CODEWARS

II SPAIN **Virtual** Edition 2022

Problems and Solutions

#	Problem	Points
1	CodewarRocks	1
2	Zenzizenzizenzic	2
3	The stocktaking	3
4	Magic number	4
5	Follow the steps	5
6	The 5 counter	5
7	Digit sum	6
8	Padel tournament	6
9	Tiling	6
10	Militar nessage	7
11	The Netflicks scheduler	8
12	Newspeak vowels	8
13	Aeronautical phraseology	9
14	Sorting photos	10
15	Take the red pill	10
16	URL shortening	10
17	The nightly chat	11
18	Anagrams	11
19	Gethenian calendar	11
20	TicTacToe checker	12
21	Roman fractions	13
22	A new hope	13
23	Lock pattern checker	13
24	Word search control	15
25	From A to Z	15
26	Mountain words	18
27	The space between us	18
28	Perfect pyramids numbers	18
29	Deterministic Finite Automaton	19
30	Circle of primes	28
31	Blackjack	30
32	Cyphers and Letters	31
33	Castellers competition	32
34	Hoverboards Olympics	38

1 CodeWars Rocks

1 points

Introduction

Did you know that this is the 2nd HP CodeWars Spain Virtual edition? We are so excited that we don't know which words to add to this text. Can you help us? Just replace, in order, each of the dash blocks of the message "--- is the --- of CodeWars. We --- you --- and have fun!" with each of the four lines received from standard input

Input

Four lines with text. The content of each line must be placed in the corresponding dash block.

<line 1>

<line 2>

<line 3>

<line 4>

Output

Print out the resulting sentence after doing the composition:

<line 1> is the <line 2> of CodeWars. We <line 3> you <line 4> and have fun!

Example

Input

```
This year
nth edition
hope
enjoy it
```

Output

```
This year is the nth edition of CodeWars. We hope you enjoy it and have fun!
```

Python

```
a = input()
b = input()
c = input()
d = input()

print( a + " is the " + b + " of CodeWars. We " + c + " you " + d + " and have
fun!")
```

2 Zenzizenzizenzic

2 points

Introduction

Zenzizenzizenzic is an old mathematical term in English with its roots in the Italian word *censo*, meaning squared. It refers to the square of the square of the square of an integer.

Input

The input consists of a single integer.

Output

The output is the zenzizenzizenzic of the input number.

Example

Input

7

Output

5764801

C++

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

int main()
{
   int a;

   std::cin >> a;

    std::cout << (int)pow(a,8) << std::endl;
}
```

3 The stocktaking

3 points

Introduction

We are running a store and have hired some people to sort the products depending on the boxes that are available. Each person writes down the quantity of products, boxes and how much is remaining, and we need to be sure that these numbers are correct.

All the boxes have the same capacity, and all the boxes are filled up to this capacity.

Input

- First line the quantity of products, greater than 0.

- Second line the quantity of boxes, greater than 0.

- Third line the quantity of what was left.

Output

The output will indicate either “CORRECT” (with the capacity of each box) or “INCORRECT.”

Example 1

Input

```
4528
4
0
```

Output

```
CORRECT, the capacity of each box is 1132
```

Example 2

Input

```
1233
7
0
```

Output

```
INCORRECT
```

Java

```java
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {

    int a, b, c;
    String res;

    // input data
    Scanner read=new Scanner(System.in);
    a=read.nextInt();
    b=read.nextInt();
    c=read.nextInt();

    if (a%b == c)
    {
      res = "CORRECT, the capacity of each box is " + a/b;
    }
    else
    {
      res = "INCORRECT";
    }
    //output data
    System.out.println(res);
  }
}
```

4 Magic number

4 points

Introduction

You collaborate in the sports section of the high school newspaper. This season the high school basketball team is playing very well and is in the first position. The newspaper director has requested you to find out the magic number required to win the season's title.

In certain sports, each game results in a win or a loss, but not a tie. The magic number indicates how close a front-running team is to win a season title. This number represents the total of additional wins by the front-running team or additional losses (or any combination thereof) by the rival team after which it is mathematically impossible for the rival team to capture the title in the remaining games. The formula to calculate the magic number is:

magic number = total number of games - number of wins by a front-running team - number of losses by the 2nd place team + 1

Can you write a program to find out the magic number?

Input

The input consists of three lines:

- The first line has the total number of games in a regular season.

- The second line has the name (in a single word) of the front-running team followed by the numbers of wins and losses during the current season.

- The third line has the name (in a single word) of the second placed team followed by the numbers of wins and losses during the current season.

Output

The output will be a single line reporting the magic number of matches that the front-running team must win to conquer the season's title.

Example

Input

```
162
Chicago 90 45
Cleveland 88 46
```

Output

```
Chicago must win 27 matches
```

Python

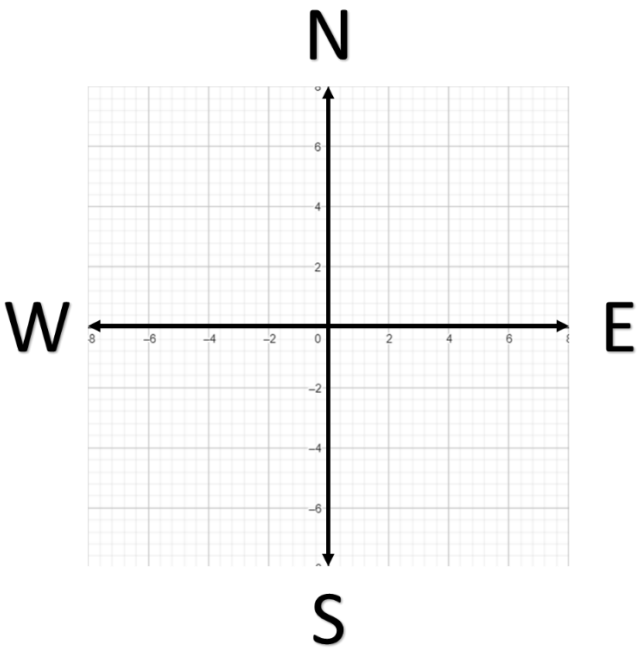
```python
numGames = int(input())
leaderTeamName, leaderTeamWins, leaderTeamLost = input().split()
secondTeamName, secondTeamWins, secondTeamLost = input().split()

leaderTeamWins = int(leaderTeamWins)
secondTeamLost = int(secondTeamLost)

magicNumber = numGames + 1 - leaderTeamWins - secondTeamLost

print(leaderTeamName + " must win " + str(magicNumber) + " matches")
```

5 Follow the steps

5 points

Introduction

The Jedi council has granted you the astromech droid C0-DY as an assistant to navigate your starship. Before planning your first space flight, you want to get familiar with the droid navigation system in an XY-Cartesian plane. Given a set of points (x0, y0), (x1, y1), ... (xn, yn) write a program that prints instructions for the droid to move to each point. Follow the order the points were given starting at the axis origin (0, 0) and moving in the cardinal directions. If one axis does not require a move, then do not print the move.



Input

The first line defines the number of points that the droid must follow.

The content of each line, after the first one, is a pair of integer values defining a (x, y) point.

Output

A pair of lines defining how to reach the corresponding (x, y) point. First line for the movement in Y-axis (North-South) and second line for the movement in X-axis (West-East).

Example

Input

```
4
3 5
2 5
2 7
9 5
```

Output

```
Walk 5 steps north
Walk 3 steps east
Walk 1 steps west
Walk 2 steps north
Walk 2 steps south
Walk 7 steps east
```

C++

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

int main()
{
   int moves;
   int x0 = 0;
   int y0 = 0;
   int x, y, moveX, moveY;

   std::cin >> moves;

   for( int i = 0; i < moves; i++)
   {
     std::cin >> x >> y;

     moveY = y - y0;
     if (moveY>0)
       std::cout << "Walk " + std::to_string(moveY) + " steps north" <<
std::endl;
     else if (moveY<0)
       std::cout << "Walk " + std::to_string(-moveY) + " steps south" <<
std::endl;

     moveX = x - x0;
     if (moveX>0)
       std::cout << "Walk " + std::to_string(moveX) + " steps east" <<
std::endl;
     else if (moveX<0)
       std::cout << "Walk " + std::to_string(-moveX) + " steps west" <<
std::endl;

     x0 = x;
     y0 = y;
   }
}
```

6 The 5 counter

5 points

Introduction

A religious sect thinks that god is number 5 and needs a program to determine in how many places is god, based on a number they obtain through an obscure and misterious process.

They ask you to write a program that calculates the number of 5's that exists in all numbers from 0 to a given number (including it).

For instance, given the number 20, there are two 5's in the numbers from 0 to 20: The one in 5 and the one in 15.

Or, another example, given the number 99, there are 20 5's in all numbers from 0 to 99 (11 in the 50's, remember that 55 has two fives, and 9 in the rest of tens).

Input

The input will be a positive integer.

Output

A number indicating the count of 5's in all numbers from zero to the given number.

Example 1

Input

```
100
```

Output

```
20
```

Example 2

Input

```
283746
```

Output

```
143145
```

Example 3

Input

```
5
```

Output

```
1
```

Python

```python
count = 0

end = input()

num = int(end)
for i in range(num+1):
        aux = str(i)
        if aux.find("5") != -1:
            count+=aux.count("5")

print(count)
```

7 Digit sum

6 points

Introduction

In Maths calculating the digit sum of a number can lead to some amazing results. Let's play a little, choose a positive two or more digit number and add the number to each of the individual digits that comprise the number. For instance, with number 23, add to it 2 and 3, giving

23 + 2 + 3 = 28

Repeating same steps with 28, we could create a series of ever-increasing numbers:

28 + 2 + 8 = 38,

38 + 3 + 8 = 49

and so on. Curiously enough, it's BEEN IMPOSSIBLE to find a formula to determine the Nth position of this series, starting with any given number. You need to follow the series step by step to reach that position and find out what the number is.

Code a program to calculate the Nth number in the series when you are provided with the first number.

Input

The input will be two lines.

The first line is the first number of the series consisting of two or more digits (>= 10).

The second figure is the number of elements the series will have including the initial number.

Both numbers will be positive, non-zero, integers. The first will be <= 1000000 and the second will be <= 1000.

Output

The output will be a single number corresponding to the Nth element calculated for the series.

Example 1

Input

23

11

Output

115

The explanation for this output follows as:

23 + 5 = 28 + 10 = 38 + 11 = 49 + 13 = 62 + 8 = 70 + 7 = 77 + 14 = 91 + 10 = 101 + 2 = 103 + 4 = 107 + 8 = 115

Example 2

Input

1234

332

Output

6655

Python

```python
initial = input()
iter = input()

suma = int(initial)

for x in range(int(iter)):
    l = len(str(suma))
    aux = 0
    for i in range(l):
        numero = str(suma)
        aux += int(numero[i])
        if i == l-1:
            suma += aux
print(suma)
```

8 Padel Tournament

6 points

Introduction

Cesar is the organizer of a Padel Tournament. Each time he receives the final score of each match he has to update the scoreboard and this is a very tedious work.

Help Cesar to write a program that given the result of a padel match, returns the final score and the winner.

In this tournament, padel matches are played as the best of 3 sets. Each set finishes when one team wins 6 games. If both teams win one set, then they are in a tie, so they play a super tie-break up to 10 points. See the following examples.

Input

The inputs of this problem are the teams names and the scores per each set:

- The first line is the name of the teams
- The second line is the score of the first set
- The third line is the score of the second set
- In case of a tie, the fourth line is the score of the super tie-break

Dash symbol, "-", is used to separate teams.

Output

One single line with the name of the winner and the final score with the format

```
<Winner team> won the match <first team's score> - <second team's score>.
```

Example 1

Input

```
Canariones - Chicharreros
6 - 0
6 - 0
```

Output

```
Canariones won the match 2 - 0.
```

Example 2

Input

```
Locomotiv - Guayamecos
1 - 6
6 - 8
```

Output

```
Guayamecos won the match 0 - 2.
```

Example 3

Input

```
Pio Pio! - GonCen
4 - 6
6 - 3
10 - 9
```

Output

```
Pio Pio! won the match 2 - 1.
```

Python

```python
# Padel Tournament problem
# Objective is to compute final score from set results

# Get input of the problem
players = input().split(' - ')
first_set = list(map(int,input().split(' - ')))
second_set = list(map(int,input().split(' - ')))

# Check result of first and second se
if (first_set[0] > first_set[1]):
    if (second_set[0] > second_set[1]):
        # Player 0 won the math 2 - 0
        print(players[0] + " won the match 2 - 0.")
    else:
        # Players are in tie
        # As they are in tie, read tie-break line
        tie_break = list(map(int,input().split(' - ')))

        if (tie_break[0] > tie_break[1]):
            print(players[0] + " won the match 2 - 1.")
        else:
            print(players[1] + " won the match 1 - 2.")

else:
    if (second_set[0] < second_set[1]):
        # Player 0 won the math 2 - 0
        print(players[1] + " won the match 0 - 2.")
    else:
        # Players are in tie
        # As they are in tie, read tie-break line
        tie_break = list(map(int,input().split(' - ')))

        if (tie_break[0] > tie_break[1]):
            print(players[0] + " won the match 2 - 1.")
        else:
            print(players[1] + " won the match 1 - 2.")
```

9 Tiling

6 points

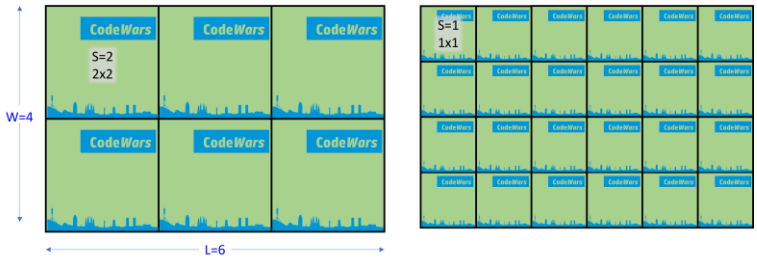
Introduction

Suppose we are tile artists that can make incredibly beautiful square tiles of any size.

The sizes of such tiles are measured in unit lengths. Let's say, we can make a tile of 3 units of size (3x3) or of 100 units of size (100x100).

There is no limit to the size of the tiles, as long as they are squared and have an integer size (1, 2, 3,...).

Now we want to cover the floor of a room using an exact amount of equal sized whole square tiles so that we do not have to cut any of the gorgeous tiles.



As you can see, for a given room we may have several possible solutions (in the picture S=2 or S=1)

We are artists, so we want to concentrate on the beauty of the design, not in the maths.

How can you help us to calculate the size S of the square tiles that will exactly cover the floor using the minimum number of whole tiles, given the length L and width W of the room's floor?

Input

Two positive integers greater than 0 that represent the length L and width W of the floor of the room.

Observation: These numbers do not need to be in order.

Output

The size S of the square tiles that exactly cover the floor of the room with the minimum number of whole tiles.

