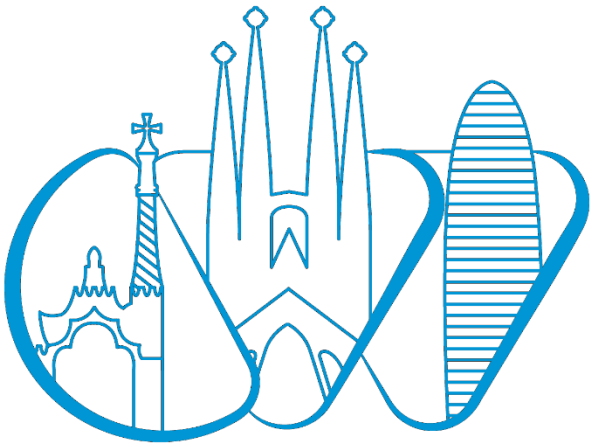


CodeWars

BCN 2024

Problems
and
Solutions

#	Problem	Points
1	Integer Pranks	1
2	Display Data In 3D	2
3	Weighting Hydrocarbons	2
4	Atmospheric Pressure Converter	2
5	The Magellanic Strain	3
6	The Triplet Treasure Hunt	3
7	Automated Assistant Referee	3
8	Problem 42	4
9	Galactic Bracket Balancing Equilibrium	4
10	Meeting Across Dune	4
11	Nilats Athletes	5
12	Repeated Paints	5
13	Upside Down	6
14	Drawing Hydrocarbons	6
15	Lost Letters Staircase	7
16	Palindrome Parade	7
17	Mars Rover	8
18	The Kiss Precise	8
19	Say That Again?	9
20	Misfit Columns	9
21	Permutation Ciphering	9
22	Chess Board	11
23	Rule 90	11
24	Mixing Hats	12
25	Gaussian Blur	13
26	Sudoku Solver	15
27	Finding Achilles	17
28	Boggle Search	19
29	Battleship Board Sketcher	20
30	Treetronomical Challenge	25
31	Sustainable Batteries	30
32	Monster Slayer	35

1 Integer Pranks

1 points

Introduction

In *Cipherworld* integers love to play tricks on programmers. Your mission, should you choose to accept it, is to write a program that can outsmart these mischievous integers. Your program should read three integers and determine if they are in successive ascending order. Can you bring order back to the world of programming and foil the pranks of these integers?

Input

The input is composed by three lines. Each line will contain a single integer. Three integer values are different.

Output

The output will return True if the three integers are received in increasing order. Otherwise, the output will be False.

Example 1

Input

```
5
10
3
```

Output

```
False
```

Example 2

Input

```
-11
-6
1
```

Output

```
True
```

Python

```python
a = int(input())
b = int(input())
c = int(input())

if (a < b and b < c):
    print("True")
else:
    print("False")
```

2 Displaying Data In 3D

2 points

Introduction

Seeing in 3D is mostly about tricking your brain. Since our eyes are placed a bit apart from each other, our brain receives two different images. Consequently, these images are shifted by a very small amount from each other. Then, our brain proceeds to merge them to see a single image. Our brain perceives depth because of the separation between these two images coming from the two eyes.

The research project you're currently focused on is about 3D glasses with augmented reality. For example, when your eyes focus on a building, the glasses display the distance in meters to reach that place. To provide the 3D effect, this data will be represented by the glasses to both eyes slightly shifted.

The very first step for this process is to support the duplication of data to be displayed. This data will then be shown to the left and right glasses. Can you write a program that reads a positive integer and duplicates all of its digits?

Input

The input is a single line with a positive integer.

Output

The output is a single line with all the digits of the input duplicated.

Example

Input

```
12345
```

Output

```
1122334455
```

Python

```python
data = input()
result = ""

for i in data:
    result = result + 2*i

print(result)
```

3 Weighting Hydrocarbons

2 points

Introduction

In organic chemistry, the hydrocarbon "alkane" is a molecule made up of linked Carbon (C) atoms with Hydrogen (H) atoms branching off each Carbon like a tree structure. Here is the internal structure of ethane (C_2H_6):

```
  H H
  | |
H-C-C-H
  | |
  H H
```

Since you are frequently going to experiment with hydrocarbons in the chemistry lab, you decided to code a program to quickly find out the atomic mass of a given molecule. Given the number of Carbon atoms in a hydrocarbon, you can easily calculate the number of Hydrogen atoms:

$$\boldsymbol{number\ of\ H\ atoms} = (\boldsymbol{number\ of\ C\ atoms} * \mathbf{2}) + \mathbf{2}$$

The atomic weight of the hydrocarbon molecule is calculated by multiplying the number of Carbon atoms by 12 (its atomic mass) and adding the number of Hydrogen atoms (the atomic mass of Hydrogen is 1).

Can you write a simple program to find the formula and atomic mass of the corresponding hydrocarbon given the number of atoms of Carbon?

Input

The input is a positive integer representing the number of Carbon atoms

Output

The formula given the number of C and H atoms along with its atomic weight.

Example 1

Input

```
3
```

Output

```
The atomic mass of C3H8 is 44
```

Example 2

Input

```
2
```

Output

```
The atomic mass of C2H6 is 30
```

Python

```python
carbon = int(input())

hydrogen = carbon * 2 + 2

mass = 12 * carbon + hydrogen

print("The atomic mass of C" + str(carbon) + "H" + str(hydrogen) + " is " +
str(mass))
```

4 Atmospheric Pressure Converter

2 points

Introduction

Changes in atmospheric pressure can tell us about the weather. If the pressure is high, it usually means we will have nice, clear weather with lots of sunshine. If the pressure is low, it might mean that rain or storms are on the way.

The pascal (Pa) is the unit of pressure in the International System of Units (SI). Your weather station reports the measured atmospheric pressure in hectopascal (hPa). But your friend Eolo always asks you about this data in millimeters of Mercury (mmHg).

Considering the conversion factors (1 mmHg = 133.322 Pa) can you write down a program that converts data from hPa to mmHg with a precision of two decimal places.

Input

A positive integer value representing the pressure in hPa units.

Output

A positive float value representing the pressure in mmHg with a precision of two decimal places.

Example

Input

```
1018
```

Output

```
763.56
```

Python

```python
pressurePa = int(input()) * 100

pressureMmHg = pressurePa  / 133.322

print("{:.2f}".format(pressureMmHg))
```

5 The Magellanic Strain

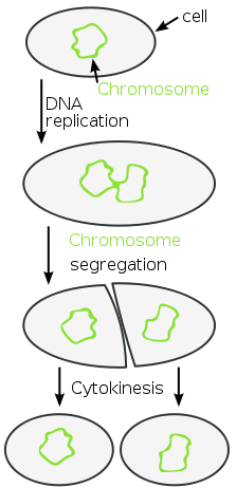
3 points

Introduction

From the Southern Hemisphere it's possible to admire the Large and Small Magellanic Clouds, which are satellite galaxies of the Milky Way. As some of the closest galaxies to our home galaxy, they stand out as big, misty blobs of light under dark skies. A few days ago, a meteor was observed falling between the two Magellanic Clouds.

Meteor falling between Magellanic Clouds

Unfortunately for mankind, after the meteor was collected, it was discovered that it was carrying an extraterrestrial microbe inside. A team of scientists are studying the growth cycle of the microbe. Like terrestrial bacteria, it follows a four-step process called binary fission to clone itself. This process takes 13 minutes.



Binary Fission

To control the Magellanic strain, the scientists urge you to develop a program to calculate the microbe population after a given amount of time.

Input

The input is composed by two positive integers:

- The first value represents the number of microbes present at the beginning of the measurement.
- The second value represents the amount of time in minutes left to allow bacteria growth.

Output

The output is a single integer reporting the total amount of microbe population expected.

Example 1

Input

```
3
0
```

Output

```
3
```

Example 2

Input

```
9
52
```

Output

```
144
```

Example 3

Input

```
1
25
```

Output

```
2
```

Python

```python
initial_population = int(input())
minutes = int(input())

growthCycle = 13

numCycles = minutes // growthCycle

final_population = initial_population * (2**numCycles)

print(final_population)
```

6 The Triplet Treasure Hunt

3 points

Introduction

In a distant land known as Numeria, there exists a mystical forest named Triplet Grove. Legends have it that deep within this forest, there lies a hidden treasure of unimaginable wealth. The catch? To unlock the treasure's location, one must possess the knowledge of the "Triplet Numbers."

The Triplet Numbers are special integers that are multiples of 3 and play a crucial role in unraveling the treasure's mystery. The forest is enchanted with a magical aura that only reveals its secrets to those who can harness the power of the Triplets.

As an aspiring treasure hunter, you find yourself standing at the edge of Triplet Grove, determined to uncover its riches. However, you have a dilemma: you don't know where to begin your search.

A wise old sage approaches you and offers guidance. She tells you that the first step in deciphering the forest's secrets is to understand the essence of the Triplet Numbers. She hands you a parchment with a cryptic message:

"Seek the sum of the Triplet Numbers to unveil the path to riches."

With these words, she disappears into the forest mist, leaving you with a task. You must write a program to calculate the sum of all multiples of 3 that are less than or equal to a given number n. This sum, as you've been told, holds the key to unlocking the treasure's location deep within Triplet Grove.

Armed with your coding skills and the wisdom of the Triplet Numbers, you embark on this journey, determined to reveal the secret of the enchanted forest and claim the long-lost treasure as your own. Now, it's your turn to write the code and solve the mystery of the Triplet Numbers!

Input

The input is a positive integer

Output

The output is a positive integer.

Example 1

Input

7

Output

9

Example 2

Input

15

Output

45

Python

```python
value = int(input())

result = 0

for i in range(1,value+1):
    if ( i % 3 == 0 ):
        result = result + i

print(result)
```

7 Automated Assistant Referee

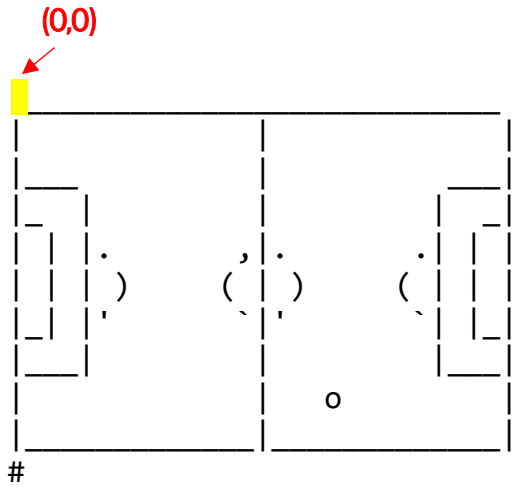
3 points

Introduction

To avoid controversy in the refereeing of soccer matches, it is planned to use a drone to perform the duties of a linesman. Before making this system official, it is important to verify its correct operation. For now the drone will only be in charge of tracking the ball during the match.

Input

You will receive a map grid made up of lines of ASCII characters of the football pitch. The map will be between 10-16 lines tall, and 29-41 characters wide. The size will depend on the drone flying altitude. The ball will be marked on the map with a "o" character. The map always ends with the character"#".

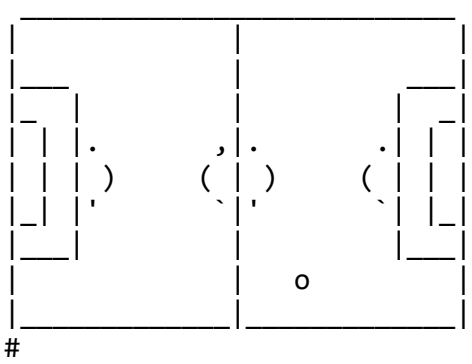


Output

Once the drone finds the ball in the map with a "o" character, the drone will report the coordinates from the map. The upper left of the map will be (X=0, Y=0). The lower right of the map will be the maximum values for X and Y.

Example

Input



Output

```
The ball is in: (18, 8)
```

Python

```python
x=0
y=0

# Read first line from input
line = input()

# While line is different from end of input,
# process the lines looking for the ball
while line != "#":
    # Find the ball in current line
    x = line.find('o')

    # When the ball is found the x-axis position is given from find method
    # and the y-axis matches with the number of line processed. Search for
    # the ball has finished
    if (x != -1):
        print("The ball is in: (" + str(x) + ", " + str(y) + ")")
        break
    # Move to next line
    line = input()
    # Increase the line counter
    y = y + 1
```

8 Problem 42

4 points

Introduction

42, what a number! Do you know that...

- It is the sum of the first six positive even numbers. It is also the sum of the numbers on a pair of dice.
- It is the atomic number of Molybdenum.
- The hypothetical efficiency of converting mass to energy, as per known formula $E=mc^2$, by having a given mass orbit a rotating black hole is 42%, the highest efficiency yet known to modern physics.
- The Orion Nebula is the popular name for Messier object M42, a magnitude 5.0 diffuse nebula in the constellation Orion.
- The ASCII code 42 is for the asterisk symbol, being a wildcard for everything.
- There are 42 US gallons in a barrel of oil.
- Level 42 is a popular English music band.
- "42" is one of the tracks on Coldplay's 2008 album Viva la Vida or Death and All His Friends.
- In Star Wars Episode IX: The Rise of Skywalker, the Festival of the Ancestors on Planet Pasaana is held every 42 years.
- Last but not least, in "The Hitchhiker's Guide to the Galaxy" novel by Douglas Adams, the number 42 is the "Answer to the Ultimate Question of Life, the Universe, and Everything", calculated by an enormous supercomputer named Deep Thought over a period of 7.5 million years. Unfortunately, no one knows what the question is.

Given the importance of such number we need your help to detect how many times it appears in a given message.

To make things easier do not consider the case of decimal values with comma or point like 42.5 or 42,5. Neither consider the case of having a prefix formed by zeros like 042.

Input

A single line with the message to be analyzed.

Output

A single line with exactly the number of occurrences of 42.

