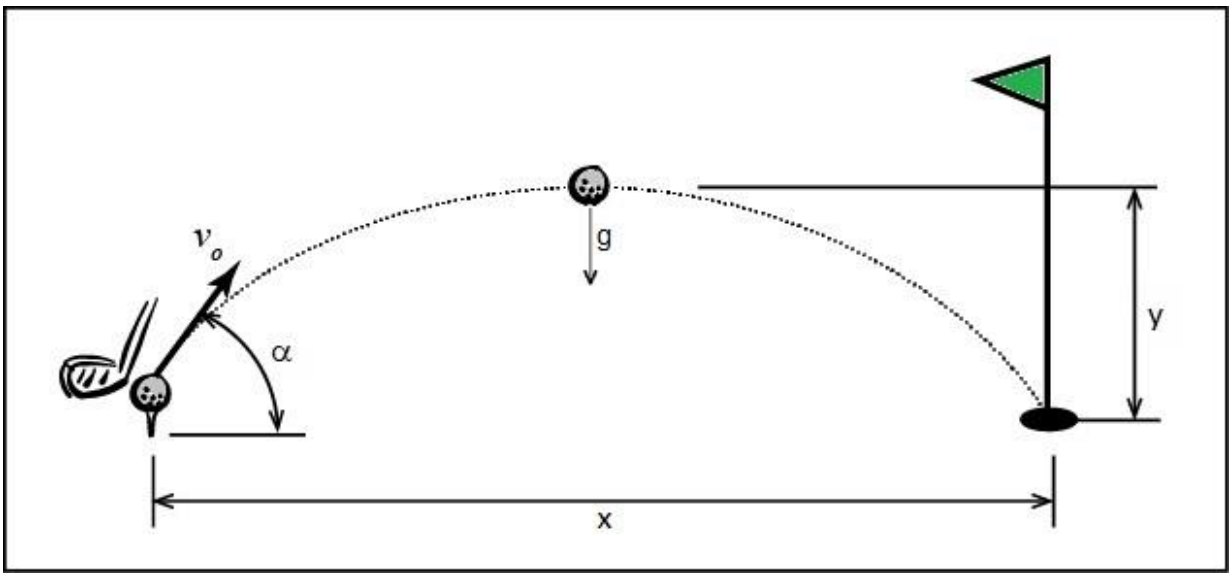
```
  for i in words:
    print("# " + i + " "*(longestWord-len(i)) + " #")

  # Compose footer
  footer = "#"*(longestWord+4)

  # Read next line
  line = input()

# Print footer
print(footer)
```

19 Golf Ball

7 points

Introduction

When a golf ball is hit, its flight follows a curved path known as parabola. The shape of the parabola is affected by two main forces, gravity, and air resistance. To keep things easier, let's assume that air resistance as negligible and just consider gravity (9.8 m/s^2).



Such movement is composed by a uniform rectilinear motion on the X-axis and an uniformly accelerated rectilinear motion on the Y-axis. So, the distance formulas are:

$$x = x_0 + v_0 * cos(\alpha) * t$$

$$y = y_0 + v_0 * sin(\alpha) * t - 1/2 * g * t^2$$

It is also assumed that the golf field is flat, that is the initial and final heights are equal.

Given a golf ball with a diameter of 42.7 millimetres hit at a certain angle and aiming to get the ball inside a 3 meters circle around a target position, could you find out the maximum and minimum speed that must be achieved providing a precision of two decimal places?

To help you in this task we recommend applying these formulas:

$$minDistance = targetDistance - 3 + (0.0427/2)$$

$$maxDistance = targetDistance + 3 - (0.0427/2)$$

$$minTime = \sqrt{\frac{2 * minDistance * sin(\alpha)}{g * cos(\alpha)}}$$

$$maxTime = \sqrt{\frac{2 * maxDistance * sin(\alpha)}{g * cos(\alpha)}}$$

$$minSpeed = \frac{minDistance}{cos(\alpha) * minTime}$$

$$\boldsymbol{maxSpeed = \frac{maxDistance}{cos(\alpha) * maxTime}}$$

HINT: Don't forget that trigonometric functions are performed in radians!

Input

Two lines are provided.

The first line is a positive representing the target distance in meters.

The second line is also positive and defines the angle in degrees.

Output

There are also two output lines. The first line contains the maximum speed in meters per second. And the second line is for the minimum speed in meters per second.

Example

Input

```
100
50
```

Output

```
The maximum speed is: 32.01 m/s.
The minimum speed is: 31.07 m/s.
```

Python

```
import math

# We need a program that returns maximum and minimum speed of the ball
# to get inside 3 meters circle around target when the initial angle is changed
# initial (constant) values

g = 9.8
ball_diameter = 0.0427 # 42.7 mm

# input
target_distance = int(input())
angle = int(input())

# Student may use this manual method to find
# the speed of the ball

min_distance = target_distance-3+(0.0427/2)
max_distance = target_distance+3-(0.0427/2)

min_time =
math.sqrt((2*min_distance*math.sin(math.radians(angle)))/(g*math.cos(math.radian
s(angle))))
max_time =
math.sqrt((2*max_distance*math.sin(math.radians(angle)))/(g*math.cos(math.radian
s(angle))))

min_speed = min_distance/(math.cos(math.radians(angle))*min_time)
max_speed = max_distance/(math.cos(math.radians(angle))*max_time)

# printing maximum and minimum (initial) speed of the ball to get inside 3
meters circle around target
print ("The maximum speed is: " + "{:.2f}".format(max_speed) + " m/s.")
print ("The minimum speed is: " + "{:.2f}".format(min_speed) + " m/s.")
```

20 Sudoku Friends

7 points

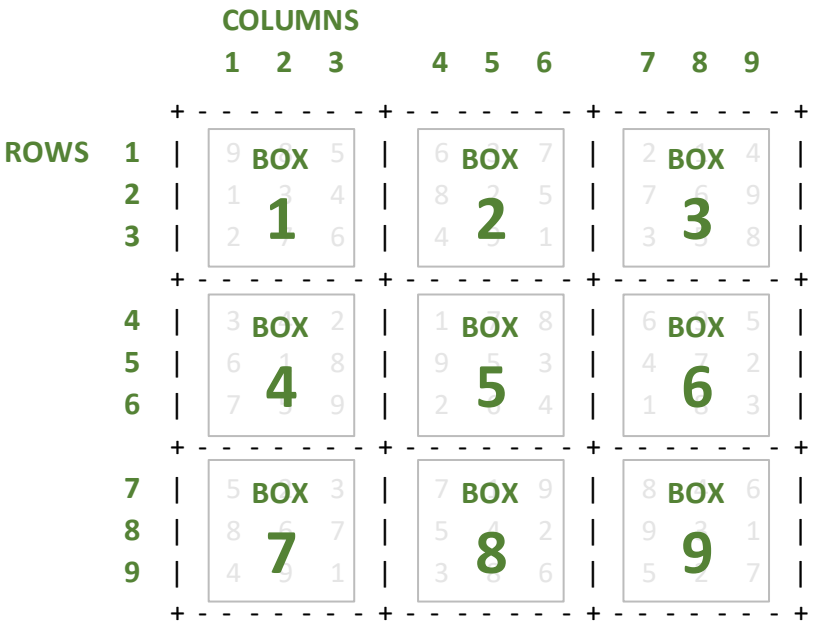
Introduction

A Sudoku is a type of puzzle where you must fill a 9x9 grid with numbers from 1 to 9 with the following rules:

- Numbers cannot repeat in a row
- Numbers cannot repeat in a column
- Numbers cannot repeat in a box

This will help you understand our coordinate system:

- A box is a 3x3 region that divides the Sudoku grid in a 3x3, normally represented with a darker outline.
- The rows are counted from top to bottom being the topmost number 1
- The columns are counted from left to right being the leftmost number 1
- The boxes are counted in reading order, being the top-left number 1 and the bottom-right number 9



In Sudoku a cell is considered a friend if its value coincides with the number of its row, column, or box. We need a program to count the number of friend cells in a given Sudoku.

Input

81 digits from 1 to 9 separated by a space representing a Sudoku.
The numbers in the Sudoku are assigned in reading order.

Ex.: The 15th digit will be in row 2, column 6, box 2.

Output

The total number of friend cells in the format:

```
Number of friends = number
```

Example

In this example, the *cells in red* are friends.

```
+------+-------+-------+
|9 8 5 | 6 3 7 | 2 1 4 |
|1 3 4 | 8 2 5 | 7 6 9 |
|2 7 6 | 4 9 1 | 3 5 8 |
+------+-------+-------+
|3 4 2 | 1 7 8 | 6 9 5 |
|6 1 8 | 9 5 3 | 4 7 2 |
|7 5 9 | 2 6 4 | 1 8 3 |
+------+-------+-------+
|5 2 3 | 7 1 9 | 8 4 6 |
|8 6 7 | 5 4 2 | 9 3 1 |
|4 9 1 | 3 8 6 | 5 2 7 |
+------+-------+-------+
```

Input (notice that although this text appears as three lines, the program input is a single line)

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

Output

```
Number of friends = 21
```

C++

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

int getIndex(int r, int c)
{
    return r * 9 + c;
}

bool inRange(int v, int a, int b)
{
    if (v < a)
        return false;
    if (v > b)
        return false;
    return true;
}

int getBox(int row, int col)
{
    if (inRange(col, 1, 3) && inRange(row, 1, 3))
        return 1;
    if (inRange(col, 4, 6) && inRange(row, 1, 3))
        return 2;
    if (inRange(col, 7, 9) && inRange(row, 1, 3))
        return 3;
    if (inRange(col, 1, 3) && inRange(row, 4, 6))
        return 4;
    if (inRange(col, 4, 6) && inRange(row, 4, 6))
        return 5;
    if (inRange(col, 7, 9) && inRange(row, 4, 6))
        return 6;
    if (inRange(col, 1, 3) && inRange(row, 7, 9))
        return 7;
    if (inRange(col, 4, 6) && inRange(row, 7, 9))
        return 8;
    if (inRange(col, 7, 9) && inRange(row, 7, 9))
        return 9;
}

bool checkIfFriend(int v, int row, int col, int box)
{
    if (v == row || v == col || v == box)
        return true;
    else
        return false;
```

```
}

std::vector<int> readSudoku()
{
    std::vector<int> sudoku;
    for (int i = 0; i < 81; i++)
    {
        int v = 0;
        std::cin >> v;
        sudoku.push_back(v);
    }
    return sudoku;
}

int main()
{
    std::vector<int> sudoku = readSudoku();
    int friendCount = 0;

    for (int r = 0; r < 9; r++)
    {
        int row = r + 1;
        for (int c = 0; c < 9; c++)
        {
            int cell = sudoku[getIndex(r, c)];
            int col = c + 1;
            int box = getBox(row, col);
            if (checkIfFriend(cell, row, col, box))
            {
                friendCount += 1;
            }
        }
    }

    std::cout << "Number of friends = " << friendCount << std::endl;

    return 0;
}
```

21 Close Encounters Of The String Kind

8 points

Introduction

An unexpected event has happened to mankind! It has been confirmed that a radio signal has been received from outer space. The signal does not contain any musical tones like in science-fiction movies. Instead, it is formed by a series of strings of characters containing only single digit numbers, letters and the number sign or hash (#).

These strings are being analyzed by top scientists in the world and the only clue up to now is that a strange pattern has been found. That is, there are exactly 3 hashes between a pair of two digits that add up to the number 10, as in these examples:

```
dhj1###9Adkjkldj
mcvnjkdf8##j#2dkL
aBc4thE#hjsldf#dJ#6dkjFkd#
```

Not all the messages follow this pattern. To quickly advance in this investigation a computer will be used to classify all the strings that are been received. Your help is key to advancing the investigation. Can you code a program that detects if a given message follows the alien pattern observed?

Note that the received message has been pre-processed to split it in smaller strings containing only one possible alien message: if during the analysis you find a *failure*, you don't need to keep analyzing the string for potential new good patterns. For example: this string in not a valid input:

`dj1#k###9adkjkldjdhj1###9adkjkldj`

so, your program doesn't need to consider it.

Input

The alien string pattern to be processed.

Output

True is printed when the pattern with exactly 3 hashes between a pair of two digits that add up to 10 is detected. Otherwise just print False.

Example 1

Input

```
dhj1###9adkjkldj
```

Output

```
True
```

Example 2

Input

```
dj1#k###9adkjkldj
```

Output

```
False
```

Example 3

Input

```
opdhdj3#kdf##6adldj
```

Output

```
False
```

Python

```python
# Auxiliar function to check whether a valid wall exists
def checkWall(word, start, end):
  subWord = word[start:end]
  count = 0
  for i in subWord:
    if i == "#":
      count = count + 1
  return count

# Read the input

word = input()

# Auxiliar variables

res = False
firstNumber = -1
secondNumber = -1
pos = 0
posFirstNumber = -1
posSecondNumber = -1

# Look for first and second numbers along the input

for i in word:
    if i.isdigit():
        if firstNumber == -1:
          firstNumber = int(i)
          posFirstNumber = pos
          #print(pos, str(i))
          res = False

        elif secondNumber == -1:
          secondNumber = int(i)
          posSecondNumber = pos
          #print(pos, str(i))

          # Once the two numbers are identified
          # Check there is a wall between them
          count = checkWall(word, int(posFirstNumber)+1, int(posSecondNumber))
          if count == 3:
            # Check first and second number add 10
            if (firstNumber + secondNumber == 10):
              res = True
              # Continue looking for next case in the input
```