Example 1

Input

6

4

Output

2

Example 2

Input

49

21

Output

7

Example 3

Input

462

1071

Output

21

Python

```python
length = int(input())
width = int(input())

while (width > 0):
    res = width
    width = length % width
    length = res

print(length)
```

10 Military message

7 points

Introduction

It is 1942 and World War II is at its final stage, so it is critical to have a reliable communication channel to transfer messages. There is a rudimentary machine for that communication, but it is quite slow (remember, it is 1942...) and all the sent and received messages have to be manually processed. To replace that slow machine, you are in charge of the important task of implementing an automatic mechanism to send and receive messages.

It is important to take also into account the military ranks when sending and receiving a message. The military ranks are, from higher to lower:

Admiral > Captain > Commander > Lieutenant > Officer

If the sender of a message has a higher rank than the receiver, the first field of the transmitted message will be the word "URGENT", and the interpretation of the recevied message must start with "<<< URGENT >>>" always (a received message could be missing that word due to an error when sending it).

Input

The first line of the input is a word which must contains either "S" or "R" to indicate that we have to "send" or "receive" a message, respectively.

If the first line is "S", it will be followed by 6 lines, each one of them being: the name of the sender, the rank of the sender, the name of the receiver, the rank of the receiver, the message text and the time in the format hhmm (being hh the hour and mm the minute).

If the first line is "R", it will be followed by a single line containing all the data of the message that has been received.

Output

If the input is a message that has to be sent (S), the output must be the transmitted message in a single line, with all the input fields separated by commas. Remember to put an extra field "URGENT" at the beginning of the message if the rank of the sender is higher than the rank of the receiver.

If the input is a received message (R), the output must be the interpretation of that message, as shown in the Example 2 below. If it is an "URGENT" message, remark that putting the word in between "<<< >>>", as shown in Example 2.

If the first line is neither "S" nor "R", you must indicate that there is an error in the input by showing the message "Invalid input.". Show that message also if any of the given military ranks is invalid. Be careful, because both sent and received messages may contain an invalid rank.

Example 1

Input

```
S
Bobby Shaftoe
Lieutenant
Lawrence Waterhouse
Officer
Bring coffee!
2357
```

Output

```
URGENT,Bobby Shaftoe,Lieutenant,Lawrence Waterhouse,Officer,Bring
coffee!,2357
```

Example 2

Input

```
R
Bobby Shaftoe,Lieutenant,Lawrence Waterhouse,Officer,Bring coffee!,2357
```

Output

```
<<< URGENT >>>
From: Bobby Shaftoe
From rank: Lieutenant
To: Lawrence Waterhouse
To rank: Officer
Content: Bring coffee!
Timestamp: 2357
```

Example 3

Input

```
M
URGENT,Bobby Shaftoe,Lieutenant,Lawrence Waterhouse,Officer,Bring
coffee!,2357
```

Output

```
Invalid input.
```

Python

```python
def main():
    # The military ranges
    ranks = ["Admiral", "Captain", "Commander", "Lieutenant", "Officer"]

    # Parse the input action
    action = input()
    if action == "R":
        # Read the input message and split it
        msgParts = input().split(",")
        printUrgent = False
        # Check if first word is URGENT
        if msgParts[0] == "URGENT":
            printUrgent = True
            msgParts = msgParts[1:]
        # Check military ranks
        if not msgParts[1] in ranks or not msgParts[3] in ranks:
            print("Invalid input.")
            return
        if ranks.index(msgParts[1]) < ranks.index(msgParts[3]):
            printUrgent = True
        # Print the message
        if printUrgent:
            print("<<< URGENT >>>")
        print("From: " + msgParts[0])
        print("From rank: " + msgParts[1])
        print("To: " + msgParts[2])
        print("To rank: " + msgParts[3])
        print("Content: " + msgParts[4])
        print("Timestamp: " + msgParts[5])
    elif action == "S":
        # Read each of the fields of the message
        p1 = input()
        r1 = input()
        p2 = input()
        r2 = input()
        msg = input()
        time = input()
        # Check military ranks
        if not r1 in ranks or not r2 in ranks:
            print("Invalid input.")
            return
        # Print the message
        msg = p1 + "," + r1 + "," + p2 + "," + r2 + "," + msg + "," + time
        if ranks.index(r1) < ranks.index(r2):
```

```
            print("URGENT," + msg)
        else:
            print(msg)
    else:
        print("Invalid input.")
    return

main()
```

11 The NetFlicks scheduler

8 points

Introduction

In the *Perféctez* family they have a lot of trust in each other. As the family´s only son, *Benigno,* has been very good, his parents have allowed him to watch *Netflicks* for a few hours each week. They've decided to use a program to schedule the hours that *Benigno* can watch his favourite series and they check how many hours he has spent watching TV every day and the hours he has left. *Benigno* never cheats, but sometimes he exceeds his allotted time so the program would also indicate how much he has exceeded his time.

Input

The input will consist of one line with at least 1 parameter (number of allowed weekly hours) and/or up to 7 extra parameters (the time spent every weekday), separated by commas (,). Extra parameters will be ignored. All parameters will have the following format:

hh:mm or h:mm

Therefore, the input format would be:

weekly_hours[, monday_hours][, tuesday_hours][, wednesday_hours][, thursday_hours][, friday_hours][, saturday_hours][, sunday_hours]

Output

The program will return a line stating the total time left for *Benigno* to watch his favourite series.

The format of the output will be:

```
XX:YY hours of viewing remaining.
```

Please note the trailing dot.

If the time is exceeded the output will be:

```
LIMIT EXCEEDED BY XX:YY hours. Benigno punished!
```

Example 1

Input

```
14:00, 2:21, 1:32
```

Output

```
10:07 hours of viewing remaining.
```

Example 2

Input

```
14:00, 2:21, 1:32, 0:00, 3:27, 3:25, 3:10
```

Output

```
00:05 hours of viewing remaining.
```

Example 3

Input

```
14:00, 2:21, 1:32, 0:00, 3:27, 3:25, 4:20, 0:00
```

Output

```
LIMIT EXCEEDED BY 01:05 hours. Benigno punished!
```

Python

```python
from datetime import timedelta


def main():
    # get hours, remove spaces, take the first 8 (maximum input)
    hours = [hour.strip() for hour in input().split(sep=',')][:8]

    # convert everything in minutes instead of using datetime objects, too much
hassle
    minutes = []
    for hour in hours:
        splitted = hour.split(sep=':')
        minutes.append(int(splitted[0])*60+int(splitted[1]))

    time_left = minutes[0]
    for i in range(1, len(minutes)):
        time_left -= minutes[i]

    if time_left >= 0:
        print(
            f"{time_left // 60:02d}:{time_left % 60:02d} hours of viewing
remaining.")
    else:
        print(
            f"LIMIT EXCEEDED BY {-time_left // 60:02d}:{-time_left % 60:02d}.
Benigno punished!")


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

12 Newspeak vowels

8 points

Introduction

The last edition of the Orwellian Newspeak English Dictionary has decided to update all the words that contain exactly two consecutive vowels and reverse them. Help them to code a simple program to translate a given word according to this new rule. Remember Big Brother is watching you!

Input

The input will be a single word.

Output

The output will be the word with two consecutive reversed vowels.

Example 1

Input

```
source
```

Output

```
suorce
```

Example 2

Input

```
plan
```

Output

```
plan
```

Example 3

Input

```
beautiful
```

Output

```
beautiful
```

Python

```python
def isVowel(c):
  c = c.lower()
  if (c=="a" or c=="e" or c=="i" or c=="o" or c=="u"):
    return True
  else:
    return False

word = input()
outputWord = word

# Main program

for i in range(0, len(word)-1):
  # Look for a candidate pair of vowels in the word
  if (isVowel(word[i]) and isVowel(word[i+1])):
    # Check that there are no vowels before or after
    if (i-1 >=0):
      if (isVowel(word[i-1])):
        # Found a vowel before, so discard this pair and continue the search
        continue
    if (i+2 <=len(word)-1):
      if (isVowel(word[i+2])):
        # Found a vowel after, so discard this pair and continue the search
        continue
    # This is a real pair of vowels swap them
    outputWord = outputWord[:i] + word[i+1] + word[i] + outputWord[i+2:]

print(outputWord)
```

13 Aeronautical phraseology

9 points

Introduction

You are about to join the Space Force Air Traffic Control, but unfortunately you skipped almost all the classes because they were online, and you were playing "among us". So, you decide to create a simple program that converts everyday phraseology to aeronautical phraseology following these conditions:

- Numbers must be converted to their names, with the first letter in upper case.

- Decimal points must be written using the word "Decimal"

- Capital letters must be converted to phonetic alphabet as shown in the table.

- Words that already follow the correct phraseology don't have to be converted.

- Words that don't follow any of these categories don't have to be converted.

- If a hash mark (#) is the only thing entered, the program must stop asking inputs

A	Alpha	K	Kilo	U	Uniform	4	Four
B	Bravo	L	Lima	V	Victor	5	Five
C	Charlie	M	Mike	W	Whiskey	6	Six
D	Delta	N	November	X	X-ray	7	Seven
E	Echo	O	Oscar	Y	Yankee	8	Eight
F	Foxtrot	P	Papa	Z	Zulu	9	Nine
G	Golf	Q	Quebec	0	Zero		
H	Hotel	R	Romeo	1	One		
I	India	S	Sierra	2	Two		
J	Juliett	T	Tango	3	Three		

Input

A sequence of strings separated by an enter.

Output

That string transcribed to the correct aeronautical phraseology, as shown below.

Example

Input

```
A123BC
D119 left Y right H
Delta Echo Golf cleared for take-off
F456 altimeter 30.05 maintain 13000
#
```

Output

```
Alpha One Two Three Bravo Charlie
Delta One One Nine left Yankee right Hotel
Delta Echo Golf cleared for take-off
Foxtrot Four Five Six altimeter Three Zero Decimal Zero Five maintain One
Three Zero Zero Zero
```

Python

```
number_names = ["Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven",
"Eight", "Nine"]

dict_string = {
    "A":"Alpha", "B":"Bravo", "C": "Charlie", "D":"Delta", "E":"Echo",
"F":"Foxtrot", "G":"Golf", "H":"Hotel", "I":"India", "J":"Juliett", "K":"Kilo",
"L":"Lima", "M":"Mike", "N":"November", "O":"Oscar", "P":"Papa", "Q":"Quebec",
"R":"Romeo", " S":"Sierra", "T":"Tango", "U":"Uniform", "V":"Victor",
"W":"Whiskey", "X":"X-ray", "Y":"Yankee", "Z":"Zulu",
    ".": "Decimal",
}

character_list = []

while True:

    input_data = input()
    if input_data == "#":
        break
    for s in input_data.split():
        found = False
        for key in dict_string.keys():
            if dict_string[key] == s:
                for k in list(s):
                    character_list.append(k)
                    found = True
        if found: character_list.append(" ")
        if not found:

            for i in list(s):
                from_key = False
                if i.isdigit():
                    character_list.append(number_names[int(i)])
                    from_key = True
                elif i in dict_string.keys():
                    character_list.append(dict_string[i])
                    from_key = True
                else:

                    character_list.append(i)

                if character_list != [] and from_key:
                    c = character_list.pop()
                    character_list.append(c)
                    if c != " ":
```

```
                        character_list.append(" ")
            c = character_list.pop()
            character_list.append(c)
            if c != " ":
                character_list.append(" ")


    character_list.append('\n')

limit = len(character_list)
iteration = 1
concat = ""
for i in character_list:
    if i != '\n':
        concat = concat + i
    else:
        print(concat)
        concat = ""
```

14 Sorting photos

10 points

Introduction

Enrique loves travelling with his friends. He enjoys discovering new places and taking photos all the time. However, when coming back home, he has to classify and archive all the pictures taken by him and his friends, and that's a very long and boring task. There are always two types of photo files, the ones taken by the smartphones and the ones taken by Enrique's reflex camera. The files have a different naming structure, which is as follows:

- Smartphone cameras use the format "IMG_YYYYMMDD_HHMMSS.jpg", where YYYY is the year, MM is the month, and DD is the day of the month. For example: IMG_20171203_213455.jpg corresponds to a photo taken in Dec 3rd, 2017, at 21:34:55.

- Reflex camera files are stored in the format "PDDMMYY_HHMMSS.jpg", where DD is the day of the month, MM is the month, and YY is the two last digits of a year (assuming it is a 21st century year). For example: P031217_213455.jpg corresponds to a photo taken in Dec 3rd, 2017, at 21:34:55.

In both cases, HHMMSS is the timestamp, with HH for the hour (0 to 23), MM for the minute (0 to 59) and SS for the seconds (0 to 59).

Given these two different formats, it is impossible to sort the files automatically by name, since all the "IMG_" ones would be placed before the "P_" ones. Enrique would like to have a Linux script to rename all the files in a way that they include the name of the place that they visited, followed by a three-digit counter, which starts from 000 for the earliest picture and grows one unit for each next photo, sorted by the date and time they were taken. The Linux command "mv" is good enough for renaming the files in this way.

Note: "mv" command syntax is:

mv <source> <destination>

Note: there will be less than 1000 photos in every collection.

Input

The input will consist of 3 lines:

- The name of the place visited on the trip. It will be always one word.

- A list of the photo files coming from the smartphone cameras, separated by a space

- A list of the photo files coming from the reflex camera, separated by a space

Output

The output will be the list of "mv" commands to execute in order to rename all the files. Enrique also wants the list to be ordered by the new file name, so the first command must be the one that converts the oldest photo.

Example 1

Input

```
Japan
IMG_20210613_104512.jpg IMG_20210612_225814.jpg
P130621_083827.jpg
```

Output

```
mv IMG_20210612_225814.jpg Japan_000.jpg
mv P130621_083827.jpg Japan_001.jpg
mv IMG_20210613_104512.jpg Japan_002.jpg
```

Example 2

Input

```
Paris
IMG_20170103_124522.jpg IMG_20170104_113321.jpg IMG_20170107_223749.jpg
P020117_203138.jpg P030117_052636.jpg
```

Output

```
mv P020117_203138.jpg Paris_000.jpg
mv P030117_052636.jpg Paris_001.jpg
mv IMG_20170103_124522.jpg Paris_002.jpg
mv IMG_20170104_113321.jpg Paris_003.jpg
mv IMG_20170107_223749.jpg Paris_004.jpg
```

Python

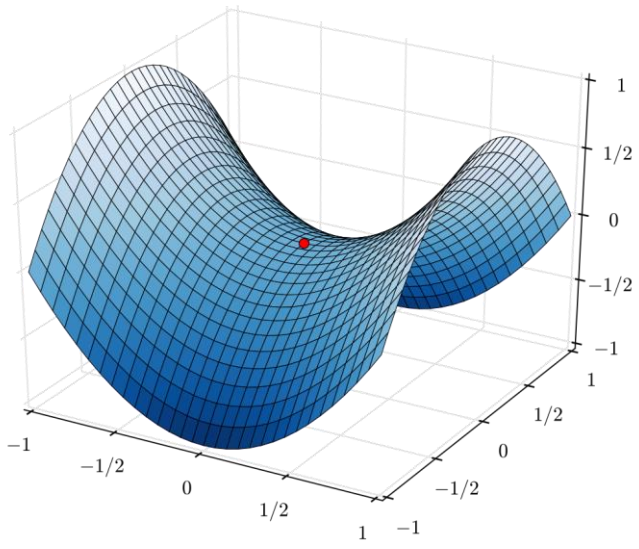
```python
# Store the input data
place = input()
imgFiles = input().split(" ")
pFiles = input().split(" ")