Example 1

Input

```
My favourite number is 42. That is a 4 next to a 2 and nothing else. So, 420 is not valid. Just 42!
```

Output

```
2
```

Example 2

Input

```
See the Orion Nebula M42 through my telescope that my grandfather bought in 1942
```

Output

```
1
```

Python

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

while True:

  anyDigitBefore = False
  anyDigitAfter = False
  index = data.find("42", index)

  # Is there any other 42 pending to review
  if index == -1:
    break

  # Check there are no any extra digits before 42
  if index > 0:
    if (data[index-1].isdigit()):
      anyDigitBefore = True

  # Check that there are no extra digits after 42
  if index+2 < length:
    if data[index+2].isdigit():
      anyDigitAfter = True

  if not(anyDigitBefore) and not(anyDigitAfter):
    count += 1

  # Move the start position past the last match
  index += 2

print(count)
```

9 Galactic Bracket Balance Equilibrium

4 points

Introduction

In a distant future, humanity has colonized numerous planets and galaxies. With advanced technology and interstellar travel, the cosmos has become the playground of explorers, adventurers, and mathematicians alike.

One day, while navigating through a wormhole, your spaceship encounters a peculiar cosmic phenomenon: a rift in space-time that distorts reality itself. Your ship's computer, equipped with advanced AI, detects a mysterious signal emanating from this rift. Upon closer inspection, you realize it's not a signal but a sequence of brackets – opening brackets '(' and closing brackets ')' – arranged in a peculiar pattern.

As an intrepid astronaut-mathematician, you are tasked with deciphering the secrets hidden within this cosmic anomaly. Your mission is to find the "Galactic Bracket Balance Equilibrium" point within the string of brackets.

An equal point in this cosmic string is the index k where the number of opening brackets '(' occurring before index k is precisely equal to the number of closing brackets ')' occurring from index k onwards. The counting for index k will start always at 0.

Picture yourself in your spacesuit, floating near this rift in space-time, laser-scanning the brackets, and running your AI algorithms to solve this otherworldly puzzle.

Write a program that takes the cosmic string as input and returns the index of the Galactic Bracket Balance Equilibrium point.

Input

A cosmic string s that consists of only two characters: '(' and ')'.

Output

An integer representing the index of the Galactic Bracket Balance Equilibrium point followed by the balanced and split bracket sequence.

Example 1

Input

```
(()())
```

Output

```
3 (()-())
```

Example 2

Input

```
(((
```

Output

```
0 -(((
```

Example 3

Input

```
((()))
```

Output

```
3 (((-)))
```

Example 4

Input

```
())(()
```

Output

```
3 ())-(()
```

Example 5

Input

```
))
```

Output

```
2 ))-
```

Python

```python
brackets = input()
index =  0

open = 0
#print("Original: " + brackets)
# Count open and close brackets
for i in range(len(brackets)):
    if brackets[i] == "(":
        open = open + 1
#    print(str(i+1) + "  " + str(len(brackets)))
#    print(brackets[:i+1] + " " + brackets[i+1:])
    close = 0
    for j in range(i+1,len(brackets)):
        if brackets[j] == ")":
            close = close + 1
    if open == close:
        index = i + 1
        break
#    print (" Loop: " + str(i) + " current open: " + str(open) + " current
close: " + str(close) + " index: " + str(index))

if index != -1:
  print(str(index) + " " + brackets[:index] + "-" + brackets[index:])
else:
  print(index)
```

10 Meeting Across Dune

4 points

Introduction

In the Dune universe, ornithopters are the primary mode of transportation on desert planet Arrakis. It is basically an aircraft that flies by flapping its wings.

Two ornithopters set out from distinct cities with the intention of meeting each other somewhere along the way, although not precisely at the midpoint due to their different speeds. Assuming a straight-line path between the two cities, you will be provided with the distance between the cities in kilometers and the speed values of the two ornithopters in kilometers per hour. Both ornithopters follow a uniform linear motion, meaning that the acceleration is 0 throughout the motion. Your objective is to determine the distance from the first city to their meeting point with a precision of three decimal places.

Input

Three lines containing each line a single integer value referring to:

- Distance between cities in kilometers
- First ornithopter speed in kilometers per hour
- Second ornithopter speed in kilometers per hour

Output

The distance in kilometers from the first city and the rendezvous point with a precision of three decimal places, that is the distance flown by first ornithopter before they meet.

Example 1

Input

```
30
1
2
```

Output

```
10.000
```

Example 2

Input

```
20
1
2
```

Output

```
6.667
```

Example 3

Input 3

```
20
2
1
```

Output

```
13.333
```

Python

```python
distance = int(input())
speedThopter1 = int(input())
speedThopter2 = int(input())

time = distance / (speedThopter1 + speedThopter2)

position = time * speedThopter1

print("{:.3f}".format(position))
```

11 Nilats Athletes

5 points

Introduction

Nilats was once an impressive athlete that is now in charge of the Naissur Sports Federation. He has been given the order to choose which athletes go to the Olympic Games and which do not.

In order to properly select the ones that will represent the country he has decided that the best option is to create an algorithm. The only problem is that he is really bad at making decisions, so he has implemented a rather particular way of choosing the athletes. Since all these incredible athletes have been showing amazing results, he has created an evaluation system and a divider, so all the athletes have been divided in groups of 8. Each of these athletes has a number (an integer, Nilats likes absolutes) that determines how likely they are to succeed in the sport they participate.

Since he is incredibly bad at properly choosing the athletes, he is thinking about an algorithm that will use those numbers that each athlete has. The logic is the following:

The algorithm should get the first number given to it and directly select that athlete, whatever number that athlete has. Then it will compare the score of the second athlete and if that athlete has a better score that the first one it will select him or her too. However, if the third one has a worse score than the second one that athlete will not go to the Olympics and it is removed from the input. If the fourth one has a greater score than the second one it will go to the Olympics. If not, it will be removed too. If they have the same score the athlete is also removed.

Could you help him write it since he is also really bad at coding?

Input

Eight integers separated by a space.

Output

From one to eight integers separated by a space.

Example 1

Input

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

Output

```
1 2 3 6
```

Example2

Input

```
8 7 6 6 7 8 9 9
```

Output

```
8 9
```

Python

```python
def nilats_sort(input_list):
    sorted_list = [input_list[0]]
    max_so_far = input_list[0]
    for num in input_list[1:]:
        if num > max_so_far:
            sorted_list.append(num)
            max_so_far = num
    return sorted_list


user_input = input()
input_numbers = user_input.split()

if len(input_numbers) > 8:
    print("Please enter no more than 8 numbers.")
else:
    input_numbers = [int(num) for num in input_numbers]

    sorted_numbers = nilats_sort(input_numbers)

    sorted_numbers_str = ' '.join(map(str, sorted_numbers))
    print(sorted_numbers_str)
```

12 Repeated Paints

5 points

Introduction

Welcome to 'The Color House,' a paint shop that features ten enchanting colors, each with a unique ID ranging from 0 to 9. The paints are stored in drawers, but they are a bit messy and can contain a variable number of paints including the possibility of repeated paint colors.

As a helpful worker at 'The Color House,' we need your magical coding skills to find out which paint color is the most popular in our drawers. Our aim is to ensure that we have enough of the favorite color in stock. Do not worry about possible ties because there will be always a single most popular and repeated paint color.

For instance, in the case of having just two drawers, if the first drawer has paints with IDs 0, 1 and 2. And the second drawer has paints with IDs 2, 2, 3, 4 and 5, then your program should reveal that the color with ID 2 is the most popular with three occurrences, considering all the paints from the drawers.

Unleash your coding magic and assist us in identifying the most repeated paint color at 'The Color House,' navigating the variable number of each paint and the chance of repeated colors within each drawer. And do not worry about possible ties because there will always be a unique most popular and repeated color paint.

Input

A single line with a variable number of drawers, each separated by a space. The content of each drawer is described by consecutive color paint ID's.

Output

The most repeated color paint ID.

Example 1

Input

```
12 222100 34401 001
```

Output

```
0
```

Example 2

Input

```
431 44420 3210 33213 03211
```

Output

```
3
```

Python

```python
ids = input().split(' ')
cont = dict()

for id in ids:
    for i in id:
        if i in cont.keys():
            cont[i] +=1
        else:
            cont[i] = 1

max_key = max(cont, key=lambda key: cont[key])

print(max_key)
```

13 Upside Down

6 points

Introduction

The Upside Down, in Stranger Things TV series, is a parallel dimension to real world. Much of it remains as a mistery, but a recent scientic investigation discovered that obscure messages coming from that dimension can be translated following the upside down rule.

Each message received is a singular word separated by a line break, or a line with the keyword UPSIDE_DOWN. To be able to read a message scientists discovered that words must be stored in the order them are received, that is, new words should be added to the end of the message. However, when the keyword UPSIDE_DOWN appears, then the order of all the words must be reversed. They also noticed that all messages end with the symbol #. By processing the message this way, you end up a clear message.

Can you write a program to help the people of Hawkins understand messages from the Upside Down?

Input

The input is a series of lines containing a single word that is part of the message alternatively will appear the word UPSIDE_DOWN.

Output

The output is a single line with the translated message.

Example 1

Input

```
of
Hawkins
UPSIDE_DOWN
town
small
UPSIDE_DOWN
in
UPSIDE_DOWN
the
to
Welcome
UPSIDE_DOWN
Indiana
#
```

Output

```
Welcome to the small town of Hawkins in Indiana
```

Python

```python
upsideDown = False
wordList = []
finalMessage = ""

wordRead = input()

while wordRead != "#":
    if wordRead == "UPSIDE_DOWN":
        wordList.reverse()
    else:
        wordList.append(wordRead)

    wordRead = input()

for word in wordList:
    finalMessage += word + " "

print(finalMessage)
```

14 Drawing Hydrocarbons

6 points

Introduction

Many chemistry lab experiments are focused on hydrocarbon alkanes. These are formed by Carbon-to-Carbon single bonds (C-C) and exist as a continuous chain of Carbon atoms also bonded to Hydrogen atoms.

Methane (CH_4), ethane (C_2H_6), and propane (C_3H_8) are the first three of a series of compounds in which any two members in a sequence differ by one Carbon atom and two Hydrogen atoms—namely, a CH_2 unit. The investigations in the lab will cover up to the first 12 members of hydrocarbon alkanes.

Here is the internal structure of Butane, an alkane having four Carbon atoms (C_4H_{10}):

```
  H H H H
  | | | |
H-C-C-C-C-H
  | | | |
  H H H H
```

Since you are going to perform several experiments, you do not want to spend too much time drawing such structures in your notebook. So you decide to automate the drawings by coding a program that, given the number of Carbon atoms, draws the corresponding hydrocarbon structure. Remember that you only need to consider the first 12 hydrocarbons.

Input

The input is a positive integer representing the number of Carbon atoms in the hydrocarbon to represent.

Output

The structure of the hydrocarbon given the number of C atoms and the corresponding bonds with H atoms.

Example 1

Input

```
1
```

Output

```
  H
  |
H-C-H
  |
  H
```

Example 2

Input

```
4
```

Output

```
  H H H H
  | | | |
H-C-C-C-C-H
  | | | |
  H H H H
```

Python

```python
carbon = int(input())

# Basic case for one single carbon

line1 = "  H"
line2 = "  |"
line3 = "H-C-H"

# Generic case for n carbons
if (carbon > 1):
  line1 = line1 + " H" * (carbon-1)
  line2 = line2 + " |" * (carbon-1)
  line3 = "H" + "-C"  * carbon + "-H"

# Printing the output
print(line1)
print(line2)
print(line3)
print(line2)
print(line1)
```

15 Lost Letters Staircase

7 points

Introduction

Have you ever wondered what happens to letters when they are not used in a text? Such letters become lost letters. Subsequently, they disappear, as they are no longer useful to a specific text, and they cascade down the exit staircase. The lost letters staircase is constructed from these lost letters, arranged in alphabetical order, one after the other. Each step consists of a letter followed by an underscore (_) character, except for last step, which comprises just the very last letter. To add a touch of humor to their disappearance, the letters are set alternatively in uppercase and lowercase, always starting with a capital letter.

Would you code a program to build up the lost letters staircase for a given text?

Input

The input will be formed by several lines of text composed exclusively by English alphabet letters, spaces and character # which marks the end of the text.

Output

The lost letters staircase formed by letters not used in the text alphabetically ordered. Just one letter per step followed by an underscore (_) character, except for last step. If there are no lost letters in the text just print out the message "There are no letters lost!".

Example 1

Input

```
It is a period of civil war Rebel spaceships
striking from a hidden base have won their
first victory against the evil Galactic
Empire#
```

Output

```
J_
  q_
    U_
      x_
        Z
```

Example 2

Input

```
Sphinx of lack quartz judge my vow#
```

Output

```
B
```

Example 3

Input

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

Output

```
There are no letters lost!
```

Python

```python
alphabet= []
for i in range(26):
    alphabet.append(0)
end = False

# Build frequency table
while not end:
    # Read line from standard output
    line = input()
    for letter in line:
        if letter == " ":
            continue
        elif letter != "#":
            letter = letter.upper()
            alphabet[ord(letter) - ord('A')] += 1
        else:
            end = True
            break

# Build staircase with letters that have zero frequency
result = ""
line = 0
nextCapital = True
staircase = []
for i in range(26):
    if alphabet[i] == 0:
        letter = chr(i+ord('A'))
        if not nextCapital:
            letter = letter.lower()
        staircase.append(line * "  " + letter + "_")
        nextCapital = not nextCapital
        line += 1

# Print result. Take into account the case of last step
steps = len(staircase)
if (steps > 0):
    for i in range(steps-1):
        print(staircase[i])
    lastStep = staircase[steps-1][:-1]
    print(lastStep)
else:
    print("There are no letters lost!")
```

16 Palindrome Parade

7 points

Introduction

Welcome to the Palindrome Parade, where words of all shapes and sizes march together, celebrating their symmetrical beauty! Your task is to organize the parade by creating a special merger for this grand event.

Given two lists, each filled with palindromic and non-palindromic words, the merging process is unique: we want to merge the lists based on whether each word is a palindrome or not. Palindromic words lead the parade, followed by non-palindromic words. So, keeping the original order from the lists, the parade will move as: first palindromic words from first list, followed by palindromic words from second list, then non-palindromic words from first list and finally non-palindromic words from second list.

Some words can appear in both lists. So, be careful that they do not appear repeated along the merged parade. Only the first appearance of a word is allowed.

Can you write a program to organize the Palindrome Parade?

Input

The input is composed by several lines: the first line contains the number of elements of first list. Followed by the list of words of first list. Next line will contain the number of elements of second list. Finally, there is the list of words from second list.

Output

The output is composed by several lines with a single word per line which considers the original order of appearance. First palindromic words from first list, followed by palindromic words from second list, then non-palindromic words from first list and finally non-palindromic words from second list.

Example 1

Input

