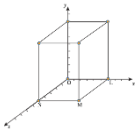
Problems

#	Problem	Points
1	Coordinates arranger	1
2	Lemonade	1
3	Where will the wedding take place?	2
4	Abel wants to play 4-9!	3
5	Rating movies	3
6	Shopping list for a party	3
7	A mirror on the Moon	3
8	The modern teacher	4
9	King Africa	4
10	BanBot	5
11	Lottery algorithm	5
12	Environmental footprint	6
13	Double checking internet facts	6
14	Zombie attack	7
15	Save the Code*Wars*	8
16	Life on Mars	9
17	Sator Arepo	9
18	The Arrangers	9
19	Pattern recognition	12
20	Droid maker	13
21	Happy new year 2019	13
22	Vigenère	13
23	X marks the spot	13
24	Smart spreadsheet	14
25	Juno's calculator	15
26	Playing 4-9!	15
27	Rolling dice game	17
28	Network graph	20
29	Blobs	22
30	Quantum gates	23
31	Settlers of Code*Wars*	24
32	The Dragon King	31

1 Coordinates arranger

1 point



Introduction

The last model of the "*dy*" augmented reality glasses includes three sensors which tell you the position of the focus point where the user is staring at. The position is represented by three values x, y and z. Your goal is to read the coordinates of the point from the sensors and print the point coordinates, arranging them according to the (x, y, z) format.

Input

The input consists of integer values that represent the coordinates x, y and z of a 3D point, one by line.
<x value>
<y value>
<z value>

Output

Print out the 3D point coordinates following this output format:
(x, y, z)

Example 1

Input
```
3
4
5
```

Output
```
(3, 4, 5)
```

Example 2

Input
```
0
-7831
2323
```

Output
```
(0, -7831, 2323)
```

Python3

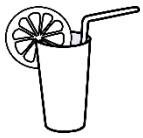
```
a=input()
b=input()
c=input()

print('(' + a + ', ' + b + ', ' + c + ')')
```

C++

```cpp
#include <iostream>
int main() {
    int x, y, z;
    std::cin >> x >> y >> z;
    std::cout << "(" << x << ", " << y << ", " << z << ")" << std::endl;
}
```

2 Lemonade

1 point



Introduction

A group of students want to earn some money and decide to sell lemonade to their neighbours. To prepare one jar of lemonade they need a pack of 6 lemons and one litre of water that they get from their houses. How many jars can they do given a certain number of lemon packs? Can you write a simple program to find out the answer?

Input

The input consists of a positive integer number representing the number of lemon packs.

Output

Print the total amount of lemonade jars that can be produced.

Example 1

Input
```
3
```

Output
```
3 lemonade jars
```

Example 2

Input
```
1
```

Output
```
1 lemonade jar
```

Python3

```python
line = int(input())

if line == 1:
    print ("{0} lemonade jar" .format(line))
else:
    print ("{0} lemonade jars" .format(line))
```

C++

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

int main() {
      int jar;
      cin >> jar;
      if (jar > 1) {
            cout << jar << " lemonade jars" << endl;
      }
      else {
            cout << jar << " lemonade jar" << endl;
      }
}
```

3 Where will the wedding take place?

2 points

Introduction

Ana is a wedding planner and she is planning Nuria's wedding. Ana must know how many people will attend the wedding, so she asks the couple, Nuria and Albert, for this number. If there are fewer than 100 guests, the wedding will take place in Monestir de Sant Cugat. If there are 100 guests or more, the wedding will take place in a very big *masia* (a large country estate). Can you help Ana to decide where the wedding will take place?
Come on, help Ana!

Input

The input consists of two integers in two lines:
<number for Nuria's guests>
<number for Albert's guests>

Output

Print out one of the following outputs stating where the wedding will take place:
The wedding will take place in Monestir de Sant Cugat.
The wedding will take place in very big masia.

Example 1

Input

```
50
20
```

Output

```
The wedding will take place in Monestir de Sant Cugat.
```

Example 2

Input

```
100
250
```

Output

```
The wedding will take place in very big masia.
```

Python3

```python
nuria = int(input())
albert = int(input())
attendees = nuria + albert
if attendees < 100:
    print("The wedding will take place in Monestir de Sant Cugat.")
else:
    print("The wedding will take place in very big masia.")
```

C++

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

int main() {
      int nuria, albert;
      cin >> nuria >> albert;
      int attendees = nuria+albert;
      cout << "The wedding will take place in ";
      cout << ((attendees < 100)?"Monestir de Sant Cugat.":"very big masia.");
     cout << endl;
}
```

4 Abel wants to play 4-9!

3 points

Introduction

Abel's parents want to buy him a new PC for his homework, and he wants to take advantage of that and get a PC that is able to run the 4-9 game, the most popular game of the year. In order to do so, he wants to write a little program to check if the PC specifications are enough to run the game.

The 4-9 game needs at least the following PC specifications to run properly:

- **Processor generation: 5**
- **Graphics card memory: 2**
- **Free storage memory: 50**

Can you help him to write this program?

Input

A sequence of 3 lines, each one with a positive integer number representing each of the PC specifications:
<Processor generation>
<Graphics card memory>
<Free storage memory>

Output

Print out one of the following outputs stating where Abel can play 4-9 or not:
You can run the game
You can NOT run the game

Example 1

Input

```
7
4
100
```

Output

```
You can run the game
```

Example 2

Input

```
5
1
500
```

Output

```
You can NOT run the game
```

Python3

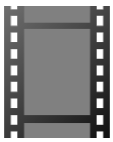
```python
processor=int(input())
graphicscard=int(input())
freestorage=int(input())

if processor>=5 and graphicscard >=2 and freestorage >= 50:
    print ("You can run the game")

else:
    print ("You can NOT run the game")
```

C++

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

int main() {
      int processor, graphic, freestorage;
      cin >> processor >> graphic >> freestorage;

      if (processor > 4 and graphic > 1 and freestorage > 49) {
          cout << "You can run the game" << endl;
      }
      Else {
          cout << "you can NOT run the game" << endl;
      }
}
```

5 Rating movies

3 points



Introduction

Your friend Ana enjoys a lot going to the cinema. She wants to keep a tracking list of the movies watched during the year and store her personal movie rating. This rating will be a number of stars, but she does not want to write them down one by one. Instead, she asked you to write a simple program to write down as many stars as the value of a given number between 0 and 100. In the case of rating a very bad movie with 0 stars, she asked you to display a single character dash.

Input

An integer number between 0 and 100.

Output

Print out the number of stars corresponding to the given number greater than zero. In the case the of reading a zero then the output will be a single dash.

Example 1

Input
5

Output
*****

Example 2

Input
0

Output
-

Python3

```
nstars =int(input())

s = ''
if nstars > 0:
    for i in range(nstars):
        s += '*'
else:
    s = '-'
print(s)
```

C++

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

int main() {
  int numberOfStars;
  string stringStars;
  // Read the number of stars from standard input
  cin >> numberOfStars;
  if (numberOfStars > 0) {
      for (int i = 0; i < numberOfStars; i++) {
          stringStars += "*";
      }
  }
  else {
      stringStars = "-";
  }
  cout << stringStars << endl;
}
```

6 Shopping list for a party

3 points

Introduction

Three boys want to throw a party and they want to bake some cakes for their friends. They have a recipe for a cake for 5 people. The ingredients needed for one cake are:

- **1 natural yogurt.**
- **3 eggs.**
- **250gr flour.**
- **125gr cocoa powder.**
- **250gr sugar.**
- **125gr olive oil.**
- **1 packet of yeast.**

The number of guests will vary, depending on each person's RSVP on social media. Once they have this number they will need to calculate the amount of each ingredient needed to bake enough cakes for everybody. In case the number of guests is not a multiple of 5, round it up to the nearest multiple of 5.

Can you write a program to solve this problem?

Input

The number of guests (including the 3 boys) who have confirmed their attendance.

Output

Print out the list of ingredients required for the total number of cakes needed for all the guests, in the same order as in the recipe, specifying the amount of each ingredient.

Example 1

Input

```
5
```

Output

```
- 1 natural yogurt.
- 3 eggs.
- 250gr flour.
- 125gr cocoa powder.
- 250gr sugar.
- 125gr olive oil.
- 1 packet of yeast.
```

Example 2

Input

```
12
```

Output

```
- 3 natural yogurt.
- 9 eggs.
- 750gr flour.
- 375gr cocoa powder.
- 750gr sugar.
- 375gr olive oil.
- 3 packet of yeast.
```

Python3

```python
import math

people = int(input())

if ((people % 5) == 0):
    people = people//5
else:
    people = people//5 + 1

print("- {0} natural yogurt." .format(people*1))
print("- {0} eggs." .format(people*3))
print("- {0}gr flour." .format(people*250))
print("- {0}gr cocoa powder." .format(people*125))
print("- {0}gr sugar." .format(people*250))
print("- {0}gr olive oil." .format(people*125))
print("- {0} packet of yeast." .format(people*1))
```

C++

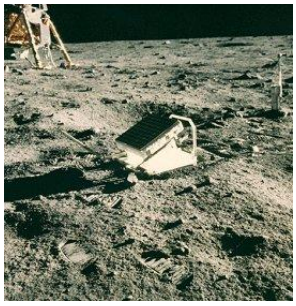
```cpp
#include <iostream>
using namespace std;

int main() {
      int people;
      cin >> people;
      int natural=1,eggs=3, flour=250, cocoa=125, sugar=250, olive=125,pack=1;

      if (people%5 == 0)
            people /= 5;
      else
            people = people/5 + 1;

      cout << "- "  << natural*people << " natural yogurt." << endl;
      cout << "- "  << eggs*people<< " eggs." << endl;
      cout << "- "  << flour*people<< "gr flour." << endl;
      cout << "- "  << cocoa*people << "gr cocoa powder." << endl;
      cout << "- "  << sugar*people<< "gr sugar." << endl;
      cout << "- "  << olive*people<< "gr olive oil." << endl;
      cout << "- "  << pack*people<< " packet of yeast." << endl;
}
```

7 A mirror on the Moon

3 points

Introduction

The Apollo 11 mission allowed mankind to step on the Moon 50 years ago. The astronauts Neil Armstrong and Buzz Aldrin, before the end of their moonwalk, installed a 2-foot wide panel studded with 100 mirrors pointing at Earth. It is called the "lunar laser ranging retroreflector array".



This array of mirrors allows to 'ping' the moon with laser pulses and measure the Earth-Moon distance very precisely. The laser pulse shoots out of a telescope on Earth, crosses the Earth-Moon divide, and hits the array. Then the mirrors send the pulse straight back where it came from. This allows, for example, to study the Moon's orbit. So, here is the simple formula to calculate this distance:

$$distance = \frac{speed\ of\ light * time\ taken\ for\ light\ to\ reflect}{2}$$

We would like to track the variability of the Earth-Moon distance so we need you to program this formula to find out the distance in kilometers. As you may know the speed of light is a constant value that is assumed as 300.000 kilometers / second. So, the parameter of this formula is the time taken for light to reflect. This value will be received as input expressed in milliseconds.