# Dictionary to link timestamps (in string format) to the original file names.
dateTimeToFileName = {}

# Calculate the dates for IMG_ files.
# Create a string containing the date and time, extracted from the file name

for file in imgFiles:
    date = file[4:12]
    time = file[13:19]
    datetime = date + time
    dateTimeToFileName[datetime] = file

# Calculate the dates for P* files.
# Create a string containing the date and time, extracted from the file name

for file in pFiles:
    date = "20" + file[5:7] + file[3:5] + file[1:3]
    time = file[8:14]
    datetime = date + time
    dateTimeToFileName[datetime] = file

# Now sort the timestamps:

sortedTimeStamps = sorted(dateTimeToFileName.keys())

counter = 0

# for every sorted timestamp, retrieve the original file name
# to convert and compose the 'mv' command

for timeStamp in sortedTimeStamps:
    originalFileName = dateTimeToFileName[timeStamp]
    counterStr = str(counter).zfill(3)          # Add padding zeroes
    newName = place + "_" + counterStr + ".jpg"
    print("mv " + originalFileName + " " + newName)
    counter += 1
```

15 Take the red pill

10 points

Introduction

Remember the movie The Matrix (1999): "..., *to enter into a matrix and locate its saddle point if it exists*", so let's try to enter in Matrix. Given an N x N matrix of integers the saddle point is the minimum element in its row and the maximum in its column. So, help Neo writing a program to find out whether a saddle point exists and which is its value.



HINT: If there is a saddle point, it will be only one.

Input

The input consists of:

- The first row contains a single integer N that defines the matrix size (N rows x N columns).
- The following N lines define the rows of the matrix with N integer values per row.

Output

If there is a saddle point, print a message saying which element of the matrix is the saddle point. Otherwise, print the error message "No saddle point in the matrix".

Example 1

Input

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

Output

```
The saddle point is 7
```

Example 2

Input

```
3
1 2 3
7 8 9
10 5 6
```

Output

```
No saddle point in the matrix
```

Python

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

matrix = []

# Read the matrix from standard input
for i in range(n):
  rowStr = input()
  rowList = [int(i) for i in rowStr.split()]
  matrix.append(rowList)

row = -1
col2 = -1
found = False

# For each column look for the largest value
# Then process the row that has this value
for col in range(n):

  largest = matrix[0][col]
  # Identify which row contains the largest value
  for i in range(n):
    if matrix[i][col] >= largest:
      largest = matrix[i][col]
      row = i

  smallest = matrix[row][0]
  # Look for smallest value in the selected row
  # and check that smallest value is in the same column
  for j in range(n):
    if matrix[row][j] <= smallest:
      smallest = matrix[row][j]
      col2 = j

  # Finally check whether the found column of the smallest
  # value is the same than the column of the largest value
  if col == col2:
    found = True
    break

if found:
  print("The saddle point is " + str(largest))
else:
  print("No saddle point in the matrix")
```

16 Shortening URLs

10 points

Introduction

Shortening URLs on the web is a technique that makes addresses easier to manage, but still allows them to direct to the required page. This can be useful when sharing specific URLs in, for example, instant messaging technologies like tweets or SMS that have a limit on the number of characters allowed.

One of the most popular techniques for shortening URLs consists of having them stored in a database associated a unique integer key. This avoids having to manage long addresses by generating a new, short URL.

Considering that an URL character can be one of the following:

- A lower case alphabet from 'a' to 'z

- An upper case alphabet from 'A' to 'Z'

- A digit from '0' to '9'

So, it sums up to a total of 62 possible characters to represent the new short URL.

Can you code a program to convert a given positive integer (up to 4294967295) to a base 62 number where digits of 62 base consists of the lower case letters a-z, the capital letters A-Z and the numbers 0–9?

Input

The input will be a positive integer.

Output

The output will be the shortened URL key expressed as base 62 number.

Example 1

Input

```
31
```

Output

```
F
```

Example 2

Input

4294967291

Output

eQPpl9

Python

```python
base62String = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
shortenedURL = ""

a = int(input())

while (a >= 62):
  remainder = a % 62
  a = a // 62
  shortenedURL = base62String[remainder] + shortenedURL

shortenedURL = base62String[a] + shortenedURL
print(shortenedURL)
```

17 The nightly chat

11 points

Introduction

Three friends who live in Madrid, Moscow and Tokyo want to chat online after work. Strange as it may sound, they only want to talk when it is dark outside each of their respective homes.

The three friends are asking you to develop a program to know at which times they can chat. Knowing that Madrid is located at GMT+1 (Greenwich Mean Time + 1 hour), Moscow is located at GMT+3 and Tokyo is located at GMT+9, and assuming that in those three cities the dusk (and the dawn) occurs at the same local time, the program has to calculate the first moment (hour and minute in GMT time) and last minute when the three friends can chat.

Since it may be that there's no time in the day where the three friends can chat, the program has to indicate that possibility.

Note: The three friends need to be able to chat for at least 1 minute or it will be considered that there is no window to do so.

As an example, let's consider the case in which the sun goes down at 19:32 and rises again at 08:43. The following will be the respective dusk and dawn times in all three places in GMT:

Local time	Madrid (GMT+1)	Moscow (GMT+3)	Tokyo (GMT+9)	Type
19:32	18:32 GMT	16:32 GMT	10:32 GMT	Dusk
08:43	07:43 GMT	05:43 GMT	23:43 GMT	Dawn

In the table, the "dusk" line can be considered the starting time. While the "dawn" line can be considered the end time.

With these settings, the chat window will be 18:32 to 23:43 always expressed as GMT.

Let's try another example. Let's consider that sundown is at 23:04 and sunrise is at 06:45. The table would then read:

Local time	Madrid (GMT+1)	Moscow (GMT+3)	Tokyo (GMT+9)	Type
23:04	22:04 GMT	20:04 GMT	14:04 GMT	Dusk
06:45	05:45 GMT	03:45 GMT	21:45 GMT	Dawn

As the chat window end time in Tokyo (21:45 GMT) would be earlier than the chat window start time in Madrid (22:04 GMT), it would not be possible to chat that day.

Input

The program will receive the GMT time of dawn and dusk for that day in the format HH:MM.

In the first example the input would be:

```
19:32
8:43
```

Output

The output will be a single line indicating either the beginning and the end times of the chat window with format "HH:MM to HH:MM GMT"

or

```
CANNOT FIND A SLOT
```

if it's not possible to find a chat window.

In the first example, the output would be:

```
18:32 to 23:43 GMT
```

Example 1

Input

```
19:32
8:43
```

Output

```
18:32 to 23:43 GMT
```

Example 2

Input

```
23:04
06:45
```

Output

```
CANNOT FIND A SLOT
```

Example 3

Input

```
21:43
6:56
```

Output

```
20:43 to 21:56 GMT
```

Python

```python
def remove_hours(hours, toRemove):
    result = hours-toRemove
    return result if result >= 0 else result + 24


def main():
    dusk = input().split(sep=':')
    dawn = input().split(sep=':')
    dusk_hours = int(dusk[0])
    dusk_minutes = int(dusk[1])
    dawn_hours = int(dawn[0])
    dawn_minutes = int(dawn[1])

    span_hours = dawn_hours-dusk_hours
    if span_hours < 0:
        span_hours += 24

    span_minutes = dawn_minutes-dusk_minutes
    if span_minutes < 0:
        span_minutes += 60
        span_hours -= 1  # in this case an hour needs to be substracted from the
span

    # there is overlap if span is more than 8 hours
    if span_hours < 8 or (span_hours == 8 and span_minutes == 0):
        print("CANNOT FIND A SLOT")
    else:
        # considering only Madrid start and Tokio end is fine, those are the
edge cases
        madrid_gmt_start_hours = remove_hours(dusk_hours, 1)
        tokyo_gmt_end_hours = remove_hours(dawn_hours, 9)
        print(f"{madrid_gmt_start_hours:02d}:{dusk_minutes:02d} to
{tokyo_gmt_end_hours:02d}:{dawn_minutes:02d} GMT")


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

18 Anagrams

11 points

Introduction

An anagram is a word formed by rearranging the letters of a different word, using all the original letters exactly once. For instance, the word *listen* can be rearranged into *silent*.

Checking for anagrams might be a tedious task, especially in the case of long words. Because of that, we ask you to help us in coding a simple program that, given an original word and a list of words, finds which of them are anagrams of the original text in the list.

Input

The input consists of two lines:

- A first one with the original word.

- A second one with the list of words to check for anagrams.

All inputs are always given in lower case.

Output

The output will be all the anagrams in the list, sorted by order of appearance.

Example 1

Input

```
abcd
abdc cdab bace dacb abce
```

Output

```
abdc cdab dacb
```

Example 2

Input

```
codewars
anagram warscode varscode warcodes rawcodes readcaos sweardoc
```

Output

```
warscode warcodes rawcodes sweardoc
```

Python

```python
def areAnagrams(a, b):
    if a == b:
        # a word is not an anagram of itself
        return False

    # Check if a and b sorted alphabetically are equal
    return sorted(a) == sorted(b)

subject = input()
words   = input().split()

first = True
for word in words:
    if areAnagrams(word, subject):
        if not first:
            print(' ', end='')
        print(word, end='')
        first = False

if not first:
    print('')
```

19 Gethenian calendar

11 points

Introduction

In the science fiction novel "The Left Hand of Darkness" by Ursula K. Le Guin, Genly Ai representing the Ecumen arrives to planet Gethen. He must adapt to its particular cold weather and specific calendar. The Gethenian year has 14 months spread in four seasons:

#	Winter	#	Spring	#	Summer	#	Autumm
1	Thern	5	Irrem	8	Some	12	Gor
2	Thanern	6	Moth	9	Ockre	13	Susymy
3	Nimmer	7	Tuwa	10	Kus	14	Grende
4	Anner			11	Hakanna		

And a Gethenian month has 26 days, where each day has a name:

#	Name	#	Name	#	Name	#	Name
1	Getheny	8	Orny	15	Odsordny	22	Odharhahad
2	Sordny	9	Harhahad	16	Odeps	23	Odguyrny
3	Eps	10	Guyrny	17	Odarhad	24	Odyrny
4	Arhad	11	Yrny	18	Onnetherhad	25	Opposthe
5	Netherhad	12	Posthe	19	Odstreth	26	Ottormenbod
6	Streth	13	Tormenbod	20	Obberny		
7	Berny	14	Odgetheny	21	Odorny		

Given two Gethenian dates considering the same year, can you write a simple program to calculate the number of days between two dates.

Input

The input will be a single line containing a pair of day and month names separated by a dash.

Output

The output will be a single number representing the number of days comprised.

Example

Input

```
Posthe Tuwa - Odyrny Kus
```

Output

```
90
```

Python

```python
months = ["Thern", "Thanern", "Nimmer", "Anner", "Irrem", "Moth",
          "Tuwa", "Osme", "Ockre", "Kus", "Hakanna", "Gor", "Susymy", "Grende"]

days = ["Getheny", "Sordny", "Eps", "Arhad", "Netherhad", "Streth",
        "Berny", "Orny", "Harhahad", "Guyrny", "Yrny", "Posthe", "Tormenbod",
        "Odgetheny", "Odsordny", "Odeps", "Odarhad", "Onnetherhad", "Odstreth",
        "Obberny", "Odorny", "Odharhahad", "Odguyrny", "Odyrny", "Opposthe",
"Ottormenbod"]

lst = [item for item in input().split()]

firstDay = days.index(lst[0]) + 1
firstMonth = months.index(lst[1]) + 1
secondDay = days.index(lst[3]) + 1
secondMonth = months.index(lst[4]) + 1

#print(str(firstDay) + " " + str(firstMonth) + " " + str(secondDay) + " " +
str(secondMonth))

res = (secondDay + (secondMonth - 1) * 26) - (firstDay + (firstMonth - 1) * 26)

print(res)
```

20 Tic Tac Toe checker

12 points

Introduction

The "Three-in-a-row" game, known in English as Tic-Tac-Toe is a game that can be traced back to Egypt, where such board games have been found on roofing tiles dating from around 1300 BC. But it has been found that in those roofing tiles the board games do not always follow current rules.

You are asked to write a program that determines whether a game follows current Tic-Tac-Toe rules and who won that game (if someone did). In the program, a Tic-Tac-Toe board can only have three symbols: X, O and _ (for the blank square).

A game is considered finished if:

a) One of the players (X or O) obtained a three-in-a-row (either horizontally, vertically or in one of the main diagonals). In this case the output will show who is the winner with the outputs "X WON" or "O WON".

b) All 9 board squares are filled with "X" or "O" and three-in-a-row is not found. In this case the output will be "IT'S A TIE"

There are two cases in which a board should be considered "NOT VALID":

- In a valid board the number of X's and O's may only differ in one, otherwise the board will be considered "NOT VALID".

- If a board shows three-in-a-row of both players, the board will also be considered "NOT VALID".

If the game is not finished and the board is not considered "NOT VALID", this means the board shows a game in a "PLAYING" state.

Input

The input will consist of three lines containing three characters (X, O or _) each.

Output

Depending on the previously indicated rules, the output will consist in a single line indicating either

- NOT VALID

- PLAYING

- IT'S A TIE

- X WON

- O WON

Example 1

Input

XXX

XOO

XOO

Output

```
X WON
```

Example 2

Input

X__

OOO

OOO

Output

```
NOT VALID
```

Python

```
def winningBoard(board, player):
#check horizontal winners
    playerCount = 0
    hwin = 0
    for line in range(0,len(board)):
        count = 0
        for col in range(0,len(board[line])):
            if board[line][col] == player:
                count = count + 1
                playerCount = playerCount + 1
        if (count == 3):
            hwin =  hwin + 1

    vwin = 0
    for line in range(0,len(board)):
        count = 0
        for col in range(0,len(board[line])):
            if board[col][line] == player:
                count = count + 1
        if (count == 3):
            vwin = vwin + 1

    dwin=False
    if ((board[0][0] == player and board[1][1] == player and board[2][2] ==
player) or
        (board[0][2] == player and board[1][1] == player and board[2][0]  ==
player)):
        dwin = True

    win = "NO"
    if hwin == 1 or vwin == 1 or dwin:
        win = "YES"
    elif hwin > 1 or vwin > 1:
        win = "INVALID"

    return playerCount,win



l1 = list(input())
l2 = list(input())
l3 = list(input())

board = [l1,l2,l3]
xCount,xWin = winningBoard(board, "X")
```

```
oCount, oWin = winningBoard(board, "O")

if (xWin == "INVALID" or oWin == "INVALID"):
    print("NOT VALID")
elif(abs(xCount - oCount) > 1):
    print("NOT VALID")
elif (xWin == "YES" and oWin == "YES"):
    print("NOT VALID")
elif(xWin == "YES"):
    print("X WON")
elif(oWin == "YES"):
    print("O WON")
elif(xCount + oCount == 9):
    print("IT'S A TIE")
else:
    print("PLAYING")
```

21 Roman fractions

13 points

Introduction

Roman numerals are a numeral system that originated in ancient Rome and remained the usual way of writing numbers throughout Europe well into the Late Middle Ages. Integer numbers in this system are represented by combinations of letters from the Latin alphabet. Modern usage employs seven symbols, each with a fixed integer value:

"M", "D", "C", "L", "X", "V", "I"

What's not so widely known is that Romans also used a method to write fractions. It was a very simple method as it only allowed to write 11 possible fractions.

The Romans used a duodecimal rather than a decimal system for fractions, as the divisibility of twelve makes it easier to handle the common fractions of 1/2, 1/3 and 1/4. Notation for fractions is mainly found on surviving Roman coins, many of which had values that were duodecimal fractions of the unit. Fractions less than 1/2 are indicated by a dot (.) for each "twelfth" (uncia in latin).

Therefore, possible fractions in latin were:

```
1/12          which is 0.083    represented by  .
2/12 = 1/6    which is 0.166    represented by  :
3/12 = 1/4    which is 0.25     represented by  :.
4/12 = 1/3    which is 0.333    represented by  ::
5/12          which is 0.416    represented by  ::.
6/12 = 1/2    which is 0.5      represented by  S
7/12          which is 0.583    represented by  S.
8/12 = 2/3    which is 0.666    represented by  S:
9/12 = 3/4    which is 0.75     represented by  S:.
10/12 = 5/6   which is 0.833    represented by  S::
11/12         which is 0.916    represented by  S::.
```

(remember that 12/12 = 1 which is I)

Note: For more information, read wikipedia.

Exercise

You are asked to write a program that transform a decimal number into roman. The number may or may not have a decimal section (therefore it may be an integer).

Given the previous instructions, the program will return the appropiate roman number string.

Note: As the roman fraction system was so weak, only the allowed decimal section will be allowed:

".083", ".166", ".25", ".333", ".416", "0.5", ".583", ".666", ".75", ".833", ".916"

Note: As 1.5 is the same as 1.500, this input will be following and it must return a valid roman value.

Input

Input will consist of an integer or decimal number (represented by a dot as separator, ie: 12.25).

Output

If the input value has a fractional part that cannot be represented as a 1/12 part the output will be:

ERROR

Otherwise, the output will be the appropiate roman number given the previously indicated instructions.

If the integer part is 0 then only the decimal part must be printed.

Example 1

Input

```
1287.916000
```

Output

```
MCCLXXXVIIS::.
```

Example 2

Input

```
0.25
```

Output

```
:.
```

Example 3

Input

```
123
```

Output

```
CXXIII
```

Python

```python
def integer_to_roman(number : int) -> str:
    roman_numbers = [
        ("M", 1000), ("CM", 900), ("D", 500), ("CD", 400),
        ("C", 100), ("XC", 90), ("L", 50), ("XL", 40),
        ("X", 10), ("IX", 9), ("V", 5), ("IV", 4), ("I", 1)
    ]

    result = ''
    for i in range(13):
        while number - roman_numbers[i][1] >= 0:
            result += roman_numbers[i][0]
            number -= roman_numbers[i][1]

    return result

def float_to_roman(number : str) -> str:
    roman_fraction = [
        (".", ".083"), (":", ".166"), (":.", ".25"),
        ("::", ".333"), ("::.", ".416"), ("S", ".5"),
        ("S.", ".583"), ("S:", ".666"), ("S:.", ".75"),
        ("S::", ".833"), ("S::.", ".916")
    ]

    if len(number) == 0:
        return ''

    for i in range(11):
        if number == roman_fraction[i][1]:
            return roman_fraction[i][0]

    return "ERROR"

def main() -> int:
    #print("Introduce numero a convertir")
    numero = input()

    elements = numero.split('.')
    if len(elements)>2:
        print("ERROR")
        return -1

    part_integer = integer_to_roman(int(elements[0]))
    part_decimal = ''
    if len(elements)==2:
        part_decimal = f".{elements[1]}"
```

```
        part_decimal = part_decimal.strip("0")
        part_decimal = float_to_roman(part_decimal)

    if part_decimal == "ERROR":
        print("ERROR")
        return -1
    print(f"{part_integer}{part_decimal}")

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

22 A New Hope

13 points

Introduction

The galaxy is under the menace of the Empire. R2-D2 must protect the message of the Princess Leia from the clutches of Darth Vader using an encrypting algorithm.

Example:

```
s = help me obi wan kenobi you are my only hope
```

Note that s will only contain characters in the range ascii[a-z] and space, which is ascii(32).

In order to encode the message, the characters are written into a grid, whose rows and columns have the following constraints, being L is the length of this message.

$$\lfloor \sqrt{L} \rfloor \leq row \leq column \leq \lceil \sqrt{L} \rceil, \text{ where } \lfloor x \rfloor \text{ is floor function and } \lceil x \rceil \text{ is ceil function}$$

After removing the spaces, the string is 34 characters long, and the sqrt of 34 is between 5 and 6, so it is written in the form of a grid with 6 rows and 6 columns.

```
helpme
obiwan
kenobi
youare
myonly
hope
```

Note:

- Ensure that rows columns >= L

- If multiple grids satisfy the above conditions, choose the one with the minimum area, i.e. rows columns.

The encoded message is obtained by displaying the characters of each column as a word, with a space between column texts. The encoded message for the grid above is:

```
hokymh ebeoyo linuop pwoane mabrl eniey
```

You are asked to write a program to encode a message as specified.

Input

The input will be a single line of text, that will be the string s.

Note thet the length of s will be no longer than 81 characters.

Output

The output will be a single line of text, that will show the encrypted message.

Example 1

Input

```
have a nice day
```

Output

```
hae and via ecy
```

Explanation

- *L* = 12, sqrt(12) is between 3 and 4

Rewritten with 3 rows and 4 columns:

```
have
anic
eday
```

Python

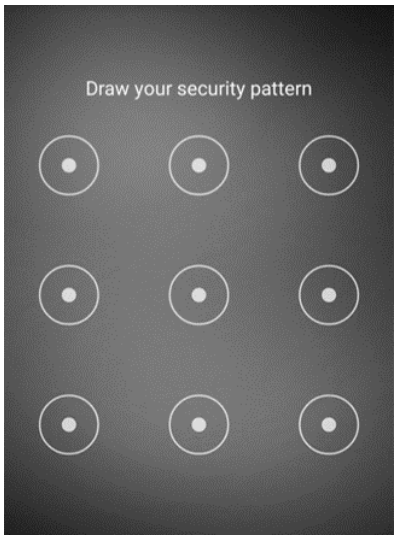
```python
import math

def encryption(s):
    clean_s = s.replace(" ", "")
    c = math.ceil(math.sqrt(len(clean_s)))
    p = ' '.join(map(lambda x: clean_s[x::c], range(c)))

    return p

def main():
    s = input()

    result = encryption(s)

    print(result)

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

23 Lock pattern checker

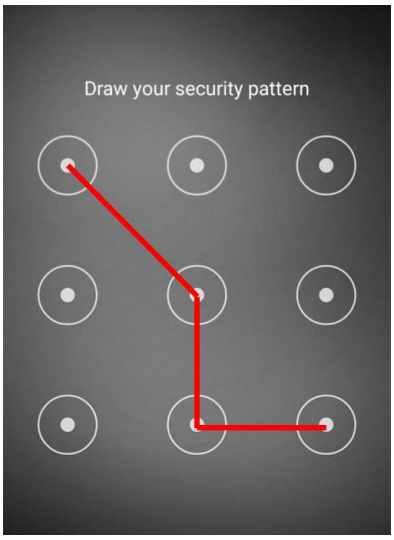
13 points

Introduction

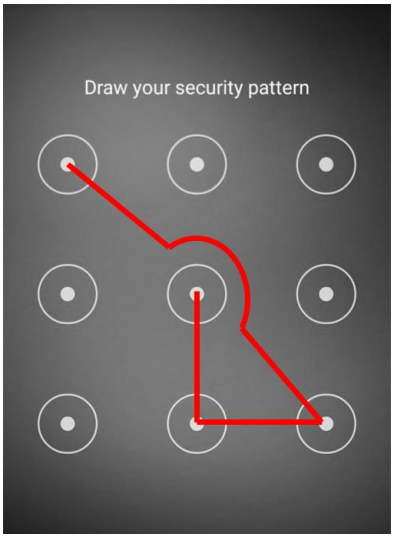
While configuring your new smartphone, you decided to set the safest gesture pattern to unlock it over a grid of 9 circles like this.



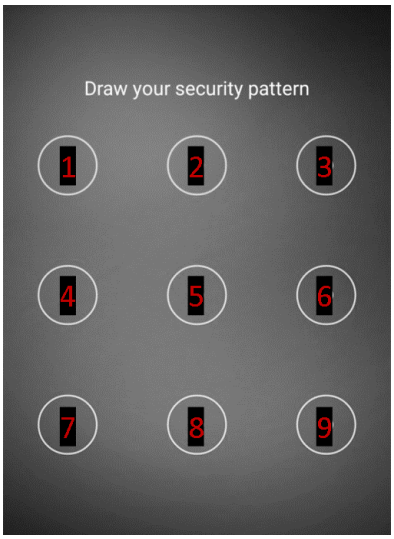
But before setting it you want to be sure that your candidate lock pattern is feasible and follows the restrictions of your smartphone operating system (A circle only can be connected with an adjacent circle and is allowed to connect the same circle more than once) for example a correct pattern is:



There will be patterns that do not follow the OS restrictions like in this example:



So to avoid any surprise you decided to code a program that checks whether a lock pattern is correct. To do so, the grid of 9 circles is numbered following this scheme:



The sequences defined by the lock pattern in the previous examples are:

1-5-8-9 for the correct lock pattern

1-9-8-5 for the incorrect lock pattern

Also take into account that it is necessary to have at least 2 circles connected to have a safe gesture pattern.

Input

The input will be a single line containing a sequence of numbers ranging from 1 to 9 (one per each circle) separated by minus sign.

Output

The output will be a single line stating whether the pattern is ok, otherwise print the first pair of circles that cannot be connected in the pattern.

Example 1

Input

```
1-5-8-9
```

Output

```
Pattern is ok
```

Example 2

Input

```
1-9-8-5
```

Output

```
Circle 1 can not connect with circle 9
```

Python

```
# * * *    1 2 3
# * * * => 4 5 6
# * * *    7 8 9

connectedNodes = [[2,4,5],
                  [1,3,4,5,6],
                  [2,5,6],
                  [1,2,5,7,8],
                  [1,2,3,4,6,7,8,9],
                  [2,3,5,8,9],
                  [4,5,8],
                  [4,5,6,7,9],
                  [5,6,8]]

result = True

# Read the pattern from standard input
pattern = input()

# Get the sequence of nodes followed
nodeSequence = [int(i) for i in pattern.split('-')]

# Get the first node and remove it from the list
nextNode = nodeSequence[0]
nodeSequence.pop(0)

# Process node after node
for i in nodeSequence:

  if not i in connectedNodes[nextNode-1]:
    result = False
    break

  nextNode = i

if result == True:
  print("Pattern is ok")
else:
  print("Circle " + str(nextNode) + " can not connect with circle " + str(i))
```

24 Word search control

15 points

Introduction

You are responsible for creating word searches, and you would like to know the proportion of letters that form part of words compared to the total number of characters in the grid. Be careful, a word can share a letter with other words, so avoid counting them twice.

Words can appear multiple times, from left to right, from right to left, from top to bottom, from bottom to top and in any diagonal direction.

Input

The input is structured as follows:

- First line: a list of the words to search.

- Second line: the number of rows.

- As many strings as the number of rows, representing the text where the words have to be found.

Output

The output must be the percentage of the characters of the grid forming the input words with respect to the total amount of characters in the grid, to exactly 2 decimal places.

Example 1

Input

```
SOL MARIA DAVID LAURA DANI
6
RMLAURA
XYOOVDU
DISRSIQ
AOTDIVD
NVHFHAX
IIANYDT
```

Output

```
The words given occupy 42.86% of the grid.
```

Example 2

Input

```
COW DEER HEN DOG HORSE PIG
7
NDOYEKNO
ZNWOCWRD
ZEDEAEOG
HENEENOD
WRUDKAOG
ZOOETGLI
MHORSESP
```

Output

```
The words given occupy 37.50% of the grid.
```

Python

```python
def findWordInDirection(word, text, charCount, i, j, di, dj):
    ii = i
    jj = j
    count = 0
    while count < len(word) and ii >= 0 and ii < len(text) and jj >= 0 and jj < len(text[ii]) and text[ii][jj] == word[count]:
        count += 1
        ii += di
        jj += dj
    if count == len(word):
        for x in range(len(word)):
            charCount[i + di*x][j + dj * x] = 1
    return

def findWord(word, text, charCount):
    for i in range(len(text)):
        for j in range(len(text[i])):
            findWordInDirection(word, text, charCount, i, j, -1, -1)
            findWordInDirection(word, text, charCount, i, j, -1,  0)
            findWordInDirection(word, text, charCount, i, j, -1,  1)
            findWordInDirection(word, text, charCount, i, j,  0, -1)
            findWordInDirection(word, text, charCount, i, j,  0,  1)
            findWordInDirection(word, text, charCount, i, j,  1, -1)
            findWordInDirection(word, text, charCount, i, j,  1,  0)
```

```
            findWordInDirection(word, text, charCount, i, j,  1,  1)
    return

def main():
    # Parse the input words
    words = input().split()

    # Parse the input text
    N = int(input())
    text = []
    for i in range(N):
        text.append(input())

    # Create a matrix to store which characters are used in any of the input
words
    charCount = [ [0] * len(text[0]) for x in range(len(text)) ]

    # Find the words in the text
    for word in words:
        findWord(word, text, charCount)

    # Count the characters marked with a 1
    count = 0
    for i in range(len(charCount)):
        count += charCount[i].count(1)
    print('The words given occupy {0:.2f}% of the grid.'.format( round(
float(count * 100) / float(len(text) * len(text[0])), 2) ))

    # Use the following lines to print the text and the marked characters
    #for i in range(len(text)):
    #    print(text[i])
    #for i in range(len(charCount)):
    #    print("".join([ str(x) for x in charCount[i] ]))
    return

main()
```

25 From A to Z

15 points

Introduction

In school you memorized the alphabet from “a” to “z” by running the letters in order "abcdef...". It has been the same for thousands of years, but what if the order of the letters were different? For example, if the letter “b” was before the letter “a” or the letter “y” was after the letter “z”. Well, now you have the chance of consider this scenario by coding a program to sort words in an alternative alphabet.

HINT: If two words contain the same letters in the same positions, then the shortest word comes first.

Input

The input consists of several lines ending with a line with a single hashtag sign "#".

- The first line contains the letters of the alphabet with a new order. The position in the line defines the order from left to right.
- The following lines contain a single word.

Output

The output prints in a single line the words ordered according to the new alphabetical order.

Example

Input