```
6
radar
apple
deed
banana
level
civic
8
orange
radar
noon
kayak
grape
table
hello
grape
```

Output

```
radar
deed
level
civic
noon
kayak
apple
banana
orange
grape
table
hello
```

Python

```python
def isPalindromic(word):
    if word == word[::-1]:
      return True
    else:
     return False

def addToList(list, paradeList, palindromic):
    for i in list:
        if i not in paradeList:
            if isPalindromic(i) == palindromic:
                paradeList.append(i)

numList1 = int(input())
list1 = []
for i in range(numList1):
    list1.append(input())

numList2 = int(input())
list2 = []
for i in range(numList2):
    list2.append(input())

paradeList = []

addToList(list1, paradeList, True)
addToList(list2, paradeList, True)
addToList(list1, paradeList, False)
addToList(list2, paradeList,  False)

for i in paradeList:
    print(i)
```

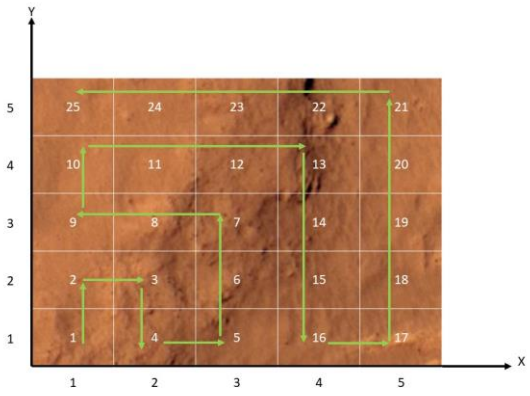
17 Mars Rover

8 points

Introduction

A new Mars rover is being developed to travel autonomously on the surface of Mars. To test its navigation, you prepared a N x N board where N represents the number of cells. The rover must move around all the cells. The navigation algorithm of the Mars rover will always start the move at cell (1,1). First, it moves up to the next cell, then a cell to the right, then a cell downward. At this point, it then moves one cell to the right, next two cells upward, and continues the move two cells to the left. This pattern will continue until the whole board is discovered. To keep things simple just assume a constant speed of one grid per second.

Let's see a graphical example with a 5 x 5 board:



Given this navigation strategy it is easy to predict the time spent to arrive at the target position (x,y). When rover is at (2,3), the rover took 8 seconds. And when it was at (5,4), the rover was at the 20th second of the movement.

Your task will be to compute the time spent to reach a given position (x,y). Please assume that N could be as large as 4096.

Input

The input is formed by two lines reporting the position in the board:

First line has a single positive integer for the x position.

Second line has a single positive integer for the y position.

Output

The output is the number of seconds to reach the given input position.

Example 1

Input

```
2
3
```

Output

```
8
```

Example 2

Input

```
5
4
```

Output

```
20
```

Python

```python
import math

x = int(input())
y = int(input())
res = 0
squareSize = 0

# Find out the side of the square that includes the position (x,y)
# considering the biggest of the two dimensions.
# Then find out value corresponding to the cell

if (x > y):
  if (x%2 == 0):
      res = x**2 - (y - 1)
  else:
      res = (x-1)**2 + y
else:
  if (y%2 == 0):
      res = (y-1)**2 + x
  else:
      res = y**2 - (x - 1)

print(res)
```

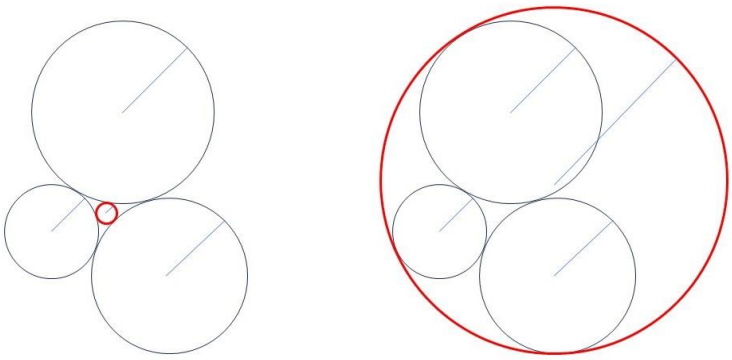
18 The Kiss Precise

8 points

Introduction

Four circles to the kissing come.
The smaller are the benter.
The bend is just the inverse of
The distance form the center.
Though their intrigue left Euclid dumb
There's now no need for rule of thumb.
Since zero bend's a dead straight line
And concave bends have minus sign,
The sum of the squares of all four bends
Is half the square of their sum.

In 1936 the mathematician Frederick Soddy published in *Nature* magazine this poem summarizing Descartes circle theorem. This theorem is about determination of a circle touching three mutually tangent circles (also called the kissing circles problem). There are two solutions: a small circle surrounded by the three original circles, and a large circle surrounding the original three. See them in red in the picture.



But if you prefer a math language description here you have the formula

$$2\left(s_1^2 + s_2^2 + s_3^2 + s_4^2\right) = (s_1 + s_2 + s_3 + s_4)^2$$

having that

$$s_i = \frac{1}{r_i}$$

where r_i is the radius of circle i.

This can be solved as a quadratic equation with two solutions. One of these solutions is positive (corresponds to inner circle), and the other is either positive or negative (corresponds to outer circle); if the second solution is negative, it must represent a circle that is internally tangent to the other three.

Can you write a program to find out the radios of inner and outer circles?

Input

The input is composed by three lines:

The first line contains the radius of the first circle.

The second line contains the radius of the second circle.

The third line contains the radius of the third circle.

Output

The output is composed by two lines:

The first line contains the radius of the inner circle with 5 decimal places.

The second line contains the radius of the outer circle with 5 decimal places.

Example

Input

```
1
2
5
```

Output

```
0.28663
-11.25437
```

C++

```
/* Given the formula for finding the radii of the Soddy circles (s4 = 1/r4)
given that si = 1/ri corresponding to radius r1, r2 and r3 of other three
circles.
 *
 * 2 * (s1^2 + s2^2 + s3^2 + s4^2) = (s1 + s2 + s3 + s4)^2
 *
 * Let's assume a = s1^2 + s2^2 + s3^2 and b = s1 + s2 + s3 to simplify so we
have
 *
 * 2 * (a + s4^2) = (b + s4)^2
 *
 * 2a + 2s4^2 = s4^2 + 2bs4 + b^2
 *
 * s4^2 - 2bs4 + 2a - b^2 = 0 (To be solved using a quadratic equation formula)
 *
 * s4 = 2b +- sqrt ((2b)^2 - 4(2a-b^2)) / 2
 */

#include<iostream>
#include<iomanip>
#include<math.h>

using namespace std;

int main( ) {

    int r1, r2, r3 ;
    double s1, s2, s3 ;
    double a, b;
    double s4a, s4b, r4a, r4b;

    // Read the three radius from input
    cin >> r1;
    cin >> r2;
    cin >> r3;

    s1 = 1 / (double)r1;
    s2 = 1 / (double)r2;
    s3 = 1 / (double)r3;

    a = pow(s1,2) + pow(s2,2) + pow(s3,2);
    b = s1 + s2 + s3;

    s4a  = (2*b + sqrt( pow(2*b,2) - 4*(2*a-pow(b,2))))/2;
    s4b  = (2*b - sqrt( pow(2*b,2) - 4*(2*a-pow(b,2))))/2;
```

```
    r4a = 1 / s4a;
    r4b = 1 / s4b;

    cout << fixed << setprecision(5) << r4a << endl;
    cout << fixed << setprecision(5) << r4b << endl;

}
```

19 Say That Again?

9 points

Introduction

AI1-C3 and B-0b are two friendly robots that wish to establish a communication protocol with some redundancy to avoid miscommunication. As they are robots, they do not communicate by words but rather with bytes (8-bit numbers) represented by integer numbers from 0 to 255. For their protocol, they came up with the simple idea to repeat each of the bytes in the message three times, so that the received byte is formed by the most common bit at each of the 8 positions within the byte.

For example, if the bytes are 24, 117 and 178, then the resulting byte is 48:

n	b7	b6	b5	b4	b3	b2	b1	b0
24	**0**	**0**	**0**	1	1	**0**	**0**	**0**
117	**0**	1	1	1	**0**	1	**0**	1
178	1	**0**	1	1	**0**	**0**	1	**0**
48	0	0	1	1	0	0	0	0

Please, help AI1-C3 and B-0b implement the program that translates the repeated bytes into the final byte on the receiving end.

Input

Several lines, each containing a triplet of bytes (separated by a single space), except the last line that contains only the character '#'. Bytes are represented by an integer number between 0 and 255 (both inclusive).

Output

One line for each triplet of bytes with the final byte according to the protocol described above. Bytes are represented by an integer number between 0 and 255 (both inclusive).

Example 1

Input

```
24 117 178
3 3 0
1 2 3
#
```

Output

```
48
3
3
```

Example 2

Input

```
4 4 2
0 0 0
255 127 128
4 2 8
#
```

Output

```
4
0
255
0
```

Python

```python
while True:
    line = input()
    if line == "#":
        break

    x, y, z = line.split()
    x = int(x)
    y = int(y)
    z = int(z)

    print((x&y) | (y&z) | (z&x))
```

20 Misfit Columns

9 points

Introduction

Calling all Number Detectives! We need your help to uncover the misfit columns in our mysterious 3x5 number matrix. Your program must identify those columns where the elements have different values. It's like a game of hide-and-seek, but with numbers! Get ready to dive into the world of matrix and expose the columns that have all their values different. In the event that a matrix doesn't have such columns, print out a single 0.

Input

The input is a 3x5 matrix. It is composed by three lines. Each line will have five positive integers separated by a space.

Output

The output will display the columns whose elements have different values in same order than the original matrix.

Example 1

Input

```
1 2 2 5 9
3 0 2 5 1
1 7 2 3 0
```

Output

```
2 9
0 1
7 0
```

Example 2

Input

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

Output

```
0
```

Example 3

Input

```
10 21 31 14 15
60 71 89 29 1
10 21 30 34 15
```

Output

```
31 14
89 29
30 34
```

Python

```python
# Initialize an empty matrix
matrix = []

# Read matrix from standard input
for i in range(3):
    row = input().split()  # Assuming elements are separated by spaces
    # Convert elements to integers if needed
    row = [int(x) for x in row]
    matrix.append(row)

# print matrix read
# for i in range(3):
#  print(matrix[i])

mischievous_columns = []
for col in range(5):  # Iterate through each column
    column_values = set()  # To store unique values in the column
    for row in range(3):  # Iterate through each row in the column
        column_values.add(matrix[row][col])
    # print(column_values)
    # If the number of unique values equals the number of rows, all values are
different
    if len(column_values) == 3:
        mischievous_columns.append(col)

# Print just 0 if there are no mischievous columns
if len(mischievous_columns) == 0:
    print("0")
else:
    # Construct a list of lines containing the mischievous columns
    output = []
    for j in range(3):
        row = ""
        for i in mischievous_columns:
            if row == "":
                row = str(matrix[j][i])
            else:
                row = row + " " + str(matrix[j][i])
        output.append(row)
    # Print the mischievous columns line per line
    for line in output:
        print(line)
```

21 Permutation Ciphering

9 points

Introduction

Permutation cyphering is an example of symmetric cryptography, a cryptographic approach where both the sender and receiver collaborate using a common encryption key. Such key is an integer of n digits ($n \leq 9$) where each digit must be between 1 and n and appear only once.

For instance, let's consider the 5-digit key '25413' to encrypt the Mandalorian message 'This is the way!!'. Since the key size is 5, the message is divided into four segments, each with 5 characters. In the case where the last segment's size is smaller than the key size, padding characters (represented by '*') are added.

1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
T	h	i	s		i	s		t	h	e		w	a	y	!	!	*	*	*
h		s	T	i	s	h	t	i			y	a	e	w	!	*	*	!	*

During the encryption process, the characters are arranged according to the key's digits corresponding to their own segment. To clarify, let's look at an example: the second character from the original message becomes the first character in the encrypted segment, the fifth character takes the second position, and so on. This pattern is applied to each segment in the process of rearranging characters.

Can you write a simple program that, given a certain key, is able to encrypt a message following the permutation cyphering?

Input

The input is composed by two lines where the first line just contains the ciphering key and the second line has the message to encrypt.

Output

The output is single line with the crypted message.

Example

Input

```
321
The force is with us!!
```

Output

```
ehTof ecrsi iw  ht!su**!
```

Python

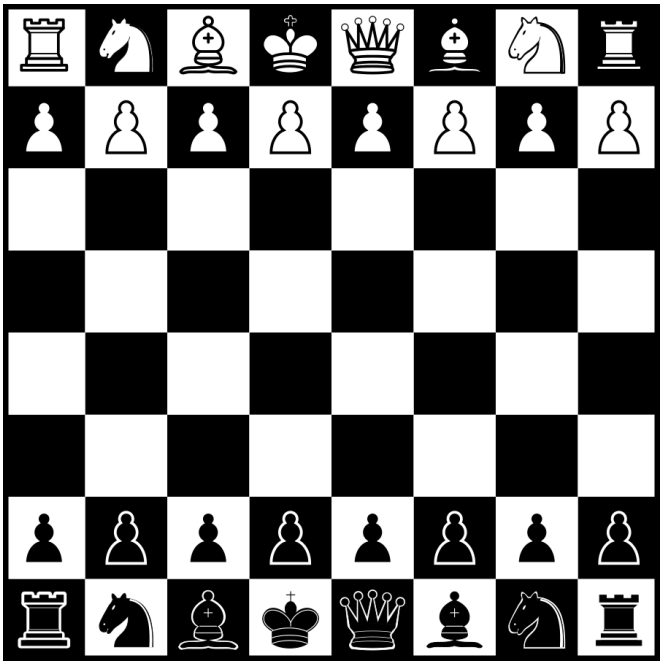
```python
key = input()
message = input()
cryptedMessage = ""
index = []