HINT: You do not need to mess with decimals to solve this formula.

Input

The time for the laser pulse to go and return expressed in milliseconds.

Output

The distance Earth-Moon expressed in kilometers.

Example

Input

```
2500
```

Output

```
375000
```

Python3

```python
time_in_ms = int(input())

# Since the time is expressed in milliseconds and the
# speed of light is expressed in km per second to avoid
# messing around with big numbers and decimals the
# speed of light is divided by 1000, that is, applying
# a conversion factor to express the speed of light in
# km per millisecond. Then we have the same units to apply
# directly the formula.

speed_of_light = 300000 // 1000
distance = (speed_of_light * time_in_ms) // 2
print(distance)
```

C++

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

int main() {
	int time;
	cin >> time;
	int sl = 300000/1000;
	cout << (time*sl)/2 << endl;
}
```

8 The modern teacher

4 points

Introduction

Silvia is a modern teacher who frequently sends messages to all her students. She likes to remind them of their homework, to congratulate them on special events and to share interesting information.

Currently, she is sending the same message to all her students, but she would like to personalize the text for each student. She has asked her best student to build a software application that receives a template message and a list of names and returns a personalized message for each name. Would you like to help her?

Input

The input consists of several lines:
The first line is the template message.
The second line is the word to replace, which has a single occurrence in the template.
The third line is the number of students to customize the messages for.
The following lines contain the students' names, one name per line.

Output

The output of the program will contain all the personalized messages, one message per line.
Each message will contain one student's name in place of the word that was to be replaced. The messages should appear in the same order as the names in the input.

Example 1

Input

```
Merry Christmas NNNN!
NNNN
4
John
Natalie
Christian
Angie
```

Output

```
Merry Christmas John!
Merry Christmas Natalie!
Merry Christmas Christian!
Merry Christmas Angie!
```

Example 2

Input

```
NAME, remember to do the homework for tomorrow.
NAME
3
Mariel
Max
Rebecca
```

Output

```
Mariel, remember to do the homework for tomorrow.
Max, remember to do the homework for tomorrow.
Rebecca, remember to do the homework for tomorrow.
```

Python3

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

for i in range(num):
    name = input()
    sentence_new = sentence.replace(word,name)
    print(sentence_new)
```

C++

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

int main() {

      string phrase, word;
      getline(cin, phrase);
      cin >> word;
      int n;
      cin >> n;
      for (int i=0; i<n; ++i) {
            string aux = phrase;
            string name;
            cin >> name;
            int pos = aux.find(word);
            aux.replace(pos, word.size(), name);
            cout << aux << endl;
      }
}
```

9 King Africa

4 points

Introduction

Joan is a big fan of King Africa. This Argentinian band led by Alan Duffy rose to prominence in 1999 with the hit "La Bomba". He likes them so much that when he talks, instead of saying the words ending with r normally, he adds an additional 'r'. This way, instead of saying 'Esta tarde va a llover', he says, 'Esta tarde va a lloverr'. He wants to translate the e-book 'Don Quijote de la Mancha', but before he does that he wants to practice a bit. He does not want to change all, only the words ending with 'r'; can you help him to write a program that given a sentence, converts it to 'King Africa' style?

Input

The input consists of one single line containing the sentence to translate.

Output

Print out the translated sentence.

Example 1

Input

```
Vamos a bailar bailar
```

Output

```
Vamos a bailarr bailarr
```

Example 2

Input

```
El desenroscador que lo desenrosque buen desenroscador sera
```

Output

```
El desenroscadorr que lo desenrosque buen desenroscadorr sera
```

Python3

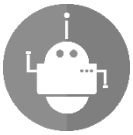
```python
sentence = input()
sentence_modified = sentence.replace('r ', 'rr ') # replacing the ending r with an double r
sentence_modified = sentence_modified.replace('R ', 'RR ') # replacing the ending R with an double r

if sentence_modified[-1] == 'r' or sentence_modified[-1] == 'R': # in case sentence ends with r without trailing spaces, add a double r
    sentence_modified = sentence_modified + sentence_modified[-1]

print(sentence_modified)
```

C++

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

int main() {

	string word;
	vector <string> v;
	while (cin >> word) v.push_back(word);

	for (int i=0; i<v.size(); ++i) {
		if (v[i][v[i].size()-1] == 'r') v[i] += "r";
		if (v[i][v[i].size()-1] == 'R') v[i] += "R";
	}

	for (string x:v)
		cout << x << " ";
	cout << endl;


}
```

10 BanBot

5 points



Introduction

You are a programmer of a game and your team has an idea to reduce the toxicity that less experienced users suffer and promote a friendly environment. For this you must program a bot that restricts the chat of the people using the word 'noob' in several ways, both in upper and lower case or mixed: `n00b, noob, n0ob, no0b`

HINT: Also, `n00B, No0b, ...` **are reasons to ban! Don't let them slip!**

Input

A single sentence with a chat fragment of a certain user.

Output

The bot should respond 'User is banned.' or 'User is exhibiting a friendly behaviour.' depending if they have used the forbidden words described above.

Example 1

Input

```
Your style is so bad, you're a noob!
```

Output

```
User is banned.
```

Example 2

Input

```
Despite you are new, you're trying your best! Keep practicing!
```

Output

```
User is exhibiting a friendly behaviour.
```

Example 3

Input

```
Go play chess, N00B
```

Output

```
User is banned.
```

Python3

```python
# List of banned words
banned_words = ['n00b', 'noob', 'n0ob', 'no0b']

# Get the input
player_text = input()

# Remove dots, commas, question and exclamation marks
player_text = player_text.replace(".","")
player_text = player_text.replace(",","")
player_text = player_text.replace("?","")
player_text = player_text.replace("!","")

# Get a list of the words after filtering
words = player_text.lower().split()

# Check if any of the words are in the list of banned words
if any(word in banned_words for word in words):
      print('User is banned.')
else:
      print('User is exhibiting a friendly behaviour.')
```

C++

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

int main() {
	string words;
	bool trobat = false;
	while (cin >> words and !trobat) {
		if (words.size() == 4 or words.size() == 5) {
			if (words[0] == 'n' or words[0] == 'N') {
				if (words[1] == 'o' or words[1] == 'O' or words[1] == '0') {
					if (words[2] == 'o' or words[2] == 'O' or words[2] == '0') {
						if (words[3] == 'b' or words[3] == 'B')  trobat=true;
					}
				}
			}
		}
	}
	if (trobat) cout << "User is banned." << endl;
	else cout << "User is exhibiting a friendly behaviour." << endl;
}
```

11 Lottery algorithm

5 points

Introduction

As Christmas has already passed, John has seen how most of his workmates left for a new life in the Caribbean Sea after they all won the 'Sorteo del Gordo de Navidad'. All won the prize, except for him, who also played but he didn't have the correct algorithm and he chose the wrong number.

After having deeply read several advanced numerology books, he is quite sure that by applying this algorithm, he will win. The algorithm consists on taking the candidate number (formed by 5 digits), dividing it by the day of the month he was born, and if the first decimal is not 0 and the second decimal equals 0, the number is a winner.

Input

Two integer values: the lottery number and the day of the month he was born.

Output

The output is 0 if it's not a good number, or 1 if it's the winning combination.

Example 1

Input
```
84367 27
```

Output
```
1
```

Example 2

Input
```
84368 27
```

Output
```
0
```

Python3

```python
entrada = input()

dos_valors = entrada.split()# Splitting the entry into two string

# converting the candidate and the day of month into an integers
candidate_number = int(dos_valors[0])
day_of_month = int(dos_valors[1])

divisio = candidate_number / day_of_month
no_unity = divisio - int(divisio) # removing the unities
primer_decimal = int(no_unity * 10) # finding th  1st decimal
no_first_decimal = no_unity * 10 - int(no_unity * 10)
segundo_decimal = int(no_first_decimal * 10) # finding the 2nd decimal

if (primer_decimal != 0) & (segundo_decimal == 0): # condition for the
winning number, outputting a 1
    print(1)
else:
    print(0)
```

C++

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

int main() {
      cout.setf(ios::fixed);
      cout.precision(2);
      double number, day, div;
      cin >> number >> day;

      div = number/day;
      int first, second;
      div *= 10;
      first = int(div)%10;
      div *= 10;
      second = int(div)%10;
      cout << ((first and !second)?1:0) << endl;
}
```

12 Environmental footprint

6 points

Introduction

Nowadays, we can buy food that comes from anywhere in the world. But this does not come for free from an environmental point of view. To transport the products from their origin to their final destination, it is needed a certain amount of energy and production carbon emissions (CO_2) is required. Sometimes, the transportation of the product requires more energy than the energy that the product gives us when eating it. The following table shows an estimate of the energy and carbon emissions per kilometers and tone for a variety of different vehicles.

Transport	Energy (MJ/(t·km))	Carbon emissions (g/(t·km) of CO_2)
Ship	0,3	23,3
Train	0,32	23,1
Road	2,12	160,1
Plane	21,01	1577,1

Given the energy by eating the product (MJ/t) and the transportion vehicle, calculate the maximum amount of kilometers that can be reached so that the energy required for the transportation does not exceed the energy produced by the product. Then, find out the associated carbon emissions produced by the transportation. For both values provide just 1 decimal of precision.

Input

The input is a sequence of lines, where each line contains a mean of transportation and the energy produced by the food in MJ/t. The sequence ends with the character #.