```
            firstNumber = -1
            secondNumber = -1
            posFirstNumber = -1
            posSecondNumber = -1
          else:
            res = False
            break

    pos = pos + 1

print(res)
```

22 Martes y Trece

8 points

Introduction

There is a popular superstition in Western countries that considers Friday the 13th is an unlucky day. In Spanish-speaking countries the same happens when the 13th day of the month falls on Tuesday, that is called "Martes y Trece". To avoid any potential unluckiness, you decided to find out in advance when the next Martes y Trece will happen. To do so you will code a program that, given a year, returns in temporal order the next Martes y Trece dates.

HINT: To help you to develop your program you can count on Zeller's congruence that calculates the day of the week for a given calendar date.

1. Given day number D, month number M and year Y
2. If M is 1 or 2, add 12 to M, and subtract 1 from Y
3. **Let C be the zero-based century (actually [Y/100]) and K the year of the century (Y mod 100).**
4. Add together the integer parts of (2.6M-5.39), (K/4) and (C/4). (The integer part of a number is the whole number part: integer part of 2.3 is 2, and of 6.7 is 6. Note that the integer part of -1.7 is -2)
5. Add to this D and K, and subtract 2C
6. Find the remainder when this **number*** is divided by 7, then the remainder is the day of the week where Sunday = 0, Monday = 1, Tuesday = 2, Wednesday = 3, Thursday = 4, Friday = 5 and Saturday = 6

   Such resulting number can be a negative value and depending on the language (C, C++ and Java) the operator '%' can lead to a misleading result. In Python this operator returns the (modulus) remaining numbers by dividing first number from the second. But same operator in C, C++ and Java strictly returns the remainder so when handling with a negative divider the value returned is negative.

Input

A single positive integer value representing the year to check.

Output

The list of dates following temporal order for that year that are Martes y trece.

Example 1

Input

```
2023
```

Output

```
Martes y Trece will occur on 13/6/2023
```

Example 2

Input

```
1998
```

Output

```
Martes y Trece will occur on 13/1/1998
Martes y Trece will occur on 13/10/1998
```

Python

```python
def switch(h):
  return {
    0 : "Sunday",
    1 : "Monday",
    2 : "Tuesday",
    3 : "Wednesday",
    4 : "Thursday",
    5 : "Friday",
    6 : "Saturday",
  }[h]

def ZellerAlgorithm(D, M, Y):
  if (M < 3) :
    M = M + 12
    Y = Y - 1
  C = Y // 100
  K = Y % 100
  h = int(2.6*M-5.39) + K//4 + C//4 + D + K - 2*C
  h = h % 7
  return(switch(h))

day = 13
year = int(input())

for month in range(1,13):
  if (ZellerAlgorithm(day, month, year) == "Tuesday"):
      print ("Martes y Trece will occur on " + str(day) + "/" + str(month) + "/"
+ str(year))
```

23 Minesweeper

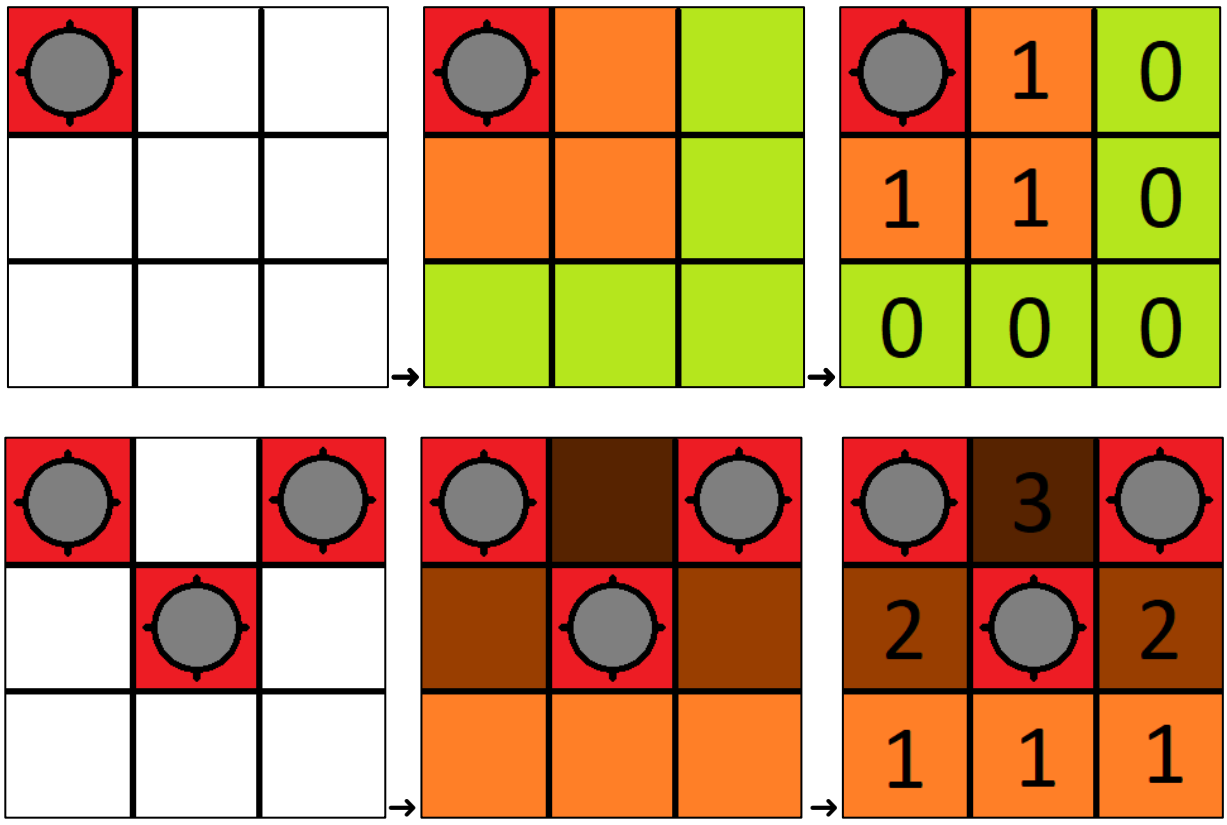
9 points

Introduction

A mission has been entrusted to you.

The government is having trouble delimiting safe zones in places where mines have been detected. The objective is to indicate the level of each perimeter of the dangerous place. To do this, you will be assigned to a control zone, and the places where the mines were detected.

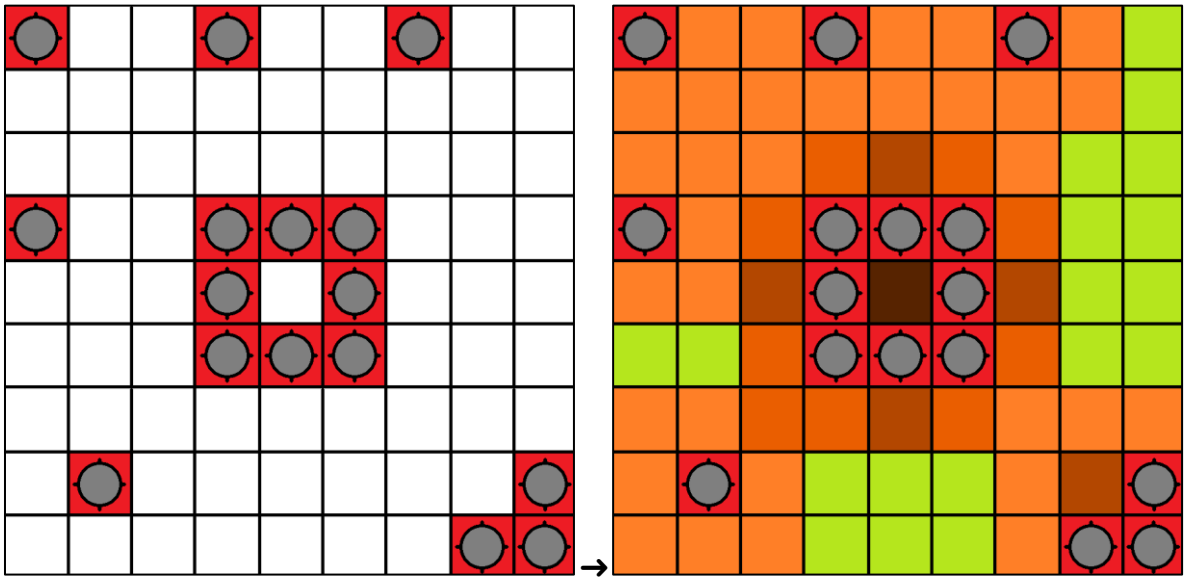
The president's words are very clear: "We need you to show us a map of the area, indicating the level of danger in terms of nearby mines, with 0 being an area that has no surrounding mines, and 8 an area completely surrounded by mines. Good luck."



There are different area levels. You must differentiate between "Easy" areas, "Medium" areas and "Hard" areas.

LEVEL	DIMENSION
Easy	3 X 3
Medium	6 X 6
Hard	9 X 9

This is an example of a Hard Level area



Input

The input consists of:

- The first line defines the area level to work in. (Easy, Medium or Hard)
- The second line is the number of mines on the area. (You must assume that there will always be at least one mine, and never more than the area capacity)
- A sequence of LINES representing each mine coordinate (x, y)

HINT: Notice that the first coordinate is (1,1).

Output

The output consists of the final map, representing the mines positions with the character "#", and each sub-area with the integer of nearby mines.

Example 1

Input

```
Easy
1
1 1
```

Output

```
#10
110
000
```

Example 2

Input

```
Easy
3
1 1
1 3
2 2
```

Output

```
#3#
2#2
111
```

Example 3

Input

```
Hard
16
1 1
1 4
1 7
4 1
4 4
4 5
4 6
5 4
5 6
6 4
6 5
6 6
8 2
8 9
9 8
9 9
```

Output

```
#11#11#10
111111110
111232100
#12###200
113#8#300
002###200
112232111
1#100013#
1110001##
```

C++

```cpp
#include <iostream>
#include <string>

using namespace std;

#define easy 3
#define medium 6
#define hard 9


int main() {
    int board_size; //easy, medium, hard
    string difficult;
    cin >> difficult;

    if(difficult == "Easy") {
        board_size = easy;
    }
    else if(difficult == "Medium") {
        board_size = medium;
    }
    else if(difficult == "Hard") {
        board_size = hard;
    }

    int board[board_size][board_size];
    int number_of_bombs;
    cin >> number_of_bombs;
    int bombs[number_of_bombs][2];

    int bombX, bombY;
    for(int i = 0; i < number_of_bombs; i++) {
        cin >> bombX >> bombY;
        bombs[i][0] = bombX - 1;
        bombs[i][1] = bombY - 1;
    }

    for(int j = 0; j < board_size; j++) {
        for(int k = 0; k < board_size; k++) {
            board[j][k] = 0;
        }
    }

    for(int l = 0; l < number_of_bombs; l++) {
        board[bombs[l][0]][bombs[l][1]] = -1;
```

```
    }



    int counter;
    for(int i = 0; i < board_size; i++) {
        for(int j = 0; j < board_size; j++) {
            counter = 0;
            if(board[i][j] != -1) {
                if(i-1 >= 0) {if(board[i-1][j] == -1) counter++;}
                if(i+1 < board_size) {if(board[i+1][j] == -1) counter++;}
                if(j-1 >= 0) {if(board[i][j-1] == -1) counter++;}
                if(j+1 < board_size) {if(board[i][j+1] == -1) counter++;}
                if(i-1 >= 0 && j-1 >= 0) {if(board[i-1][j-1] == -1) counter++;}
                if(i+1 < board_size && j+1 < board_size) {if(board[i+1][j+1] ==
-1) counter++;}
                if(i-1 >= 0 && j+1 < board_size) {if(board[i-1][j+1] == -1)
counter++;}
                if(i+1 < board_size && j-1 >= 0) {if(board[i+1][j-1] == -1)
counter++;}
                board[i][j] = counter;
            }
        }
    }

    for(int i = 0; i < board_size; i++) {
        for(int j = 0; j < board_size; j++) {
            if(board[i][j] == -1) {
                cout << "#";
            }
        else{
            cout << board[i][j];
        }
    }
    cout << endl;
    }

    return 0;
}
```

24 Synthetic Division

10 points

Introduction

What a name! The synthetic division refers to the method to divide a polynomial by the binomial $(x - c)$ where c is a constant. Consider the case of dividing

$$(-3x^3 + 5x - 2)/(x - 5)$$

Do not be afraid since Ruffini's rule will help you to do so. Let's see how it works with the previous example. First, it begins by drawing a couple of crossed lines and put the c value at left.

```
     |
   5 |
-------------------------
     |
```

Next step is to write the coefficients of the polynomial ordered from highest to lowest degree at the top. If some degree is missing, put it as a zero in its corresponding place. In this case the coefficients are -3, for x^3, 5 for x and -2 as an independent term.

```
     |   -3    0    5   -2
   5 |
-------------------------
     |
```

Now copy the coefficient of highest degree, which is -3, at the top just under horizontal line.

```
     |   -3    0    5   -2
   5 |
-------------------------
     |   -3
```

Multiply this number by the value of c, which is 5, and the result is put next above the horizontal line.

```
     |   -3    0    5   -2
   5 |       -15
-------------------------
     |   -3
```

Then add the values in the second column, write the result under the horizontal line and repeat the multiplication with the value of c.

```
     |   -3    0    5   -2
   5 |       -15  -75
-------------------------
     |   -3  -15
```

Again, it is time to add the numbers in the column and put the result down the horizontal line.

```
     |   -3    0    5   -2
   5|        -15  -75
-------------------------
     |   -3  -15  -70
```

Repeat these steps until reaching the last column.