# Fill the message with padding if needed.
# Beware of a special case when key is just 1.
if len(key) > 1 and (len(message) % len(key)) > 0:
    padding = len(key) - (len(message) % len(key))
    message = message + padding * "*"

# Create the index vector to store the cypher order
for i in key:
    index.append(int(i)-1)

# Construct the cyphered message by accessing to the original message following
the
# cypher sequence.
for i in range(len(message)):
    cryptedMessage = cryptedMessage + message[index[i%len(key)] + i//len(key) *
len(key)]

print(cryptedMessage)
```

22 Chess Board

11 points

Introduction

After the last World Chess Championship, a fever to play chess has risen around the globe resulting in a high demand of chess boards.



It's a big deal to build chess boards nowadays. It's even better if you allow people to customize their boards. For example, instead of having solid black squares, you might offer the chance to print any pattern like a letter, a number or even a symbol. Another feature could be to allow customers to define the size of the squares in their chess board. Before starting a massive production of chess boards, your company wants to ensure that the software is ready to support the printing of customized patterns. Can you write a program that prints a chess board given a pattern defined by a specific character and square size?

Input

A line with a single character that represents the pattern to use,

A line with a single positive number ($1 < size < 15$) that provides the size of each square.

Output

Print the requested chess board.

Example 1

Input

```
#
1
```

Output

```
-----------------
| |#| |#| |#| |#|
-----------------
|#| |#| |#| |#| |
-----------------
| |#| |#| |#| |#|
-----------------
|#| |#| |#| |#| |
-----------------
| |#| |#| |#| |#|
-----------------
|#| |#| |#| |#| |
-----------------
| |#| |#| |#| |#|
-----------------
|#| |#| |#| |#| |
-----------------
```

Example 2

Input

```
X
3
```

Output

```
---------------------------------
|   |XXX|   |XXX|   |XXX|   |XXX|
|   |XXX|   |XXX|   |XXX|   |XXX|
|   |XXX|   |XXX|   |XXX|   |XXX|
---------------------------------
|XXX|   |XXX|   |XXX|   |XXX|   |
|XXX|   |XXX|   |XXX|   |XXX|   |
|XXX|   |XXX|   |XXX|   |XXX|   |
---------------------------------
|   |XXX|   |XXX|   |XXX|   |XXX|
|   |XXX|   |XXX|   |XXX|   |XXX|
|   |XXX|   |XXX|   |XXX|   |XXX|
---------------------------------
|XXX|   |XXX|   |XXX|   |XXX|   |
|XXX|   |XXX|   |XXX|   |XXX|   |
|XXX|   |XXX|   |XXX|   |XXX|   |
---------------------------------
|   |XXX|   |XXX|   |XXX|   |XXX|
|   |XXX|   |XXX|   |XXX|   |XXX|
|   |XXX|   |XXX|   |XXX|   |XXX|
---------------------------------
|XXX|   |XXX|   |XXX|   |XXX|   |
|XXX|   |XXX|   |XXX|   |XXX|   |
|XXX|   |XXX|   |XXX|   |XXX|   |
---------------------------------
|   |XXX|   |XXX|   |XXX|   |XXX|
|   |XXX|   |XXX|   |XXX|   |XXX|
|   |XXX|   |XXX|   |XXX|   |XXX|
---------------------------------
|XXX|   |XXX|   |XXX|   |XXX|   |
|XXX|   |XXX|   |XXX|   |XXX|   |
|XXX|   |XXX|   |XXX|   |XXX|   |
---------------------------------
```

C++

```cpp
#include <iostream>

void printRow(bool beginWithEmpty, char symbol, int size, char verticalLine)
{
    for (int j = 0; j<4; j++)
    {
        char first = ' ';
    char second = symbol;
    if (!beginWithEmpty)
    {
        first = symbol;
            second = ' ';
    }
        std::cout << std::string(size, first) << verticalLine
<<  std::string(size, second) << verticalLine;
    }
}

int main()
{
    char symbol;
    int size;
    char horizontalLine = '-';
    char verticalLine = '|';

    std::cin >> symbol;

    std::cin >> size;

    std::cout << std::string(((size+1)*8)+1, horizontalLine) << std::endl;

    // Repeat per row (8)
    for (int i = 0; i<8; i++)
    {
        for (int j = 0; j<size; j++)
    {
        std::cout << verticalLine;
        if (i % 2 == 0)
        {
            printRow(true, symbol, size,verticalLine);
        }
        else
        {
            printRow(false, symbol, size,verticalLine);
```

```
        }
        std::cout << std::endl;
    }
        std::cout << std::string(((size+1)*8)+1, horizontalLine) << std::endl;
    }
}
```

23 Rule 90

11 points

Introduction

Cellular automata (CA) are mathematical models for systems in which simple components (known as cells) act together following certain rules to produce complex behaviour. They can be used to model biological processes, simulate chemical reactions or study physical phenomena.

The simple one-dimensional CA consists of a single row of cells, where each cell can be in one of two possible states (0 or 1), plus a set of rules for evolving the whole CA state. It can be graphically represented as a sequence of numbers, where each number represents a cell value.

To evolve the current state of the CA, the rules act over the cell neighbourhood, that is, the cells placed at the left and right of a given cell. Rule 90 states that each cell's new value is the *exclusive or* of the two neighbouring (left and right) cell values. So, the next state of this particular CA follows this transition table:

Current pattern	000	001	010	011	100	101	110	111
New state for center cell	0	1	0	1	1	0	1	0

It is known as rule 90 because concatenating the new states for center cell of this table results into the binary number 01011010 that equals to decimal number 90.

To make things funnier an exception has been introduced to this rule. For the left-most and right-most cells, just copy the value of the next or previous cell respectively. That is, given the CA state 00110, the next state will be 01111.

The size of our CA is 64 cells and when representing the state 0s must be replaced by "-" and 1s replaced by "*".

Just one final word about complexity. Compare the output of the examples and notice how example 1 ends up forming Sierpinski fractal while example 2 looks like a random form. The output pattern depends critically on our initial conditions. Regular initial conditions provide regular output, but random initial conditions create somehow chaotic output. Hello Complexity World!

Can you write a program that evolves a given cellular automata in a row of a certain number of steps?

Input

The input will be a pair of lines.

The first line contains the original state of the cellular automata represented by 64 digits.

The second line represents the number of steps the cellular automata will execute reporting its output.

Output

A printed line representing the state of the cellular automata for each of the steps executed.

Example 1

Input

```
0000000000000000000000000000001000000000000000000000000000000000
32
```

Output

```
-------------------------------*-------------------------------
------------------------------*-*------------------------------
-----------------------------*---*-----------------------------
----------------------------*-*-*-*----------------------------
---------------------------*-------*---------------------------
--------------------------*-*-----*-*--------------------------
-------------------------*---*---*---*-------------------------
------------------------*-*-*-*-*-*-*-*------------------------
----------------------*--------------*----------------------
---------------------*-*------------*-*---------------------
--------------------*---*----------*---*--------------------
-------------------*-*-*-*--------*-*-*-*-------------------
------------------*-------*-------*-------*------------------
-----------------*-*-----*-*-----*-*-----*-*-----------------
----------------*---*---*---*---*---*---*---*----------------
---------------*-*-*-*-*-*-*-*-*-*-*-*-*-*-*---------------
--------------*-------------------------------*--------------
-------------*-*-----------------------------*-*-------------
------------*---*---------------------------*---*------------
-----------*-*-*-*-----------------------*-*-*-*-----------
----------*-------*---------------------*-------*----------
---------*-*-----*-*-------------------*-*-----*-*---------
--------*---*---*---*-----------------*---*---*---*--------
-------*-*-*-*-*-*-*-*---------------*-*-*-*-*-*-*-*-------
------*--------------*--------------*--------------*-------
```

```
------*-*------------*-*------------*-*------------*-*-------
-----*---*----------*---*----------*---*----------*---*------
----*-*-*-*--------*-*-*-*--------*-*-*-*--------*-*-*-*-----
---*-------*-------*-------*-------*-------*-------*-------*----
--*-*-----*-*-----*-*-----*-*-----*-*-----*-*-----*-*-----*-*---
-*---*---*---*---*---*---*---*---*---*---*---*---*---*---*---*--
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
```

Example 2

Input

```
10101111100001111111111111101111111111101111111111111101110110101
32
```

Output

```
*-*-*****---*************-**********-************-***-**-*-*
----*---**--**-----------*-*--------*-*----------*-*-*-**----
---*-*-*******----------*---*-------*---*---------*------***---
--*----*------**--------*-*-*-*-----*-*-*-*-------*-*----**-**--
-*-*--*-*----****------*-------*---*-------*-----*---*--***-***-
*---**---*--**--**----*-*-----*-*-*-*-----*-*---*-*-*-***-*-*-**
-*-****-*-*********--*---*---*-------*---*---*-*------*-*-----**
*--*--*---*-------***-*-*-*-*-*-----*-*-*-*-*---*----*---*---***
-**-**-*-*-*-----**-*----------*---*--------*-*-*--*-*-*-*-**-*
***-**------*---***--*--------*-*-*-*-------*-----**--------**--
*-*-***----*-*-**-***-*------*-------*-----*-*---****------****-
----*-**--*----**-*-*--*----*-*-----*-*---*---*-**--**----**--**
---*--****-*--***----**-*--*---*---*---*-*-*-*--*******--*******
--*-***--*--***-**--***--**-*-*-*-*-*-*-------***-----****-----*
-*--*-***-***-*-*****-*****-----------*-----**-**---**--**---*-
*-**--*-*-*-*---*---*-*---**----------*-*---***-***-********-*-*
--****-------*-*-*-*---*-****--------*---*-**-*-*-*-*------*----
-**--**-----*-------*-*--*--**------*-*-*--**--------*----*-*---
********---*-*-----*---**-*****----*-----*****------*-*--*---*--
*------**-*---*---*-*-***-*---**--*-*---**---**----*---**-*-*-*-
-*----***--*-*-*-*----*-*--*-******---*-****-****--*-*-***------*
*-*--**-***-------*--*---**--*---**-*--*--*-*--***----*-**----*-
```

```
---****-*-**-----*-**-*-*****-*-***--**-**---***-**--*--***--*-*
--**--*---***---*--**---*---*---*-*****-***-**-*-****-***-***---
-*****-*-**-**-*-*****-*-*-*-*--*---*-*-*-**---*--*-*-*-*-**--
**---*---**-**---*---*----------**-*-*-----***-*-**--------***-
***-*-*-***-***-*-*-*-*--------***----*----**-*---***------**-**
*-*-----*-*-*-*--------*------**-**--*-*--***--*-**-**----***-**
---*---*-------*------*-*----***-****---***-***--**-***--**-*-**
--*-*-*-*-----*-*----*---*--**-*-*--**-**-*-*-*****-*-******--**
-*-------*---*---*--*-*-*-****----****-**-----*---*---*---**-***
*-*-----*-*-*-*-*-**------*--**--**--*-******--*-*-*-*-*-***-*-*
```

Python

```python
################################
# Cellular automata function #
################################

def CellularAutomata(initialState, numSteps):

   # build cellular automata from initialState
   initialState_map = map(int, initialState)
   ca = list(initialState_map)

   # new cellular values
   ca_new = ca[:]

   # dictionary mapping the cell value to characters '-' and '*'
   translateDic = {0:'-', 1:'*'}

   # print out initial step
   print(''.join( [translateDic[e] for e in ca_new]))
   # peform next steps
   step = 1
   while(step < numSteps):
      ca_new = []
      # loop through 0 to 63 and store the current cell status in ca_new list
      for i in range(0,64):
         # inside cells - check the neighbor cell state
         if i > 0 and i < 63:
            if ca[i-1] == ca[i+1]:
               ca_new.append(0)
            else:
               ca_new.append(1)
```

```
        # left-most cell : check the second cell
        elif(i == 0):
           if ca[1] == 1:
              ca_new.append(1)
           else:
              ca_new.append(0)

        # right-most cell : check the second to the last cell
        elif(i == 63):
           if ca[62] == 1:
              ca_new.append(1)
           else:
              ca_new.append(0)

     # print out current cell state
     print(''.join( [translateDic[e] for e in ca_new]))

     # update cell list
     ca = ca_new[:]

     # increase step count
     step += 1

################
# Main program #
################

# read initial state represented by 64 consecutive cells in a single line
initialState = input()

# read number of steps to execute
numSteps = int(input())

# run the cellular automata with input paramaters
CellularAutomata(initialState, numSteps)
```

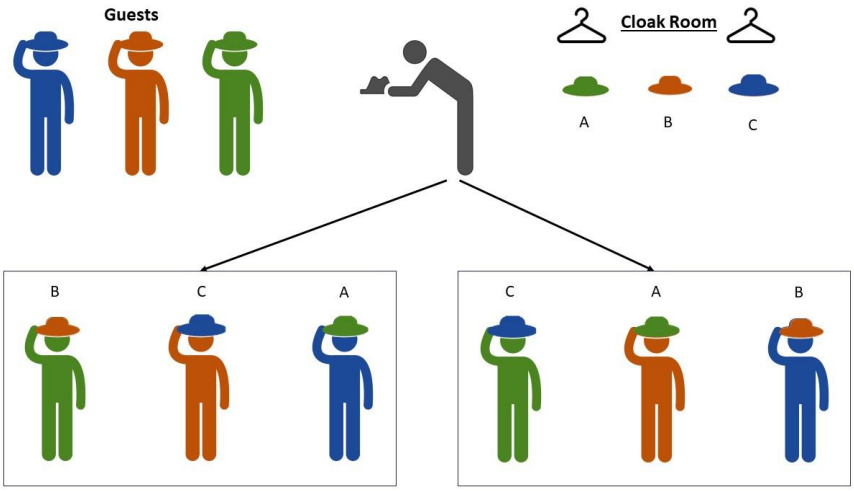
24 Mixing Hats

12 points

Introduction

At a fancy party, guests arrive and check their hats in the cloak room. Starting with a single capital letter "A" and following alphabetical order, the cloak room attendant gives a ticket with that letter to the guest and attaches a label with the same letter to the hat. Once the party starts, the attendant wants to have some fun, so he starts exchanging the hat labels. By the time guests are ready to leave, none of them has their original hat back.

In case of having three guests at the party, when the hats arrive to the cloak room, the hats will receive labels A, B and C. To make it funnier, the attendant makes sure that no guest has their original hat. So he has 2 different options when mixing things up, he can mix them in a B, C, A manner or a C, A, B manner. If he did it in any other way, like C, B, A, guest B would have his original hat, which the attendant wants to avoid.



Can you write a program to calculate all the possible combinations of labels that ensures that nobody gets their hat back?

Input

The input is the number of guests attending the party: a number between 2 and 7, both included.

Output

The output is the list of possible hat labels combinations ordered alphabetically.

Example

Input

3

Output

BCA

CAB

Python