Output

The output is a triplet for each line of the input. Each triplet consists of:

- **The mean of the transportation.**
- **The maximum number of km that can travelled while the energy required does not exceed the energy produced by the food.**
- **The CO_2 emissions produced by the transportation for the maximum number of km.**

Example

Input

```
Ship 10650
Plane 10650
#
```

Output

```
Ship 35500.0 827150.0
Plane 506.9 799434.3
```

Python3

```
# Store in a dictionray the table < transport: energy/tone*km and carbon
emissions/tone*km >
transport = {'Ship':[0.3,23.3], 'Train':[0.32,23.1], 'Road':[2.12,160.1],
'Plane':[21.01,1577.1]}

# Read the data input
data = input()

# Until the character # is read perform this loop
while data != "#":

  # Collect the data entered and split them into two fields: vehicle and
energy of the product
  vehicle, energy = data.split(" ")

  # Find out the max kilometers that can be transported
  maxKm = float(energy) / transport[vehicle][0]

  # Calculate the carbon emissions produced by the transportation
  carbonEmissions = maxKm * transport[vehicle][1]

  # Print the result
  print( vehicle, "{0:.1f}".format(maxKm), "{0:.1f}".format(carbonEmissions)
)

  # Process the next line
  data = input()
```

C++

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

#define mp make_pair

int main() {
	cout.setf(ios::fixed);
	cout.precision(1);
	string transport;
	map<string,pair<float,float> > tr;
	tr.insert(mp("Ship", mp(0.3, 23.3)));
	tr.insert(mp("Train", mp(0.32, 23.1)));
	tr.insert(mp("Road", mp(2.12, 160.1)));
	tr.insert(mp("Plane", mp(21.01, 1577.1)));

	while (cin >> transport and transport != "#") {
		float energy;
		cin >> energy;
		float km, coxkm;
		km = energy/tr[transport].first;
		coxkm = (energy/tr[transport].first)*tr[transport].second;
		cout << transport << ' ' << km << ' ' << coxkm << endl;
	}
}
```

13 Double checking Internet facts

6 points

Introduction

You are attending your daily English class at the Upside-Down school, and your teacher tells you and your classmates about letter frequency. She says the letters E, T, A, O and I are the most used in the English language. Since you're a curious guy, you raise your hand and ask the teacher where she learned about this, and she mentions that she read an article on Wikipedia. You know that Wikipedia has a lot of information, but since everyone can edit it, you do not always trust everything it says. To make sure that the information is right, you decide to create a program to verify the letter frequcency in different texts.

Input

The input is a text in English.

Output

Print each letter of the Upside-Down English Alphabet with their corresponding number of appearances.

Example

Input

```
In a 2017 German study, researchers at Ruhr-University Bochum compared the ability of gamers and non-gamers to remember information from cue cards and then combine that information to predict weather conditions. The video gamers showed greater retention of the cue card knowledge and made better predictions, especially in conditions of uncertainty.
```

Output

```
z = 0
y = 5
x = 0
w = 3
v = 2
u = 7
t = 26
s = 13
r = 26
q = 0
p = 4
o = 23
n = 25
m = 13
l = 4
k = 1
j = 0
i = 22
h = 10
g = 6
f = 6
e = 37
d = 15
c = 14
```

```
b = 5
a = 22
```

python3

```python
text = input()

text = text.lower()
listletters = "abcdefghijklmnopqrstuvwxyz"
listletters = listletters[::-1]

for x in listletters:
	count = 0
	for y in text:
		if x == y:
			count += 1
	print(x,"=",count)
```

C++

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

int main() {
	map<char,int> m = {{'a',0},{'b',0},{'c',0},{'d',0},{'e',0},{'f',0},{'g',0},
		{'h',0},{'i',0},{'j',0},{'k',0},{'l',0},{'m',0},{'n',0},
		{'o',0},{'p',0},{'q',0},{'r',0},{'s',0},{'t',0},{'u',0},
		{'v',0},{'w',0},{'x',0},{'y',0},{'z',0}};

	char lletra;
	while (cin >> lletra) {
		if (isupper(lletra)) lletra=tolower(lletra);
		if (m.find(lletra) != m.end())
			++m[lletra];
	}
	for (auto it = m.rbegin(); it != m.rend(); ++it) {

		cout << it->first << " = " << it->second << endl;
	}
}
```

14 Zombie attack

7 points

Introduction

The scientists of your city are preparing for an imminent zombie outbreak. They have asked you to write a program that simulates the evolution of a zombie apocalypse in a city, assuming a zombie enters the city at noon. Researchers has shown that:

- **Every night each zombie infects 2 healthy inhabitants of the city**
- **25% of the zombies (rounded up) die when the sun rises**
- **An infected inhabitant becomes a zombie when the sun sets**

Your job is to create a zombie simulator which, given the city's population, calculates how many days will pass until the city is completely infected.

Input

The initial population before the first zombie arrives.

Output

Print out the number of days required by the zombies to infect the entire population.

Example

Input

```
600
```

Output

```
7 days
```

Python3

```python
import math

population = int(input())
zombies = 1
day = 0
while population > 0:
    day += 1
    new_zombies = min(population, 2 * zombies)
    new_dead = math.ceil(0.25 * zombies)
    population -= new_zombies
    zombies += new_zombies - new_dead

print(day, "days")
```

C++

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

int main() {
	int pop;
	cin >> pop;
	int days = 0;
	int zombie = 1;
	while (pop > 0) {
		int infected = min(pop,zombie*2);
		int dead = ceil(zombie*0.25);
		pop -= infected;
		zombie += infected - dead;
		++days;
	}
	cout << days << " days" << endl;
}
```

15 Save the CodeWars

8 points

Introduction

A hacker has infiltrated our Judge code! He has introduced a virus into our system that threatens the competition. It has kidnaped the server! We need you, our most talented young programmers, to deactivate the virus in order to save the event and our amazing prizes!

Our smart HP CodeWars volunteers' team has discovered that to deactivate the virus we need to enter a 4 numbers combination related to the Fibonacci numbers.

The Fibonacci numbers are the sequence of numbers defined by the linear recurrence equation:

$$F_n = F_{n-1} + F_{n-2}$$

where $F_1 = 1$ **and** $F_2 = 1$**.**

Therefore, the Fibonacci numbers are 1, 1, 2, 3, 5, 8, 13, 21, ...

We need your help to write a program that computes the 4 Fibonacci numbers that the virus is asking for.

Input

The input of the program is one line with 4 numbers, greater than 0, and separated by white spaces. Each one of these numbers marks a position of an element in the Fibonacci sequence (for example, 1 marks the first element, F_1**.**

Output

The output of the program must be the 4 Fibonacci numbers (F_x**) in the given positions, separated by white spaces.**

Example

Input

```
10 2 12 5
```

Output

```
55 1 144 5
```

Python3

```
# Read the input with the four numbers to request Fibonacci(n). The numbers
are separated by space.
data = input()

# Auxiliar recursive function to calculate the fibonacci number n
def fibonacci(n):

  # Default values for Fib(0), Fib(1) and Fib(2)
  fib = [0,1,1]

  if ( n > 0 and n <= 2):
    return fib[1]
  elif ( n == 0 ):
    return fib[0]
  else:
    for i in range(3,n+1):
      # Add a new element to the Fibonacci table
      fib.append(0)
      # Calculate this value
      fib[i] += fib[i-1] + fib[i-2]
    return fib[n]

# Split the data into a list of strings
numbers = data.split(" ")

# Convert the list of strings in a list of numbers
numbers = list(map(int, numbers))

# Apply the fibonacci function to the four numbers and print the output
print(str(fibonacci(numbers[0])) + " " +str(fibonacci(numbers[1]))+ " "
+str(fibonacci(numbers[2]))+ " " +str(fibonacci(numbers[3])))
```

C++

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

int fib(int n) {
      if (n == 0 or n == 1) return n;
      int f1,f2,f;
      f1 = 1; f2 = 0;
      for (int i=2; i<=n; ++i) {
            f = f1 + f2;
            f2 = f1;
            f1 = f;
      }
      return f;
}

int main() {
      for (int i=0; i<4; ++i) {
            int n;
            cin >> n;
            cout << ((i)?" ":"") << fib(n);
            if (i==3) cout << endl;
      }
}
```

16 Life on Mars

9 points

Introduction

The first human expedition on Mars has identified a new strand of microorganisms. The new life is based on a single stranded RNA-like code but uses only 3 different nucleotides or "letters" to encode its genetic information: A (Adenine), T (Thymine) and U (Uracil). The biology of the cell is based on simple proteins formed by a short series of amino acids encoded by pairs of the genetic code letters (codons). For instance: the sequence AT gives the amino acid Cysteine; the sequence TU encodes for Arginine (see table). There are especial codons for starting a coding sequence (AU) and for the end of a sequence (TA) and (TT). Note that the especial sequence UT makes the cell discard (delete) the last amino acid coded and there is no multiple START signal on a given sequence.

CODON PAIR	Amino Acid / SIGNAL
AA	Glycine
AT	Cysteine
AU	START
TA	END
TT	END
TU	Arginine
UA	Serine
UT	DELETE
UU	Lysine

Write a program that given a string containing a nucleotide sequence, returns the (comma separated) sequence of amino acids (if any) encoded in the sequence or "None" if no valid sequence is found. To be valid, a sequence must have a single START, at least one amino acid and an END signal.

Input

The input is a string with a nucleotide sequence using single capital letters.

Output

Print out the sequence of amino acid names (fist letter of the amino acid name in capital letter), separated by a comma.

Example 1

Input
AUAATUTT

Output
Glycine,Arginine

Example 2

Input
AUAATUUTUUATTT

Output
Glycine,Lysine,Cysteine

Example 3

Input
AUAATUUTUUAT

Output
None

Python3

```
import sys
"""
ribosome This is the method that converts a RNA sequence into a series of aminoacid that form a
protein.
 This function is implementing a state machine able to read alien genomic sequeces.
The ARN to aminoacid translation table is defined in the  variable ARN.
Valid sequences are  found between a single START and the first END thereafter
 Here the kids have to implement a state machine. The implementation can be short (less than 40
lines) but tricky.
While we have not yet found a START sequence, we have to parse letters one by one. After we find
the first START,
we parse letters in pairs till we find the first END
 :param sequence: String with the ARN string with letters A,T or U
:returns: string with all the aminoacids sepparated with comma or the string "None" of no valid
sequence found.
"""
def ribosome(sequence):
    ARN = {'AA': 'Glycine', 'AT': 'Cysteine', 'AU': 'START', 'TA': 'END','TT': 'END','TU':
'Arginine','UA': 'Serine','UT': 'DELETE','UU' : 'Lysine'}
    aminoList = []
    seq = sequence
    coding = False # True when we are in a coding area (found START but not yet END)
    exit = False
    while not exit:
        if len(seq) < 2:
            break
        codon = seq[:2]
        signal = ARN[codon]
        if signal == 'START':
            coding = True
            seq = seq[2:] # found START, skip 2 letters
        elif signal == 'END':
            if coding:
                exit = True # No need to keep looking for more sequences
            else:
                seq = seq[1:] # found end but, we had not yet found a start, so keep searching
skipping just 1 letter
            coding = False
        else:
            if coding:
                if signal != 'DELETE':
                    aminoList.append(signal)
                else:
                    aminoList.pop()
                seq = seq[2:]
            else:
                seq = seq[1:]
    if len(aminoList) == 0 or not exit: #
        return 'None'
    else:
        return ','.join(aminoList)

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

sequence = input()
aminoList = ribosome(sequence)
print(aminoList)
```