```
xdhwjtklnoapqersuivmygzbcf
me
here
crazy
is
alphabet
world
making
hello
this
hereafter
```

#

Output

```
hello here hereafter world this alphabet is making me crazy
```

Python

```python
# Comparison function given two words
# according to the order stored in the dictionary
def isGreaterThan(word1, word2):
  for i in range(min(len(word1), len(word2))):
    if alphabetDict[word1[i]] < alphabetDict[word2[i]]:
      return False
    if alphabetDict[word1[i]] > alphabetDict[word2[i]]:
      return True
  # Finally consider shorter words go before longer words
  if (len(word1) > len(word2)):
    return True
  else:
    return False

#
# Main program
#

# Read the new alphabet and store each element ordered in a dictionary
alphabetOrder = input()
alphabetDict = {}
idx = 0
for i in alphabetOrder:
  alphabetDict[i] = idx
  idx += 1

# Read the list of words to be sorted.
wordsList = []
word = input()
while word != "#":
  wordsList.append(word)
  word = input()

# Selection sort algorithm
for i in range(len(wordsList)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(wordsList)):
```

```
        if isGreaterThan(wordsList[min_idx], wordsList[j]):
            min_idx = j

    # Swap the found minimum element with
    # the first element
    wordsList[i], wordsList[min_idx] = wordsList[min_idx], wordsList[i]

# Finally print out the words ordered
finalOutput = ""
for i in wordsList:
  if finalOutput == "":
    finalOutput = i
  else:
    finalOutput = finalOutput + " " + i
print(finalOutput)
```

26 Mountain words

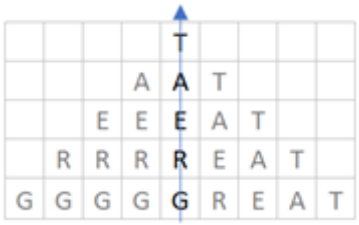
18 points

Introduction

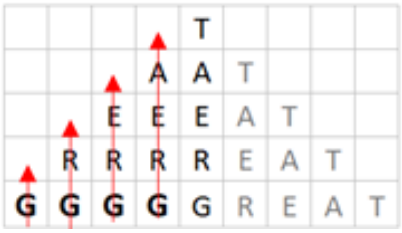
Given a word, let's draw a mountain to create a nice ASCII landscape view.

```
G R E A T  ⟹        T
                  A A T
                E E E A T
              R R R R E A T
            G G G G G R E A T
```

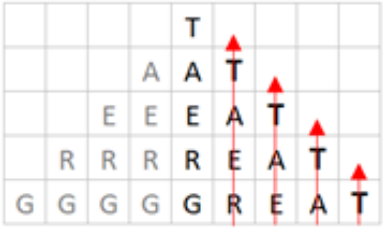
The whole word is printed vertically in the middle column, defining the mountain peak.



Then the left-hand side of the mountain is formed by vertical words that start at the base with the first letter of the word and go up vertically up to the diagonal.



Finally, for the right-hand side of the mountain the reverse principle applies, it's formed by vertical words starting at the base with the appropriate letter so that when going up vertically, they all end with the last letter of the word in the diagonal.



Input

Each line will contain one upper case word up to 12 characters long. The last line of the input contains a single #.

Output

Print the corresponding mountains, one after the other, for given words.

Example 1

Input

```
HAT
GREAT
FOUR
#
```

Output

```
            T
          AAT         R
   T     EEEAT       UUR
  AAT   RRRREAT     OOOUR
HHHAT GGGGGREAT  FFFFOUR
```

Example 2

Input

```
A
#
```

Output

```
A
```

Python

```python
# Read the mountain words
word = input()
longestWord = 0
mountainWords = []
while word != "#":
    if len(word) > longestWord:
        longestWord = len(word)
    mountainWords.append(word)
    word = input()
#print (mountainWords)

# Create the mountains
mountains = []
for word in mountainWords:
    for i in range(1,2*len(word)):
        slice = []
        if i <= len(word):
          for j in range(i):
              slice.append(word[j])
          mountains.append(slice)
        if i > len(word):
          for j in range(2*len(word)-i):
              slice.append(word[i%len(word)+j])
          mountains.append(slice)
    mountains.append([])
#print (mountains)

# Fill missing spaces in the lists with blanks
for i in mountains:
    for j in range(longestWord-len(i)):
        i.append(" ")

# Print the output
for j in range(longestWord):
    output = ""
    for i in mountains:
        output=output+(i[longestWord-j-1])
    output=output.rstrip()
    print(output)
```

27 The space between us

18 points

Introduction

Given the current pandemic situation the number of people permitted indoors must be controlled. This is especially critical in places such an emergency room of a hospital. You are working in a team developing a system to measure the capacity of a room. It consists of a couple of sensors installed at the door that detects flow of people, particularly the entrance and the exit. At the end of the day the system sends the collected data for further analysis. So, let's code a program that analyses this data and finds out the maximum number of people in the room and the time they stay together also reporting a list of names ordered by arrival time. In case the maximum number of people is repeated several times a day it is important to report all the occurrences.

Hint: Note that if a person arrives at the same time that another one leaves, they are not considered to be in the room at the same time since they do not stay at the same time.

Input

The input consists of a variable number of lines:

- The first line contains an integer bigger than 0 that represents the number of people that entered into the emergency room.

- The next lines contain three fields separated by spaces: patient name, arrival and leaving times expressed in 24h format (HH:MM)

Output

The output should be a series of lines like this:

Max capacity of *X* people during *Y* minutes with following people: *list of names present in the room*

where *X* is the maximum number of people present in the room, *Y* is the number of minutes that the *X* persons stay in the room together and the list of the names of people present in the room ordered by arrival time and separated by commas.

Example 1

Input

```
3
Alexandra 09:30 10:30
Bob 09:50 10:20
Carol 09:45 10:15
```

Output

```
Max capacity of 3 people during 25 minutes with following people: Alexandra,
Carol, Bob
```

Example 2

Input

```
6
Ann 07:31 09:36
Billy 07:53 09:19
Charles 07:45 09:15
David 10:20 13:45
Esther 11:30 19:33
Frances 10:55 11:52
```

Output

```
Max capacity of 3 people during 82 minutes with following people: Ann,
Charles, Billy
Max capacity of 3 people during 22 minutes with following people: David,
Frances, Esther
```

Python

```python
# Auxiliar function that returns the arrival time:
def getTime(elem):
  return elem[0]

# Auxiliar function get total amount of minutes from a pair of timestamps.
def getMinutes(t1, t2):
  ArrivalHour = int(t1[0:2])
  ArrivalMinutes = int(t1[3:5])
  LeavingHour = int(t2[0:2])
  LeavingMinutes = int(t2[3:5])
  if (LeavingMinutes < ArrivalMinutes):
    LeavingMinutes += 60
    LeavingHour -= 1
  totalTime = (LeavingHour - ArrivalHour) * 60 + (LeavingMinutes -
ArrivalMinutes)
  return totalTime

# Initialize variables
numMembers = int(input())
timeSlots = []
currCapacity = 0
maxCapacity = 0
timeArrivalMaxCapacity = 0
timeLeavingMaxCapacity = 0
peopleMaxCapacity = []
peopleCurrCapacity = []
outputStringList = []

# Get arrival and leaving time for each person
for i in range(numMembers):
  line = input().split()
  name = line[0]
  arrival = line[1]
  leaving = line[2]
  timeSlots.append([arrival, name, "Arrival"])
  timeSlots.append([leaving, name, "Leaving"])

# Sort the people by arrival time
timeSlots.sort(key=getTime)

# Traverse the list of time slots to look for the max capacity
for i in timeSlots:
  # Add 1 when someone arrives
  if i[2] == "Arrival":
    currCapacity += 1
```

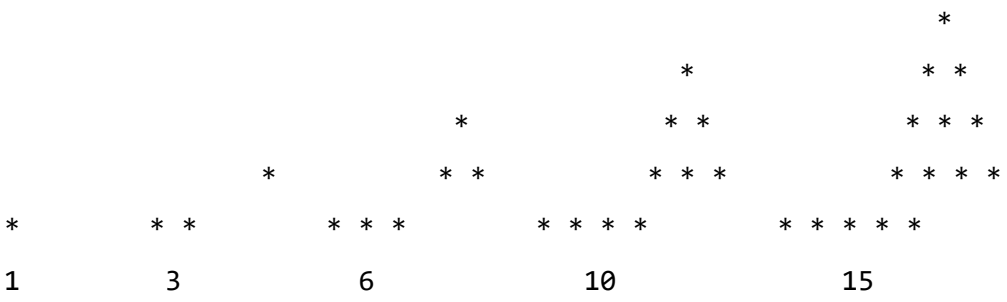
```
    peopleCurrCapacity.append(str(i[1]))

  # Substract 1 when someone leaves
  if i[2] == "Leaving":
    currCapacity -= 1
    peopleCurrCapacity.remove(str(i[1]))
    # Get time when the max capacity ends because someone leaves
    if (currCapacity == maxCapacity -1):
      timeLeavingMaxCapacity = i[0]
      # Get total time in minutes
      totalTime = getMinutes(timeArrivalMaxCapacity, timeLeavingMaxCapacity)
      outputStringList.append("Max capacity of " + str(maxCapacity) + " people
during " + str(totalTime) + " minutes with following people: " + strMaxCapacity)

  # Check max capacity. Whenever new max is detected get time,
  # max capacity and list of names
  if currCapacity >= maxCapacity:
    # Reset the list when a new max number of people is found:
    if currCapacity > maxCapacity:
      outputStringList.clear()
    # Process data corresponding to the maximum detected
    timeArrivalMaxCapacity = i[0]
    maxCapacity = currCapacity
    peopleMaxCapacity = peopleCurrCapacity.copy()
    strMaxCapacity = ', '.join([str(elem) for elem in peopleMaxCapacity])



# Print out the output
for i in  outputStringList:
  print(i)
```

28 Perfect pyramids

18 points

Introduction

A perfect pyramid is a regular pyramid where the 4 sides have the shape of an equilateral triangle, as you can see from the following examples:



As shown in the these examples, there is a relation between the N^{th} pyramid and the required dots to represent it. For example, the 4^{th} pyramid (in the diagram above) is represented by 1 + 2 + 3 + 4 = 10 dots. You can see that the N^{th} pyramid is represented by the sum of all the numbers until N.

Can you write a program to check whether a given number corresponds to the number of dots in a side of a perfect pyramid?

Input

The input will be a single line containing a positive number greater or equal to 1.

Output

The output will print "True" if the input is the number of dots of the side of a perfect pyramid, or "False" if it is not.

Example 1

Input

```
6
```

Output

```
True
```

Example 2

Input

```
5706588888630
```

Output

```
True
```

Example 3

Input

```
5706588888631
```

Output

```
False
```

Python

```
#
# A triangular number t meets the equation t = Sum(1..n) = n*(n+1)/2 that is,
#
#       t = (n^2i +i n) / 2
#
# and we get this quadratic equation:
#
#        n^2 + n - 2t = 0
#
# That can be solved as:
#
#            n = (-1 +/- sqrt(1+8t)) / 2
#
# The negative solution for n is discarded. And the positive solution for n is
#
#            n = (-1 + sqrt(1+8t)) / 2
#
# If t is a triangular number, n must be positive integer number, that follows
#
# a) 1+8t is a perfect square and
#
# b) -1 + sqrt(1+8t) is divisible by 2. This is true since 1+8t is odd and since
#    it is a perfect square the square root is also odd. Finally substracting
#    one unit the result is even
#
# Then to check if t is a triangular number: (1 + 8 * t) == (round(sqrt(1 + 8 *
t)))

import math

t = int(input())

n = 1 + 8 * t
m = round(math.sqrt(n))

#print(n, m**2)

print(n == m**2)
```

29 Deterministic Finite Automaton

19 points

Introduction

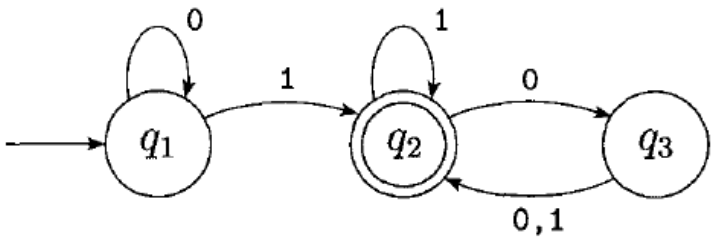
A Deterministic Finite Automaton (DFA) is a simple computational model that accepts or rejects a given string of symbols by running through a state sequence uniquely determined by the string.

A DFA can be formally defined as a 5-tuple (Q, **Σ**, **δ**, q_0, F), where,

1. Q is the finite set of available states,
2. **Σ** is the finite set of accepted symbols, also called the alphabet,
3. δ: Q **X** **Σ** → Q is the transition function,
4. q_0 ∈ Q is the start state, and
5. F ⊆ Q is the set of accepted states.

As said, DFAs are capable of processing strings whose characters are part of the DFA's alphabet. Starting in state q_0, the automaton takes one symbol of the input string at a time and performs the corresponding state transition defined by its δ(q, c) (being q ∈ Q the current state and c ∈ Σ the symbol being processed). The process finishes when the last symbol has been processed. At this point, we say that the DFA accepts the input string if it is in a state q ∈ F. Otherwise, the input is rejected.

As an example, in the following figure it is represented a three-state DFA that we call M.



Formally described:

M = (Q, **Σ**, δ, q_0, F), where

1. Q = {q_1, q_2, q_3},

2. **Σ** = {0, 1},

3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state, and

5. F = {q_2}

Notice that M will accept strings such as 1, 1111, 0010010000 or 0101000111001. Contrary, strings like 0, 0000, 111000 or 1010101010 aren't accepted. Indeed, M recognizes the language of strings that contain at least one 1 and an even number of 0s follow the last 1.

Let's try to build a program that allows us to determine if a DFA formally defined would accept or not a given string.

Input

The input will consist in several rows:

- 1st will come the set of states of the DFA.

- 2nd the alphabet supported by the DFA.

- 3rd the start state.

- 4th the set of accepted states (note that there could exist more than one accepted state).

- N rows, one per state and in order of appearance, defining the transition function for such state using pairs of symbol-destination.

- Finally, the input string.

For the sake of clarity, the input for trying to discern if M (our previous example DFA) would accept 100010 would be:

q1 q2 q3

0 1

q1

q2

0 q1 1 q2

0 q3 1 q2

0 q2 1 q2

100010

Output

The output will be a single sentence depending on the evaluation's outcome:

- If the input string has symbols that aren't from the DFA's alphabet: "Invalid input string!"

- If the input string is accepted: "Input string accepted"

- If the input string is rejected: "Input string rejected"

- If the definition does not correspond with a DFA: "This is not a deterministic finite automaton!"

Note that the later situation may arise when:

- A transition involving a symbol not existing in the DFA's alphabet is defined.

- A transition involving a state not existing in the set of states of the DFA is defined.

- There isn't exactly a single transition defined for each pair of state-symbol. If that rule is broken we would be in front of a Nondeterministic Finite Automaton (NFA), so we better leave that topic for another year :)

Example 1

Input