```
     |   -3    0    5   -2
   5|        -15  -75 -350
-------------------------
     |   -3  -15  -70 -352
```

Last number at the right, that is -352, is the remainder of the division. And the polynomial quotient of the division is built from the coefficient numbers that are previous to the remainder from left to right providing as a result $-3x^2 - 15x - 70$

Now that you have refreshed how the Ruffini's rule work, can you write a program to perform a synthetic division?

Important note: The expected length of the horizontal line is 5 dashed characters per each number plus an extra dash aligned with the vertical line.

Input

Two lines form the input. First line contains the coefficients of the dividend where a zero represents any missing terms. Second line have a single number representing the c constant of the binomial divisor.

Output

The final table after applying Ruffini's rule. Please note that per each number a fixed size of 5 positions is defined in order to have the number properly printed in columns.

Example

Input

```
-3 0 5 -2
5
```

Output

```
     |   -3    0    5   -2
   5|        -15  -75 -350
-------------------------
     |   -3  -15  -70 -352
```

Python

```python
# Read from input the polynomial coefs and the divisor
coefs = input().split()
coefs = [eval(i) for i in coefs]
divisor = int(input())
aux = []
res = []

# Do the ruffini calculation
for i in range(len(coefs)):
  if i == 0:
    res.append(coefs[i])
    aux.append("     ")
  else:
    aux.append(divisor * int(res[i-1]))
    res.append(divisor * int(res[i-1]) + coefs[i])

# Print out the results

row1 = "     |"
row2 = f'{divisor:5d}' + "|"
row3 = "     |"

for i in coefs:
  row1 = row1 + f'{i:5d}'

for i in aux:
  if i == "     ":
    row2 = row2 + i
  else:
    row2 = row2 + f'{i:5d}'

for i in res:
  row3 = row3 + f'{i:5d}'

print(row1)
print(row2)
print("-"*len(row3))
print(row3)
```

25 The Sheldon Prime

11 points

Introduction

The 73rd episode of the TV series "*The Big Bang Theory*" is very special for math lovers. In it, Sheldon Cooper asks Raj, Howard, and Leonard "What is the best number? By the way, there is only one correct answer". Sheldon explains to them that the best number is 73 because 73 is the 21st prime number. Its mirror, 37, is the 12th prime, which in turn is the mirror of 21!!! Mathematicians have named this number the Sheldon prime.

Since we are math lovers, we would like to find if there are other numbers like 73 or that are somehow related to it. To this end, we ask you to make a program that, given a natural number, indicates what type of relation it has with the Sheldon prime according to these rules:

1. A number N is a **Sheldon prime** if:
   - *N* is prime (e.g., 73)
   - *M*, which is the mirror of *N*, is prime (37)
   - The position of *N* in the prime numbers (21st) is the mirror of the position of *M* (12th)
2. A number *N* is a **relative** of the Sheldon prime if:
   - *N* is prime (e.g., 769)
   - *M*, which is the mirror of *N*, is prime (967)
   - The position of *N* (136th) is a permutation (with the same digits) of the position of *M* (163rd)
3. A number *N* is a **close friend** of the Sheldon prime if:
   - *N* is prime (e.g., 1409)
   - *M*, which is the mirror of *N*, is prime (9041)
   - The position of *N* (223rd) and the position of *M* (1123rd) are primes
4. A number *N* is a **friend** of Sheldon prime if:
   - *N* is prime (e.g., 17)
   - *M*, which is the mirror of *N*, is prime (e.g., 71)

Notice that every number can only fit in one of the categories, giving higher priority to 1 (i.e., Sheldon prime) and less priority to 4 (i.e., a friend).

Input

The input is a natural number

Output

The output is a message indicating the type of the input number with the format shown below:

- If it is a Sheldon prime the message is: "Number *N* is a Sheldon prime!"
- If it is a relative of Sheldon prime: "Number *N* is a Sheldon prime relative"
- If it is a close friend of Sheldon prime: "Number *N* is a close friend of Sheldon prime"
- If it is a friend of Sheldon prime: "Number *N* is a friend of Sheldon prime"
- If it is a number not related to a Sheldon prime: "Number *N* is not related to Sheldon prime"

Note that *N* should be the input number

Example 1

Input

```
73
```

Output

```
Number 73 is a Sheldon prime!
```

Example 2

Input

```
769
```

Output

```
Number 769 is a Sheldon prime relative
```

Example 3

Input

```
9
```

Output

```
Number 9 is not related to Sheldon prime
```

Example 4

Input

```
17
```

Output

```
Number 17 is a friend of Sheldon prime
```

Example 5

Input

```
1409
```

Output

```
Number 1409 is a close friend of Sheldon prime
```

C++

```
// Given a number, return if it is the Sheldon prime, a Sheldon prime relative,
a close friend, a friend, or nothing
//
// Sheldon prime:
//   - N is prime
//   - rev(N) is prime
//   - pos(N) == rev(pos(rev(N)))
//
// Sheldon prime relative (medium/complex):
//   - N is prime
//   - rev(N) is prime
//   - pos(N) == anyPermutation(pos(rev(N)))
//
// Close friend of Sheldon prime (easy/medium):
//   - N is prime
//   - rev(N) is prime
//   - pos(N) and pos(rev(N)) are prime
//
// Friend of Sheldon prime (easy):
//   - N is prime
//   - rev(N) is prime

#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <bits/stdc++.h>


using namespace std;


/*** Types ***/


// Type of numbers
enum Number_t
{
        SheldonPrime,
        Relative,
        CloseFriend,
        Friend,
        Other
};
```

```
/*** Local functions ***/


// Return whether a number is prime or not
bool isPrime(uint32_t num)
{
        if (num < 2)
        {
                return false;
        }

        bool isPrime = true;

    // Just to save some time
    if ((num > 2) && ((num % 2) == 0))
    {
        isPrime = false;
    }

        for (uint32_t i = 3; (i <= (num / 2)) && isPrime; i+=2)
        {
                if ((num % i) == 0)
                {
                        isPrime = false;
                }
        }

        return isPrime;
}


// Reverse a given number
uint32_t reverseNumber(uint32_t num)
{
    string numStr = to_string(num);
    reverse(numStr.begin(), numStr.end());
    return static_cast<uint32_t>(stoul(numStr));
}


// Recursive function that adds to 'permutations' all the combinations given a
number prefix ('numPrefix') and a series of digits
void getAllPermutations(const vector<int> & digits, uint32_t numPrefix,
map<uint32_t, bool> & permutations)
```

```
{
    for (size_t i = 0; i < digits.size(); i++)
    {
        // New prefix
        uint32_t num = numPrefix * 10 + digits[i];

        // Remove the used digit from remainderDigits
        vector<int> remainderDigits = digits;
        remainderDigits.erase(remainderDigits.begin() + i);

        // Check if there are more digits to consume
        if (remainderDigits.size() > 0)
        {
            getAllPermutations(remainderDigits, num, permutations);
        }
        else
        {
            permutations[num] = true;
        }
    }
}


// Return all the possible permutations of 'num' in 'permutations'
void getPermutationList(uint32_t num, map<uint32_t, bool> & permutations)
{
    string numStr = to_string(num);

    // Digits in num
    vector<int> digits;
    for (auto c : numStr)
    {
        digits.push_back(static_cast<int>(c - '0'));
    }

    getAllPermutations(digits, 0, permutations);
}


/** Main **/

int main()
{
    // Get the input number
    uint32_t num;
    cin >> num;
```

```
    // TODO: Remove. Just to check all the numbers
    //for (num = 0; num < 100000; num++)
    //{

    Number_t numType = Other;

    // Get the reverse number
    uint32_t revNum = reverseNumber(num);

    // Check if num and revNum are primes
    bool numIsPrime = isPrime(num);
    bool revNumIsPrime = numIsPrime && isPrime(revNum);

    // To know the positions of the primes, calculate all of the primes until
the max of (num, revNum)
    uint32_t maxNum = 0;
    if (numIsPrime && revNumIsPrime)
    {
        maxNum = max(num, revNum);
    }

    // Calculate all the primes and their positions until maxNum (prime, pos)
    map<uint32_t, uint32_t> primeNumbers;
    uint32_t count = 0;
    for (uint32_t n = 1; n <= maxNum; n++)
    {
        if (isPrime(n))
        {
            count++;
            primeNumbers[n] = count;
        }
    }

    if (numIsPrime && revNumIsPrime)
    {
        // At least, they are friends
        numType = Friend;

        // Positions
        uint32_t posNum = primeNumbers[num];
        uint32_t posRevNum = primeNumbers[revNum];
        uint32_t revPosRevNum = reverseNumber(posRevNum);

        // Check for Sheldon prime
        if (posNum == revPosRevNum)
        {
            numType = SheldonPrime;
```

```
        }
        else
        {
            // Check for relative
            map<uint32_t, bool> permutationList;
            getPermutationList(posRevNum, permutationList);
            if (permutationList.find(posNum) != permutationList.end())
            {
                numType = Relative;
            }
            else
            {
                // Check for close friends
                bool isPosNumPrime = primeNumbers.find(posNum) !=
primeNumbers.end();
                bool isposRevNumPrime = primeNumbers.find(posRevNum) !=
primeNumbers.end();
                if (isPosNumPrime && isposRevNumPrime)
                {
                    numType = CloseFriend;
                }
            }
        }
    }

    // Print the result
    switch (numType)
    {
        case Other:
            cout << "Number " << num << " is not related to Sheldon prime" <<
endl;
            break;

        case Friend:
            cout << "Number " << num << " is a friend of Sheldon prime" << endl;
            //cout << "Number " << num << " is friend" << endl;
                break;

        case CloseFriend:
            cout << "Number " << num << " is a close friend of Sheldon prime" <<
endl;
            //cout << "Number " << num << " is close friend" << endl;
            break;

        case Relative:
            cout << "Number " << num << " is a Sheldon prime relative" << endl;
            //cout << "Number " << num << " is relative" << endl;
```

```
            break;

        case SheldonPrime:
            cout << "Number " << num << " is a Sheldon prime!" << endl;
            break;

        default:
            cout << "ERROR" << endl;
    }

    return 0;
}
```

26 Chain Reaction

12 points

Introduction

We want to implement a simulation of how subatomic particles behave when a chain reaction occurs. The first model is a very basic simplification, but it will help improving future versions.

The objective is to shoot a particle (the detonator) to space with particles (reactors) and figure out the final state of the reaction.

The reactors particles are represented as circles with the following parameters:

- x: integer, position in x-axis
- y: integer, position in y-axis
- r: integer >0, particle radius
- e: integer >=0, reaction radius

The detonator particle is represented as reactors, but without reaction radius.
The simulation will start shooting the detonator to a given position.
All reactors colliding with the detonator will be hit and start a chain reaction.
When a reactor is hit, they will explode and hit any other reactors in the reaction radius.
To implement this simulation, we can use the formula to know if two circles intersect or not.

Given 2 A and B circles with parameters (x1,y1,r1) and (x2,y2,r2):

- Distance d between circles centers d = sqrt((x1 – x2)(x1 – x2) + (y1 – y2)(y1 – y2))
- If d <= r1 – r2: Circle B is inside A.
- If d <= r2 – r1: Circle A is inside B.
- If d < r1 + r2: Circle intersects each other.
- If d== r1 + r2: Circle A and B are in touch with each other.
- Otherwise, Circles A and B do not overlap.

We will consider that a particle is hit also when they are touched.

Input

A line with detonator impact coordinates and its radius.
A line with the number of particles (>=0).
A line per particle, with coordinates, radius and reaction radius

Output

A list with all particles, in the same order of input, saying if they were hit or not

Example 1

Input

```
0 0 1
4
0 0 1 3
0 3 1 2
0 6 1 1
0 9 1 1
```

Output

```
(0, 0) HIT
(0, 3) HIT
(0, 6) HIT
(0, 9) NOT HIT
```

Example 2

Input

```
10 10 1
4
10 13 1 3
13 10 2 3
10 7 1 3
7 10 2 3
```

Output

```
(10, 13) NOT HIT
(13, 10) HIT
(10, 7) NOT HIT
(7, 10) HIT
```

C++

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <math.h>

using namespace std;


float getDistance(int x1, int y1, int r1, int x2, int y2, int r2) {
    return sqrt((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2));
}

bool detonates(float distance, int r1, int r2) {
    if(distance <= (r1 - r2)) {
        return true;
    }
    if(distance <= (r2-r1)) {
        return true;
    }
    if(distance < (r1 + r2)) {
        return true;
    }
    if(distance == (r1 + r2)) {
        return true;
    }
    return false;
}


int main() {

    int detonator[3]; //[0]: x, [1]: y, [2]: r
    cin >> detonator[0] >> detonator[1] >> detonator[2];

    int nReactors;
    cin >> nReactors;
    vector<vector<int>> reactors;

    int value; //x, y, r, e
    for (int i = 0; i < nReactors; i++) {
        vector<int> reactor;
        for(int j  = 0; j < 4; j++) {
            cin >> value;
            reactor.push_back(value);
```

```
        }
        reactors.push_back(reactor);
    }

    vector<int> hits;
    for(int i = 0; i < nReactors; i++) {
        hits.push_back(0);
    }

    float distance;
    for(int i = 0; i < nReactors; i++) {
        distance = getDistance(detonator[0], detonator[1], detonator[2],
reactors[i][0], reactors[i][1], reactors[i][2]);
        if(detonates(distance, detonator[2], reactors[i][2])) {
            hits[i] = 1;
        }
    }

    bool newDetonation = true;
    while(newDetonation) {
        newDetonation = false;
        for(int i = 0; i < nReactors; i++) {
            if(hits[i] == 1) {
                for(int j = 0; j < nReactors; j++) {
                    if(j != i && hits[j] == 0) {
                        distance = getDistance(reactors[i][0], reactors[i][1],
reactors[i][2], reactors[j][0], reactors[j][1], reactors[j][2]);
                        if(detonates(distance, reactors[i][3], reactors[j][2]))
{
                            hits[j] = 1;
                            newDetonation = true;
                        }
                    }
                }
            }
        }
    }


    for(int i = 0; i < nReactors; i++) {
        cout << "(" << reactors[i][0] << ", " << reactors[i][1] << ") ";
        if(hits[i] == 0) {
            cout << "NOT HIT";
        } else {
            cout << "HIT";
        }
```

```
        if(i < nReactors-1) {
            cout << endl;
        }
    }

    return 0;
}
```

27 3D Box Drawing

13 points

Introduction

Drawing with ASCII characters is always fun. But let's go a step further and draw in 3D . To keep things simple, we ask you just draw a box using its 3 dimensions as input; that is 3 positive integer numbers corresponding to the box width, height and depth provided in this order. The characters to use to depict the box are "_", "/ " and "\ ". Have fun!