C++

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main() {
	bool start = false;
	string rna;
	cin >> rna;
	vector<string> v;
	int i = 0;
	while (i < rna.size()-1) {
		if (!start) {
			if (rna[i] == 'A' and rna[i+1] == 'U') {
				i += 2;
				start = true;
			}
			else ++i;
		}
		else {
			if (rna[i] == 'A') {
				if (rna[i+1] == 'A') v.push_back("Glycine");
				else if (rna[i+1] == 'T') v.push_back("Cysteine");
				else continue;
			}
			else if (rna[i] == 'T') {
				if (rna[i+1] == 'U') v.push_back("Arginine");
				else start = false;
			}
			else {
				if (rna[i+1] == 'A') v.push_back("Serine");
				else if (rna[i+1] == 'T') v.pop_back();
				else v.push_back("Lysine");
			}
			i += 2;
		}
	}

	if (!v.size() or start) cout << "None" << endl;
	else {
		for (int i=0; i<v.size(); ++i) {
			if (i > 0) cout << ',';
			cout << v[i];
		}
		cout << endl;
	}
}
```

17 Sator Arepo

9 points

Introduction

The five words SATOR, AREPO, T ENET, OPERA, and ROTAS form a known lettered magic square.

When it is read horizontally (backwards and forwards) and vertically (up and down) it always forms the same palindrome:

S A T O R A R E P O T E N E T O P E R A R O T A S

It was written in Latin and was found for the first time signed up in the ruins of Pompeii on a column from the 1st century AD. Since its discovery, it was found in many ancient tombs and temples. Some people have attributed magical properties to it, considering it one of the broadest magical formulas in the West.

We would like to have a program to detect lettered magic squares like Sator Arepo. Can you help us?

Input

The input consists of five words of five characters in different lines

Output

A string describing the input as "Magic Square like Sator Arepo" or "Not a Magic Square"

Example 1

Input

```
SATOR
AREPO
TENET
OPERA
ROTAS
```

Output

```
Magic Square like Sator Arepo
```

Example 2

Input

```
HELLO
PATER
SILLY
JIMMY
BUZZY
```

Output

```
Not a Magic Square
```

Python3

```
# Lines
lines = ["","","","",""]

# Columns
columns = ["","","","",""]

lines[0] = input()
lines[1] = input()
lines[2] = input()
lines[3] = input()
lines[4] = input()

# Fill the content of each column from lines
for i in range(5):
  columns[i] = lines[0][i] + lines[1][i] + lines[2][i] + lines[3][i] +
lines[4][i]

# Check that lines and columns are palindromes
if (lines[0] == lines[4][::-1] and lines[1] == lines[3][::-1] and lines[2] ==
lines[2][::1] and \
    columns[0] == columns[4][::-1] and columns[1] == columns[3][::-1] and
columns[2] == columns[2][::-1] and \
    lines[0] == columns[0] and lines[1] == columns[1] and lines[2] ==
columns[2] and lines[3] == columns[3] and lines[4] == columns[4]):
  print("Magic Square like Sator Arepo")
else:
  print("Not a Magic Square")
```

C++

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

int main() {
	vector<string> v(5);
	for (int i = 0; i < 5; ++i) {
		cin >> v[i];
	}
	bool trobat = false;

	for (int k= 0; k < 5 and !trobat; ++k) {
		for (int i = k, j = k; i < 5 and !trobat; ++i, ++j) {
			if (v[i][k] != v[k][j]) trobat = true;
		}
	}
	for (int i = 0, x = 4; i < 5 and !trobat; ++i, --x) {
		for (int j = 0, z = 4; j < 5 and !trobat; ++j, --z) {
			if (v[i][j] != v[x][z]) trobat = true;
		}
	}
	if (!trobat) cout << "Magic Square like Sator Arepo" << endl;
	else cout << "Not a Magic Square" << endl;
}
```

18 The Arrangers

9 points

Introduction

The time has come. After more than ten years and twenty films building up to the final climax, "The Arrangers" is premiering in cinemas today. It's full of action, emotion and bad jokes. It's going to be HUGE.

At least as huge as the queue.

You are the first of your friends to arrive and there is no more room inside, so you must buy all the tickets. You can only buy one ticket at a time, so if you need more than one, you have to go back to the end of the line until you have as many as you need.

Each ticket takes exactly one minute to buy. You need to calculate how many minutes you will spend in the queue, so you can tell your friends when to arrive.

Input

The input consists of two lines where:
The first line is a series of positive integers indicating the number of tickets that each person wants to buy.
The second line is your initial position in the queue, starting from "1".

Output

Print out a single integer indicating how many minutes you will be in the queue until you buy all the tickets.

Example 1

Input

```
2 2 4 1 3
3
```

Output

```
12
```

Example 2

Input

```
5 1 3 2
2
```

Output

```
2
```

Python3

```
ll = input()
llista = ll.split(' ')
llista = list(map(int, llista))
pos = int(input())

time = 0
pos -= 1

while True:
      if not llista[pos]:
            break
      for i in range(len(llista)):
            if not llista[pos]:
                  break
            if llista[i]:
                  time += 1
                  llista[i] -= 1

print(time)
```

C++

```cpp
#include <iostream>
#include <sstream>
#include <queue>
#include <stdint.h>

int main()
{
    uint32_t buyTime = 0;

    // Read the tickets each person wants to buy
    std::string ticketsString;
    std::getline(std::cin, ticketsString);
    std::istringstream iss(ticketsString);

    std::queue<uint32_t> ticketsQueue;
    uint32_t ticket = 0;
    while (iss >> ticket)
    {
        ticketsQueue.push(ticket);
    }

    // Get our position
    uint32_t myPosition = 0;
    std::cin >> myPosition;
    if ( (myPosition <= 0) || (myPosition > ticketsQueue.size()) )
    {
        std::cout << "Error" << std::endl;
        return -1;
    }

    // Check how many time we will spend in the queue to buy all the tickets
    while (true)
    {
        // One more minute spent in the queue
        buyTime++;

        // Adavance the turn
        uint32_t nextTickets = ticketsQueue.front();
        ticketsQueue.pop();
        nextTickets--;
        myPosition--;

        // Check if we are finished
        if ( (nextTickets == 0) && (myPosition == 0) )
        {
            break;
        }
```

```
        // Go back to the waiting line
        if (nextTickets > 0)
        {
            ticketsQueue.push(nextTickets);
        }

        // Update our position if we just bought a ticket
        if ( myPosition == 0 )
        {
            myPosition = ticketsQueue.size();
        }
    }

    // Print out the time we will spend
    std::cout << buyTime << std::endl;
    return 0;
}
```

19 Pattern recognition

12 points

Introduction

Pattern recognition is a popular branch of Artificial Intelligence and is defined as the process of classifying input data into objects or classes based on key features. It has applications in computer vision like face detection and stop sign detection in autonomous car driving.

In its basic form a pattern recognition system consists of three stages:

1. **A set of sensors gather the observations to be classified or described.**
2. **A feature extraction mechanism computes numeric or symbolic information from the observations.**
3. **A classification or description scheme classifies the observations relying on the extracted features in the previous stage.**

Here is a proposal to do the first steps in pattern recognition. Can you develop a program to just classify the input received as a series of lines received as one of these polygons: square, rectangle or triangle?

Input

The input will be a series of lines with any ASCII characters drawing a polygon form, that can only be a square, a rectangle or a triangle.

Output

Print out the sentence "I see a X" where X will be the polygon recognized, that is square, rectangle or triangle.

Example 1

Input

```
###
# #
###
```

Output

```
I see a square
```

Example 2

Input

```
*****
*   *
*****
```

Output

```
I see a rectangle
```

Example 3

Input

```
#
```

Output

```
I see a square
```

Example 4

Input

```
$
$$
$ $
$  $
$$$$$
```

Output

```
I see a triangle
```

Python3

```
import sys

rows = 0
columns = 0
prevColumns = 0
triangleFound = False

for i in sys.stdin:
  prevColumns = columns
  i=i.lstrip() # Remove whitespaces from left
  i=i.rstrip() # Remove whitespaces from right
  # Discard any empty line
  if i !="":
    columns = len(i) # Get the number of columns from current row
    if (prevColumns != columns and rows != 0):
      triangleFound = True
    rows += 1

if (triangleFound):
  print("I see a triangle")
elif (rows == columns):
  print("I see a square")
else:
  print("I see a rectangle")
```

C++

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

int main() {
	string pattern;
	int rows, prevC, col;
	rows = prevC = col = 0;
	bool triangle = false;
	while (getline(cin,pattern)) {
		prevC = col;
		int size1a = pattern.find_first_not_of(' ');
		int size1b = pattern.find_first_not_of('\t');

		pattern.erase(0,size1a);
		pattern.erase(0,size1b);
		if (pattern != "") {
			col = pattern.size();
			if (prevC != col and rows != 0)
				triangle = true;
			++rows;
		}
	}
	if (triangle) cout << "I see a triangle" << endl;
	else if (rows == col) cout << "I see a square" << endl;
	else cout << "I see a rectangle" << endl;
}
```

20 Droid maker

13 points

Introduction

In 1999 the movie Star Wars: Episode I The Phantom Menace premiered worldwide. This prequel movie explained the life of a young Anakin Skywalker, also known as the evil Darth Vader. One of his hobbies was to construct his own droids, like C-3PO. We challenge you to follow in the footsteps of Anakin. Well, we are not asking you to rule the galaxy, just to share the same hobby building and programming little robots. But since you are in galaxy far, far away, you cannot program the droids using binary numeral systems (based in two digits: 0 and 1) because this galaxy uses ternary numeral systems.

A ternary numeral system has three as its base with three digits: 0, 1 and 2. So, instead of speaking of bits (binary digits) you will manage trits (trinary digits). Below an example:

Decimal	Binary	Ternary
0	0	0
1	1	1
2	10	2
3	11	10
4	100	11
5	101	12
6	110	20
7	111	21
8	1000	22
9	1001	100
10	1010	101
11	1011	102
12	1100	110
...	...	...

Consider the decimal number 42, as an example, its representation in binary is 101010. Meaning that the position of 2^5 is 1, 2^4 is 0, 2^3 is 1, 2^2 is 0, 2^1 is 1 and 2^0 is 0.

Accordingly, 42 in ternary the positions are represented as 3^3 is 1, 3^2 is 1, 3^1 is 2 and 3^3 is 0.

So, you will need to find a way to convert all your decimal data into ternary data. Can you write a program to have your data expressed in trits?

Input

An integer positive decimal number

Output

The corresponding value of the input number translated to ternary numeral system.

Example 1

Input
10

Output
101

Example 2

Input
1977

Output
2201020

Python3

```python
import sys

num = int(input())

def convertToTrits(n):
  a = n // 3
  b = n % 3
  if (a == 0):
    return str(b)
  else:
    return(convertToTrits(a) + str(b))

res = convertToTrits(num)

print(res)
```

C++

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

string convertToTri(int n) {
      int a = n/3;
      int b = n%3;
      if (!a) return to_string(b);
      else {
            return (convertToTri(a) + to_string(b));
      }
}

int main() {
      int n;
      cin >> n;
      cout << convertToTri(n)     << endl;
}
```

21 Happy new year 2019!

13 points

Introduction

Although today we are already March 2nd we are still celebrating that 2019 is happy year. Probably you will ask yourself why? The answer is simple 2019: is a happy number.

A happy number is a number that, if you square its digits, add them together, and then take the result and square its digits and add them together, and keep repeating that over and over, your end result is the number 1.