```
def permutation(lst):
    if len(lst) == 0:
        return []
    if len(lst) == 1:
        return [lst]

    l = [] # empty list that will store current permutation

    for i in range(len(lst)):
        m = lst[i]

        # Extract lst[i] or m from the list. remainderLst is remaining list
        remainderLst = lst[:i] + lst[i+1:]

        # Generating all permutations where m is first element
        for p in permutation(remainderLst):
           l.append([m] + p)
    return l

def anyMatching(a, b):
  res = False

  for i in range(0,len(a)):
    if a[i] == b[i]:
      return True
  return res

# 2 <= number <= 7
number = int(input())

# list of identifiers for hats
hats = []

# Generate the hats sequence received in order
for i in range(0,number):
```

```
    hats = hats + [chr(i+65)]

#print(hats)

# All the permutations without repetitions are generated
# then the permutations that have a match against original
# order are discarded.
for p in permutation(hats):
    if not anyMatching(hats, p):
      print(''.join(map(str, p)))
```

25 Gaussian Blur

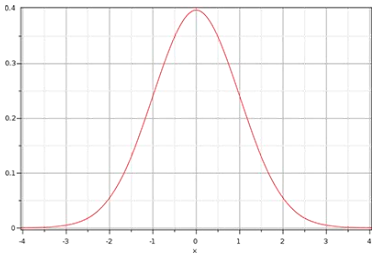
13 points

Introduction

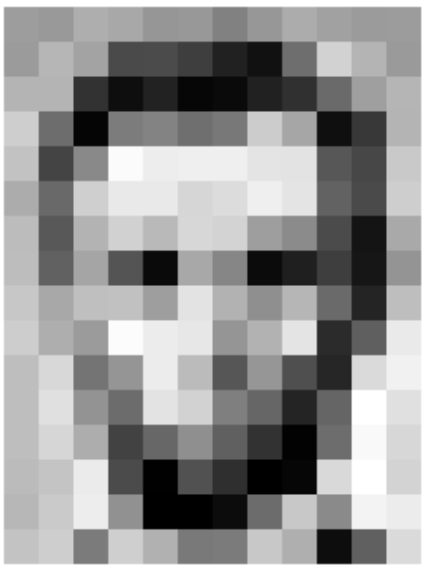
For sure, you've seen many times in photos, the typical effects where the out-of-focus parts are softened or even the whole image seems like viewed through a translucent screen.

Original image | After applying Gaussian Blur

Some of these effects are accomplished using a Gaussian Blur which is the application of a mathematical function to an image in order to blur it. A Gaussian curve or Gaussian function has the following shape:



As you may know an image is represented as a matrix of pixel values.



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

So let's consider a simple way to apply a Gaussian Blur to a given image. To make things easier assume that only some points along the Gaussian curve are considered. These points, also called "weights", show how much softness we want at that place. Higher numbers mean more softness. The selected points are collected in a one-dimensional weight table like this:

```
[0.06136 0.24477 0.38774 0.24477 0.06136]
```

This is also called Gauss kernel window. Then, placing this window centered on a certain pixel, we would sample every other pixel in the window using the corresponding weight, and the sum of all them would be the blurred value of the pixel in the center:

Image size = 7 x 7

125	125	75	125	255	75	45
100	100	75	50	40	25	0
100	150	175	190	225	245	255
255	255	250	245	255	255	255
250	245	240	235	225	220	215
200	150	200	100	150	100	50
50	25	20	0	25	25	0

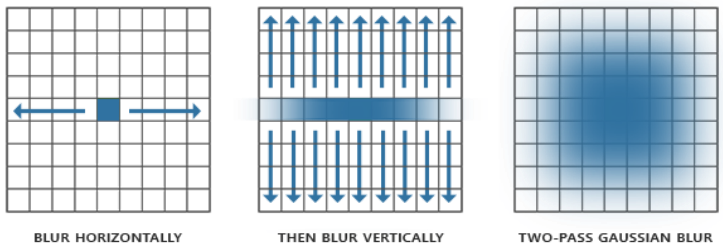
Gauss kernel window

0.06136	0.24477	0.38774	0.24477	0.06136

Original Pixel Value = 75

Blurred Pixel Value = 100 x 0.06136 + 100 x 0.24477 + 75 x 0.38774 + 50 x 0.24477 + 40 x 0.06136 = 74.386

The operation to be done consists of a first horizontal blur on the image, followed by another vertical blur on the resulting image:



So, for every pixel we'll put the one-dimensional window horizontally around it and compute the blurred value using the weights in the window. Then, in the resulting image, for each pixel we'll put the one-dimensional window vertically around it and compute the blurred value using the weights. We can repeat this process any number of times if we want a more blurred result. Corner pixels and pixels situated along the sides of the image are missing some neighboring pixels. When placing the window around these pixels, assume that the value of missing neighbors is '0'.

Write a program that applies the Gaussian Blur a specified number of times to a given image using previous one-dimensional window weight table.

Input

First line, a positive number indicating how many times we want to blur the image. Then, a line with the size of the image separated by a white space (rows columns).

And finally, the image pixel values.

Output

The output is the pixel values of the blurred image, rounded to the nearest integer.

Example

Input

```
4
5 5
73 54 80 45 73
37 63 54 54 18
97 37 80 80 97
27 45 97 18 18
45 45 18 45 97
```

Output

```
10 16 18 16 10
16 25 28 25 16
18 28 32 28 18
15 24 27 24 15
9 15 17 15 10
```

Python

```
def main():
    WEIGHTS = [0.06136, 0.24477, 0.38774, 0.24477, 0.06136]

    #INPUT
    n_times_blur = int(input())

    n_rows, n_columns = input().split()
    n_rows, n_columns = int(n_rows), int(n_columns)

    image = []
    for i in range(n_rows):
        row = list(map(int, input().split()))
        image.append(row)

    #FUNCTIONS
    def apply_weights(matrix, weights):
        n_weights = len(weights)
        n_zeros = n_weights//2
        zeros = [0]*n_zeros

        new_matrix = []
        for row in matrix:
            row_with_zeros = zeros + row + zeros
            new_row = []
            for i in range(len(matrix[0])):
                new_value = sum([row_with_zeros[i+k] * weights[k] for k in
range(n_weights)])
                new_row.append(new_value)
            new_matrix.append(new_row)
        return new_matrix

    def transpose(matrix):
        result = []
        for i in range(len(matrix[0])):
            new_row = []
            for row in matrix:
                new_row.append(row[i])
            result.append(new_row)
        return result

    def round_matrix(matrix):
        result = []
        for row in matrix:
            new_row = [round(x) for x in row]
            result.append(new_row)
```

```
        return result

    def apply_gaussian_blur(image, n_times):
        new_image = image.copy()
        for i in range(n_times):
            new_image = apply_weights(new_image, WEIGHTS)
            new_image = transpose(new_image)
            new_image = apply_weights(new_image, WEIGHTS)
            new_image = transpose(new_image)
        new_image = round_matrix(new_image)
        return new_image

    #OUTPUT
    blurred_image = apply_gaussian_blur(image, n_times_blur)
    for row in blurred_image:
        print(*row)

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

26 Sudoku Solver

15 points

Introduction

As an aspiring programmer and sudoku fan, you have decided to become the number one sudoku completionist. You want to create a program to solve all the sudokus of the newspaper you buy every day.

Before starting, you write down the rules needed to complete a sudoku:

- You can put a number between 1-9 in each slot.

- No number can be repeated in any given row.

- No number can be repeated in any given column.

- No number can be repeated in any given 3x3 "small" square.

Now that you are ready, you will need to write a program that can solve any (solvable) sudoku to become the best.

IMPORTANT: In case of a sudoku that have multiple solutions, any valid solution is accepted.

Input

A sudoku table with the initial numbers needed to complete it. The missing numbers will be represented with a blank space, even if they are at the end of the line.

Output

The completed sudoku table. In case of multiple sudoku solutions, only one completed table.

Example 1

Input

```
 1 |2  |3 4
2 5| 6 | 7
  8| 9 |
---+---+---
3  |12 |  8
6  |8 7|  2
1  | 34|  5
---+---+---
   | 8 |9
 2 | 4 |5 6
7 4|  5| 8
```

Output

```
916|278|354
235|461|879
478|593|621
---+---+---
357|126|498
649|857|132
182|934|765
---+---+---
561|382|947
823|749|516
794|615|283
```

Example 2

Input

```
  3|685|47
5  |  4|162
7  | 19|538
---+---+---
   |9  |
39 |  1| 2
 65|  3|  1
---+---+---
4  | 6 |9 3
9  |342| 87
   |1  |  4
```

Output

```
123|685|479
589|734|162
746|219|538
---+---+---
817|926|345
394|851|726
265|473|891
---+---+---
472|568|913
951|342|687
638|197|254
```

Python

```python
def createInitialSudoku():
    initialSudoku = [
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]
    ]
    posRow = 0
    for i in range(11):
        rowRead = input()
        if rowRead[0] != '-':
            posCol = 0

            for j in range(11):
                if rowRead[j] != '|':
                    if rowRead[j] != ' ':
                        initialSudoku[posRow][posCol] = int(rowRead[j])

                    posCol += 1


            posRow += 1

    return initialSudoku

def isLegal(sudokuMatrix, row, column, num):

    #We check if the number is already in this row
    for check in range(9):
        if sudokuMatrix[row][check] == num:
            return False

    #We check if the number is already in this column
    for check in range(9):
        if sudokuMatrix[check][column] == num:
            return False

    #We check if the number is already in the box
    boxStartRow = row - row % 3
    boxStartCol = column - column % 3
```

```
    for i in range(boxStartRow, boxStartRow + 3):
        for j in range(boxStartCol, boxStartCol + 3):
            if sudokuMatrix[i][j] == num:
                return False

    return True

def solveSudoku(sudokuMatrix, row, column):

    if row == 8 and column == 9:
        return True
    if column == 9:
        row += 1
        column = 0
    #We check if the curent tile does contain a number
    if sudokuMatrix[row][column] != 0:
        return solveSudoku(sudokuMatrix, row, column+1)

    #We try all possible numbers
    for num in range(1, 10):
        if isLegal(sudokuMatrix, row, column, num):
            #If the number is legal, we put into the sudoku
            sudokuMatrix[row][column] = num

            if solveSudoku(sudokuMatrix, row, column+1):
                return True
            sudokuMatrix[row][column] = 0

    return False

def printSudoku(sudokuMatrix):
    for i in range(9):
        if i == 3 or i == 6:
            print("---+---+---")
        for j in range(9):
            if j == 3 or j == 6:
                print("|", end="")
            print(sudokuMatrix[i][j], end="")
        print("")

############################ MAIN FUNCTION ##################################

sudokuMatrix = createInitialSudoku()
if solveSudoku(sudokuMatrix, 0, 0):
    printSudoku(sudokuMatrix)
else:
    print("There is no solution")
```

27 Finding Achilles

17 points

Introduction

For sure you known of Achilles, one of the greatest Greek warriors in the Trojan war. Despite his mighty power he had a weakness at his heel. Did you know that there are also numbers named after him? Just like Achilles, these numbers are powerful but imperfect.

Usually a positive integer is a powerful number if, when doing its factorization, every prime factor appears at least squared in the factorization. However, Achilles numbers have a weakness: that is the only way they can be represented. They cannot be represented as m^k, where m and k are positive integers greater than 1. If a powerful number has both representations, it will *not* be an Achilles number.

Let's see some examples:

- 50 is not a powerful number because one of its prime factors is not squared: $2 \cdot 5^2$

- 784 is a powerful number as its prime factors show $2^4 \cdot 7^2$, but it is not an Achilles number because it can be represented as a power in the form of 28^2.

- 200 is a powerful number as its prime factors are $2^3 \cdot 5^2$, and it is an Achilles number since it's not possible to be represented as m^k.

Can you code a program to detect whether a given positive number is or is not an Achilles number?