HINTS: Notice that horizontal line at the top has an extra character "_" versus the two other horizontal lines. Also remember not to put any extra white space after the last right drawing character of each row.

Input

The first line contains a single positive integer defining the width of the box.
The second line contains a single positive integer defining height of the box.
The third line contains a single positive integer defining the depth of the box.

Output

The representation in 3D of the corresponding box with the given dimensions.

Example 1

Input

```
1
1
1
```

Output

```
 _ _
/ _ /\
\ _ \ /
```

Example 2

Input

```
8
4
2
```

Output

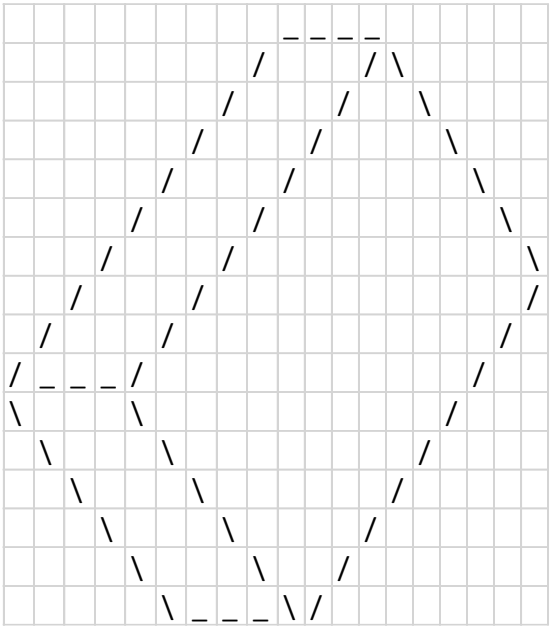
```
   _ _ _ _ _ _ _ _
  /           /\
 /_ _ _ _ _ _ _ /  \
 \              \  /
  \              \/
   \              \
    \              \
     \ _ _ _ _ _ _ _\/
```

Example 3

Input

```
3
6
9
```

Output



Python

```python
width = int(input())
height = int(input())
depth =  int(input())

# Basic case
if (width == 1 and height == 1 and depth == 1):
    print(" __")
    print("/_/\\")
    print("\_\/")
else:
   # build plane using width and depth
   row = ""
   # Drawing top line for top plane
   print("".join(depth*" ") + "".join((width+1)*"_"))

   currHeight = 0
   offset = 0
```

```
    row = "/" + "".join(width*" ")+ "/"

    for i in range(depth):
        output = ""

        if i == depth-1:
            # Drawing bottom line for top plane
            output = "/"+"".join(width*"_")+"/"
        else:
            # Drawing intermediate lines for top plane
            output = "".join((depth-i-1)*" ")+row

        if currHeight < height:
            # Adding height perspective
            offset = 2*currHeight
            output = output + "".join(offset*" ") + "\\"
        else:
            # Adding depth perspective once height is achieved
            output = output + "".join((offset+1)*" ") + "/"
        currHeight = currHeight + 1

        print (output)

    # ending the box considering the height

    row = "\\" + "".join(width*" ")+ "\\"
    for i in range(height-1):
        if currHeight <= height-1:
            # If height was not achieved continue building it
            output = "".join((i)*" ")+row +"".join((offset+1)*" ")+"\\"
            currHeight = currHeight + 1
        else:
            # Otherwise continue closing depth perspective
            output = "".join((i)*" ")+row +"".join(offset*" ")+"/"
            # discount one space for the left shifting and another for the right
perspective
            offset = offset - 2

        print (output)

    # Drawing bottom line for bottom plane
    print("".join((height-1)*" ") + "\\" + "".join(width*"_")+"\\"+"/")
```

28 Flags

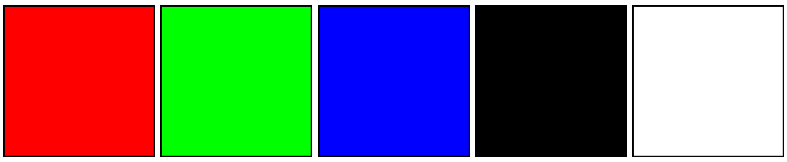
14 points

Introduction

Lots of colors can be represented with RGB color model. RGB means: "Red" "Green" "Blue", because these are the primary colors that, applying some combinations of light for every value, can become to another color.

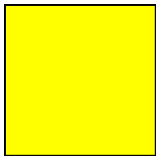
For each of these three colors, the light will be represented as an integer between 0 and 255 (both included).

Let's see some examples:



Color= [R], [G], [B]
RED = [255], [0], [0]
GREEN = [0], [255], [0]
BLUE = [0], [0], [255]
BLACK = [0], [0], [0]
WHITE = [255], [255], [255]

So, to make, for example, a yellow color, we know that it is a composition of GREEN and RED:



YELLOW: [255], [255], [0]

With this color model you can represent and recognize colors.

This is a table with all the Spanish Communities' flag colors, represented with RGB model. **Every flag has at least two colors.** This table has the colors **sorted by abundancy**. So, the first color is the most abundant color in the flag. In the table, there is a maximum of three color representations.

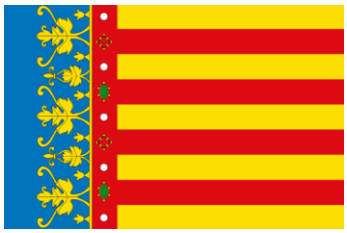
Table of Spanish Communities Flags

- Andalucia = [[0, 102, 51], [255, 255, 255], [255, 228, 77]]	- Comunidad Valenciana = [[0, 114, 188], [218, 18, 26], [252, 221, 9]]
- Aragon = [[252, 221, 9], [218, 18, 26], [15, 71, 175]]	- Extremadura = [[100, 0, 67], [255, 255, 255], [0, 0, 0]]
- Canarias = [[255, 255, 255], [7, 104, 169], [255, 204, 0]]	- Galicia = [[0, 153, 204], [255, 255, 255], [0, 91, 191]]
- Cantabria = [[255, 255, 255], [237, 28, 36], [0, 113, 188]]	- Islas Baleares = [[252, 221, 9], [218, 18, 26], [255, 255, 255]]
- Castilla-La Mancha = [[162, 28, 28], [255, 204, 0], [0, 0, 0]]	- La Rioja = [[181, 41, 33], [255, 255, 255], [0, 0, 0]]
- Castilla y Leon = [[116, 44, 100], [255, 255, 255], [252, 221, 9]]	- Pais Vasco = [[213, 43, 30], [255, 255, 255], [0, 155, 72]]
- Catalunya = [[252, 221, 9], [218, 18, 26]]	- Principado de Asturias = [[0, 102, 255], [247, 212, 23]]
- Comunidad de Madrid = [[198, 11, 30], [255, 255, 255]]	- Region de Murcia = [[156, 31, 45], [252, 183, 20]]
- Comunidad Foral de Navarra = [[237, 45, 29], [227, 228, 229], [234, 193, 2]]	

HINTS: Notice that the list is sorted alphabetically. Also, there are no accents or special characters like "ñ". There is a .txt file where you can find this list in the "Guides and tools" section.

Examples:

Comunidad de Madrid = [[198, 11, 30], [255, 255, 255]]



Comunidad Valenciana = [[0, 114, 188], [218, 18, 26], [252, 221, 9]]

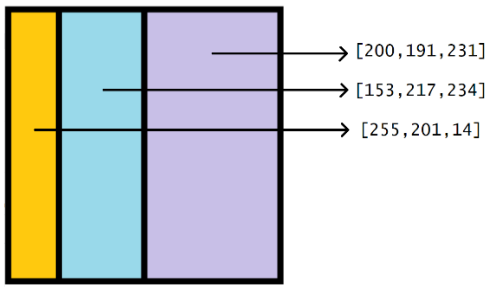
Goal

We want you to write a program that, given a color or colors (maximum of three colors) represented with RGB model, return the flag that best matches. If there is a tie, return all the flags that tied.

The rules are simple:
To compare two colors, we will sum the absolute difference for each RGB value. For example, comparing:

[0, 0, 10] and [5, 0, 10] → The difference is (|0-5|+|0-0|+|10-10|) = 5

[255, 55, 0] and [0, 254, 10] → The difference is (|255-0|+|55-254|-|0-10|) = 466



If only one color is given, we compare it **with the most abundant color** of every flag (The first color on the Community value).	If a second color is given, we compare its match **only to the second color** of every flag.	If a third color is given, we compare its match **only to the third color** of every flag.
Random input example [123, 211, 46] → [200, 191, 231] |123 - 200| + |211 - 191| + |46 - 231| = 282 **282**	Random input example [123, 211, 46] → [200, 191, 231] [35, 106, 200] → [153, 217, 234] |123 - 200| + |211 - 191| + |46 - 231| = 282 |35 - 153| + |106 - 217| + |200 - 234| = 263 **282 + 263 = 545**	Random input example [123, 211, 46] → [200, 191, 231] [35, 106, 200] → [153, 217, 234] [200, 52, 117] → [255, 201, 14] |123 - 200| + |211 - 191| + |46 - 231| = 282 |35 - 153| + |106 - 217| + |200 - 234| = 263 |200 - 255| + |52 - 201| + |117 - 14| = 307 **282 + 263 + 307= 852**

HINTS: Notice that if the input contains three colors, the communities' flags with less than three colors are not going to be considered as possible matches.

Input

One, two or three lines representing a color expressed using RGB color model.
A single character '#' marks the end of the input lines.

Output

Return the flag that best matches, or all the flags that tied with the first position.
Then, return the flag or flags with the second position.
For every flag returned, print its difference.
The output message must follow this format:

```
1st community flag: COMMUNITY with difference: DIFFERENCE
2nd community flag: COMMUNITY with difference: DIFFERENCE
```

If there's more than one flag for a position (for example, there is a tie-on 1st position), **sort it alphabetically** and print the message with "flags" instead of "flag". For example:

```
1st community flags: COMMUNITY with difference: DIFFERENCE
1st community flags: COMMUNITY with difference: DIFFERENCE
2nd community flag: COMMUNITY with difference: DIFFERENCE
```

Example 1

Input

```
0 0 0
#
```

Output

```
1st community flag: Andalucia with difference: 153
2nd community flag: Extremadura with difference: 167
```

Example 2

Input

```
255 230 10
220 20 30
#
```

Output

```
1st community flags: Aragon with difference: 21
1st community flags: Catalunya with difference: 21
1st community flags: Islas Baleares with difference: 21
2nd community flag: Cantabria with difference: 301
```

Example 3

Input

```
252 221 9
218 18 26
200 200 200
#
```

Output

```
1st community flag: Islas Baleares with difference: 165
2nd community flag: Aragon with difference: 339
```

Python

```
flags = {} #flags[] = [] Spanish Community, RGB colors

flags['Andalucia'] = [[0, 102, 51], [255, 255, 255], [255, 228, 77]]
flags['Aragon'] = [[252, 221, 9], [218, 18, 26], [15, 71, 175]]
flags['Canarias'] = [[255, 255, 255], [7, 104, 169], [255, 204, 0]]
flags['Cantabria'] = [[255, 255, 255], [237, 28, 36], [0, 113, 188]]
flags['Castilla-La Mancha'] = [[162, 28, 28], [255, 204, 0], [0, 0, 0]]
flags['Castilla y Leon'] = [[116, 44, 100], [255, 255, 255], [252, 221, 9]]
flags['Catalunya'] = [[252, 221, 9], [218, 18, 26]]
flags['Comunidad de Madrid'] = [[198, 11, 30], [255, 255, 255]]
flags['Comunidad Foral de Navarra'] = [[237, 45, 29], [227, 228, 229], [234,
193, 2]]
flags['Comunidad Valenciana'] = [[0, 114, 188], [218, 18, 26], [252, 221, 9]]
flags['Extremadura'] = [[100, 0, 67], [255, 255, 255], [0, 0, 0]]
flags['Galicia'] = [[0, 153, 204], [255, 255, 255], [0, 91, 191]]
flags['Islas Baleares'] = [[252, 221, 9], [218, 18, 26], [255, 255, 255]]
flags['La Rioja'] = [[181, 41, 33], [255, 255, 255], [0, 0, 0]]
flags['Pais Vasco'] = [[213, 43, 30], [255, 255, 255], [0, 155, 72]]
flags['Principado de Asturias'] = [[0, 102, 255], [247, 212, 23]]
flags['Region de Murcia'] = [[156, 31, 45], [252, 183, 20]]



color = ""
colors = []
while color != "#":
    color = input()
    if color != "#":
        color = color.split(" ")
        colors.append(color)


communities_diff = {} #Dict to save all the differences
for k in flags.keys():
    communities_diff[k] = 0


for i in range(0, len(colors)):
    for k, v in flags.items():
        if len(v) <= i:
            del communities_diff[k]
        if k in communities_diff:
            for j in range(0, 3):
                if v[i][j] > int(colors[i][j]):
```

```
                    communities_diff[k] = communities_diff[k] + (v[i][j] -
int(colors[i][j]))
                elif v[i][j] < int(colors[i][j]):
                    communities_diff[k] = communities_diff[k] +
(int(colors[i][j]) - v[i][j])

#print(communities_diff)

winners_dict = {}
second_list = {}

winners_dict[min(communities_diff, key=communities_diff.get)] =
communities_diff[min(communities_diff, key=communities_diff.get)]
end = False
second_list_control = False

while(not end):
    if not second_list_control:
        maxValue = winners_dict[min(communities_diff, key=communities_diff.get)]
    else:
        maxValue = newValue
        second_list[min(communities_diff, key=communities_diff.get)] = maxValue
    del communities_diff[min(communities_diff, key=communities_diff.get)]
    newValue = communities_diff[min(communities_diff, key=communities_diff.get)]
    if(newValue != maxValue):
        if second_list_control == True:
            end = True
        second_list_control = True
    else:
        if second_list_control == False:
            winners_dict[min(communities_diff, key=communities_diff.get)] =
communities_diff[min(communities_diff, key=communities_diff.get)]

if len(winners_dict) > 1:
    for k, v in winners_dict.items():
        print("1st community flags:", k, "with difference:", v)
else:
    for k, v in winners_dict.items():
        print("1st community flag:", k, "with difference:", v)

if len(second_list) > 1:
    for k, v in second_list.items():
        print("2nd community flags:", k, "with difference:", v)
else:
    for k, v in second_list.items():
        print("2nd community flag:", k, "with difference:", v)
```

29 Top Pizza

15 points

Introduction

The restaurant Mario & Luigi cooks the best pizza in town, and it is so successful that thousands of pizza orders are being received every day. This is the list of their delicious pizza types:

Rustica	Romana
Prosciutto e funghi	Funghi
Pesto Genovese	Bianca
Carbonara	Sicilian
California	Hawaiian
Pinsa Romana	Caprese
Vegetariana	Quattro formaggi
Diavola	Pepperoni
Quattro stagioni	Calzone
Frutti di mare	Margherita
Prosciutto	Napoletana

HINTS: There is a .txt file where you can find this list in the "Guides and tools" section.