If there are happy numbers, then there must be unhappy numbers. Those numbers for which this process does not end in 1 are considered unhappy numbers.

Input

An integer number greater than zero.

Output

Print out a string with the input number stating whether the number is happy or unhappy.

Example 1

Input

```
2
```

Output

```
2 is an unhappy number!
```

Example 2

Input

```
2019
```

Output

```
2019 is a happy number!
```

Python3

```
num = input()

# Make a copy of the original number entered
originalNumber = num

# Use a set to store all the number that are processed
past = set()

# Iterate searching for number 1 until a cycle is detected
while (int(num) != 1):
 # Find out the number resulting of applying the sum of all the digits
squared
 res = 0
 for i in num:
  res = res + pow(int(i),2)
 num = str(res)

 # Check whether this number has been already processed and we are cycling
 # Then break the search and report the number as unhappy
 if num in past:
  break
 # Add the resulting number to the set of past numbers visited
 past.add(num)

if(int(num) == 1):
 print(originalNumber + " is a happy number!")
else:
 print(originalNumber + " is an unhappy number!")
```

C++

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

int suma(int n) {
      int suma=0;
      while (n != 0) {
            suma=suma+(n%10)*(n%10);
            n/=10;
      }
      return suma;
}
bool is_happy(int n) {
      if (suma(n)==1) return true;
      else if (suma(n)==4)return false;
      else return is_happy(suma(n));
}
int main() {
      int n;
      cin >> n;
      if (is_happy(n)) {
            cout << n << " is a happy number!" << endl;
              }
      else {
            cout << n << " is an unhappy number!" << endl;
              }
}
```

22 Vigenère

13 points

Introduction

A 16th century French diplomat, Blaise de Vigenère, created a very simple cipher that is moderately difficult to decipher. It is an improvement on the Caesar cipher to reduce the effectiveness of performing frequency analysis on the ciphered text.

The Vigenère cipher uses a 26×26 table with A to Z as the row and column headings. This is known as the Vigenère table. The first row of this table contains the 26 letters in the English alphabet in order. Starting with the second row, each contains the letters in the same order shifted one position to the left one compared to the previous row in a cyclical manner. For example, when B is shifted to the first position on the second row, the letter A moves to the end of the row.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Given this poly-alphabetical table the cipher uses a keyword for shifting the alphabet used to encode each character of the message. The first letter of the keyword indexes the row and the first letter of the message indexes the column (for which it serves as the heading). Then the second letter of the keyword is used for the second letter of the message. Once all the letters of the keyword have been used we go back to the beginning of the keyword.

Example: Given the message to cipher "MAYTHECODEBEWITHYOU" with the keyword "LUCAS" the message ciphered is "XUATZPWQDWMYYILSSQU".

The deciphering process consists of

1. **picking a letter in the coded message (M) and its corresponding letter in the keyword(K)**
2. **using the letter in the keyword (K) to determine the row**
3. **searching that row for the column which contains the message letter (M); and finally**

4. **identifying the letter at the head of that column which is the letter found in the original message.**

Input

The input is split in three lines in capital letters.
The first line contains a single character: C for cipher or D for decipher.
The second line contains the message to encrypt or decrypt.
The third line contains the keyword.

Output

Print out the encoded (encrypted) message if the first line contains a C; the decoded message if the first line contains a D.

Example 1

Input

```
C
ILIKETOCRYPTMYMESSAGES
HPCODEWARS
```

Output

```
PAKYHXKCIQWIOMPIOSRYLH
```

Example 2

Input

```
D
PIUTXRJYKGYTCRDGEPYWYTFAHWOAXW
HPCODEWARS
```

Output

```
ITSFUNNYTOREADACIPHEREDMESSAGE
```

Python3

```
import string

option = input()
message = input()
key = input()

message = message.upper()
key = key.upper()

lengthMessage = len(message)
lengthKey = len(key)

# Create Vignere table
listAlfabet =
['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X',
'Y','Z']
lengthAlfabet = len(listAlfabet)

vigenereTable = []
for i in range(lengthAlfabet):
 vigenereTable = vigenereTable + [listAlfabet]
 listAlfabet = listAlfabet[1:] + list(listAlfabet[0])

if (option == "C"):
 for i in range(lengthMessage):
  j = key[i%lengthKey]
  print(vigenereTable[string.ascii_uppercase.index(j)][string.ascii_uppercase.index(message[i])],
end="")

if (option == "D"):
 for i in range(lengthMessage):
  j = key[i%lengthKey]
  pos = vigenereTable[string.ascii_uppercase.index(j)].index(message[i])
  print(listAlfabet[pos], end="")

print()
```

C++

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

int main() {
	char opt;
	string message;
	string key;
	cin >> opt >> message >> key;
	vector<char> v = {'A','B','C','D','E','F','G','H','I','J','K','L','M',
			'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
	if (opt == 'C') {
		for (int i=0; i<message.size(); ++i) {
			int j = int(key[i%key.size()])%65;
			int p = int(message[i])%65;
			int pos = (p+j)%26;
			cout << v[pos];
		}
		cout << endl;
	}
	else {
		for (int i=0; i<message.size(); ++i) {
			int j = int(key[i%key.size()])%65;
			int p = int(message[i])%65;
			int pos = p-j;
			if (pos < 0) pos += 26;
			cout << v[pos];
		}
		cout << endl;
	}
}
```

23 X marks the spot

13 points

Introduction

Archaeology is the search for fact. Not truth. If it's truth you're interested in, Dr.Tyree's philosophy class is right down the hall… So, forget any ideas you've got about lost cities, exotic travel, and digging up the world. We do not follow maps to buried treasure, and "X" never, ever marks the spot.

Henry Jones Jr.
Indiana Jones and The Last Crusade (1989)

As you may know, Indiana Jones' speech to his students was not completely accurate. A short while later he discovered in an Italian library that sometimes, indeed, "X" marks the spot. This is how he uncovered the tomb of a First Crusade knight while looking for the Holy Grail.

In the digital world you should practice the drawing of an "X" of different sizes in case someday you will want to hide a digital treasure under it. You can do it using only the characters "X", "\", and "/".

Input

The input will be a single positive integer indicating the height of the "X".

Output

Print out the X following the pattern show in the examples.

Example 1

Input
1

Output
X

Example 2

Input
2

Output
```
\/
/\
```

Example 3

Input
3

Output
```
\ /
 X
/ \
```

Example 4

Input
4

Output
```
\   /
 \ /
 / \
/   \
```

Python3

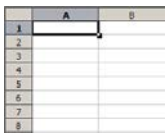
```python
rows_cols = int(input())

if rows_cols == 1:
 print("X")
elif (rows_cols % 2) == 0:
 for i in range(rows_cols//2):
  print(" " * i + "\\" + " " * 2*(rows_cols//2-(i+1)) + "/")
 for i in range(rows_cols//2):
  print(" " * (rows_cols//2-(i+1)) + "/" + " " * 2*i + "\\")
elif (rows_cols % 2) == 1:
 for i in range(rows_cols//2):
  print(" " * i + "\\" + " " * (2*(rows_cols//2-(i+1))+1) + "/")
 print(" " * ((rows_cols//2)) + "X")
 for i in range(rows_cols//2):
  print(" " * (rows_cols//2-(i+1)) + "/" + " " * ((2*i)+1) + "\\")
```

C++

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

int main() {
	int n;
	cin >> n;
	if (n == 1) cout << 'X' << endl;
	else if (n%2 == 0) {
		for (int i=0; i<n/2; ++i)
			cout << string(i,' ') << "\\" << string(2*(n/2-(i+1)),' ') << '/' << endl;
		for (int i=n/2-1; i>=0; --i)
			cout << string(i,' ') << "/" << string(2*(n/2-(i+1)),' ') << '\\' << endl;
	} else {
		for (int i=0; i<n/2; ++i)
			cout << string(i,' ') << "\\" << string(2*(n/2-(i+1))+1,' ') << '/' << endl;
		cout << string(n/2,' ') << 'X' << endl;
		for (int i=n/2-1; i>=0; --i)
			cout << string(i,' ') << "/" << string(2*(n/2-(i+1))+1,' ') << '\\' << endl;
	}
}
```

24 Smart spreadsheet

14 points



Introduction

Spreadsheets are a basic tool used in many jobs from all around the world. These computer applications are heavily used for organization, analysis and storage of data in tabular form. Since we consider it a market segment that has still room for innovation and improvement, at HP we have started the development of our own spreadsheet app and expect it to be released later this year.

However, creating the most awesome printers ever leaves us with too little time to create our own spreadsheet. Particularly, we are stuck with the integration of variables to perform computations. What we want is to be able to store partial results into variables that can be used later in another operation.

More specifically, we want our product to support the basic operators "+", "-", "*" and "/". Moreover, all the operations run by the app will be issued using any of the following formats:

- **variable = number op number**
- **variable = $variable op number**
- **variable = number op $variable**
- **variable = $variable op $variable**

Where "variable" stands for the name of a variable, "number" for any integer value (including the negative ones) and "op" for any of the aforementioned valid operators. Notice that when we use a variable as an operand, it is preceded by a "$" symbol.

Please, can you help us with this?

Input

The program must read a variable number of lines, each one following any of the previous formats. The program ends when an "end" tag is read.

Output

For each input line representing an operation, the program must print its outcome (the value that is assigned to the variable).

Example 1

Input
```
a = 4 + 3
b = 10 + $a
end
```

Output
```
7
17
```

Example 2

Input
```
hewlett = 27 - 7
packard = $hewlett / 5
hewlett = $packard * $packard
CodeWars = $hewlett + 1971
Palo = $CodeWars - 2018
Alto = -15 - $Palo
end
```

Output
```
20
4
16
1987
-31
16
```

Python3

```python
line = input()
var = dict()

def computeValue(vari):
	if '$' in vari:
		vari = vari.replace('$','')
		return int(var[vari])
	else:
		return int(vari)

def operation(n, m, c):
	if c == '+':
		return n+m
	elif c == '-':
		return n-m
	elif c == '*':
		return n*m
	else:
		return n//m

while line != 'end':
	lines = line.split(" ")
	num1 = computeValue(lines[2])
	num2 = computeValue(lines[4])
	op = lines[3]
	var[lines[0]] = operation(num1, num2, op)
	print(var[lines[0]])
	line = input()
```

C++

```
#include <iostream>
#include <stdint.h>
#include <stdlib.h>
#include <string>
#include <map>

std::map<std::string, int32_t> variables;

int32_t computeValue(std::string op)
{
	if (op[0] == '$')
	{
		op.erase(0,1);
		return variables[op];
	}
	else
	{
		return atoi(op.c_str());
	}
}

int main()
{
    std::string dest;