Example 1

Input

```
2
```

Output

```
2 is NOT an Achilles number
```

Example 2

Input

```
200
```

Output

```
200 is an Achilles number
```

C++

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

using namespace std;


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

    bool isAchilles = true;

    vector<int> factors;

    if(number == 1) {
        isAchilles = false;
    }

    int aux = number;
    while(aux > 1) {
        for(int i = 2; i <= aux; i++) {
            if(aux % i == 0) {
                aux = aux / i;
                factors.push_back(i);
                break;
            }
        }
    }

    int size = factors.size();
    int factorAux;
    bool found;
    for(int j = 0; j < size; j++){
        found = false;
        factorAux = factors[j];
        for(int k = 0; k < size; k++) {
            if(j != k) {
                if(factorAux == factors[k]) {
                    found = true;
                }
            }
        }
        if(!found) {
            isAchilles = false;
```

```
        }
    }

    double auxResult;
    for(int l = 1; l < number; l++){
        for(int m = 1; m < number; m++) {
            auxResult = pow(l,m);
            if((double)number == auxResult) {
                isAchilles = false;
            }
        }
        if(!isAchilles) {break;}
    }



    if(isAchilles) {
        cout << number << " is an Achilles number";
    }
    else{
        cout << number << " is NOT an Achilles number";
    }
    return 0;
}
```

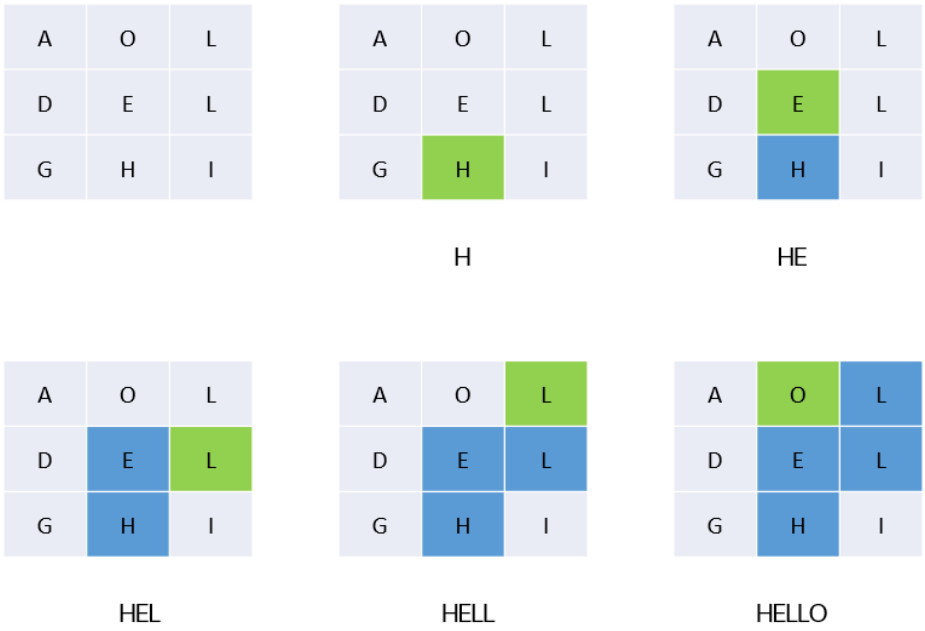
28 Boggle Search

19 points

Introduction

Boggle is a word game where players try to find as many words as possible by connecting adjacent letters on a rectangular lettered grid. It's a fun and challenging game that tests your vocabulary and pattern recognition skills.

The following sequence shows the steps to find the word HELLO on a board.



Please notice that given a letter, the valid adjacent letters to continue composing a word are the letters that can be found up, down, left and right to the last letter selected. A letter cannot be visited twice. In case of words with a letter repeated consecutively, that letter must appear repeated also in the board. To report the sequence to find a word follow this coordinate system:

	0	1	2
0	A	O	L
1	D	E	L
2	G	H	I

So, the coordinates sequence for HELLO is (2,1), (1,1), (1,2), (0,2) and (0,1).

To make things easier a word will appear only once in a boggle board. Can you write a program that finds out a word and provides the corresponding sequence of coordinates?

Input

First line contains the word to find.
Second line is a positive integer representing the number of rows of the boggle board.
Finally, the content of every row with the letters contained in it.

Output

The output is the sequence of coordinates to find the word. Otherwise, the program must print out that the word has not been found.

Example 1

Input

```
HELLO
3
A O L
D E L
G H I
```

Output

```
The word HELLO can be composed following the path: (2,1), (1,1), (1,2), (0,2) and (0,1).
```

Example 2

Input

```
HELP
5
A O L S
D E L O
G H I P
X D F Q
H E L X
```

Output

```
The word HELP has not been found.
```

Python

```python
# To check if a given [i,j] is within matrix range
def isValidPos(i,j):
    if (i >= 0 and i < rows and j >= 0 and j < columns):
        return True
    else:
        return False

# To find out the list of valid neighbours for a given matrix position
def possibleNeighbours(i,j):
    # Candidates are  up, left, right and down neighbours
    neighbours = [[i-1,j],[i,j-1],[i,j+1],[i+1,j]]
    toDelete = []

    # Check whether are within matrix limits and are nor part of the solution
    for x in neighbours:
        if (not isValidPos(x[0],x[1])) or x in solution:
            toDelete.append(x)

    # Remove invalid positions
    for x in toDelete:
        neighbours.remove(x)

    return neighbours

def wordIsCompleted(word, i, j, solution):
    #print(word, str(rows), str(columns), str(i),str(j))
    if word == "":
        #print("Word found!")
        return True
    else:
        neighbours = possibleNeighbours(i, j)

        # Check if there is a possible solution
        #print("Possible neighbours: " + str(neighbours))
        for i in neighbours:
            if matrix[i[0]][i[1]] == word[0]:
                #print("FOUND at " + str(i[0]) + "," + str(i[1]))
                solution.append([i[0],i[1]])
                #print("ongoing solution is : " + str(solution))
                if (wordIsCompleted(word[1:], i[0], i[1], solution) == True):
                    return True
                else:
                    solution.pop()
        return False
```

```
## Main program

matrix = []
solution = []

# Read word to search in the board
word = input()

# Read board from standard input
rows = int(input())

for i in range(rows):
    currentRow = input().split()
    matrix.append(currentRow)

# Find out number of columns
columns = len(matrix[0])

wordFound = False

# Traverse the whole matrix looking for first letter
for i in range(rows):
    for j in range(columns):
        if wordFound == False:
            #print("Checking: " + str(i) + "," + str(j))
            if matrix[i][j] == word[0]:
                #print("FOUND at " + str(i) + "," + str(j))
                solution.append([i,j])
                if wordIsCompleted(word[1:], i, j, solution) == True:
                    wordFound = True
                else:
                    solution = []

if solution == []:
    print ("The word " + word + " has not been found.")
else:
    res = "The word " + word + " can be composed following the path: "
    for i in solution:
        if i == solution[-1]:
            # Remove last comma and add last coordinates.
            res = res[:-2]
            res += " and (" + str(i[0]) + "," + str(i[1]) + ")."
        else:
            # Concatenate coordinates
            res += "(" + str(i[0]) + "," + str(i[1]) + "), "
    print(res)
```

29 Battleship Board Sketcher

20 points

Introduction

Brugilda is developing a smartphone application to play the well-known *Battleship* game (aka *Hundir la flota*). She is starting with a very basic user interface to sketch the board and the result of several shots.

Battleship is a strategy-type guessing game of two players. Each player has a board with rows and columns and places its own fleet of warships secretly. The objective is to destroy the opposing player fleet by calling "shots", trying to guess the position of the different warships.

The fleet consists in different warships of several sizes:

Warship Name	Size
Carrier	5
Battleship	4
Destroyer	3
Submarine	3
Patrol Boat	2

The size of each ship indicates the number of cells it occupies when placed on the board. Remember that players must leave a safety perimeter of 1 cell around each warship, except in case the warship is placed at any corner or along the board's edges.

The board must have 10 rows and 10 columns. Rows named from 1 to 10 and columns from A to J.

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

Check the following example picture. Several boats have been added to the board:

- Carrier warship horizontally at location "C3"
- Patrol Boat vertically at "F6"
- Destroyer horizontally at A8
- Submarine located at H10, also horizontally

Notice how the 1 cell safety perimeter is respected in all cases.

	A	B	C	D	E	F	G	H	I	J
1										
2										
3			C	C	C	C	C			
4										
5										
6						P				
7						P				
8	D	D	D							
9										
10								S	S	S

Calling a shot requires to specify the column (A, B, C, ...) and the row (1,2,3,...). If a warship receives as many shots as its size, it is considered destroyed, so all the cells around the ship must be immediately marked.

Following the example, we draw many shots on the current board. The Carrier received a shot at "D3", the Destroyer stays intact, the Submarine also received one shot at H10 and the Patrol Boat received two shots ("F6" and "F7") being completely destroyed.

	A	B	C	D	E	F	G	H	I	J
1			X		X					
2		X								
3	X		C	X	C	C	C			
4		X								
5					X	X	X			
6	X				X	X	X			
7		X			X	X	X			
8	D	D	D		X	X	X			
9									X	
10								X	S	S

Code Requirements

Would you like to help Brugilda writing her prototype? The code must accomplish the following requirements:

- Input the number of ships.
- Input the position (cell column and row) and orientation of each boat (horizontal and vertical).
- Input several shot calls, each one defined as a cell (column and row).
- With input data, draw a 10x10 board with ships properly placed, shots received and a player health status.
- After all input data is entered the prototype must ensure that a valid board will be drawn:
  - Ships do not exceed board dimensions.
  - There are no ships overlapping. That is, a board cell can only contain one type of warship.
  - Once ships are placed, they respect the 1 cell safe perimeter around each ship.

Input

The input of this problem consists in 3 lines:

- First line is the number of ships. For example: `Number of ships:4`
- Subsequent lines correspond to the description and position of each boat in the form of `Boat Name,Cell,Orientation`

```
Carrier,C3,H
Destroyer,A8,H
Patrol Boat,F6,V
Submarine,H10,H
```

Third line corresponds to the shots, separated by comas: `Shots:D3,F6,F7`

It is not mandatory to enter any shot for the prototype to draw the board.

Output

The game board with the ships in its positions and the shots marked as "X". Each boat is identified as follows:

- Carrier: `C`
- Battleship: `B`
- Destroyer: `D`
- Submarine: `S`
- Patrol Boat: `P`

The board must show named columns and rows using `|` as column separator. The player health status will use symbols "*" and "-" to indicate remaining 'life' of each boat. See the example:

```
  |A|B|C|D|E|F|G|H|I|J|
1 | | |X| |X| | | | | |
2 | |X| | | | | | | | |
3 |X| |C|X|C|C|C| | | |
4 | |X| | | | | | | | |
5 | | | | |X|X|X| | | |
6 |X| | | |X|X|X| | | |
7 | |X| | |X|X|X| | | |
8 |D|D|D| |X|X|X| | | |
9 | | | | | | | | |X| |
10| | | | | | | |X|S|S|

Your Board Status:
Carrier at C3: ****-
Destroyer at A8: ***
Patrol Boat at F6: --
Submarine at H10: **-
```

Finally, as the prototype must check some board validity parameters, different output messages will be showed according to the error found at input data:

- If any input ship, when placed at the board, exceeds board dimensions: `ERROR: At least one warship's location exceeds board's dimensions`
- If any input ship, when placed at the board, causes overlap with another ship: `ERROR: There is overlap between at least two ships`

- After placing warships, if the 1 cell safety perimeter is not satisfied: `ERROR: Safety perimeter is not respected`
- If any shot goes outside of the board: `ERROR: There is at least one shot going outside the board`

Example 1

Input

```
Number of ships:4
Carrier,C3,H
Destroyer,A8,H
Patrol Boat,F6,V
Submarine,H10,H
Shots:C1,E1,B2,A3,D3,B4,A6,E6,F6,G6,F7,B7,H10,I9
```

Output

```
  |A|B|C|D|E|F|G|H|I|J|
1 | | |X| |X| | | | | |
2 | |X| | | | | | | | |
3 |X| |C|X|C|C|C| | | |
4 | |X| | | | | | | | |
5 | | | | |X|X|X| | | |
6 |X| | | |X|X|X| | | |
7 | |X| | |X|X|X| | | |
8 |D|D|D| |X|X|X| | | |
9 | | | | | | | | |X| |
10| | | | | | | |X|S|S|

Your Board Status:
Carrier at C3: ****-
Destroyer at A8: ***
Patrol Boat at F6: --
Submarine at H10: **-
```

Example 2

Input

```
Number of ships:1
Carrier,H10,V
Shots:A7
```

Output

```
ERROR: At least one warship's location exceeds board's dimensions
```

Example 3

Input

```
Number of ships:3
Carrier,C4,H
Battleship,D3,V
Submarine,F5,H
Shots:D7
```

Output

```
ERROR: There is overlap between at least two ships
```

Example 4

Input

```
Number of ships:1
Carrier,D3,H
Shots:L17
```

Output

```
ERROR: There is at least one shot going outside the board
```

Python

```python
import sys

####################
## CAPTURE INPUT ##
#
n_ships = int(input().split(':')[1]) # Number of ships

input_ships = []
for k in range(n_ships):
   line = input()
   input_ships.append(line.split(','))

# Input shots:
shots = input().split(':')[1].split(',')

# Uncomment following line for testing purposes
# input_ships = [["Destroyer","A1","V"],["Carrier","C2","H"], ["Patrol Boat",
"E4", "H"]]

############################
## DEFINE SCRIPT PARAMETERS
#
# Define index for columns and rows
col_index = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]

# Define Ships
def_ships = {'Carrier':5, 'Battleship':4, 'Destroyer':3, 'Submarine':3, 'Patrol
Boat':2}

# Define Heart Unicode Symbols
full_heart = "*"
empty_heart = "-"

####################
## PROCESS BOARD ##
#
# Define empty 10x10 board.
n_rows = 10
n_cols = 10

board = []
for k in range(n_rows):
    board.append([" "] * n_cols)

# Fill Board with ships and modify input_ships to handle full ship
```

```
# location and status. Use try:catch: to handle ships exceeding board size.
try:
    for ship in input_ships:

        ship_type = ship[0] # Ship type (Carrier, Destroyer, etc)
        ship_col = ord(ship[1][0]) - ord("A") # Get index of ship column
        ship_row = int(ship[1][1:len(ship[1])]) - 1 # Get index of ship row
        ship_dir = ship[2] # Get if ship is in H or V direction
        ship_size = def_ships[ship_type]

        # Remove ship direction. Will fill with all ship cells
        ship.pop(2)

        if ship_dir == 'H':
            for col in range(ship_col,ship_col+ship_size):
                if board[ship_row][col] == " ":
                    # Cell is empty, fill it!
                    board[ship_row][col] = ship_type[0]
                else:
                    # Cell is not empty! Error, overlap!
                    print("ERROR: There is overlap between at least two ships")
                    sys.exit()