Since certain pizza types are more requested than others, some of the pizza's names occur many times in the list of orders. At the end of the day Mario and Luigi want to know the total number of pizzas correctly received. They also need to get the list of pizzas: sorted first by decreasing request order and then alphabetically by name. If a pizza appears twice or more in the list, write the number of repetitions just next to the pizza name. All pizza orders are digitally processed but unfortunately due to communication errors sometimes an invalid pizza name is received. Your program must deal with such incorrect names and list them by order of appearance at the end of the list without considering if they are repeated.

Input

The input is formed by a list of the pizzas ordered by the end of each day. The list should be ended by a hashtag '#' character.

Output

The output reports the total number of valid pizza request received in a line with the format:

```
Received valid pizza requests: number
```

Followed by a list of the pizzas sorted first by number of requests then by name. In the case that a pizza appears twice or more in the list, write the number of repetitions just next the pizza name.

Then a line containing three dashes —

Followed by the total number of invalid requests:

```
Invalid requests: number
```

Finally, a list of the incorrect pizza names sorted by appearance, without considering if they are repeated.

See the examples for clarification.

Example 1

Input

```
Romana
Rumana
Pepperoni
Margherita
Margherita
Romana
Quattro formaggi
Quottro formagge
#
```

Output

```
Received valid pizza requests:  6
Margherita 2
Romana 2
Pepperoni
Quattro formaggi
---
Invalid requests:  2
Rumana
Quottro formagge
```

Example 2

Input

```
Rockmana
Piperoni
Sizilian
Diabolik
#
```

Output

```
Received valid pizza requests:  0
---
Invalid requests:  4
Rockmana
Piperoni
Sizilian
Diabolik
```

Example 3

Input

```
Diavola
Diavola
Diavola
Diavola
Diavola
Diavola
Calzone
Quattro stagioni
Frutti di mare
Frutti di mare
Calzone
Prosciutto
Romana
Calzone
Calzone
Diavola
Romana
Funghi
Bianca
Calzone
Sicilian
Calzone
Hawaiian
Calzone
Caprese
Quattro formaggi
Quattro formaggi
Pepperoni
Calzone
#
```

Output

```
Received valid pizza requests:  29
Calzone 8
Diavola 7
Frutti di mare 2
Quattro formaggi 2
Romana 2
Bianca
Caprese
Funghi
Hawaiian
Pepperoni
Prosciutto
Quattro stagioni
Sicilian
---
Invalid requests:  0
```

C++

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

using namespace std;

int main() {

    map<string, int> requests = {
        {"Bianca",0},
        {"California",0},
        {"Calzone",0},
        {"Caprese",0},
        {"Carbonara", 0},
        {"Diavola",0},
        {"Frutti di mare",0},
        {"Funghi",0},
        {"Hawaiian",0},
        {"Margherita",0},
        {"Napoletana",0},
        {"Pepperoni",0},
        {"Pesto Genovese", 0},
        {"Pinsa Romana",0},
        {"Prosciutto",0},
        {"Prosciutto e funghi", 0},
        {"Quattro formaggi",0},
        {"Quattro stagioni",0},
        {"Romana",0},
        {"Rustica", 0},
        {"Sicilian",0},
        {"Vegetariana",0}
    };

    std::map<string,int>::iterator it;
    int validCounter = 0;
    int invalidCounter = 0;

    vector<string> badPizzas;
    string pizza;

    getline (cin,pizza);
    int actualValue;
    while(pizza != "#"){
        it = requests.find(pizza);
```

```
        if(it != requests.end()) {
            validCounter++;
            actualValue = it->second;
            actualValue++;
            it->second = actualValue;
        }
        else{
            invalidCounter++;
            badPizzas.push_back(pizza);
        }

        getline (cin,pizza);
    }

    cout << "Received valid pizza requests:  " << validCounter << endl;

    int maxValue;
    bool finish = false;
    string topPizza;
    while(finish == false){
        maxValue = 0;
        for(it = requests.begin(); it != requests.end(); it++){
            if(it->second > maxValue){
                maxValue = it->second;
                topPizza = it->first;
            }
        }
        it = requests.find(topPizza);
        it->second = 0;
        if(maxValue == 0) finish = true;
        else{
            cout << topPizza;
            if(maxValue > 1) {
                cout << " " << maxValue << endl;
            }
            else cout << endl;
        }
    }
    cout << "---" << endl;
    cout << "Invalid requests:  " << invalidCounter;

    int size = badPizzas.size();
    if(size > 0) {
        cout << endl;
    }
    for(int i = 0; i < size; i++) {
        cout << badPizzas[i];
```

```
        if(i != size-1){
            cout << endl;
        }
    }
    return 0;
}
```

30 MotoHP

16 points

Introduction

The organizers of the MotoHP championship have suffered a short-circuit in their data centers and have lost the program to elaborate the classifications after the races. They did not have a backup of the algorithm, so they are counting on you to create a script that prints the results of the championship, taking the riders info and the race results as inputs.

The MotoHP season consists of three competitions: one for riders, another one for the teams, and one more for the brands of the motorbikes. This means that there are three different charts. As they would like to use your script for future seasons, the number of riders, teams and brands can't be known in advance.

The MotoHP races are very exciting, with many overtakes and lots of adrenaline on the track. Only the first 7 riders get points, according to the following table:

1st	10p
2nd	8p
3rd	6p
4th	4p
5th	3p
6th	2p
7th	1p
8th to last	0p

The points are assigned to the rider, but also for their team and the brand of their motorbike. For example, the rider in 2nd position gets 8 points, as well as their team (which might also get points for other riders of that team) and the brand (which might also get points from other riders/teams using that brand).

In order to promote speed on the track, the organization also awards the fastest lap rider with 1 extra point. This point also goes for their team and brand.

In the most exciting seasons, there have been ties for first place in any of the three championships. In these cases, the rider/team/brand with more wins along the season takes the first place, without any modification in the points count.

Input

The input will consist of two parts.

1) The first part shows lines describing each rider, with their name (a three-letter abbreviation, usually from their last name), the name of their team and the brand of motorbike.
2) A line containing"#" as separator.

3) The second part describes the results of the races in the championship. Each line is one race, and it shows a list of the riders sorted by position (starting with the winner), separated by "_" characters, and at the end a "|" separator and the name of the fastest lap rider
4) A final line containing "#"

Assumptions:

- There will always be at least 3 riders, 3 teams and 3 brands.
- A rider will only belong to one team during the entire season.
- A team will use one single motorbike brand for all its riders during the entire season.
- The number of race finishers is variable. This means that there can be races where there are many riders that get 0 points, as well as races where there are so few finishers that some point-awarded positions are empty.
- Only ties for 1st position must be solved. There can't be ties for the 2nd or 3rd position.
- The fastest lap rider can only redeem their extra point if they have finished the race (regardless of position).

Output

As mentioned before, the output must consist of three classifications, each showing the top 3 riders/teams/brands with most points, along with the number of points and wins, using the following structure:

Riders Classification:

1 - (rider name) - (rider points) pts - (rider wins) wins

2 - (rider name) - (rider points) pts - (rider wins) wins

3 - (rider name) - (rider points) pts - (rider wins) wins

Teams Classification:

1 - (team name) - (team points) pts - (team wins) wins

2 - (team name) - (team points) pts - (team wins) wins

3 - (team name) - (team points) pts - (team wins) wins

Brands Classification:

1 - (brand name) - (brand points) pts - (brand wins) wins

2 - (brand name) - (brand points) pts - (brand wins) wins

3 - (brand name) - (brand points) pts - (brand wins) wins

Example 1

Input

```
AAA SuperTeam Hondamm
BBB SuperTeam Hondamm
CCC MegaTeam Hondamm
DDD MegaTeam Hondamm
EEE UltraTeam YeahMaha
FFF UltraTeam YeahMaha
GGG NotSoGoodTeam Tuzuki
HHH NotSoGoodTeam Tuzuki
#
DDD_CCC_BBB_AAA_EEE_FFF_GGG_HHH|BBB
FFF_EEE_CCC_BBB_AAA_DDD_GGG_HHH|EEE
DDD_CCC_BBB_AAA_EEE_FFF_GGG_HHH|HHH
#
```

HINT: Note that there will be a tie in points for the first place of the Riders championship between riders DDD and CCC. However, DDD won 2 races and CCC 0.

Output

```
Riders Classification:
1 - DDD - 22 pts - 2 wins
2 - CCC - 22 pts - 0 wins
3 - BBB - 17 pts - 0 wins
Teams Classification:
1 - MegaTeam - 44 pts - 2 wins
2 - UltraTeam - 29 pts - 1 wins
3 - SuperTeam - 28 pts - 0 wins
Brands Classification:
1 - Hondamm - 72 pts - 2 wins
2 - YeahMaha - 29 pts - 1 wins
3 - Tuzuki - 4 pts - 0 wins
```

Python

```python
## MOTO GP WITH UNTIE

# These dictionaries can be sorted by value
from collections import defaultdict

# Data initialization

# Dictionaries to link rider names to the teams and brands
namesToTeams = {}
namesToBrands = {}

# Dictionaries to store the points and wins of the riders, the teams and the
brands
namesAndPoints = defaultdict(int)
namesAndWins = {}
teamsAndPoints = defaultdict(int)
teamsAndWins = {}
brandsAndPoints = defaultdict(int)
brandsAndWins = {}

# Each position in the array is a position in the race
points = [10,8,6,4,3,2,1]

# Helper function to encapsulate points assignment to a rider + team + brand
def assignPoints(rider, points):

    namesAndPoints[rider] += points
    teamsAndPoints[namesToTeams[rider]] += points
    brandsAndPoints[namesToBrands[rider]] += points

# Helper function to encapsulate wins assignment to a rider + team + brand
def assignWins(rider):

    namesAndWins[rider] += 1
    teamsAndWins[namesToTeams[rider]] += 1
    brandsAndWins[namesToBrands[rider]] += 1

# MAIN PROGRAM
# Read riders info

line = input()

while(line != "#"):

    # Extract name, team and brand from every line
```

```
    riderInfo = line.split()
    name = riderInfo[0]
    team = riderInfo[1]
    brand = riderInfo[2]

    # Link names to teams and brands
    namesToTeams[name] = team
    namesToBrands[name] = brand

    # Initialize charts
    namesAndPoints[name] = 0
    namesAndWins[name] = 0
    teamsAndPoints[team] = 0
    teamsAndWins[team] = 0
    brandsAndPoints[brand] = 0
    brandsAndWins[brand] = 0

    line = input()

# End of riders info
# Now read races results

line = input()

while(line != "#"):

    # Part before "|" is a list of rider names separated by "_"
    raceResults = line.split("|")[0].split("_")

    # Part after "|" is string containing the fastest lap rider
    fastLapRider = line.split("|")[1]

    # Read only the riders that get points
    for i in range(min(len(points),len(raceResults))):

        rider = raceResults[i]

        # Assign points to the rider, the team and the brand
        assignPoints(rider, points[i])

        # Assign victory counts
        if (i == 0):
            assignWins(rider)

    # Assign extra point to the fastLapRider, only if he has finished the race
    if fastLapRider in raceResults:
        assignPoints(fastLapRider,1)
```

```
    line = input()


# Now sort the charts according to the points

ridersClassification = sorted(namesAndPoints, key=namesAndPoints.get,
reverse=True)
#print(ridersClassification)
teamsClassification = sorted(teamsAndPoints, key=teamsAndPoints.get,
reverse=True)
#print(teamsClassification)
brandsClassification = sorted(brandsAndPoints, key=brandsAndPoints.get,
reverse=True)
#print(brandsClassification)

# untie riders championship

first = ridersClassification[0]
second = ridersClassification[1]
thereIsTie = namesAndPoints[first] == namesAndPoints[second]
shouldSwap = namesAndWins[first] < namesAndWins[second]

if (thereIsTie and shouldSwap):
    ridersClassification.pop(0)            # Remove it from the list
    ridersClassification.insert(1,first)   # Add it back in 2nd place

# untie teams championship

first = teamsClassification[0]
second = teamsClassification[1]
thereIsTie = teamsAndPoints[first] == teamsAndPoints[second]
shouldSwap = teamsAndWins[first] < teamsAndWins[second]

if (thereIsTie and shouldSwap):
    teamsClassification.pop(0)            # Remove it from the list
    teamsClassification.insert(1,first)   # Add it back in 2nd place

# untie brands championship

first = brandsClassification[0]
second = brandsClassification[1]
thereIsTie = brandsAndPoints[first] == brandsAndPoints[second]
shouldSwap = brandsAndWins[first] < brandsAndWins[second]

if (thereIsTie and shouldSwap):
    brandsClassification.pop(0)            # Remove it from the list
```