    // Iterate over all the operations requested
    while (std::cin >> dest && dest.compare("end"))
    {
       // Read the operands and the action to perform
       std::string op1, op2, action;
       std::cin >> op1 >> op1 >> action >> op2;

       int32_t op1Value = computeValue(op1);
       int32_t op2Value = computeValue(op2);
       int32_t result;
       if (action == "+")
       {
		result = op1Value + op2Value;
       }
       else if (action == "-")
       {
		result = op1Value - op2Value;
       }
       else if (action == "*")
       {
		result = op1Value * op2Value;
       }
       else if (action == "/")
       {
		result = op1Value / op2Value;
       }

       std::cout << result << std::endl;
       variables[dest] = result;
    }
}
```

25 Juno's calculator

15 points

Introduction

The roman goddess Juno is the protector and special counselor of the state. She needs to manage the state accounting, but she is very busy, and she asks you to help her.

Could you write a program that quickly calculates mathematical operations? There are no parentheses, priorities or operator associativity, the operations must be done in the order they are read. Remember that Juno only knows Roman Numerals, so you will have to do some translating!

Input

An undetermined amount of pairs of lines. The first line from each pair is a Roman Number, the second one is a mathematical operation ("+", "-", "*", "/", "%", "="). Note that:

- **"/" is the integer division operator**
- **"%" is the modulo operator that finds the remainder after division of one number by another**
- **"=" is only allowed once at the end of the input**

Output

The result of the series of operations in the order they are read. The output must be written in roman numbers. The result will be always greater than 0 and less than 4000

Example 1

Input

```
I
+
II
+
IV
+
VI
=
```

Output

```
XIII
```

Example 2

Input

```
I
+
II
+
IV
+
VI
*
II
=
```

Output

```
XXVI
```

Example 3

Input

```
I
+
II
+
IV
+
VI
-
XII
=
```

Output

```
I
```

Example 4

Input

```
I
+
II
+
IV
+
VI
/
II
=
```

Output

```
VI
```

Example 5

Input

```
I
+
II
+
IV
+
VI
%
X
=
```

Output

```
III
```

Python3

```
roman = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}

def roman2dec(rom):
      aux = 0
      for i in range(len(rom)):
            n = roman[rom[i]]
            if i+1 < len(rom) and roman[rom[i+1]] > n:
                  aux -= n
            else:
                  aux += n
      return aux

def operate(r, opt, num):
      if opt == '+':
            r += num
      elif opt == '-':
            r -= num
      elif opt == '*':
            r *= num
      elif opt == '%':
            r %= num
      else:
            r //= num
      return r

def dec2roman(val):
      rv = [1,4,5,9,10,40,50,90,100,400,500,900,1000]
      rl = ['I','IV','V','IX','X','XL','L','XC','C','CD','D','CM','M']
      romanL = ""
      for i in range(12,-1,-1):
            while val >= rv[i]:
                  val -= rv[i]
                  romanL += rl[i]
      return romanL

rom = input()
op  = input()

res = roman2dec(rom)
while op != '=':
      rom = input()
      res = operate(res,op,roman2dec(rom))
      op = input()

print(dec2roman(res))
```

C++

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

map<char, int> rtd = {{'I', 1}, {'V', 5}, {'X', 10}, {'L', 50}, {'C', 100}, {'D', 500}, {'M',
1000}};

unsigned int romanToDecimal(string roman)
{
    int dec = 0;
    for (int i = 0; i < roman.length(); i++)
    {
        int digit = rtd[roman[i]];
        //If the next letter is bigger, the current one substarcts
        if ( i+1 < roman.length() && rtd[roman[i+1]] > digit)
            dec -= digit;
        else //Otherwise it sums
            dec += digit;
    }
    return dec;
}

string decimalToRoman(int decimal)
{
    int thresholds[13] = {1,4,5,9,10,40,50,90,100,400,500,900,1000};
    string letters[13] = {"I","IV","V","IX","X","XL","L","XC","C","CD","D","CM","M"};
    string roman = "";
    for ( int i = 12; i >= 0; i--) {
        while(decimal >= thresholds[i]){
            decimal -= thresholds[i];
            roman += letters[i];
        }
    }
    return roman;
}

//Calculates result <- result op num
void applyOperator(int& result, const char& op, const int& num){
    if(op == '+'){
        result += num;
    }
    else if(op == '-'){
        result -= num;
    }
    else if(op == '*'){
        result *= num;
    }
    else if(op == '/'){
        result /= num;
    }
    else {
        result %= num;
    }
}

int main()
{
    string num;
```

```
    char op;
    cin >> num >> op;
    int result = romanToDecimal(num);
    while (op != '=' ){
        cin >> num;
        applyOperator(result, op, romanToDecimal(num));
        cin >> op;
    }

    cout << decimalToRoman(result) << endl;
}
```

26 Playing 4-9!

15 points

Introduction

HP is developing an online videogame called "4-9". In a "4-9" game, several players battle over a large field trying to be the last man standing, who turns out to be the single winner of the round. However, if players are being eliminated, the battlefield can start to be too big to allow the remaining players to find their opponents, which could lead the game to take too much to complete in a reasonable amount of time.

To come up with that, when a certain number of players have been eliminated, the game itself delimits a small area in the field where the players must move in. After a given time, all the players laying outside this "safety area" are eliminated, forcing them to get closer and closer while trying to survive if they want to win the round.

Here, we want you to help us develop the algorithm in charge of detecting the position of the players in the field once the "safety area" has been deployed. Basically, we need to know what happens with each player: do they continue playing or are eliminated?

Input

The input of your program should, first, read the size of the side of the battlefield (it has always a square shape). Later, we provide you a snapshot of the current state of the field in form of matrix. Each position of the matrix holds a single character that can stand for the following:

- **If it is a ".", it means that in such position of the field there is nobody.**
- **If it is a capital letter, it means that the player identified with such character is in that position of the field.**
- **Finally, the character "!" will be used to delimit the "safety area", which will always have a rectangular shape.**

Output

The output of your program must be a sentence per each player in the battle zone indicating whether they are alive or have been eliminated. More specifically, for a given player X, we will be saying "Player X has been eliminated" if it has been eliminated or "Player X is still alive" if it is inside the "safety area". The order in which the players must be mentioned has to be from top to bottom and from left to right.

Example

Input

```
6
.....F
.!!!!.
.!O.!.
.!.C!.
.!..!.
.!!!!.
```

Output

```
Player F has been eliminated
Player O is still alive
```

```
Player C is still alive
```

Python3

```python
# to use 'read' install -> pip3 install jutge
size = read(int)
field = [[0 for i in range(size)] for j in range(size)]

left = -1; right = -1; up = -1; down = -1
for i in range(size):
	for j in range(size):
		field[i][j] = read(chr)
		if field[i][j] == '!':
			if up == -1:
				up = i
			if left == -1:
				left = j
			down = i; right = j

for i in range(size):
	for j in range(size):
		if field[i][j] != '!' and field[i][j] != '.':
			if i > up and i < down and j > left and j < right:
				print('Player ' + field[i][j] + ' is still alive')
			else:
				print('Player ' + field[i][j] + ' has been eliminated')
```

C++

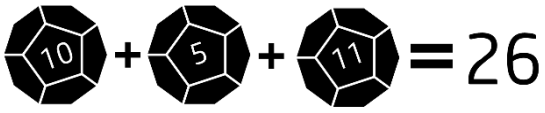
```
#include <iostream>
#include <stdint.h>
#include <vector>

int main()
{
	// Read matrix size
	uint32_t size;
	std::cin >> size;

	// Read matrix and detect safety area
	std::vector< std::vector<char> > field(size, std::vector<char>(size));
	int32_t up = -1, down = -1, left = -1, right = -1;
	for (uint32_t i = 0; i < size; ++i)
	{
		for (uint32_t j = 0; j < size; ++j)
		{
			std::cin >> field[i][j];
			if (field[i][j] == '!')
			{
				if (up == -1)
				{
					up = i;
				}
				if (left == -1)
				{
					left = j;
				}
				down = i;
				right = j;
			}
		}
	}

	// Show player's status
	for (uint32_t i = 0; i < size; ++i)
	{
		for (uint32_t j = 0; j < size; ++j)
		{
			if (field[i][j] != '!' && field[i][j] != '.' && std::isupper(field[i][j]))
			{
				if (i > up && i < down && j > left && j < right)
				{
					std::cout << "Player " << field[i][j] << " is still alive" <<
std::endl;
				}
				else
				{
					std::cout << "Player " << field[i][j] << " has been eliminated"
<< std::endl;
				}
			}
		}
	}
}
```

27 Rolling dice game

17 points

Introduction

James Bond enters a casino to play his favourite game: Hold'em Texas Poker. But he got a very bad surprise! In this casino, they only play a very weird game: Rolling Dice! The casino manager throws some dice, and players must bet against the total result that will appear. The total result is the sum of each individual die result.



For every game, the casino manager may roll a different type of die, and a different amount of dice of the selected type! So, which combinations are the most probable?

You are asked to write a program that computes the probability of a certain result given the number of dice and the number of faces of those dice.

HINT: The probability of a certain event is computed as the "positive cases" divided by "the total number of cases". For example, given a die of 6 faces, the probability of getting a 3 is 1/6 = 0.167 and the probability of getting a 22 is 0/6 = 0.000.

Input

The input is set by three non-negative integers separated by spaces:

- **The first number is the amount of dice.**
- **The second number is the amount of faces per die greater than 3.**
- **The third number is the number of which you want to know its probability.**

Output

A sentence specifying the probability for the given input following this format:
`The probability of getting a W with X dice of Y faces is X.XXX`
The probability X must be round to the third decimal and always printed out with 3 decimals.