```
q1 q2 q3 q4
0 1
q1
q4
0 q2 1 q1
0 q3 1 q1
0 q3 1 q4
0 q4 1 q4
011001100
```

Output

```
Input string accepted
```

Example 2

Input

```
s q1 q2 r1 r2
a b
s
q1 r1
a q1 b r1
a q1 b q2
a q1 b q2
a r2 b r1
a r2 b r1
baba
```

Output

```
Input string rejected
```

Example 3

Input

```
q1 q2
a b c
q1
q1
a q2 b q1 c q2
a q1 b q1
abc
```

Output

```
This is not a deterministic finite automaton!
```

Python

```python
def dfa_execution():
    states = str(input()).split()
    alphabet = str(input()).split()
    start_state = str(input())
    accepted_states = str(input()).split()

    transition_function = {}
    for s in states:
        transitions = str(input()).split()
        if len(transitions)/2 != len(alphabet):
            return False

        transition_function[s] = {}
        for t in range(0, len(transitions), 2):
            if transitions[t] not in alphabet:
                return False
            if transitions[t] in transition_function[s].keys():
                return False
            if transitions[t + 1] not in states:
                return False
            transition_function[s].update({transitions[t] : transitions[t + 1]})

    current = start_state
    input_string = str(input())
    for i in input_string:
        if i not in alphabet:
            print("Invalid input string!")
            return True
        current = transition_function[current][i]

    if current in accepted_states:
        print("Input string accepted")
    else:
        print("Input string rejected")
    return True

if not dfa_execution():
    print("This is not a deterministic finite automaton!")
```

30 Circle of primes

28 points

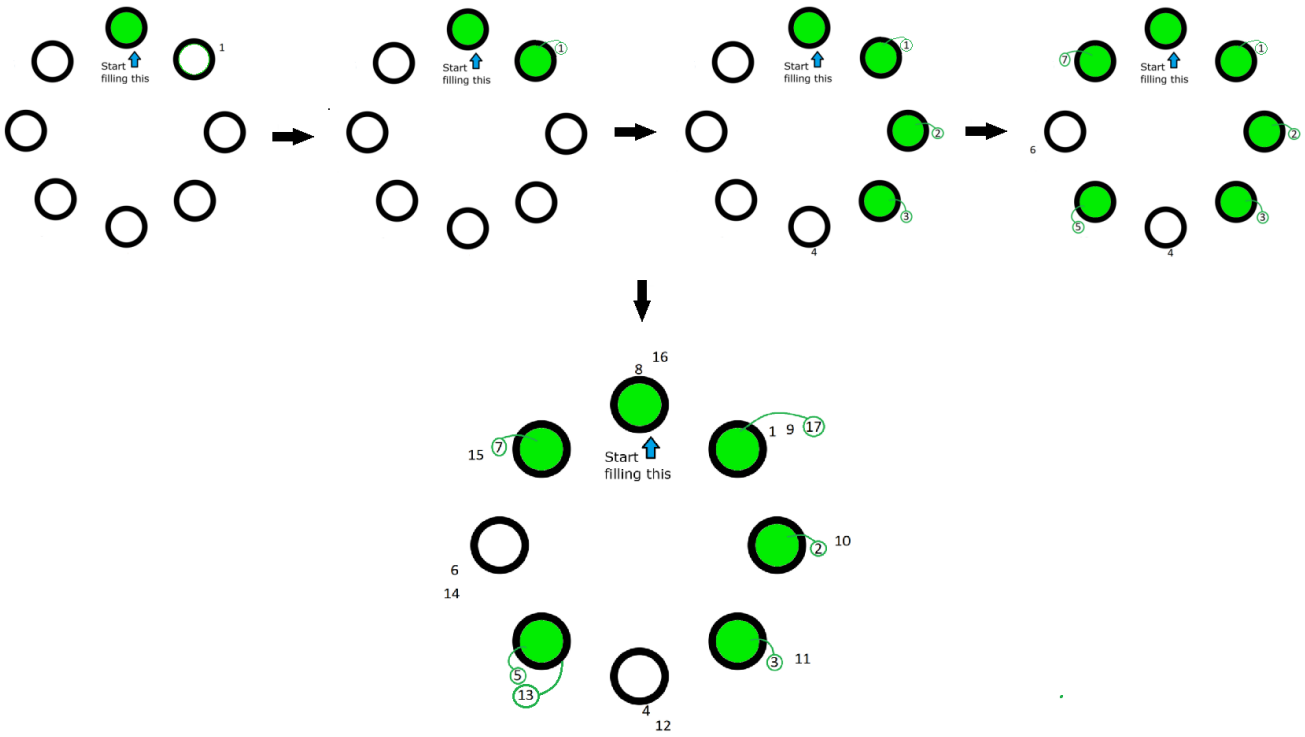
Introduction

Pol works in a chemical factory and these days he is in charge of preparing the containers to be filled by the centrifuges. The number of containers to be prepared for a given centrifuge is determined by its size and the product inside.

But since preparing those containers is a monotonous, boring task, Pol thought of a game to entertain himself while doing the work:

For a given centrifuge he takes the necessary number of containers and arranges them in a circle. He then begins preparing one of them. Once finished, he starts counting the containers beginning with the first, advancing one container at a time, clockwise. When the count arrives to a prime number, he prepares that container (if it is not prepared already) and continues counting. That way, he goes in circles around the arranged wheel of containers until he prepares them all.

Note that containers already prepared are included in the count.



When playing this game, Pol observed that, depending on the number of containers forming the circle, some of them wouldn't be prepared no matter how many times he went around the circle. If the number of containers in the circle is not prime then not all of the containers will be filled.

Pol would like to know beforehand if a given circle of containers could be completed using the game. And, in case it couldn't, which containers would be left unprepared, so that he could prepare them in advance. Can you write a program to help him figure it out?

Note: Pol needs to do his job, so the program can't take a long time to compute the solution.

Input

The input consists of two lines. The first line will be the number of centrifuges that need to be emptied.

The second line will have as many numbers as centrifuges indicated in the first line. The numbers in the second line will be the number of containers needed for each centrifuge (so the number of elements in the circle when playing the game).

Output

The output should be a line for each centrifuge indicating whether all the containers in the circle will be prepared or not. And, in the latter case, a sentence indicating the positions of the containers that won't be prepared.

If the circle can be filled the line will state: "The circle can be completed".

If the circle cannot be filled the line will state: "The circle cannot be completed. Container/s X,Y,Z, won't be prepared.", being X,Y,Z,A,B... the positions of the containers that will be never be filled.

Example 1

Input

```
3
5 9 17
```

Output

```
The circle can be completed.
The circle cannot be completed. Container/s 6, won't be prepared.
The circle can be completed.
```

Example 2

Input

```
2
22 23
```

Output

```
The circle cannot be completed. Container/s 4, 6, 8, 10, 12, 14, 16, 18, 20,
won't be prepared.
The circle can be completed.
```

Python

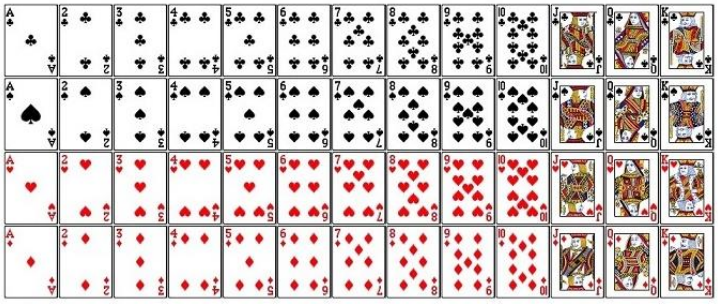
```python
def isPrime(num):
    if num == 1:
        return False
    for i in range(2,num):
        if num % i == 0:
            return False
    return True

def findDivisors(num, divisors):
    for i in range(2,num):
        if num % i == 0:
            divisors.append(i)

def GCDgreaterThanOne(divisorsN, divisorsK):
    for divisor in divisorsK:
        for d in divisorsN:
            if divisor == d:
                return True
    return False

numCentrifuges = input()
circlesSize = list(map(int,input().split(' ')))

for size in circlesSize:
    if isPrime(size) or size == 4:
        print("The circle can be completed.")
    else:
        print("The circle cannot be completed. Container/s ", end="")
        divisorsN = []
        findDivisors(size, divisorsN)
        for k in range(1,size):
            if isPrime(k):
                continue
            divisorsK = []
            findDivisors(k, divisorsK)
            if GCDgreaterThanOne(divisorsN, divisorsK):
                k2 = str(k)+", "
                print(k2, end="")
        print("won't be prepared.")
```

31 Blackjack

30 points

Introduction

Blackjack is a famous card-based game of chance. It is played using a poker deck. Each card represents its numeric value, except figure cards (Jack, Queen and King) that all score the same 10 points. The Ace card can score either 1 or 11 points depending on what is of most benefit to the player.



The goal is to reach 21 points without going over. At the beginning of the game the player is dealt 2 cards, and the croupier, who represents the casino bank, is dealt 1 card. From that moment, the player can request another card as many times as they want until they reach 21 or go over. Alternatively, the player can stand (i.e., take no more cards) and stay with the cards they already have. If the player stands with 21 or less, then the croupier begins to deal themselves cards until they reach 17. The croupier will ask for another card while their sum is less than 17 and stand if their sum is 17 or greater. Once both have been planted or, have gone over 21, the winner of the hand is decided.

If the player has gone over 21, they lose, regardless of what the croupier has. If the croupier has gone over 21 and the player has not, the player wins. Whenever both are at 21 or less, the one with the highest amount wins. If both have the same sum, the hand ends in a draw. The only exception would be if either the player or the croupier achieves the sum of 21 by using only two cards, this is called a 'Blackjack', in this case it is the 'Blackjack' holder that always wins the hand.

As said before, note that the score of each Ace could be either 1 or 11, it always gives the best advantage for winning. For example, with the cards Ace, plus a 9, the Ace will score 11 (11 + 9 = 20). But with cards Ace, plus 9, plus 5, the Ace will score 1 (1 + 9 + 5 = 15), otherwise the sum is bigger than 21 (11 + 9 + 5 = 25).

Can you find out the odds that I have to win, draw, and lose if I stand with a certain sum, knowing which is the initial card the croupier has and how many cards of each type are still to come out?

A very simple case would be, for example, the player is planted with 17, the croupier's card is 10 and, in the deck remains 1 seven, 2 nines and 1 five. In this case, the player's odds of losing would be 50%, the odds of a draw would be 25% and those of a win 25%.

Input

- The first line will be S, a number storing the total sum got when the player is planted. A "B" letter, meaning a Blackjack, will appear instead of a number when the total sum is 21 with two cards.

- Second line is C a number that represents the value of a croupier's initial card.

- The next ten lines will be Dn, that is the number of cards of each type (Ace, two, three, ..., nine, ten plus figures) that remain in the deck. Notice that more than one deck can be used in the game.

Output

The output will be three lines. The first line will contain the probability of the player winning, the second the probability of the player being in a tie, and the third the probability of the player losing. All probabilities must be expressed with one decimal, truncating the remaining decimal places (do not round the number).

Restrictions to be considered

$3 < S < 22$

$0 < C < 11$

$0 <= Dn <= 128$

Example 1

Input

```
17
10
0
0
0
0
1
0
1
0
2
0
```

Output

```
25.0
25.0
50.0
```

Example 2

Input

```
13
2
24
24
24
24
24
24
24
24
24
128
```

Output

```
39.9
0.0
60.0
```

C

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <math.h>

uint8_t blackjack_jugador = 0;

void calc_blackjack(uint32_t *array, uint32_t suma, uint32_t as, uint32_t
jugador, uint32_t ind, uint8_t n_croupier,  double *prob) {

    // Copia local del array
    uint32_t local[10];
    memcpy(local, array, sizeof(uint32_t) * 10);

    // Robamos la carta
    if (ind < 10) {
        //printf("Carta robada: %d\n", ind+1);
        local[ind]--;
        if (ind == 0) {
            // Hemos robado AS
            as++;
        } else {
            // Suma el valor correcto.
            suma += ind+1;
        }
        n_croupier++;
    }

    uint32_t suma_max = as ? suma + as + 10 : suma;
    uint32_t suma_min = suma + as;

    // Calculamos las cartas en la baraja
    uint32_t n_cartas = 0;
    for (uint32_t i = 0; i < 10; i++) {
        n_cartas += local[i];
    }

    // Utilizamos la suma max para decidir si tomamos otra carta.
    // Parte recursiva
    if (((suma_max < 17) || ((suma_max > 21) && (suma_min < 17))) && n_cartas) {

        // Pedimos carta.
        for (uint32_t i = 0; i < 10; i++) {
```

```
            if (local[i] > 0) {
                // Podemos tomar esta carta
                double prob_camino = ((double)local[i])/((double)n_cartas);

                double prob_subcamino[3];

                // Inicializacion prob_subcamino
                for (uint32_t j = 0; j < 3; j++) {
                    prob_subcamino[j] = 0;
                }

                calc_blackjack(local, suma, as, jugador, i, n_croupier,
prob_subcamino);

                // Actualizar Probabilidades de salida
                for (uint32_t j = 0; j < 3; j++) {
                    prob[j] += prob_camino * prob_subcamino[j];
                }

            }
        }
    } else {
        // Parte No recursiva
        if ((suma_max == 21) && (n_croupier == 2)) {
            if (blackjack_jugador == 1) {
                //Empate
                prob[2] = 0.0;
                prob[1] = 1.0;
                prob[0] = 0.0;
        //printf("=\n");
            } else {
                prob[2] = 1.0;
                prob[1] = 0.0;
                prob[0] = 0.0;
        //printf("-\n");
            }
        } else if (blackjack_jugador) {
                prob[2] = 0.0;
                prob[1] = 0.0;
                prob[0] = 1.0;
        //printf("+\n");
        } else {
            if (suma_min > 21) {
                prob[2] = 0.0;
                prob[1] = 0.0;
                prob[0] = 1.0;
        //printf("+\n");
```

```
            }
            else if (suma_max > 21) {
                if (suma_min > jugador) {
                    prob[2] = 1.0;
                    prob[1] = 0.0;
                    prob[0] = 0.0;
            //printf("-\n");
                }
                else if (suma_min == jugador) {
                    prob[2] = 0.0;
                    prob[1] = 1.0;
                    prob[0] = 0.0;
            //printf("=\n");
            } else {
                    prob[2] = 0.0;
                    prob[1] = 0.0;
                    prob[0] = 1.0;
            //printf("+\n");
                }
            }
            else {
                if (suma_max > jugador) {
                    prob[2] = 1.0;
                    prob[1] = 0.0;
                    prob[0] = 0.0;
            //printf("-\n");
                }
                else if (suma_max == jugador) {
                    prob[2] = 0.0;
                    prob[1] = 1.0;
                    prob[0] = 0.0;
            //printf("=\n");
                } else {
                    prob[2] = 0.0;
                    prob[1] = 0.0;
                    prob[0] = 1.0;
            //printf("+\n");
                }
            }
        }
    }
}

int main() {

    uint32_t jugador, croupier, nums[10];
    char jugador_char[10];
```

```
    uint8_t n_croupier = 1;

    // Lectura de datos
    if (!scanf("%d", &jugador)) {
    if (scanf("%s", jugador_char)) {
        if (jugador_char[0] == 'B') {
            blackjack_jugador = 1;
            jugador = 21;
        }
    }
    }

    scanf("%d", &croupier);
    for (int i = 0; i < 10; i++) {
        scanf("%d", &(nums[i]));
    }

    uint32_t as = 0;
    if ((croupier == 1) || (croupier == 11)) {
        croupier = 0;
        as = 1;
    }

    // Probabilidades obtenidas
    double prob[3];
    // Inicializacion prob
    for (uint32_t j = 0; j < 3; j++) {
        prob[j] = 0;
    }

    calc_blackjack(nums, croupier, as, jugador, 10, n_croupier, prob);

    for (int i = 0; i < 3; i++) {
        prob[i] = floor(prob[i]*1000)/10;
    }

    printf("%.1f\n%.1f\n%.1f\n", prob[0], prob[1], prob[2]);

    return 0;
}
```

32 Ciphers and letters

31 points

Introduction

Charly is making some puzzles for a newspaper. In this puzzle, some letters are replaced by digits, and the objective is to figure out the value of each letter. The puzzle consists of a sum of two numbers.