```
    brandsClassification.insert(1,first)   # Add it back in 2nd place


# Print the results (only the first three positions)

print("Riders Classification:")

for i in range(3):
    name = ridersClassification[i]
    print(i+1, "-", name, "-", namesAndPoints[name], "pts -",
namesAndWins[name], "wins")

print("Teams Classification:")

for i in range(3):
    name = teamsClassification[i]
    print(i+1, "-", name, "-", teamsAndPoints[name], "pts -",
teamsAndWins[name], "wins")

print("Brands Classification:")

for i in range(3):
    name = brandsClassification[i]
    print(i+1, "-", name, "-", brandsAndPoints[name], "pts -",
brandsAndWins[name], "wins")
```

31 Time Is Gold

25 points

Introduction

The subway system of Barcelona is one of the best ways to move around the city; it's fast and respectful with environment!

It has a system based on "Lines", where each line is represented with a number and a color. Each line is a set of Stations that one train will cross, in order.

For example, this is the first line, that is represented as L1 with the color red:

And this is the second line, represented as L2 with the color purple:

As you can see, there are some stations that have a connection with other lines, shown at the bottom of the image. This means that these stations are also on other lines. So, if you want to go from a station in L1, to a station in L2, you will have to change your train at some point. This is called a TRANSFER.

We want you to make a program that, given a starting station and a goal station, returns the best path to take.

To simplify the problem, we will consider only the first five lines of Barcelona's subway system.

Each station has an associated time requirement (and only one, no matter which line it's located on) that represents how much time in seconds is required to reach it from any of its neighboring stations.

Each transfer requires 300 extra seconds.

The tables below contain the stations and times grouped by lines. There is a .txt file where you can find this list in the

HINTS: There is a .txt file where you can find this list in the "Guides and tools" section.

Stations L1	Time
Hospital de Bellvitge	100
Bellvitge	108
Av. Carrilet	204
Rbla. Just Oliveras	173
Can Serra	182
Florida	130
Torrassa	146
Santa Eulalia	123
Mercat Nou	197
Placa de Sants	133
Hostafrancs	164
Espanya	149
Rocafort	172
Urgell	109
Universitat	141
Catalunya	190
Urquinaona	166
Arc de Triomf	217
Marina	207
Glories	280
Clot	155
Navas	216
La Sagrera	186
Fabra i Puig	210
Sant Andreu	153
Torras i Bages	138
Trinitat Vella	146
Baro de Viver	179
Santa Coloma	104
Fondo	144

Stations L2	Time
Parallel	208
Sant Antoni	163
Universitat	141
Passeig de Gracia	127
Tetuan	217
Monumental	152
Sagrada Familia	114
Encants	148
Clot	155
Bac de Roda	108
Sant Marti	217
La Pau	207
Verneda	118
Artigues Sant Adria	121
Sant Roc	209
Gorg	109
Pep Ventura	212
Badalona Pompeu Fabra	196

Stations L3	Time
Zona Universitaria	165
Palau Reial	216
Maria Cristina	162
Les Corts	164
Placa del Centre	120
Sants Estacio	204
Tarragona	202
Espanya	149
Poble Sec	212
Parallel	208
Drassanes	176
Liceu	176
Catalunya	190
Passeig de Gracia	127
Diagonal	184
Fontana	161
Lesseps	197
Vallcarca	203
Penitents	106
Vall d'Hebron	150
Montbau	195
Mundet	201
Valldaura	136
Canyelles	109
Roquetes	206
Trinitat Nova	162

Stations L4	Time
La Pau	207
Besos	209
Besos de Mar	126
El Maresme Forum	208
Selva de Mar	187
Poblenou	134
Llacuna	178
Bogatell	176
Ciutadella Vila Olimpica	154
Barceloneta	103
Jaume I	126
Urquinaona	166
Passeig de Gracia	127
Girona	213
Verdaguer	182
Joanic	135
Alfons X	220
Guinardo Hospital de Sant Pau	171
Maragall	206
Llucmajor	213
Via Julia	135
Trinitat Nova	162

Stations L5	Time
Cornella Centre	201
Gavarra	201
Sant Ildefons	108
Can Boixeres	216
Can Vidalet	190
Pubilla Cases	186
Ernest Lluch	158
Collblanc	149
Badal	165
Placa de Sants	133
Sants Estacio	204
Entenca	112
Hospital Clinic	101
Diagonal	184
Verdaguer	182
Sagrada Familia	114
Sant Pau Dos de Maig	215
Camp de l'Arpa	120
La Sagrera	186
Congres	120
Maragall	206
Virrei Amat	149
Vilapicina	194
Horta	175
El Carmel	218
El Coll La Teixonera	162
Vall d'Hebron	150

Input

The input consists of 2 lines:

- The first line defines the starting station.
- The second line is the goal station

Output

The output consists of:

- The final time of the path, expressed as:
  Total time: *_time_* seconds
- The final path:
  Best path from *_source_* to *_destination_*: *_source_*, *_station1_*, *_station2_*, ..., *_destination_*

Important! It is not possible to tie, all the inputs have a unique solution.

Example 1

Input

```
Bellvitge
Hospital de Bellvitge
```

Output

```
Total time: 100 seconds
Best path from Bellvitge to Hospital de Bellvitge:
Bellvitge, Hospital de Bellvitge
```

Example 2

Input

```
Can Serra
Tetuan
```

Output

```
Total time: 2108 seconds
Best path from Can Serra to Tetuan:
Can Serra, Florida, Torrassa, Santa Eulalia, Mercat Nou, Placa de Sants,
Hostafrancs, Espanya, Rocafort, Urgell, Universitat, Passeig de Gracia,
Tetuan
```

Example 3

Input

```
Urgell
Clot
```

Output

```
Total time: 1354 seconds
Best path from Urgell to Clot:
Urgell, Universitat, Passeig de Gracia, Tetuan, Monumental, Sagrada Familia,
Encants, Clot
```

Python

```python
from copy import copy

class Station:
    def __init__(self, name, line, time):
        self.name = name
        self.line = line
        self.time = time
        self.connections = []
        self.timeAcc = 0

        self.prevStation = None
        self.visited = False


    def __eq__(self, other):
        """Overrides the default implementation"""
        if isinstance(other, Station):
            return self.name == other.name
        return False

    def check(self, other):
        """Overrides the default implementation"""
        if isinstance(other, Station):
            return ((self.name == other.name) and (self.line == other.line))
        return False

    def addConnection(self, station):
        stationConection = copy(station)

        self.connections.append(stationConection)

    def visit(self, prevStation):
        self.prevStation = prevStation
        self.visited = True

    def mark(self):
        self.visited = True

    def asString(self):
        connectionsStr = "[ "
        for connection in self.connections:
            connectionsStr += "{} T-{} ".format(connection.name,
connection.time)
        connectionsStr += "]"
```

```
        return "{} L{} Time {} - Connections {}".format(self.name, self.line,
self.time, connectionsStr)

    def timeListFrom(self, originStation):
        time = [self.time]
        if (originStation == self.prevStation):
            return time + [originStation.time]
        else:
            time += self.prevStation.timeListFrom(originStation)
        return time

    def stationListFrom(self, originStation):
        stations = [self.name]
        if (originStation == self.prevStation):
            return stations + [originStation.name]
        else:
            stations += self.prevStation.stationListFrom(originStation)
        return stations

    def print(self):
        print(self.asString())

class QueueStations:
    def __init__(self):
        self.queue = []

    def size(self):
        return len(self.queue)

    def empty(self):
        return (len(self.queue) == 0)

    def append(self, station):
        self.queue.append(station)

    def pop(self):
        index = -1
        time = 9999
        for i, station in enumerate(self.queue):
            if (station.time < time):
                index = i
        return self.queue.pop(index)

    def popSimple(self):
        return self.queue.pop(0)

    def print(self):
```

```
        for s in self.queue:
            print("Q - {}".format(s.asString()))

def isTransfer2(s1, s2, s3, lines):
    return getLine(s1, s2, lines) != getLine(s2, s3, lines)

def getLine(s1, s2, lines):
    for i in range(len(lines)):
        if s1 in lines[i] and s2 in lines[i]:
            return i
    return -1

def getStation(name, stations):
    for s in stations:
        if s.name == name:
            return s
    return None

def validate_solution(path, lines, stations):
    # Validate solution
    time = 0
    for i in range(len(path)):
        station = path[i]
        trans = False

        stationObj = getStation(station, stations)

        if i > 1 and isTransfer2(path[i-2], path[i-1], path[i], lines):
            trans = True
            time += 300
            print("Time {:5d}, +300, --- TRANSFER ---".format(time))
        if i > 0:
            time += stationObj.time
            line = getLine(path[i], path[i-1], lines) + 1
        else:
            line = getLine(path[i], path[i+1], lines) + 1
        print("Time {:5d}, +{:3d}, L {} {}".format(time, stationObj.time, line,
stationObj.name))

L1Names = ["Hospital de Bellvitge", "Bellvitge", "Av. Carrilet", "Rbla. Just
Oliveras", "Can Serra", "Florida","Torrassa", "Santa Eulalia", "Mercat Nou",
"Placa de Sants", "Hostafrancs", "Espanya", "Rocafort","Urgell", "Universitat",
"Catalunya", "Urquinaona", "Arc de Triomf", "Marina", "Glories", "Clot",
"Navas","La Sagrera", "Fabra i Puig", "Sant Andreu", "Torras i Bages", "Trinitat
Vella", "Baro de Viver", "Santa Coloma", "Fondo"]
L1Times = [100, 108, 204, 173, 182, 130, 146, 123, 197, 133, 164, 149, 172, 109,
141, 190, 166, 217, 207, 280, 155,216, 186, 210, 153, 138, 146, 179, 104, 144]
```

```
L2Names = ["Parallel", "Sant Antoni", "Universitat", "Passeig de Gracia",
"Tetuan", "Monumental", "Sagrada Familia", "Encants", "Clot", "Bac de Roda",
"Sant Marti", "La Pau", "Verneda", "Artigues Sant Adria","Sant Roc", "Gorg",
"Pep Ventura", "Badalona Pompeu Fabra"]
L2Times = [208, 163, 141, 127, 217, 152, 114, 148, 155, 108, 217, 207, 118, 121,
209, 109, 212, 196]
L3Names = ["Zona Universitaria", "Palau Reial", "Maria Cristina", "Les Corts",
"Placa del Centre", "Sants Estacio", "Tarragona", "Espanya", "Poble Sec",
"Parallel", "Drassanes", "Liceu", "Catalunya", "Passeig de Gracia", "Diagonal",
"Fontana", "Lesseps", "Vallcarca", "Penitents", "Vall d'Hebron",
"Montbau","Mundet", "Valldaura", "Canyelles", "Roquetes", "Trinitat Nova"]
L3Times = [165, 216, 162, 164, 120, 204, 202, 149, 212, 208, 176, 176, 190, 127,
184, 161, 197, 203, 106, 150,195, 201, 136, 109, 206, 162]
L4Names = ["La Pau", "Besos", "Besos de Mar", "El Maresme Forum", "Selva de
Mar", "Poblenou", "Llacuna","Bogatell", "Ciutadella Vila Olimpica",
"Barceloneta", "Jaume I", "Urquinaona", "Passeig de Gracia","Girona",
"Verdaguer", "Joanic", "Alfons X", "Guinardo Hospital de Sant Pau", "Maragall",
"Llucmajor","Via Julia", "Trinitat Nova"]
L4Times = [207, 209, 126, 208, 187, 134, 178, 176, 154, 103, 126, 166, 127, 213,
182, 135, 220, 171, 206, 213,135, 162]
L5Names = ["Cornella Centre", "Gavarra", "Sant Ildefons", "Can Boixeres", "Can
Vidalet", "Pubilla Cases","Ernest Lluch", "Collblanc", "Badal", "Placa de
Sants", "Sants Estacio", "Entenca", "Hospital Clinic","Diagonal", "Verdaguer",
"Sagrada Familia", "Sant Pau Dos de Maig", "Camp de l'Arpa", "La
Sagrera","Congres", "Maragall", "Virrei Amat", "Vilapicina", "Horta", "El
Carmel", "El Coll La Teixonera", "Vall d'Hebron"]
L5Times = [201, 201, 108, 216, 190, 186, 158, 149, 165, 133, 204, 112, 101, 184,
182, 114, 215, 120, 186, 120, 206,149, 194, 175, 218, 162, 150]

LNames = [L1Names, L2Names, L3Names, L4Names, L5Names]
LTimes = [L1Times, L2Times, L3Times, L4Times, L5Times]

stations = []

for i in range(5):
    stationsInLine = []
    lName = LNames[i]
    lTime = LTimes[i]

    # Create all stations in Line
    for index, name in enumerate(lName):
        station = Station(name, i+1, lTime[index])
        stationsInLine.append(station)

    # Make connections in line
    N = len(stationsInLine)
    for index, station in enumerate(stationsInLine):
```

```
        if (index != 0):
            station.addConnection(stationsInLine[index-1])

        if (index != (N-1)):
            station.addConnection(stationsInLine[index+1])

    # Make connections with other lines
    for indexInLine, stationInLine in enumerate(stationsInLine):
        for indexGlobal, stationGlobal in enumerate(stations):
            if (stationInLine == stationGlobal):
                stationInLine.addConnection(stationGlobal)
                stationGlobal.addConnection(stationInLine)