Example 1

Input
```
2 6 8
```

Output
```
The probability of getting a 8 with 2 dice of 6 faces is 0.139
```

Example 2

Input
1 12 11

Output
The probability of getting a 11 with 1 dice of 12 faces is 0.083

Python3

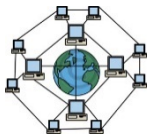
```
import itertools

## Define Rolling Dice
# N is number of dice
# M is number of faces of dice

data = input().split()
N = int(data[0])
M = int(data[1])
expctd_result = int(data[2])

cases = list(itertools.product(range(1,M+1),repeat=N))
n_cases = len(cases)

results = [sum(x) for x in cases]

probabilities = dict((x, round(results.count(x)/float(n_cases),3)) for x in set(results))

# If statement to prevent expct_result not in set(results)
if not(expctd_result in set(results)):
  expctd_p = 0.0
else:
  expctd_p = probabilities[expctd_result]

print("The probability of getting a " + str(expctd_result) + " with " + str(N) + " dice of " + str(M) +
" faces is " + "{:.3f}".format(expctd_p))
```

C++

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

int ft(int n) {
	if (n == 0) return 1;
	return ft(n-1)*n;
}

int binomial(int n,int k) {
	return (ft(n)/(ft(n-k)*ft(k)));
}

int main() {
	cout.setf(ios::fixed);
	cout.precision(3);

	int dices, faces, num;
	cin >> dices >> faces >> num;
	double val = 0;
	if (dices == 1) {
		if (dices*faces >= num) val = (double)dices/(double)faces;
	}
	else {
		if (dices*faces >= num) {

			int total = pow(faces,dices);
			int fin = (faces-dices)/num;
			for (int k=0; k<=fin; ++k) {
				int c2 = binomial(faces-num*k-1, dices-1);
				int c1 = binomial(dices,k);
				val += pow(-1,k) * c1 * c2;
			}
			val /= (double)total;
		}
	}
	cout << "The probability of getting a " << num << " with " << dices << " dice ";
	cout << "of " << faces << " faces is " << val << endl;
}
```

28 Network graph

20 points

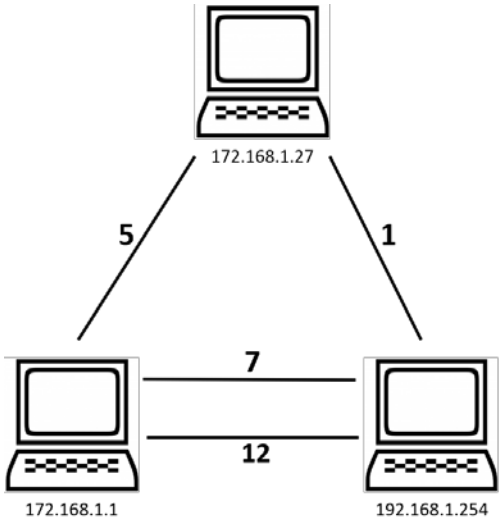


Introduction

As we are little hackers we need to send some packages through a computer network to avoid being detected by other spies.

Each computer in the network has its own identifier called IP address which is formed by 4 numbers separated by dots (e.g. 192.168.1.1). One computer can have one or more connection links to one or more computers in the network.

In the computer network you are going to use, it has some costs to send a package on a link between two computers. Notice that the cost is represented as the number over the link to find out its cost. And as you would imagine we have a short budget, so the lower price the better.



Could you program the way to find a path into the computer networks proposed to send that package?

HINT: All the IP address follow the format W.X.Y.Z where W, X, Y and Z are integer values in the range of [0..255].

Input

The input consists of several lines.
The first line will contain the IP address of the computer that sends the message.
The second line will contain the IP address of the computer that receives the message.
A variable number of lines will contain the different links that belong to the computer network. The syntax for each line will be:
ip_address_computer_x:cost:ip_address_computer_y

Output

A positive integer value representing the output will be the lower cost possible to connect the computer that sends the message and the computer that receives it.

Example 1

Input
172.168.1.1
172.168.1.254
172.168.1.1:7:172.168.1.254
172.168.1.1:12:172.168.1.254
172.168.1.1:5:172.168.1.27
172.168.1.254:1:172.168.1.27

Output
6

Example 2

Input
172.168.1.27
172.168.1.254
172.168.1.1:7:172.168.1.254
172.168.1.1:12:172.168.1.254
172.168.1.1:5:172.168.1.27
172.168.1.254:1:172.168.1.27

Output
1

Python3

```
from sys import stdin
import copy

inf = 99999

def get_origin(connection):
    return connection.split(':')[0]

def get_target(connection):
    return connection.split(':')[2]

def get_cost(connection):
    return connection.split(':')[1]

def solution(origin, target, network, visited_nodes):
    # initialize variables
    local_network = copy.deepcopy(network)
    min_path_cost = inf

    # get connections to be analyzed: these are the ones that has origin and are not visited now
    connections = list(filter(lambda x: get_origin(x) == origin or get_target(x) == origin,
local_network))
    for visited in visited_nodes:
        connections_to_remove = list(filter(lambda x: get_origin(x) == visited
                                                      or get_target(x) == visited, connections))
        for connection_to_remove in connections_to_remove:
            connections_to_remove.remove(connection_to_remove)

    connections.sort(key=(lambda x: int(get_cost(x))))

    # mark as visited node
    visited_nodes.append(origin)

    while connections:
        connection_to_explore = connections.pop(0)
        local_network.remove(connection_to_explore)

        # if connection reaches target return cost
        if get_origin(connection_to_explore) == target or get_target(connection_to_explore) ==
target \
                and int(get_cost(connection_to_explore)) < min_path_cost:
            return int(get_cost(connection_to_explore))
```

```
        else:
            node_to_explore = get_origin(connection_to_explore) if
get_origin(connection_to_explore) != origin \
                else get_target(connection_to_explore)
            this_path_cost = solution(node_to_explore, target, local_network, visited_nodes)
            if this_path_cost is not None:
                this_path_cost += int(get_cost(connection_to_explore))
                if this_path_cost < min_path_cost:
                    min_path_cost = this_path_cost
                else:
                    pass
            else:
                pass

    return min_path_cost if min_path_cost != inf else None

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

# Read the origin node from standard input
origin_node = input()

# Read the target node from standard input
target_node = input()

# Create an empty network list
network = []

# Read the network data from following lines and remember to remove the trailing \n character
from each line
for line in stdin:
  # Appedd the connection between two nodes to the network data list
  network.append(line.strip())

# Finally execute the solution based on Dijkstra shortest path algorithm
print(solution(origin_node, target_node, network, []))
```

C++

```
#include <vector>
#include <iostream>
#include <queue>
#include <map>
#include <limits>
using namespace std;

typedef pair<int,int> WArc;
typedef vector<vector<WArc> > Wgraph;
const int infinit = numeric_limits<int>::max();

int dijkstra(const Wgraph& G, int initial, int final) {
	int n = G.size();
	vector<int> d(n, infinit); d[initial] = 0;
	vector<int> p(n, -1);
	vector<bool> S(n, false);
	priority_queue<WArc, vector<WArc>, greater<WArc> > Q;
	Q.push(WArc(0, initial));
	while (not Q.empty()) {
		int u = Q.top().second; Q.pop();
		if (not S[u]) {
			S[u] = true;
			for (WArc a : G[u]) {
				int v = a.second;
				int c = a.first;
				if (d[v] > d[u] + c) {
					d[v] = d[u] + c;
					p[v] = u;
					Q.push(WArc(d[v], v));
				}
			}
		}
	}
	return d[final];
}

int main() {
	string dest, orig;
	cin >> orig >> dest;
	string path;
	int i = 0;
	map<string,int> m;
	vector<string> entry;
	while (cin >> path) {
		entry.push_back(path);
		int pos =  path.find(":");
		int posr = path.rfind(":");
		string ori = path.substr(0,pos);
		string dst = path.substr(posr+1);
		if (m.find(ori) == m.end()) m[ori] = i++;
		if (m.find(dst) == m.end()) m[dst] = i++;
	}
	int n = m.size();
	Wgraph g(n);

	for (string path:entry) {
		int pos =  path.find(":");
		int posr = path.rfind(":");
		int cost = stoi(path.substr(pos+1, posr-pos));
```

```
                string ori = path.substr(0,pos);
                string dst = path.substr(posr+1);
                g[m[ori]].push_back({cost, m[dst]});
                g[m[dst]].push_back({cost, m[ori]});
        }
        int node  = m[orig];
        int final = m[dest];

        cout << dijkstra(g, node, final) << endl;
}
```

29 Blobs

22 points

Introduction

Given a binary (black and white) image (2D matrix), calculate how many blobs it contains. A blob is single black pixel or a set of connected black pixels.

Two black pixels are connected if there is a path of black pixels between them. Adjacent pixels (A) of a pixel (P) are those connected horizontally and vertically, like in the representation below:

```
 A
APA
 A
```

Input

A line with 2 positive numbers, which are the columns and the rows of the image, followed by as many lines as image rows, where the white pixels are depicted by '.' and the black ones by '#'.

Output

The number of blobs in the image.

Example 1

Input

```
7 5
.......
.##....
.##.##.
.....#.
.......
```

Output

```
2
```

Example 2

Input

```
3 4
...
.#.
.#.
...
```

Output

```
1
```

Python3

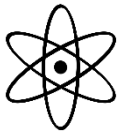
```python
# This function erases in a recursive way all the black pixels connected to the pixel (x,y)
def eraseBlob(img, w, h, x, y, fg, bg):
  #print(x,y, img[x][y])
  if img[x][y] != fg:
    return
  else:
    img[x][y] = bg
    if x+1 < h:
        eraseBlob(img,w,h,x+1,y,fg,bg)
    if x-1 >= 0:
        eraseBlob(img,w,h,x-1,y,fg,bg)
    if y+1 < w:
        eraseBlob(img,w,h,x,y+1,fg,bg)
    if y-1 >= 0:
        eraseBlob(img,w,h,x,y-1,fg,bg)

# Debug: print image
def printImg(img, w, h):

  for i in range(h):
    for j in range(w):
      print(img[i][j], end="")
    print("")
  print("")

################################################
# Read size w x h
w, h = map(int, input().split())

# Initialize image
img = [x[:] for x in [['x'] * w] * h]

# Read image
for i in range(h):
  line = input()
  for j in range(w):
    img[i][j] = line[j]

#printImg(img,w,h)

# Count blobs
bg = "."
fg = "#"
count = 0
for i in range(h):
  for j in range(w):
    # We found a new part
    if img[i][j] == fg:
      count += 1
      eraseBlob(img,w,h,i,j,fg,bg)

print(count)

#printImg(img,w,h)
```

C++

```
#include <bits/stdc++.h>
using namespace std;

void erase(vector<vector<char> > &v, int i, int j) {
	int n = v.size();
	int m = v[0].size();
	if (v[i][j] == '.') return;
	else {
		v[i][j] = '.';
		if (i+1 < n) erase(v,i+1,j);
		if (i-1 >= 0) erase(v,i-1,j);
		if (j+1 < m) erase(v,i,j+1);
		if (j-1 >= 0) erase(v,i,j-1);
	}
}

int main() {
	int n, m;
	cin >> n >> m;
	vector<vector<char> > v(m,(vector<char>(n)));
	for (auto &x:v)
		for (auto &y:x)
			cin >> y;