For example:

```
 DAME
+
 MAS
------
AMOR
```

Each letter can only be replaced by a different digit (from 0 to 9). In this example the value of the letters in the solution is:

D = 8

A = 9

M = 5

E = 6

S = 1

O = 4

R = 7

as DAME + MAS = 8956 + 591 = 9547, which is AMOR.

Note: For this example, there's more than one solution, for example 8951 + 592 = 9543

In the tests used to check your code there will be only one possible solution.

Before sending the puzzles to the newspaper Charly needs to check if they are correct. Could you help him?

Exercise

We ask you to write a program that receives three words (first addend, second addend and result):

DAME

MAS

AMOR

and returns a line like with

- if there is a solution, the numeric values of the sum and the result all in a row:

8956+591=9547

- if there is no solution, the program will return the text

NO SOLUTION

Constraints for the words:

- Words must be only letters and must be capital letters.

- Words cannot be longer than 10 letters.

- Only letters from english alphabet (no Ñ or Ç, for example).

Input

The input will be three lines, each one with one word

```
<first addend>
<second added>
<result>
```

Output

If there is a solution:

A line representing the sum, using only numbers and the symbols '+' and '='

first added as number+second added as number=result as number

If there is no solution:

NO SOLUTION

Example 1

Input

```
LET
LEE
ALL
```

Output

```
156+155=311
```

Example 2

Input

```
GIVEN
ME
MORE
```

Output

```
NO SOLUTION
```

Python

```python
# for debug
def corch(s, i):
    return s[:i] + '[' + s[i] + ']' + s[i+1:]

# for debug
def printCorch(A, B, R, solution, n):
    print(corch(A,n), '\t', corch(decrypt(A, solution), n))
    print(corch(B,n), '\t', corch(decrypt(B, solution), n))
    print(corch(R,n), '\t', corch(decrypt(R, solution), n))

def decrypt(s, sol):
    ns = ''
    for c in s:
        ns += sol[c] if c in sol else c
    return ns

def getOptions(c, solution, options):
    if c == '_':
        return c
    elif c in solution:
        return solution[c]
    else:
        return options

def getOptionsB(b, b_opt, a, va):
    if a == b:
        return va
    else:
        return sublist(b_opt, va)

def getOptionsR(r, r_opt, a, va, b, vb):
    if r == a:
        return va
    elif r == b:
        return vb
    else:
        return sublist(r_opt, [va, vb])

# Return list a without elements in list b
def sublist(a, b):
    if len(a) == 1:
        return a
    else:
        return [x for x in a if x not in b]
```

```
# transform string to digit (note that special char '_' is mapped to 0)
def digit(c):
    if c == '_':
        return 0
    else:
        return int(c)

def printSolution(A,B,R, solution):
    A = A.replace('_', '')
    B = B.replace('_', '')
    R = R.replace('_', '')
    print(decrypt(A,solution) + '+' + decrypt(B,solution) + '=' +
decrypt(R,solution))
    #print(A + '+' + B + '=' + R)
    #print('')

def solve(A, B, R, carry, options, solution, n, allSolutions=True):
    # A, B and R must have same length
    if len(A) != len(B) or len(A) != len(R):
        print("ERROR!!!")
        return False

    # If no more columns and carry is zero...
    if n == -1 and carry == 0:

        # ...we found a valid solution!
        printSolution(A, B, R, solution)
        global numSolutions
        numSolutions += 1
        return True

    # Get current column
    a = A[n]
    b = B[n]
    r = R[n]

    # DEBUG
    if 0:
        print("~~~~~ Column " + str(n) + "  ~~~~~~~~~~~~")
        printCorch(A, B, R, solution, n)

    # Get options for each letter. Use solution only if exists
    a_opt = getOptions(a, solution, options)
    b_opt = getOptions(b, solution, options)
    r_opt = getOptions(r, solution, options)
```

```
    solved = False

    # Test all possible combinations
    for va in a_opt:
        for vb in getOptionsB(b, b_opt, a, va):
            for vr in  getOptionsR(r, r_opt, a, va, b, vb):

                # Check if combination works, taking into account the carry from
previous column
                sum = digit(va) + digit(vb) + carry
                if sum % 10 == digit(vr):
                    #print('--------------------------------')
                    #print(a,'=',digit(va),
b,'=',digit(vb),'carry',carry,r,'=',digit(vr))
                    # Current solution is valid, prepare for recursion
                    # Compute new carry
                    newCarry = 1 if sum > 9 else 0
                    # Add to solution current combination
                    newSolution = solution.copy()
                    newSolution[a] = va
                    newSolution[b] = vb
                    newSolution[r] = vr
                    # Remove from options current combination
                    newOptions = options.copy()
                    newOptions = sublist(newOptions, [va, vb, vr])


                    #printCorch(A, B, R, solution, n)
                    #print('--------------------------------')

                    # Solve for next column
                    if solve(A, B, R, newCarry, newOptions, newSolution, n-1):
                        solved = True

                        if not allSolutions:
                            return True

    return solved

def crypto(A, B, R):

    letters = set()
    for c in A+B+R:
        if not c in letters:
            letters.add(c)

    # More letters than digits
```

```
    if len(letters) > 10:
        print("NO SOLUTION")
        return

    # Make all inputs of the same size
    n = max([len(A), len(B), len(R)])
    Ap = "_" * (n - len(A)) + A
    Bp = "_" * (n - len(B)) + B
    Rp = "_" * (n - len(R)) + R

    # Init empty solution
    solution = {}

    # Digits still usable
    options = ['0','1','2','3','4','5','6','7','8','9']

    solved = solve(Ap, Bp, Rp, 0, options, solution, n-1, True)

    if numSolutions > 1:
        print("INVALID INPUT, MORE THAN 1 SOLUTION")
    elif not solved:
        print("NO SOLUTION")

# Global
numSolutions = 0
crypto(input(),input(),input())
```

33 Castellers competition

32 points

Introduction

In Catalonia, there is a tradition called "Castellers", which consists of building a tower made of people. There are several competitions related to this tradition, in which different teams of "Castellers" from all Catalonia try to build the most complex human tower. Can you implement a program that can evaluate each tower and decide who is the winner of the competition?

Input

A list of lines, where each line contains the name of a team followed by a '-' and the name of the tower built by that team. The name of the tower is what defines the tower shape, and the name are 3 words that follow these rules:

- The first word defines the width of the tower (i.e the amount of people at each level of the tower). This first word may be a number or a word, and if it is a word it can be "pilar" to define a tower of a 1 person per level or "torre" to define a tower of 2 people per level. If it is a number, it is always a positive number.

- The second word is always "de", it does not define anything about the shape of the tower.

- The third word is a number that defines the height of the tower (including the ground level), which will be always greater than or equal to 4.

- The name may be followed by "amb folre" to indicate that the second level (the one over the ground level) of the tower has extra people.

- If there is "amb folre" after the name, it may also contain "amb manilles" to indicate that the third level of the tower has extra people too.

Output

The representation of the human towers for all the teams, followed by the score of each team and the name of the winner or winners.

As you can see in the examples below, there are some rules to properly draw the human towers:

- Each person on the tower is represented with the character "|" and, in the same level, is separated by a single space " "

- Each tower is sepparated with four "_" at each side of the tower.

- The base has to be wider than the tower, and it is represented with 2 extra people at each side, and the characters "/" and "\" at each side too.

- The "amb folre" option is represented with the first level being as wide as the ground level.

- The "amb manilles" option is represented as the "amb folre", but with 1 less person per side.

- The last level of the tower is ALWAYS just a single person, called "anxaneta". Is represented always on the center of the tower

- The rest of the levels have as many people as the width of the tower.

Regarding the score, it is computed following these rules:

- If the width of the tower is less than 10, add "10 - width" points for each level of the tower.

- If the width of the tower is greater or equal than 10, add "1" points for each level of the tower.

- If the tower has "amb folre", subtract 5 points.

- If the tower has "amb manilles", subtract 8 points.

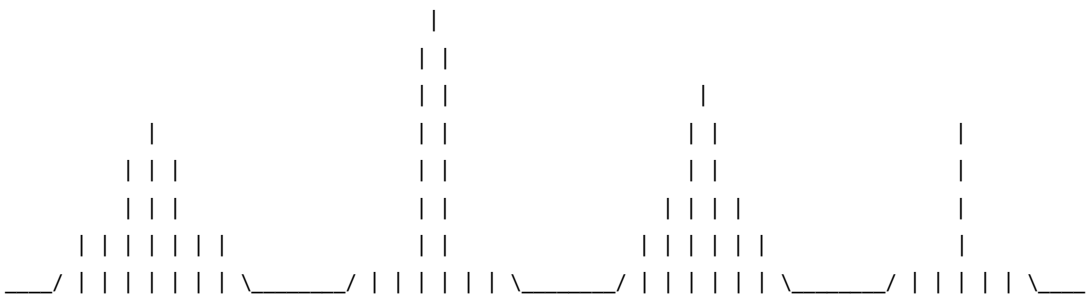
- The score must be greater or equal than 0.

Example 1

Input

```
La colla castellera - 3 de 5 amb folre
The Catalan Lions - 2 de 8
En Pinxu i en Panxu - torre de 6 amb folre amb manilles
Olakase - pilar de 5
```

Output



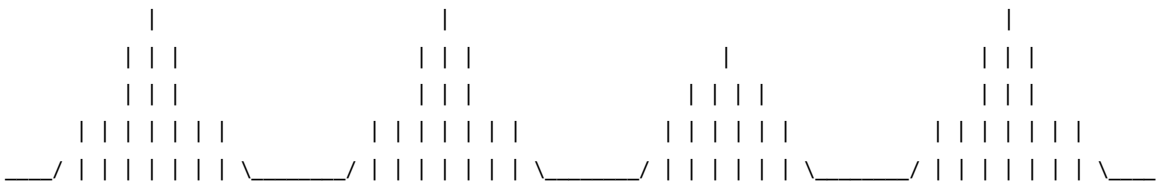
```
La colla castellera: 30
The Catalan Lions: 64
En Pinxu i en Panxu: 35
Olakase: 45
The winner is The Catalan Lions with 64 points.
```

Example 2

Input

```
La colla castellera - 3 de 5 amb folre
The Catalan Lions - 3 de 5 amb folre
En Pinxu i en Panxu - torre de 4 amb folre amb manilles
Olakase - 3 de 5 amb folre
```

Output



```
La colla castellera: 30
The Catalan Lions: 30
En Pinxu i en Panxu: 19
Olakase: 30
The winners are La colla castellera, The Catalan Lions and Olakase with 30
points.
```