    # Add stations in line to global station list
    for station in stationsInLine:
        stations.append(station)

startName = input()
endName = input()

queueToExplore = QueueStations()
startStation = []
endStation = []
for station in stations:
    if station.name == startName:
        station.mark()
        queueToExplore.append(station)
        startStation.append(station)
    if station.name == endName:
        endStation.append(station)

while (not queueToExplore.empty()):
    stationPop = queueToExplore.popSimple()

    if (stationPop.name != endName):

        for connection in stationPop.connections:
            for station in stations:
                if ((connection.check(station)) and (not station.visited)):
                    station.visit(stationPop)
                    queueToExplore.append(station)

bestTime = 999999
bestSIndex = -1
bestEIndex = -1

for i, sStation in enumerate(startStation):
```

```
    for j, eStation in enumerate(endStation):
        stationTravelList = eStation.stationListFrom(sStation)
        timeTravelList = eStation.timeListFrom(sStation)

        travelListNoRepeats = []
        timeListWithTransferTime = []
        for index, stationName in enumerate(stationTravelList):
            if (stationName in travelListNoRepeats):
                timeListWithTransferTime.append(300)
            else:
                travelListNoRepeats.append(stationName)
                timeListWithTransferTime.append(timeTravelList[index])

        timeListWithTransferTime.reverse()
        timeListWithTransferTime.pop(0)
        timeTravel = sum(timeListWithTransferTime)

        if (timeTravel < bestTime):
            bestTime = timeTravel
            bestSIndex = i
            bestEIndex = j

stationTravelList =
endStation[bestEIndex].stationListFrom(startStation[bestSIndex])
timeTravelList = endStation[bestEIndex].timeListFrom(startStation[bestSIndex])

travelListNoRepeats = []
timeListWithTransferTime = []
for index, stationName in enumerate(stationTravelList):
    if (stationName in travelListNoRepeats):
        timeListWithTransferTime.append(300)
    else:
        travelListNoRepeats.append(stationName)
        timeListWithTransferTime.append(timeTravelList[index])

timeListWithTransferTime.reverse()
timeListWithTransferTime.pop(0)
timeTravel = sum(timeListWithTransferTime)

travelListNoRepeats.reverse()
path = ", ".join(travelListNoRepeats)

print("Total time: {} seconds".format(timeTravel))
print("Best path from {} to {}:".format(startName, endName))
print("{}".format(path))
```

32 Mutant Mushrooms

30 points

Introduction

Doctor Crazyus Maximus has found a mechanism to manipulate the DNA of mushrooms to make them replicate at a very fast rate, but this genetic manipulation shows a very strange effect: depending on the mutation type, the mushrooms are replicated following specific patterns.

After some experimentation, Crazyus found 4 relevant facts:

1. When mushrooms replicate, they do not fill a space which is already filled with some other mushroom of the same species.

2. The mutant mushrooms replication is quite aggressive, and they destroy other species. It seems that there is one gene of the mutation process that is determining the resilience of the mushroom, thus the mushrooms with higher resilience are the ones that prevail in case of direct contact. You can assume that different mushrooms will never have the same resilience.

3. The growth of the mutant mushrooms is very fast, but also their death. It seems that there is another gene that defines how many days a mushroom will be alive, with no error. Luckily, the "age" of the mushroom is not inherited when the mushroom is replicated, so new mushrooms can have their own life, starting from 0.

4. Once a mushroom passes its maximum age, it stops replicating and dies, leaving the space empty at the end of the day. But the mushroom will still be present during the day, so surrounding mushrooms of the same type won't be able to replicate on it.

Can you help Doctor Crazyus to study the mutant mushrooms with your program to simulate their behaviour?

Input

The input is structured as follows:

- A line with a positive integer, indicating how many species of mutant mushrooms will be simulated

- For each of the mushroom species:

    - A line with a character representing the label for the mushroom.

    - A line with 2 positive integers, the first one being the resilience of the mushroom and the second one the number of days of life for the mushroom. To avoid having a dirty output, the number of days is limited in the range [0, 9].

    - A line with 2 positive integers, being the first one the rows of the mushroom growth pattern and the second one the columns of the pattern.

    - The growth pattern, represented by a grid of 0s and 1s.

- A line with 2 positive integers: the rows and the columns of the simulated experiment.

- The simulated experiment, which is formed by a regular grid according to the provided dimensions. The map indicates the initial places for the mushroom species (it may be more than one initial place for each species). The character '.' indicates an empty space that can be covered by a mushroom, and the character '_' indicates a space outside the simulation (so it must never be covered).

- A line with a positive integer indicating the number of days for the simulation.

NOTE: When applying the mushroom growth pattern, the mushroom is always in the center. So, you can assume that the growth pattern dimensions are always odd numbers.

Output

The program must show the status of the simulation on each day, representing all the mushrooms that are alive and their age (see the example below).

Example

Input

```
3
A
30 4
3 3
010
101
010
C
20 5
5 5
10001
01010
00000
01010
10001
L
10 6
3 5
11000
10000
10011
12 22
A..............._.....L
..............__......
......__......__......
......__..............
..............__....
..............__....
..........__..........
............__........
.....__..............
.....__..............
```

```
..........__.........
C......__...........L
6
```

Output

```
Mushrooms at day 0
A............_.....L   0............_.....0
............__......   ............__......
......__......__......   ......__......__......
......__............   ......__............
..............__....   ..............__....
..............__....   ..............__....
.........__.........   .........__.........
...........__........   ...........__........
.....__.............   .....__.............
.....__.............   .....__.............
..........__.........   ..........__.........
C......__...........L   0......__...........0

Mushrooms at day 1
AA............_...L.L   10............_...0.1
A............__...L..   0............__...0..
......__......__......   ......__......__......
......__............   ......__............
..............__....   ..............__....
..............__....   ..............__....
.........__.........   .........__.........
...........__........   ...........__........
.....__.............   .....__.............
..C..__.............   ..0..__.............
.C........__......LL.   .0........__......00.
```

```
C....._.........L.L   1....._.........0.1
```

Mushrooms at day 2

```
AAA...........__.LLL.L   210...........__.001.2
AA............__.L.LLL   10............__.0.100
A....._......__.L..LL   0....._......__.0..00
......__.............   ......__.............
..............__....   ..............__....
..............__....   ..............__....
.........__.........   .........__.........
C...C.....__........   0...0.....__........
.C.C.__.............   .0.0.__.............
C.C..__.........LLL..   0.1..__.........000..
.C.C....._....LLLL.   .1.0....._....0011.
C.C.C..__........LLLLL   2.0.0..__........00102
```

Mushrooms at day 3

```
AAAA..........._LLLLLL   3210..........._011203
AAA...........__LLLLLL   210...........__010211
AA...._.....__.LLLLL   10...._.....__.10011
A....._.......L..LL.L   0....._.......0..00.0
..............__....   ..............__....
..C...C........__....   ..0...0........__....
.C.C.C...._.........   .0.0.0...._.........
C.C.C....._........   1.0.1....._........
.C.C.__........LLLL...   .1.1.__........0000...
C.C.C__........LLLLL..   1.2.0__........00111..
.C.C.C....._..LLLLLLL   .2.1.0....._..0011220
C.C.C..__......LLLLLLL   3.1.1..__......0011213
```

```
Mushrooms at day 4
.AAAA.........L_LLLLLL   .3210.........0_122314
AAAA..........__LLLLLL   3210..........__121322
AAA..__.....L__LLLLLL    210..__.....0__021122
AA..C.__C....L.LLLLLLL   10..0.__0....0.1001101
AC.C.C.C.....L..__.LLL   00.0.0.0.....0..__.000
C.C.C.C.........__....   0.1.0.1.........__....
.C.C.C.C..__..........   .1.1.1.0..__..........
C.C.C.C.C...__LLLL....   2.1.2.0.0...__0000....
.C.C.__C.....LLLLLL...   .2.2.__0.....001111...
C.C.C__......LLLLLLLL.   2.3.1__......00112220.
.C.C.C.....__LLLLLLLLL   .3.2.1.....__001122331
C.C.C.C__....LLLLLLLLL   4.2.2.0__....001122324

Mushrooms at day 5
..AAAA......L.L_LLLLLL   ..3210......0.1_233425
.AAAA.C...CLL.__LLLLLL   .3210.0...000.__232433
AAAA.C__.C.LLL__LLLLLL   3210.0__.0.001__132233
AAA.C.__C..LLLLLLLLLLL   210.1.__1..00102112212
AA.C.C.C.C.L.LLL__LLLL   10.1.1.1.0.0.100__0111
A.C.C.C.C.CL..LL__LLLL   0.2.1.2.0.00..00__0000
.C.C.C.C.C__LLLLL.....   .2.2.2.1.0__00000.....
C.C.C.C.C..L__LLLL....   3.2.3.1.1..0__1111....
.C.C.__C.C.LLLLLLLLL..   .3.3.__1.0.001122220..
C.C.C__.C.CLLLLLLLLLL.   3.4.2__.0.00011223331.
.C.C.C.C.C.__LLLLLLLLL   .4.3.2.0.0.__112233442
..C.C.C__..LLLLLLLLLLL   ..3.3.1__..00112233435
```

```
Mushrooms at day 6
...AAAAC.CLCL.L_LLLLL.    ...32100.0001.2_34453.
..AAAAC.CLCLLL__LLLLLL    ..32101.001110__343544
.AAAAC__.CLCLL__LLLLLL    .32101__.10012__243344
AAAAC.__CLCLCLLLLLLLL     32102.__20010213223323
AAAC.C.C.CLCLLLL__LLLL    2102.2.2.1000211__1222
AAC.C.C.CLCLLLLL__LLLL    103.2.3.10110011__1111
AC.C.C.C.C__LLLLLLLLL     03.3.3.2.1__1111100000
C.C.C.C.CLCL__LLLLL...    4.3.4.2.2001__22220...
.C.C.__C.CLCLLLLLLL..     .4.4.__2.10012233331..
C...C__.CLCLLLLLLLLLL     4...3__.10111223344420
...C.C.C.CL__LLLLLLLL     ...4.3.1.10__223344553
..C.C.C__LCLCLLLLLLL.     ..4.4.2__001022334454.
```

C++

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <cassert>

struct Mushroom
{
    char label = '.';
    int age = 0;
    int ageOfDeath = 0;
    int power = 0;
    std::vector<std::string> pattern;

    void kill()
    {
        label = '.';
        age = 0;
        ageOfDeath = 0;
        power = 0;
        pattern.resize(0);
    }
};

void printGrid(const std::vector<std::vector<Mushroom>>& grid, int day)
{
    std::cout << "Mushrooms at day " << day << std::endl;
    for (size_t row = 0; row < grid.size(); ++row)
    {
        for (size_t col = 0; col < grid[row].size(); ++col)
        {
            std::cout << grid[row][col].label;
        }
        std::cout << "   ";
        for (size_t col = 0; col < grid[row].size(); ++col)
        {
            if (grid[row][col].label == '.' || grid[row][col].label == '_')
            {
                std::cout << grid[row][col].label;
            }
            else
            {
                std::cout << grid[row][col].age;
            }
        }
```

```
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

std::map< char, Mushroom> parseMushrooms()
{
    int numMushrooms;
    std::cin >> numMushrooms;
    std::map<char, Mushroom> mushrooms;
    for (int i = 0; i < numMushrooms; ++i)
    {
        Mushroom mushroom;
        int rows, cols;
        std::cin >> mushroom.label >> mushroom.power >> mushroom.ageOfDeath >>
rows >> cols;
        mushroom.pattern.resize(rows);
        for (int row = 0; row < rows; ++row)
        {
            std::cin >> mushroom.pattern[row];
            assert(mushroom.pattern[row].size() == cols);
        }
        mushrooms[mushroom.label] = mushroom;
    }
    return mushrooms;
}

std::vector<std::vector<Mushroom>> parseGrid(const std::map<char, Mushroom>&
mushrooms)
{
    // Parse the input grid as strings
    int rows, cols;
    std::cin >> rows >> cols;
    std::vector<std::string> gridChars(rows);
    std::getline(std::cin, gridChars[0]);
    for (size_t row = 0; row < gridChars.size(); ++row)
    {
        std::cin >> gridChars[row];
    }
    // Convert the input characters to mushrooms
    Mushroom noMushroom;
    noMushroom.label = '.';
    std::vector<std::vector<Mushroom>> grid(gridChars.size(),
std::vector<Mushroom>(gridChars[0].size()));
    for (size_t row = 0; row < grid.size(); ++row)
    {
        for (size_t col = 0; col < grid[row].size(); ++col)
```

```
        {
            if (gridChars[row][col] == '.')
            {
                grid[row][col].label = '.';
            }
            else if (gridChars[row][col] == '_')
            {
                grid[row][col].label = '_';
            }
            else
            {
                auto iterator = mushrooms.find(gridChars[row][col]);
                assert(iterator != mushrooms.end());
                grid[row][col] = iterator->second;
            }
        }
    }
    return grid;
}

void applyMushroomGrowth(
    const Mushroom& srcMushroom,
    std::vector<std::vector<Mushroom>>& dstGrid,
    size_t row, size_t col)
{
    int patternHalfRows = srcMushroom.pattern.size() / 2;
    int patternHalfCols = srcMushroom.pattern[0].size() / 2;
    for (size_t rPat = 0; rPat < srcMushroom.pattern.size(); ++rPat)
    {
        for (size_t cPat = 0; cPat < srcMushroom.pattern[rPat].size(); ++cPat)
        {
            if (srcMushroom.pattern[rPat][cPat] == '1')
            {
                int r = int(row) - patternHalfRows + int(rPat);
                int c = int(col) - patternHalfCols + int(cPat);
                if (r >= 0 && c >= 0 && r < dstGrid.size() && c <
dstGrid[0].size())
                {
                    if (dstGrid[r][c].label != '_')
                    {
                        if (dstGrid[r][c].label == '.' ||
                            ((dstGrid[r][c].label != srcMushroom.label) &&
(srcMushroom.power > dstGrid[r][c].power)))
                        {
                            dstGrid[r][c] = srcMushroom;
                            dstGrid[r][c].age = 0;
                        }
```

```
                    }
                }
            }
        }
    }
}

int main()
{
    std::vector<std::vector<Mushroom>> grid = parseGrid(parseMushrooms());
    int days;
    std::cin >> days;
    for (int day = 0; day < days; ++day)
    {
        printGrid(grid, day);
        // Increase the age of the mushrooms
        for (size_t row = 0; row < grid.size(); ++row)
        {
            for (size_t col = 0; col < grid[row].size(); ++col)
            {
                if (grid[row][col].label != '.' && grid[row][col].label != '_')
                {
                    ++grid[row][col].age;
                }
            }
        }
        // Make the mushrooms grow (only those that are still alive)
        std::vector<std::vector<Mushroom>> newGrid = grid;
        for (size_t row = 0; row < grid.size(); ++row)
        {
            for (size_t col = 0; col < grid[row].size(); ++col)
            {
                if (grid[row][col].label != '.' && grid[row][col].label != '_'
                    && (grid[row][col].age < grid[row][col].ageOfDeath))
                {
                    applyMushroomGrowth(grid[row][col], newGrid, row, col);
                }
            }
        }
        // Kill all the mushrooms that reached the age of death
        for (size_t row = 0; row < newGrid.size(); ++row)
        {
            for (size_t col = 0; col < newGrid[row].size(); ++col)
            {
                if (newGrid[row][col].label != '.' && newGrid[row][col].label !=
'_' && newGrid[row][col].age >= newGrid[row][col].ageOfDeath)
                {
```

```
                    newGrid[row][col].kill();
                }
            }
        }
        // Update the grid with the new grid
        grid.swap(newGrid);
    }
    printGrid(grid, days);
    return 0;
}
```