                # Add cell location to input_ships
                loc = '%s%s' % (col_index[col], ship[1][1:len(ship[1])])
                ship.append(loc)

        elif ship_dir == 'V':
            for row in range(ship_row,ship_row+ship_size):
                if board[row][ship_col] == " ":
                    # Cell is empty, fill it!
                    board[row][ship_col] = ship_type[0]
                else:
                    # Cell is not empty! Error, overlap!
                    print("ERROR: There is overlap between at least two ships")
                    sys.exit()

                # Add cell location to input_ships
                loc = '%s%s' % (ship[1][0], str(row+1))
                ship.append(loc)

        # Add Ship Size to input_ships to track ship live status
        ship.append(ship_size)

        # Remove duplicated location of first cell
        ship.pop(1)
```

```
except IndexError:
    # See sample-2.in and sample-2.out
    print("ERROR: At least one warship's location exceeds board's dimensions")
    sys.exit()

# CHECK CELL SAFETY PERIMETER
for row in range(n_rows):
    for col in range(n_cols):
        warship = board[row][col]

        if warship == " ":
            pass; # Is a water cell, do nothing

        else:
            # Define search ROI to test safety perimeter
            row0 = row - 1 if (row - 1)>=0 else row
            row1 = row + 1 if (row + 1)<=n_rows-1 else row
            col0 = col - 1 if (col - 1)>=0 else col
            col1 = col + 1 if (col + 1)<=n_cols-1 else col

            # Get content of ROI
            roi_values = []
            for r in range(row0, row1+1):
                for c in range(col0, col1+1):
                    # value is the content of the cell
                    value = board[r][c]

                    # Check if water cell
                    if not value == " ":
                        # If not water, then check if differ from "cell"
                        if not value == warship:
                            print("ERROR: Safety perimeter is not respected")
                            sys.exit()
####################
## PROCESS SHOTS ##
#
# shots = ['A1','A2','E2','E4','F4']

# I use try:except: to handle empty shots case
try:
    for shot in shots:
        shot_col = ord(shot[0]) - ord("A")
        shot_row = int(shot[1:len(shot)]) - 1

        if (shot_col < 0 or shot_col > 9) or (shot_row < 0 or shot_row > 9):
            print("ERROR: There is at least one shot going outside the board")
            sys.exit()
```

```
        board[shot_row][shot_col] = "X"


        for ship in input_ships:
            if shot in ship:
                # Remove one live to ship status
                ship[-1] = ship[-1] - 1

                # Check if ship is sunken
                if ship[-1] == 0:
                    # Draw "X" in all cells around ship
                    for k in range(1,1+def_ships[ship[0]]):

                        ship_col = ord(ship[k][0]) - ord("A")
                        ship_row = int(ship[k][1:len(ship[k])]) - 1 # Get index
of ship row

                        for col in range(ship_col-1, ship_col+2):
                            for row in range(ship_row-1,
ship_row+2):
                                # Check that indices are positive
                                low_bound = (row >= 0) and (col >= 0)
                                up_bound = (row <= 9) and (col <= 9)

                                if low_bound and up_bound:
                                    board[row][col] = "X"
except IndexError:
    pass

############################
## PRINT BATTLESHIP BOARD ##
#
# Print Col Header
head = ''.join(["|%s" % (k) for k in col_index])
head = head + '|'
head = '  '+head

print(head)

# Print Board Rows
row_index = 1
cell_separator = "|"
for row in board:
    endChar = ''
    if len(str(row_index)) == 1:
        endChar = ' '
```

```
    else:
        endChar = ''

    # Print row index
    print(str(row_index), end=endChar)

    # Print each row
    for cell in row:
        print(cell_separator + cell, end="")

    print(cell_separator, end="\n")
    row_index = row_index + 1

##########################
## PROCESS BOARD STATUS ##
#
# Print your ships status
print("\nYour Board Status:")
for ship in input_ships:

    ship_live = full_heart * (ship[-1]) + empty_heart * (def_ships[ship[0]]-
ship[-1])
    ship_status = "%s at %s: %s" % (ship[0], ship[1], ship_live)
    print(ship_status)
```

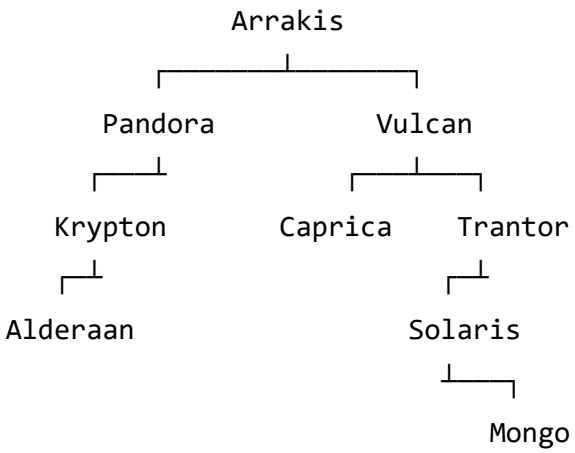
30 Treetronomical Challenge

25 points

Introduction

As an interstellar gardener you are tasked with measuring the cosmic trees of the galaxy. These unique trees are binary, wherein each tree node is connected at most to two child nodes: the left child and right child. In this tree a node represents a planet and the connections between planets are branches that stretch across light-years. These branches which connect planets are a type of wormhole through which you can travel instantly. Your spaceship can jump from planet to planet using these wormholes. Each jump consumes one unit of fuel. Your mission is to calculate the cosmic diameter of these wondrous trees, which is the longest distance between any two planets connected to the tree. Since your spaceship has limited fuel, you need to find the cosmic diameter in advance to know how much fuel is needed to explore the far reaches of a given cosmic tree of the galaxy.

Here is an example of a cosmic tree (the same that is used in Example 3):



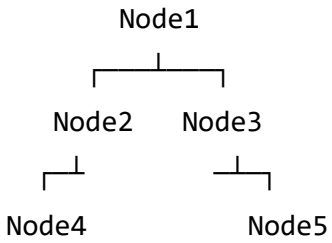
The root node for the tree is planet Arrakis. It has two child nodes: on the left the planet Pandora and on the right the planet Vulcan. Pandora is the root of a tree with just one planet at the left: Krypton. On the other hand, Vulcan has two child planet nodes: Caprica on the left and Trantor on the right. And so, it continues for the following planets on the tree.

The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

Can you write a program that calculates the cosmic diameter of these trees?

Input

A cosmic tree is represented using parentheses that follow the binary tree structure.



The notation "Node1(Node2,Node3)" represents a binary tree structure in a human-readable format. Here's how this notation can be interpreted: "Node1" represents the root node of the binary tree, "Node2" is the left child node and "Node3" is the right child node. In this notation, you can see that the tree structure is defined recursively. "Node2" and "Node3" can themselves be binary tree structures following the same notation, allowing for the representation of complex binary trees. So, this whole tree can be represented as Node1(Node2(Node4),Node3(,Node5)). Please notice that there are not any white space and that a single left node is noted as (Node4) while in case of a single right node a comma precedes the node (,Node5).

Output

The output is a positive integer with the cosmic diameter.

Example 1

Input

```
PlanetA(PlanetB(PlanetC))
```

Output

```
3
```

Example 2

Input

```
PlanetA(,PlanetB)
```

Output

```
2
```

Example 3

Input

```
Arrakis(Pandora(Krypton(Alderaan)),Vulcan(Caprica,Trantor(Solaris(,Mongo))))
```

Output

```
8
```

Python

```python
class TreeNode:
    def __init__(self, value):
        self.name = value
        self.left = None
        self.right = None

# Auxiliar function to draw a tree
def printTree(root, depth=0, prefix="Root: "):
    if root is not None:
        print(" " * (depth * 4) + prefix + root.name)
        if root.left is not None and root.right is not None:
            printTree(root.left, depth + 1, "L-- ")
            printTree(root.right, depth + 1, "R-- ")
        elif root.left is not None:
            printTree(root.left, depth + 1, "L-- ")
        elif root.right is not None:
            printTree(root.right, depth + 1, "R-- ")

# Function to find height of a tree
def height(root):
    if (root == None):
        return 0

    left = height(root.left)
    right = height(root.right)

    return max(left, right) + 1

# Computes the diameter of binary tree with given root.
def diameter(root):
    if root is None:
        return 0

    lheight = height(root.left)
    rheight = height(root.right)

    ldiameter = diameter(root.left)
    rdiameter = diameter(root.right)

    return max(lheight + rheight + 1, max(ldiameter, rdiameter))

# Parentheses are used to represent the binary tree structure.
# You start with an open parenthesis for the root, then represent
# the left subtree, followed by the right subtree. Close the
# parenthesis when you finish the subtree.
```

```
#
# Example:
#
#     1
#    / \
#   2   3
#  / \
# 4   5
#
# Using this approach, this tree would be represented as 1(2(4,5),3)

def buildTree(input):
    #print("tree to process: " + input)
    # Check whether input is empty
    if input == "":
        return None

    # First find node root
    index = input.find("(")
    if index != -1:
        rootName = input[:index]
    else:
        rootName = input
    node = TreeNode(rootName)

    #print("root: " + rootName)

    # Check whether are left or right subtrees
    if index == -1:
        return node

    # Now split left and right trees,
    # remove root and front, also remove rear bracket
    remaining = input[len(node.name)+1:-1]
    cntBrackets = 0
    pos = 0
    left=""
    right=""

    for i in range(len(remaining)):
        if remaining[i] == "(":
            cntBrackets += 1
        elif remaining[i] == ")":
            cntBrackets -= 1
        elif remaining[i] == ",":
            if cntBrackets == 0:
                pos = i
```

```
                break;

    if pos != 0:
        left = remaining[:pos]
    else:
        if remaining[0] != ",":
            left = remaining
    #print("left: " + left)
    node.left = buildTree(left)

    if remaining[pos] == ",":
        right = remaining[pos+1:]
        #print("right: " + right)
        node.right = buildTree(right)

    return node

data = input()
root = buildTree(data)
#printTree(root)
print(diameter(root))
```

31 Sustainable Batteries

30 points

Introduction

Welcome to the R&D laboratory! We discovered a new material that could open the door to more sustainable batteries because it has a very special property: it can be charged with electrical current and its molecules accumulate energy, expanding the material. Then, when the material is under a constant pressure along its surface, the molecules start moving, producing an electrical current when they collide with other molecules. However, since this new material is not completely perfect, the molecules may collide with "obstacles" due to the material's impurity.

We want to analyze the trajectory of the molecules placed at different positions. Could you help us by creating a simulator for the molecules' movement? The simulator must fulfill the following requirements:

- The simulation must generate a sequence of 2D grids, each grid representing the position of the molecules and the obstacles at the same time sample (the duration of the simulation is specified as part of the input).
- The molecules will be placed at different positions within a 2D grid, each one with its own initial direction, and all of them will move at the same time with the same speed.
- The obstacles are placed at arbitrary positions within a 2D grid and they do not move during the simulation.
- When a molecule collides with an obstacle or the edges of the 2D grid, the molecule bounces and its direction is inverted accordingly. Therefore, a molecule cannot be placed outside the 2D grid or in the same position than an obstacle.
- When a molecule collides with another molecule, the collision must be marked in the 2D grid. Therefore, different molecules may be placed in the same position of the 2D grid. After collision the molecules won't change their trajectory.

Important! Obstacles may be next to other obstacles (in adjacent cells), but they will never form gaps of only 1 cell between obstacles (they must be at least 2 cells away of other obstacles).
Use the web-based tool in **Guides & Tools tab** that we implemented to see the output in a more visual way. It will help you to implement the solution :)

The input describes the dimensions of the 2D grid of the simulation, the number of frames that the simulator must provide and the initial state of the molecules and obstacles. More precisely:

- The 1st line is the word "rows" followed by a positive integer higher or equal than 3, which indicates the number of rows of the 2D grid.
- The 2nd line is the word "cols" followed by a positive integer higher or equal than 3, which indicates the number of columns of the 2D grid.
- The 3rd line is the word "frames" followed by a positive integer higher or equal than 2, which indicates the number of frames of the 2D grid that the simulation must compute.
- The rest of the lines describe the initial state of the molecules and obstacles, until no more lines are provided:
    - If the line starts with the word "molecule", it is followed by 5 numbers:

        1st ID of the molecule, which will be used to represent the molecule in the 2D grid. Different molecules can have the same ID, it is only used to have different types of molecules.

        2nd an integer in the range [0, rows) which indicates the initial row of the molecule.

        3rd an integer in the range [0, cols) which indicates the initial column of the molecule.

        4th an integer in the range [-1, 1] which indicates the initial direction in the "rows" axis.

        5th an integer in the range [-1, 1] which indicates the initial direction in the "cols" axis.

    - If the line starts with the word "obstacle", it is followed by 2 numbers:

        1st an integer in the range [0, rows) which indicates the row of the obstacle.

        2nd an integer in the range [0, cols) which indicates the column of the obstacle.

Output

The output must be a sequence of 2D matrices following a JSON format for arrays (see the output to clarify this format), providing as many matrices as indicated in the "frames" number from the input, being the first matrix the initial state of the 2D grid, with all the particles and obstacles in their initial position.

Each matrix must have size "rows x cols" and must be filled according to the following rules:

- The position of the molecules in the matrix must be filled with their ID.

- The position of the obstacles in the matrix must be filled with value -1.

- If 2 or more molecules are in the same position of the 2D grid, that cell must be filled with value -2.

- Any other cell without molecules or obstacles must be filled with value 0.

Example 1

Input

```
rows 8
cols 12
frames 6
molecule 1 0 0 1 1
molecule 2 3 4 1 1
molecule 1 7 8 -1 -1
obstacle 2 2
obstacle 2 8
obstacle 3 8
obstacle 5 3
obstacle 5 4
obstacle 5 5
```

Output

```
[
[
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 2, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
],
[
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0],
```

```
[0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
],
[
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 2, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
],
[
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, 0, 2, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
],
[
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
],
```

```
[
[0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]
]
```

Example 2

Input