Python

```python
import sys

def printCastles(castles):
    # Get max height
    maxHeight = 0
    for castle in castles:
        maxHeight = max(maxHeight, len(castle[1]) - 1)

    # Loop over the castles, printing each row of the castle from top to bottom
    for row in range(maxHeight - 1, -1, -1):
        for castle in castles:
            if row == 0:
                print("____" + castle[1][min(row, len(castle[1]) - 1)] + "____",
end="")
            else:
                print("    " + castle[1][min(row, len(castle[1]) - 1)] + "    ",
end="")
        print()

    # Print the teams and their scores
    winners = []
    winnersScore = 0
    for castle in castles:
        print(castle[0] + ": " + str(castle[2]))
        if castle[2] == winnersScore:
            winners.append(castle[0])
        elif castle[2] > winnersScore:
            winners = [castle[0]]
            winnersScore = castle[2]
    assert(len(winners) > 0)
    if len(winners) == 1:
        print("The winner is " + winners[0] + " with " + str(winnersScore) + "
points.")
    else:
        print("The winners are ", end="")
        for i in range(len(winners) - 1):
            if i > 0:
                print(", ", end="")
            print(winners[i], end="")
        print(" and " + winners[len(winners) - 1] + " with " + str(winnersScore)
+ " points.")
    return

def buildCastle(height, width, withFolre, withManilles):
    castle = []
```

```
    for row in range(height):
        if row == 0:
            castleRow = "/ | | " + " ".join(["|"] * width) + " | | \\"
        elif row == 1 and withFolre:
            castleRow = "  | | " + " ".join(["|"] * width) + " | |  "
        elif row == 2 and withManilles:
            castleRow = "    | " + " ".join(["|"] * width) + " |    "
        elif row == height - 1:
            castleRow = "      " + " " * (width - 1) + "|" + " " * (width - 1) +
"      "
        else:
            castleRow = "      " + " ".join(["|"] * width) + "      "
        castle.append(castleRow)
    castle.append("      " + " ".join([" "] * width) + "      ")
    return castle

def main():
    castles = []

    # Parse input and create each of the input castles
    for line in sys.stdin.readlines():
        parts = line.rstrip().split("-")
        team = parts[0].rstrip()
        towerWords = parts[1].split()
        # Parse width
        if towerWords[0] == "pilar":
            width = 1
        elif towerWords[0] == "torre":
            width = 2
        else:
            width = int(towerWords[0])
        # Parse height
        height = int(towerWords[2])
        # Parse folre
        if len(towerWords) >= 5 and towerWords[3] == "amb" and towerWords[4] ==
"folre":
            withFolre = True
        else:
            withFolre = False
        # Parse manilles
        if len(towerWords) >= 7 and towerWords[5] == "amb" and towerWords[6] ==
"manilles":
            withManilles = True
        else:
            withManilles = False
        # Compute the score
```

```
        score = max(height * max(10-width, 1) - withFolre * 5 - withManilles *
8, 0)
        # Build the castle
        castles.append([team, buildCastle(height, width, withFolre,
withManilles), score])

    # Print the castles
    printCastles(castles)

    return

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

34 Hoverboard Olimpics

38 points

Introduction

The amazing Hoverboard Olimpics have arrived for one more year one year more. The participants are super excited to give their best and be the winners of the competition. The one with the higher score will be the winner. Could you help the judges to evaluate each participant?

Input

The first line is a number with the height of the map, which will be followed by that amount of lines. These lines describe the terrain of the competition, and they will only contain the character '_', '/', '\' and '|'. The character '|' indicates the finish line of the race, which is placed at the end of all the lines of the map.

Then, there will be a list of participants, where each participant will start with the keyword "Participant", followed by the name and the number of actions that they do during the race. Each action is defined by an integer representing the horizontal position at which the action is executed and the name of the action. The horizontal position starts at 0 and will be always less than the length of the race map lines.

There are 3 types of actions:

- ramp-up: this action must occur in the previous position just before a '/', and the

participant must then follow the '/' characters until the end of this ramp.

- ramp-down: this action must occur in the previous position just before a '\', and the participant must then follow the '\' characters until the end of this ramp.

- jump: this action may occur at any position, and the participant will go one row up in the following position and will continue on that row until the jump finishes. This action is followed by a positive number indicating the length of the jump.

- loop: this action must occur in any of the '_' characters of a valid loop (which is a closed sequence as you can see in the examples below) and the participant will follow the whole loop.

Tips:

- Only one action is executed at a time (example: no ramp-up will be executed during a jump).

- You can assume that all actions are valid (example: "ramp" actions will always be in positions just before '/' or '\').

- You can also assume that the top row of the map is always an empty line ending with the '|', so the participant will be always inside the map even if they do a jump in the highest '_' of the map.

- All the lines of the map are of the same length (so the map is always a 2D rectangle).

- A participant may take the same loop twice, but at different positions

Output

For each participant, the map describing the path that they follow and their score. Use the symbol '.' to indicate the jumps of the participant.

The score is very easy to compute: it is just adding the height of the participant at each position, including during the execution of an action.

Tips:

- The participants use a hoverboard, so they 'hover', but they do not 'fly'! So their path must always be on a ground character '_' when they are not in the middle of an action. For example, if the jump of a participant ends and that ending position has no ground, the participant will immediatley fall until some ground is reached.

- Note that '/' and '\' are used to create ramps, but also loops, so the same character can be part of a ramp and a loop at the same time, it depends on the path of the participant.

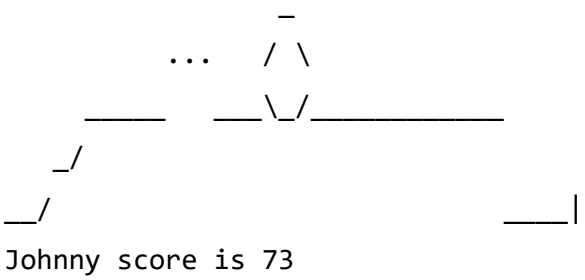
Example 1

Input

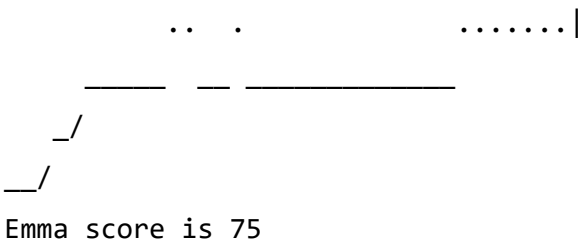
```
7
                                  |
                      __          |
                  _     /  \      |
                 / \   /    \     |
     _____  ____\_/___\_____\__   |
   _/__    ____     / \\     \    |
__/____\____________\_/_\____/____|
Participant Johnny
```

```
Action 1 ramp_up
Action 3 ramp_up
Action 9 jump 3
Action 17 loop
Participant Emma
Action 1 ramp_up
Action 3 ramp_up
Action 9 jump 2
Action 13 jump 1
Action 27 jump 10
Participant Colton
Action 1 jump 3
Action 6 ramp_down
Action 10 jump 4
Action 21 loop
Participant Sakura
Action 1 ramp_up
Action 21 loop
Action 25 loop
Action 26 loop
Action 27 loop
Action 28 loop
```

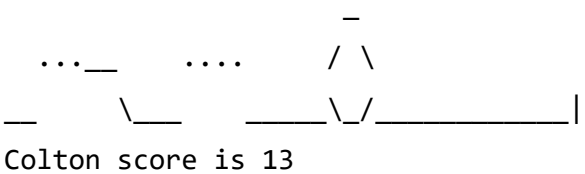
Output



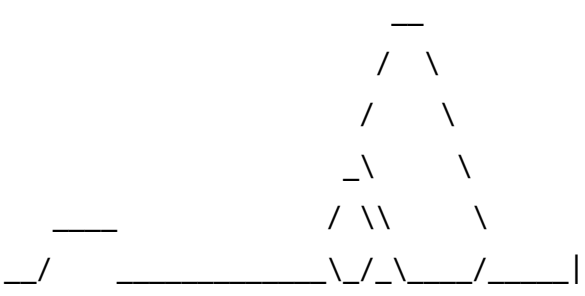
Johnny score is 73



Emma score is 75



Colton score is 13

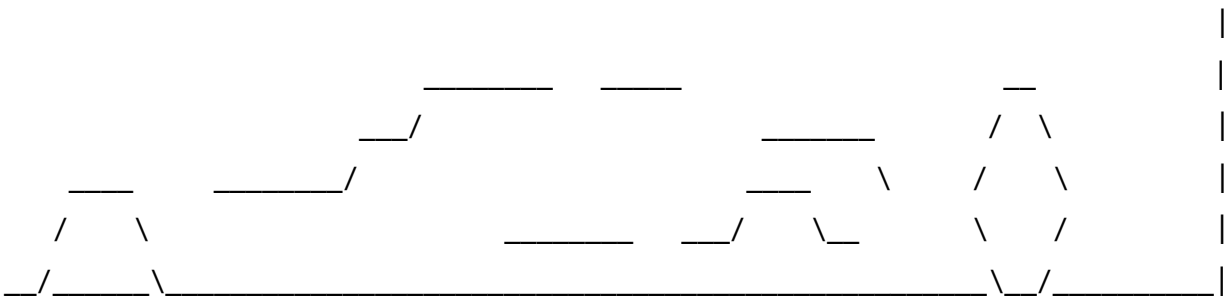


Sakura score is 128

Example 2

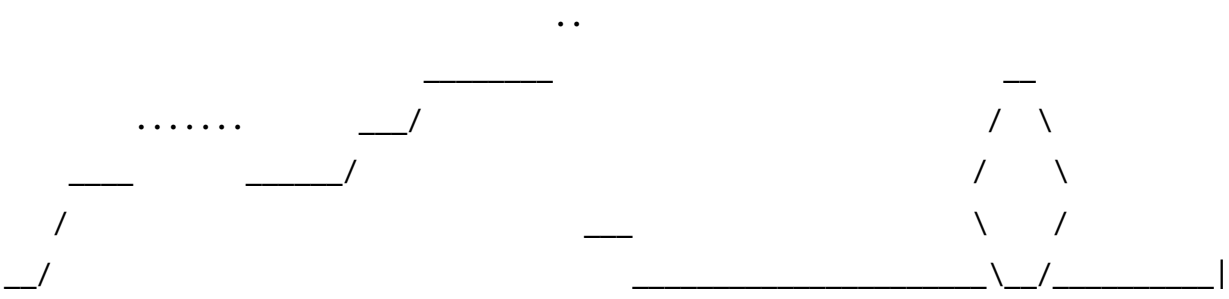
Input

6

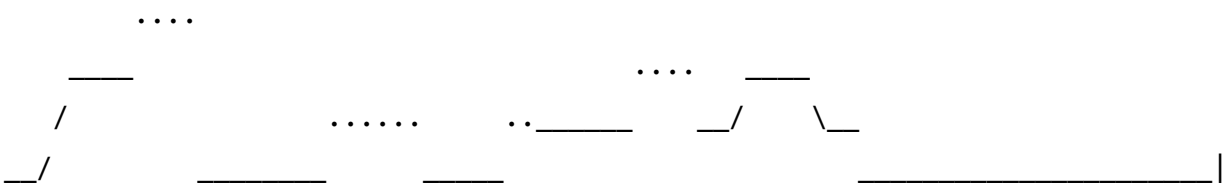


```
Participant Natasha
Action 1 ramp_up
Action 7 jump 7
Action 20 ramp_up
Action 24 ramp_up
Action 33 jump 2
Action 62 loop
Action 63 loop
Participant John
Action 1 ramp_up
Action 7 jump 4
Action 19 jump 6
Action 30 jump 2
Action 38 jump 4
Action 44 ramp_up
Action 49 ramp_down
Participant Alex
Action 1 ramp_up
Action 7 jump 5
Action 20 ramp_up
Action 24 ramp_up
Action 33 jump 4
Action 41 jump 6
Action 63 loop
```

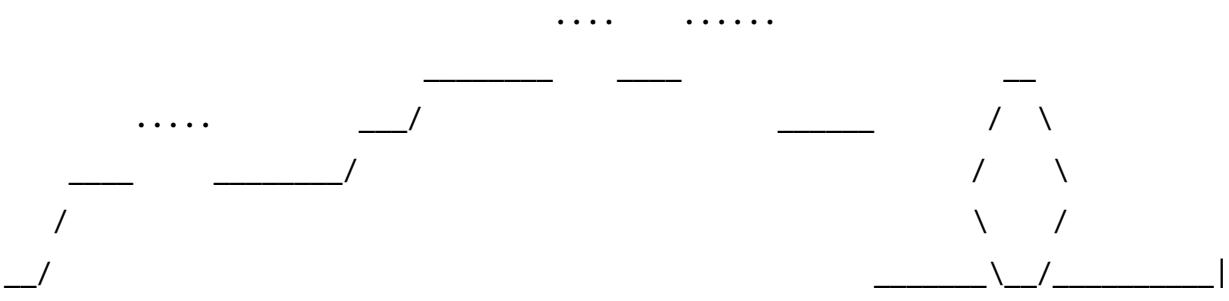
Output



```
Natasha score is 141
```



```
John score is 57
```



```
Alex score is 190
```

Python

```
class Action:
    def __init__(self, params):
        self.pos = int(params[0])
        self.name = params[1]
        if self.name == 'jump':
            self.length = int(params[2])

    def __str__(self):
        s = str(self.pos) + " " + self.name
        if self.name == 'jump':
            s+= " " + str(self.length)
        return s

class Participant:
    def __init__(self, name):
        self.name = name
        self.actions = []

def debug(*argv):
    if False:
        s = ''
        for arg in argv:
            s += str(arg) + ' '
        print(s)

def printScore(name, score):
    print(name, 'score is', str(score))

def printRoute(route):
    n = len(route)
    m = len(route[0])
    for i in range(n-1,-1,-1):
        s = ''
        for j in range(m):
            s += route[i][j]
            #print(route[i][j])
        print(s)

def play(map, part):

    # (0,0) is left-bottom edge
    x = 0  # horizontal pos
    y = 0  # vertical pos
    score = 0
```

```
    # init route
    n = len(map)
    m = len(map[0])
    route = [ [' ']*m for i in range(n)  ]

    # Do all actions
    for action in part.actions:
        debug(part.name, 'next action:', action )

        # All actions are started in next pos except loop
        trigger = action.pos - 1 if action.name == 'loop' else action.pos

        while x <= trigger: #Advance until reach pos
            debug('   ', x,y, map[y][x])

            if map[y][x] != ' ': # move forward
                debug('    move forward')
                score += y
                if route[y][x] != '/' and route[y][x] != '\\': # Ramps are
prefered in output if used
                    route[y][x] = '_'
                x += 1
            else: # fall down
                while map[y][x] != '_' and y > 0: # Once falling down, only stop
in ground
                    y -= 1

        debug('action pos reached!')
        if action.name == 'ramp_up':
            while map[y][x] == '/':
                debug('ramping-up!')
                score += y
                route[y][x] = '/'
                x += 1
                y += 1

        elif action.name == 'ramp_down':
            y -= 1
            while map[y][x] == '\\':
                score += y
                route[y][x] = '\\'
                if y-1 >= 0 and map[y-1][x+1] == '\\':
                    y -= 1
                x +=1

        elif action.name == 'jump':
```

```
            y += 1
            for i in range(action.length):
                debug('jumping!', x, y)
                if map[y][x] == '|': # end of map reached! End jumping and do
nothing, end managed apart
                    break
                else:
                    score += y
                    route[y][x] = '.'
                    x += 1

        elif action.name == 'loop':
            debug('start loop!', x, y, map[y][x])
            # save start of loop
            loop_x = x
            loop_y = y
            going_up = True
            score += y
            route[y][x] = map[y][x]
            x += 1
            while x != loop_x or y != loop_y: # until end of loop reached
                debug('looping!', x, y, map[y][x], going_up)
                score += y
                if route[y][x] != '/' and route[y][x] != '\\': # Ramps are
prefered in output if used
                    route[y][x] = map[y][x]

                # move to next
                if map[y][x] == '_':
                    if y == loop_y:    # lower part of the loop
                        x += 1
                    else:                           # upper part of the loop
                        going_up = False
                        if map[y-1][x-1] == '/':
                            y -= 1
                        x -= 1

                elif map[y][x] == '/':
                    if going_up:
                        y += 1
                        if map[y][x] != '\\':
                            x += 1
                    else:
                        y -= 1
                        if map[y][x] == ' ':
                            x -= 1
```

```
                elif map[y][x] == '\\':
                    if going_up:
                        y += 1
                        x -= 1
                    else:
                        if y != loop_y or map[y][x+1] != '_': # keep looping
                            y -= 1
                        x += 1

            debug('loop completed!')

    debug('No more actions!')

    while map[y][x] != '|': #Advance until end of map
            debug('   ', x,y, map[y][x])

            if map[y][x] != ' ': # move forward
                debug('    move forward')
                score += y
                if route[y][x] != '/' and route[y][x] != '\\': # Ramps are
prefered in output if used
                    route[y][x] = '_'
                x += 1
            else: # fall down
                debug('    fall down!')
                while map[y][x] != '_' and y > 0: # Once falling down, only stop
in ground
                    y -= 1

    # Final reached :)
    score += y
    route[y][x] = '|'

    return route, score

##########################################
import sys

# Read map. Store it in inverse order to have the (0,0) coordinate in left-
bottom edge
n = int(input())
map =  []
for i in range(n):
    l = input()
    if l[-1] != '|':
        # Check that input is OK
        print("ERROR!!!")
```

```
        exit()
    map.insert(0, l)

# Read all participants
participants = []
for line in sys.stdin:
    s = line.split()
    if len(s) > 0:
        if s[0] == 'Participant':
            name = s[1]
            participants.append(Participant(name))
        elif s[0] == 'Action':
            participants[-1].actions.append(Action(s[1:]))

# Let's play!
for p in participants:
    [route, score] = play(map, p)
    printRoute(route)
    printScore(p.name, score)
```