33 Voxels

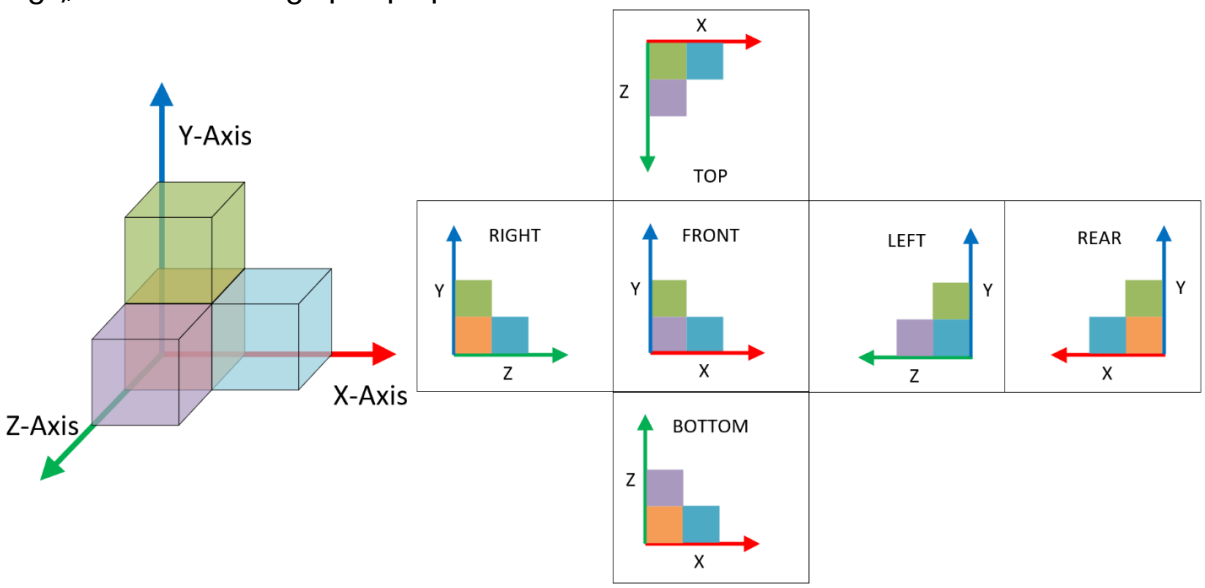
35 points

Introduction

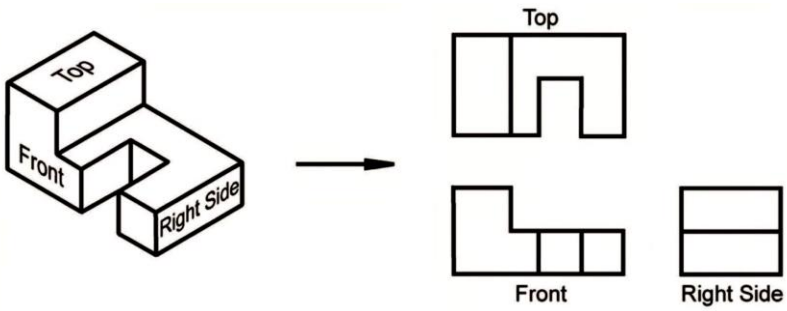
We need a program to draw an orthographic projection of a 3D model (this is the projection of the model in 2 dimensions), which is represented by a set of voxels (a voxel is a cube positioned in the space according to the 3D coordinates (x, y, z)).
The 6 projections are:

View name	Projection axes	View axis
FRONT	xy	+z
REAR	xy	-z
TOP	xz	+y
BOTTOM	xy	-y
LEFT	zy	+x
RIGHT	zy	-x

An example of the 3D model, formed by 4 voxels of different color (the orange one is hidden in the image), and their 6 orthographic projections:



We want to draw only the voxel's edges that are visible and are not touching other edges, as shown in the image below:



Input

The first line indicates the type of projection.

The second line is a positive integer that indicates the number of voxels forming the 3D model.

Finally, the sequence of voxels of the 3D model, each one of them defined by a triplet of "X Y Z" coordinates. Each coordinate is an integer in the range [0, 10].

Output

Voxels are drawn using '+', '-' and '|' symbols.

This is the representation of a single voxel:

```
+-+
| |
+-+
```

The output must be the 2D projection of the input 3D model according to the provided projection type, within a drawing space of 11x11 voxels (see the examples below).

The drawing space is framed by # symbols.

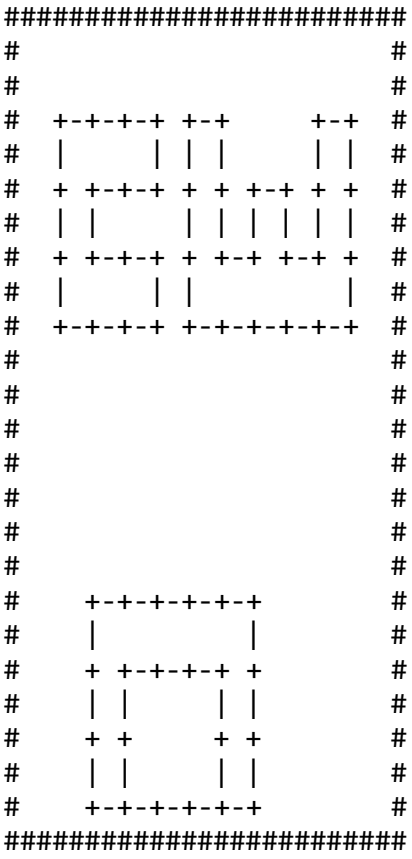
Notice that the origin of coordinates (0, 0) of each 2D projection is a different corner of the drawing space.

Example 1

Input

```
FRONT
38
2 0 3
3 0 3
4 0 3
5 0 3
6 0 3
2 1 3
3 1 3
4 1 3
5 1 3
6 1 3
2 2 3
3 2 3
4 2 3
5 2 3
6 2 3
3 0 4
4 0 4
5 0 4
3 1 4
4 1 4
5 1 4
1 9 0
1 8 0
1 7 0
2 9 0
2 7 0
3 9 0
3 7 0
5 9 0
5 8 0
```

Output



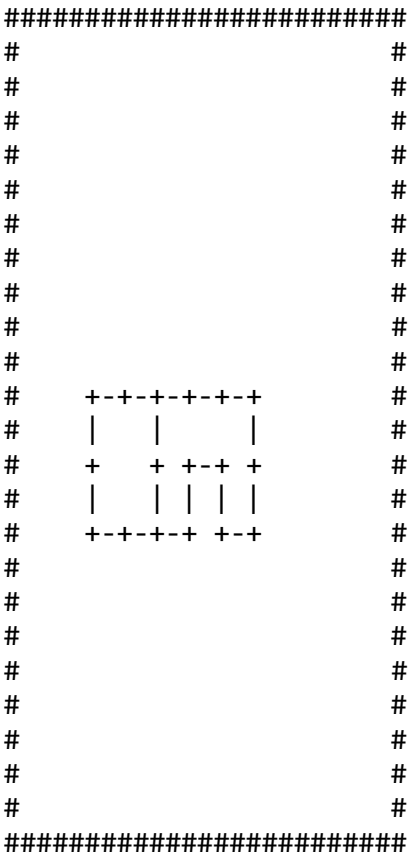
```
5 7 0
6 7 0
7 8 0
7 7 0
8 7 0
9 9 0
9 8 0
9 7 0
```

Example 2

Input

```
TOP
13
2 1 5
3 1 5
4 1 5
5 1 5
6 1 5
2 1 6
3 1 6
4 1 6
6 1 6
2 2 5
3 2 5
2 2 6
3 2 6
```

Output



C++

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

constexpr int MAX_COORD = 10;

struct Voxel
{
    int x, y, z;
    Voxel(int xx, int yy, int zz) : x(xx), y(yy), z(zz) {};
};

void project(const vector<Voxel> &voxels, std::string projection)
{
```

```
    // Projection axis
    //      Y
    //      ^
    //      |
    //      |
    //      |
    //      |
    //      /---------> X
    //     / (0,0,0)
    //    /
    //   Z
    vector<int> X; // Horizontal axis
    vector<int> Y; // Vertical axis
    vector<int> Z; // Depth axis

    // We will store the (x,y,z) points based on the projection
    // to simplify the rasterization process
    for (auto voxel : voxels)
    {
        if ( projection == "FRONT" )
        {
            X.push_back(voxel.x);
            Y.push_back(voxel.y);
            Z.push_back(voxel.z);
        }
        else if ( projection == "REAR" )
        {
            X.push_back(MAX_COORD-voxel.x);
            Y.push_back(voxel.y);
            Z.push_back(MAX_COORD-voxel.z);
        }
        else if ( projection == "TOP" )
        {
            X.push_back(voxel.x);
            Y.push_back(MAX_COORD-voxel.z);
            Z.push_back(voxel.y);
        }
        else if ( projection == "BOTTOM" )
        {
            X.push_back(voxel.x);
            Y.push_back(voxel.z);
            Z.push_back(MAX_COORD-voxel.y);
        }
        else if ( projection == "RIGHT" )
        {
            X.push_back(voxel.z);
            Y.push_back(voxel.y);
```

```
            Z.push_back(voxel.x);
        }
        else if ( projection == "LEFT" )
        {
            X.push_back(MAX_COORD-voxel.z);
            Y.push_back(voxel.y);
            Z.push_back(MAX_COORD-voxel.x);
        }
        else
        {
            cout << "INVALID VIEW " << projection << endl;
            return;
        }
    }

    // Length of each axis (both included)
    int len = MAX_COORD + 1;

    // Matrix with z-depth map: each cell represent the depth of each x,y
coordinate
    vector<vector<int>> zmap(len, vector<int>(len, -1));

    // painter algorithm: raster from far to near
    for (int depth = 0; depth < len; depth++)
    {
        for (int i = 0; i < voxels.size(); ++i)
        {
            if ( Z[i] == depth )
            {
                zmap[ X[i] ][ Y[i] ] = depth;
            }
        }
    }

    auto leftEdge   = [&](int x, int y) { return (x == 0        or zmap[x][y] !=
zmap[x-1][y  ]); };
    auto rightEdge  = [&](int x, int y) { return (x == len-1    or zmap[x][y] !=
zmap[x+1][y  ]); };
    auto topEdge    = [&](int x, int y) { return (y == 0        or zmap[x][y] !=
zmap[x  ][y-1]); };
    auto bottomEdge = [&](int x, int y) { return (y == len-1    or zmap[x][y] !=
zmap[x  ][y+1]); };

    // Matrix with the projection
    // Note that the matrix is transpoed in order to draw X in horizontal axis
and Y in vertical axis
    vector<vector<char>> proj(len*2+1, vector<char>(len*2+1, ' '));
```

```
    // draw projection
    for (int i = 0; i < len; ++i)
    {
        for (int j = 0; j < len; ++j)
        {
            const int pi = i*2 + 1;
            const int pj = j*2 + 1;
            if ( zmap[i][j] >= 0)
            {
                if ( leftEdge(i,j) )                        proj[pj  ][pi-1] =
'|';
                if ( rightEdge(i,j) )                       proj[pj  ][pi+1] =
'|';
                if ( topEdge(i,j) )                         proj[pj-1][pi  ] =
'-';
                if ( bottomEdge(i,j) )                      proj[pj+1][pi  ] =
'-';
                if ( leftEdge(i,j) or topEdge(i,j) )        proj[pj-1][pi-1] =
'+';
                if ( leftEdge(i,j) or bottomEdge(i,j) )     proj[pj+1][pi-1] =
'+';
                if ( rightEdge(i,j) or topEdge(i,j) )       proj[pj-1][pi+1] =
'+';
                if ( rightEdge(i,j) or bottomEdge(i,j) )    proj[pj+1][pi+1] =
'+';
            }
        }
    }
    char border = '#';
    cout << std::string(11*2+3, border) << endl;
    for (int i = proj.size() - 1; i >= 0; --i)
    {
        cout << border;
        for (int j = 0; j < proj.front().size(); ++j)
        {
            cout << proj[i][j];
        }
        cout << border << endl;
    }
    cout << std::string(11*2+3, border) << endl;
}

int main()
{
    string projection;
    cin >> projection;
```

```
    int numVoxels;
    cin >> numVoxels;

    vector<Voxel> voxels;
    {
        int x, y, z;
        while (cin >> x >> y >> z)
        {
            // Just for debug, all inputs must be correct
            if (x > MAX_COORD or x < 0 ) return 0;
            if (y > MAX_COORD or y < 0 ) return 0;
            if (z > MAX_COORD or z < 0 ) return 0;

            voxels.push_back(Voxel(x,y,z));
        }
    }

    project(voxels, projection);

    // Just for debug, all inputs must be correct
    if (voxels.size() != numVoxels)
    {
        cout << endl << "WARNINIG: More voxels than specified!" << endl;
    }
    return 0;
}
```