```
rows 10
cols 23
frames 3
molecule 1 0 2 1 1
molecule 2 5 11 -1 1
molecule 3 6 17 1 1
obstacle 2 3
obstacle 2 4
obstacle 2 5
obstacle 2 10
obstacle 2 11
obstacle 2 12
obstacle 2 13
obstacle 2 16
obstacle 3 2
obstacle 3 6
obstacle 3 9
obstacle 3 16
obstacle 4 2
obstacle 4 6
obstacle 4 9
```

```
obstacle 4 16
obstacle 5 2
obstacle 5 3
obstacle 5 4
obstacle 5 5
obstacle 5 6
obstacle 5 9
obstacle 5 16
obstacle 6 2
obstacle 6 6
obstacle 6 9
obstacle 6 16
obstacle 7 2
obstacle 7 6
obstacle 7 10
obstacle 7 11
obstacle 7 12
obstacle 7 13
obstacle 7 16
obstacle 7 17
obstacle 7 18
obstacle 7 19
obstacle 7 20
```

Output

```
[
[
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, 0, 0, -1, 0, 0, 0, 0, 0,
0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, -1, -1, -1, -1, 0, 0, -1, 0, 2, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 3, 0, 0, 0, 0, 0],
```

```
[0, 0, -1, 0, 0, 0, -1, 0, 0, 0, -1, -1, -1, -1, 0, 0, -1, -1, -1, -1, -1, 0,
0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
],
[
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, 0, 0, -1, 0, 0, 0, 0, 0,
0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 2, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, -1, -1, -1, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 3, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, 0, -1, -1, -1, -1, 0, 0, -1, -1, -1, -1, -1, 0,
0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
],
[
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, 0, 0, -1, 0, 0, 0, 0, 0,
0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 2, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 3, 0, 0, 0],
[0, 0, -1, -1, -1, -1, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[0, 0, -1, 0, 0, 0, -1, 0, 0, 0, -1, -1, -1, -1, 0, 0, -1, -1, -1, -1, -1, 0,
0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]
]
```

Python

```python
import sys
from copy import deepcopy
from typing import List


# Some global constants
CELL_EMPTY = 0
CELL_WALL = -1
DEBUG = False


class Molecule:
  def __init__(self, id: int, row: int, col: int, step_r: int, step_c: int):
    self.id = id
    self.row = row
    self.col = col
    self.step_r = step_r
    self.step_c = step_c
    return


def generate_frames(molecules: List[Molecule], frame_rows: int, frame_cols: int,
num_frames: int, frame_template: List[List[int]]) -> List[List[List[int]]]:
  frames = []
  for num_frame in range(num_frames):
    # Create a frame with the current state
    if DEBUG:
      print("Frame " + str(num_frame))
    frame = deepcopy(frame_template)
    for molecule in molecules:
      assert molecule.row >= 0
      assert molecule.row < frame_rows
      assert molecule.col >= 0
      assert molecule.col < frame_cols
      assert frame[molecule.row][molecule.col] != CELL_WALL
      if frame[molecule.row][molecule.col] == 0:
        frame[molecule.row][molecule.col] = molecule.id
      elif frame[molecule.row][molecule.col] > 0:
        frame[molecule.row][molecule.col] = -2
      if DEBUG:
        print(f"molecule {molecule.id} at ({molecule.row}, {molecule.col}) with
dir ({molecule.step_r},{molecule.step_c})")
    frames.append(frame)
    if DEBUG:
      for row in frame:
```

```
        for col in row:
          if col == 0:
            print(".", end="")
          elif col == -1:
            print("O", end="")
          elif col == -2:
            print("X", end="")
          else:
            print(col, end="")
        print()
      print()

    # Update the molecules state
    for molecule in molecules:
      moleculeRowIsEdge = (molecule.row == 0 and molecule.step_r < 0) or
(molecule.row == frame_rows-1 and molecule.step_r > 0)
      moleculeColIsEdge = (molecule.col == 0 and molecule.step_c < 0) or
(molecule.col == frame_cols-1 and molecule.step_c > 0)
      if moleculeRowIsEdge or moleculeColIsEdge:
        # If the molecule is on the frame edges, fix the direction depending on
the situation
        if moleculeRowIsEdge:
          molecule.step_r = -molecule.step_r
        if moleculeColIsEdge:
          molecule.step_c = -molecule.step_c
      else:
        # If not in the frame edges, check if there is some wall that requires
to fix the molecule direction

        # Bounce in horizontal walls
        if frame_template[molecule.row + molecule.step_r][molecule.col] ==
CELL_WALL:
          molecule.step_r = -molecule.step_r

        # Bounce in vertical walls
        if frame_template[molecule.row][molecule.col + molecule.step_c] ==
CELL_WALL:
          molecule.step_c = -molecule.step_c

        # Bounce in corners (be careful because the previous 'ifs' are already
contemplating "concave corners", so
        # we want to detect here only convex corners)
        if (molecule.step_r != 0 and molecule.step_c != 0 and
          frame_template[molecule.row + molecule.step_r][molecule.col +
molecule.step_c] == CELL_WALL and
          frame_template[molecule.row +
molecule.step_r][molecule.col                  ] != CELL_WALL and
```

```
          frame_template[molecule.row                    ][molecule.col +
molecule.step_c] != CELL_WALL):
          molecule.step_r = -molecule.step_r
          molecule.step_c = -molecule.step_c

      # Update the molecule position
      molecule.row = molecule.row + molecule.step_r
      molecule.col = molecule.col + molecule.step_c

  return frames


def print_frames(frames: List[List[List[int]]]) -> None:
  print("[")
  for frameIdx in range(len(frames)):
    print("[")
    for rowIdx in range(len(frames[frameIdx])):
      print(frames[frameIdx][rowIdx],end="")
      if rowIdx == len(frames[frameIdx])-1:
        print("")
      else:
        print(",")
    if frameIdx == len(frames)-1:
      print("]")
    else:
      print("],")
  print("]")
  return


def read_input_line(expectation: str = "") -> List[str]:
  line = sys.stdin.readline().rstrip()
  if len(line) > 0:
    line_words = line.split(" ")
    assert len(line_words) > 0
    if len(expectation) > 0:
      assert line_words[0] == expectation
    return line_words
  else:
    return []


def main() -> None:
  # Read the input rows, cols and frames
  frame_rows = int(read_input_line("rows")[1])
  frame_cols = int(read_input_line("cols")[1])
  num_frames = int(read_input_line("frames")[1])
```

```
  # Create the empty template of a frame
  frame_template = [ [0] * frame_cols for x in range(frame_rows) ]

  # Read the molecules and obstacles
  molecules = []
  line_words = read_input_line()
  while len(line_words) > 0:
    if line_words[0] == "molecule":
      assert len(line_words) == 6
      molecules.append(Molecule(int(line_words[1]), int(line_words[2]),
int(line_words[3]), int(line_words[4]), int(line_words[5])))
    elif line_words[0] == "obstacle":
      assert len(line_words) == 3
      frame_template[int(line_words[1])][int(line_words[2])] = -1
    else:
      assert False # This should never happen if the input is correct
    line_words = read_input_line()

  # Generate and print the frames
  print_frames(generate_frames(molecules, frame_rows, frame_cols, num_frames,
frame_template))
  return


main()
```

32 Monster Slayer

35 points

Introduction

There is a new video game called Monster Slayer, in which each player controls a group of heroes that go into dungeons and kill all the monsters there. The faster they clear the dungeons, the more points they gain.

In each dungeon there may be different classes of monsters: Trolls, Golems, Demons and Dragons. Each monster will have an associated elemental type: Fire, Ice, Earth. The heroes' equipment also has an elemental type, so that it is more effective depending on the elemental type of the monster. The effectiveness rules are as follows (effective equipment, ineffective equipment):

- Fire monsters: Earth, Ice

- Ice monsters: Fire, Earth

- Earth monsters: Ice, Fire

In Monster Slayer, the more monsters you kill in a dungeon and the less turns it takes you to do it, the more points you get. Noelia is a competitive player, and she wants to go up in the online rankings.

Noelia is confident that she can complete every dungeon with her heroes so she has elaborated a table with the number of turns it takes to kill a monster in the best case, depending on the effectiveness of the equipment:

Monster Class	Effective	Neutral	Ineffective
Dragon	7 turns	9 turns	12 turns
Demon	3 turns	7 turns	11 turns
Golem	2 turns	5 turns	7 turns
Troll	2 turns	3 turns	5 turns

Furthermore, the heroes can change the equipment type in the middle of a dungeon **only once**, *by using 2 turns*.

Noelia's favorite equipment is Fire, then Ice and lastly Earth. If she has multiple best options, she will always choose her heroes to stay for the longest time in the equipment she likes the most.

Taking all of this information into account, can you write a program to help Noelia decide which equipment type (with or without changes) allows her to clear the dungeon faster?

Input

The first line will be the number of monsters in a dungeon. The following lines will contain the order in which monsters appear, each monster class and its type.

Output

The output should be a line containing the initial equipment to use, the change of equipment to perform if it's the case followed by the lines containing the monsters to slay with that equipment.

Example 1

Input

```
5
Earth Golem
Ice Troll
Ice Dragon
Earth Demon
Ice Demon
```

Output

```
Initial equipment: Ice
Earth Golem
Ice Troll
Ice Dragon
Earth Demon
Changing to Fire equipment
Ice Demon
```

When she had to fight against the Ice Demon with an Ice equipment the effectiveness is neutral, but if she changes her equipment to the one that is effective, even if she waste 2 turns she will kill the monster faster.

Example 2

Input

```
5
Fire Dragon
Ice Dragon
Fire Dragon
Ice Dragon
Fire Dragon
```

Output

```
Initial equipment: Fire
Fire Dragon
Ice Dragon
Fire Dragon
Ice Dragon
Fire Dragon
```

Although having multiple options that result in the same number of turns spent (like swapping to Earth equipment before the last enemy), this is the only valid output because it is the one that spends more time in the Fire equipment (this being Noelia's favorite equipment)

Python

```python
import sys

ADVANTAGE = 0
NEUTRAL = 1
DISADVANTAGE = 2

ELEMENT = 0
RACE = 1

DRAGON = 0
DEMON = 1
GOLEM = 2
TROLL = 3

#We define the number of turns for each monster
MONSTER_TURNS = [
    [7, 9, 12], #DRAGON TURNS
    [3, 7, 11], #DEMON TURNS
    [2, 5, 7], #GOLEM TURNS
```

```
    [2, 3, 5] #TROLL TURNS
]

#We define the elements list in the order of Natalia's liking
ELEMENTS = ["Fire", "Ice", "Earth"]

#This function will get the number of turns spent killing the monster
def calcSpentTurns(currentMonster, playerElement):

    #We first check if we will have advantage, disadvantage or we will be
neutral
    if playerElement == currentMonster[ELEMENT]:
        strategy = NEUTRAL
    else:
        if currentMonster[ELEMENT] == "Fire":
            if playerElement == "Earth":
                strategy = ADVANTAGE
            else:
                strategy = DISADVANTAGE
        elif currentMonster[ELEMENT] == "Ice":
            if playerElement == "Fire":
                strategy = ADVANTAGE
            else:
                strategy = DISADVANTAGE
        else:
            if playerElement == "Ice":
                strategy = ADVANTAGE
            else:
                strategy = DISADVANTAGE

    #Then we return the number of turns with the according monster and
advantage/disadvantage/neutral
    if currentMonster[RACE] == "Dragon":
        return MONSTER_TURNS[DRAGON][strategy]
    elif currentMonster[RACE] == "Demon":
        return MONSTER_TURNS[DEMON][strategy]
    elif currentMonster[RACE] == "Golem":
        return MONSTER_TURNS[GOLEM][strategy]
    else:
        return MONSTER_TURNS[TROLL][strategy]

#Function that will print the final output
def printFinalCombination(monsters, turnEquipmentChange, initialElement,
elementChange):
    #Print initialEquipment
    print("Initial equipment: " + initialElement)
    for i in range(len(monsters)):
```

```
        if i == turnEquipmentChange and elementChange != initialElement:
            #If we reach the turn when we do the equipment change, and we change
to a different element,
            # we print it
            print("Changing to " + elementChange + " equipment")
        #Print the monsters we are fighting
        print(monsters[i][ELEMENT] + " " + monsters[i][RACE])


###################### MAIN FUNCTION ####################

lowestTurnCount = sys.maxsize
finalPosTurnChange = sys.maxsize

monsters = []

numMonsters = int(input())

#We store all the monsters that we will fight
for i in range(numMonsters):
    monsterRead = input()
    monsterData = monsterRead.split(" ")
    monsters.append(monsterData)

#We test starting with all 3 different elements
for initialElement in range(3):

    #We loop through all possible positions where equipment could be changed
    for currentChange in range(numMonsters):
        #We do the 1st monster manually as we don't want to swap at the
begining. This also works as a
        # turn reset between turn calculations
        turnSum = calcSpentTurns(monsters[0], ELEMENTS[initialElement])

        #We loop through the monsters that we will slay before changing
equipment
        for currentMonster in range(1, currentChange):
            turnSum += calcSpentTurns(monsters[currentMonster],
ELEMENTS[initialElement])

        #We test both elements that we can swap to
        for newElement in range(3):
            #We use an auxiliar variable to be able to calculate both element's
turns
            #We check if we are calculating the next monsters with the elements
changed
            if newElement != initialElement:
```

```
                #If the element is changed, we add the 2 turns that action takes
                turnSumNewElement = turnSum + 2
            else:
                turnSumNewElement = turnSum
            #We loop through the remaining monsters with the new element
            for currentMonster in range(currentChange, numMonsters):
                turnSumNewElement += calcSpentTurns(monsters[currentMonster],
ELEMENTS[newElement])

            #If we find a combination with a lower turn count, we store it
            #NOTE: If a turnCount is equal to the lowest, it will be discarded,
and since we check
            # in order of Natalia's likings, we will always store the best
option according to her likings.
            if turnSumNewElement < lowestTurnCount:
                lowestTurnCount = turnSumNewElement
                finalPosTurnChange = currentChange
                finalInitialElement = ELEMENTS[initialElement]
                finalElementChange = ELEMENTS[newElement]

printFinalCombination(monsters, finalPosTurnChange, finalInitialElement,
finalElementChange)
```