	int count = 0;
	for (int i=0; i<m; ++i) {
		for (int j=0; j<n; ++j) {
			if (v[i][j] == '#') {
				++count;
				erase(v,i,j);
			}
		}
	}
	cout << count << endl;
}
```

30 Quantum gates

23 points



Introduction

Quantum computing is the new paradigm of the computation, it is said to be the next technology revolution! It takes advantage of the ability of subatomic particles to exist in more than one state at any time. Due to the way the tiniest of particles behave, operations can be done much more quickly and use less energy than classical computers.

In classical computing, a bit is a single piece of information that can exist in two states: 1 or 0. Quantum computing uses quantum bits, aka qubits. These qubits have also two states, but these states can exist simultaneously because of the superposition of these values. This is the magic of quantum computing, a qubit can have the two values at the same time until it is measured!

This behavior is described in terms of the probability that when the qubit is measured becomes 0 or 1. It is expressed through bra-ket notation that basically means to put everything inside | and >, so the 0 now is |0>, the 1 became |1> and the qubit q is notated as |q>. So,

$$|q>= a|0> +b|1>$$

means that the probability to be |0> is a2 and the probability to be 1 is b2 where the coefficients a and b are complex numbers. Usually qubits are also expressed following as a vector:

$$|q>= \begin{vmatrix} a \\ b \end{vmatrix}$$

As an example, to describe a qubit with the same probability (50%) to become a 0 or 1 then the value of the coefficients a and b should be:

$$a^2 = \frac{1}{2} \rightarrow a = \sqrt{\frac{1}{2}}$$

$$b^2 = \frac{1}{2} \rightarrow b = \sqrt{\frac{1}{2}}$$

And the qubit will be:

$$|q>= \frac{1}{\sqrt{2}}|0> +\frac{1}{\sqrt{2}}|1>$$

As in classical computing, there are logical gates (AND, OR, NOT, ...) to define the circuits; in quantum computing they are known as quantum gates and are used to create quantum circuits. Single-qubit gates are the simplest and these are the Pauli (X, Y and Z) and the Hadamard (H) gates. Since a qubit can be thought of like an imaginary sphere. Whereas a classical bit can be in two states – at either of the two poles of the sphere – a qubit can be any point on the sphere. Consequently, the quantum gates would manipulate the qubit direction inside the sphere. To support such transformations the quantum gates are represented using matrices.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

But despite all this complexity, to solve a quantum circuit is as easy as doing operations with matrix and vectors, so if we represent the qubits as vectors, to solve a quantum circuit is to multiply a matrix with a vector.

Then the circuit, given the input qubit |q> and the quantum gates G1, G2, G3..., GN in line the resulting qubit |q'> will be doing the following operations:

- **|q1> = G1 * |q>**
- **|q2> = G2 * |q1>**
- **|q3> = G3 * |q2>**
- **...**
- **|q'> = GN * |q(n-1)>**

We ask you to program an algorithm that given an input qubit and a quantum circuit using the quantum gates described before, it outputs the resulting qubit.

Input

The input is described in five lines. The first two lines describe the coefficient a, the first line represents the real part of coefficient and the second is the complex (i) part of coefficient. The third and fourth line describe the coefficient b, the third line represents the real part of coefficient and the fourth is the complex (i) part of coefficient. Finally, the last line represents the quantum circuit as a string with the sequence of gates

Output

Write output in terms of (a1 + a2i) |0> + (b1 + b2i) |1> with three decimals precision.

Example 1

Considering the qubit |q> = 1/√2+0i|0> + 1/√2+0i|1> and this quantum circuit: a H gate following by an X gate.

Input
```
0.707
0
0.707
0
HX
```

Output
```
(0.000+0.000i)|0> + (1.000+0.000i)|1>
```

Example 2

Considering the qubit |q> = 1+0i|0> + 0+0i|1> and this quantum circuit: just an X gate.

Input
```
1
0
0
0
X
```

Output
```
(0.000+0.000i)|0> + (1.000+0.000i)|1>
```

Example 3

Considering this qubit |q> = 1.006|0> + 0+0.111i|1> and this quantum circuit: a Z gate followed by a H gate.

Input
```
1.006
0
0
0.111
ZH
```

Output
```
(0.711-0.078i)|0> + (0.711+0.078i)|1>
```

Example 1 developed step by step

Given the qubit (0.707+0.0i)|0> + (0.707+0.0i)|1> and the circuit HX find out the resulting qubit. First step means to apply the gate H to the current qubit:

$$|q1>= H \times |q>= \frac{1}{\sqrt{2}} \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \times \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

For the second step just apply the gate X to the qubit |q1>:

$$|q2>= X \times |q1>= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

So, the resulting qubit is (0.000+0.000i)|0> + (0.100+0.000i)|1>

Matrix operations

Multiplying a real value (A) and a matrix:

$$A \times \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} A \times a & A \times b \\ A \times c & A \times d \end{bmatrix}$$

Multiplying a matrix and a vector:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} A \times a + B \times b \\ A \times c + B \times d \end{bmatrix}$$

Basic operations with complex numbers

Given two complex numbers z1 and z2,

- **z1 = x1 + (y1)i**
- **z2 = x2 + (y2)i**

then the basic operations are:

- **Adding two complex numbers: z1 + z2 = (x1 + x2) + (y1 + y2)i**
- **Subtracting two complex numbers: z1 - z2 = (x1 - x2) + (y1 - y2)i**
- **Multiplying a complex number z1 by a real number A: A * z1 = A * x1 + (A * y1)i**
- **Multiplying two complex numbers: z1 * z2 = (x1 * x2) - (y1 * y2) + (x1 * y2 + x2 * y1)i**

Python3

```
# Import mathematical libs needed for complex numbers and other functions
import cmath
import math

# Each quantum gates is represented using by a matrix

# Hadamard (H) gate
h = 1/math.sqrt(2)
H = [ complex(h,0), complex(h,0), complex(h,0), complex(-h,0) ]

# Pauli-X gate
X = [ complex(0,0), complex(1,0), complex(1,0), complex(0,0) ]

# Pauli-Y gate
Y = [ complex(0,0), complex(0,-1), complex(0,1), complex(0,0) ]

# Pauli-Z gate
Z = [ complex(1,0), complex(0,0), complex(0,0), complex(-1,0) ]

# Auxiliar matrix
m = [ complex(0,0), complex(0,0), complex(0,0), complex(0,0) ]

# Read the input
# Real part for a
real_part = float(input())
# Imaginary part for a
imaginary_part = float(input())
# Build the complex number z1
z1 = complex(real_part, imaginary_part)
# Real part for b
real_part = float(input())
# Imaginary part for b
imaginary_part = float(input())
# Build the complex number z2
z2 = complex(real_part, imaginary_part)
# Read the description of the circuit as a string
quantum_circuit = input()

# Internal check to validate that the entered data is a valid qubit: a^2 + b^2 = 1
res = pow(z1,2) + pow(z2,2)
if round(res.real) != 1:
  print("Invalid qbit data entered! Please review that the condition a^2 + b^2 = 1 is correct")
else:

  # Initialize the qubit with a and b factors read from input
  q1 = z1
  q2 = z2

  for i in quantum_circuit:
    # print(q1, q2)

    # Get the values for the current quantum gate
    if i == "H":
      m = H

    if i == "X":
      m = X

    if i == "Y":
```

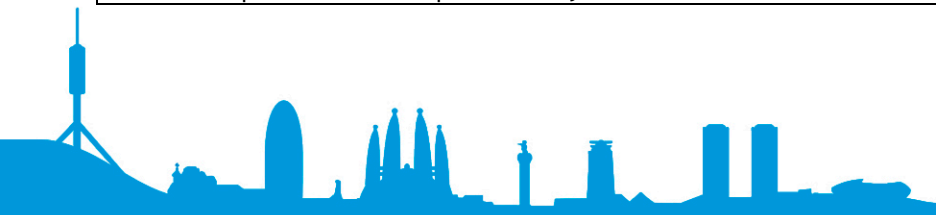
```
      m = Y

    if i == "Z":
      m = Z

    # Apply the multiplying quantum gate matrix - qubit vector and store the values in temporal
variables
    t1 = m[0] * q1 + m[1] * q2
    t2 = m[2] * q1 + m[3] * q2

    # Finally update the qubit (a and b factors) from the temporal variables
    q1 = t1
    q2 = t2

  print("("+"{:0.3f}".format(round(q1.real,3))+"{:+0.3f}".format(round(q1.imag,3))+"i)|0> +
("+"{:0.3f}".format(round(q2.real,3))+"{:+0.3f}".format(round(q2.imag,3))+"i)|1>")
```

C++

```
#include <stdint.h>
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <cmath>

typedef struct TSComplexNumber
{
        float real;
        float i;
} ComplexNumber;

typedef struct TSQubit
{
        ComplexNumber alfa;
        ComplexNumber beta;
} Qubit;

typedef struct TSQuantumGate
{
        float scalar;
        ComplexNumber matrix[2][2];
}QuantumGate;

QuantumGate H;
QuantumGate X;
QuantumGate Y;
QuantumGate Z;

void initMatrices()
{
        H.scalar = 1/sqrt(2);
        H.matrix[0][0].real =  1; H.matrix[0][0].i = 0;
        H.matrix[0][1].real =  1; H.matrix[0][1].i = 0;
        H.matrix[1][0].real =  1; H.matrix[1][0].i = 0;
        H.matrix[1][1].real = -1; H.matrix[1][1].i = 0;

        X.scalar = 1;
        X.matrix[0][0].real = 0; X.matrix[0][0].i = 0;
        X.matrix[0][1].real = 1; X.matrix[0][1].i = 0;
        X.matrix[1][0].real = 1; X.matrix[1][0].i = 0;
```

```
        X.matrix[1][1].real = 0; X.matrix[1][1].i = 0;

        Y.scalar = 1;
        Y.matrix[0][0].real = 0; Y.matrix[0][0].i =  0;
        Y.matrix[0][1].real = 0; Y.matrix[0][1].i = -1;
        Y.matrix[1][0].real = 0; Y.matrix[1][0].i =  1;
        Y.matrix[1][1].real = 0; Y.matrix[1][1].i =  0;

        Z.scalar = 1;
        Z.matrix[0][0].real =  1; Z.matrix[0][0].i = 0;
        Z.matrix[0][1].real =  0; Z.matrix[0][1].i = 0;
        Z.matrix[1][0].real =  0; Z.matrix[1][0].i = 0;
        Z.matrix[1][1].real = -1; Z.matrix[1][1].i = 0;
}

/*
  c1 = a1 + (b1)i
  c2 = a2 + (b2)i
  c1 + c2 = a1+a2 +(b1 + b2)i
 */
ComplexNumber SumComplexs(ComplexNumber &a_firstComplex, ComplexNumber &a_secondComplex)
{
        ComplexNumber tmpRes;
        tmpRes.real = a_firstComplex.real + a_secondComplex.real;
        tmpRes.i    = a_firstComplex.i + a_secondComplex.i;

        return tmpRes;
}


/*
  c1 = a1 + (b1)i
  c2 = a2 + (b2)i
  c1 * c2 = (a1*a2)-(b1*b2)+(a1*b2 + a2*b1)i
 */
ComplexNumber ProductComplexs(ComplexNumber &a_firstComplex, ComplexNumber &a_secondComplex)
{
        ComplexNumber tmpRes;
        tmpRes.real = a_firstComplex.real*a_secondComplex.real -
a_firstComplex.i*a_secondComplex.i;
        tmpRes.i    = a_firstComplex.real*a_secondComplex.i +
a_firstComplex.i*a_secondComplex.real;

        return tmpRes;
}

/*
  c1 = a1 + (b1)i
  A*c1  = (A*a1)+(A*b1)i
 */
ComplexNumber ProductScalarComplex(float a_scalar, ComplexNumber &a_complex)
{
        ComplexNumber tmpRes;
        tmpRes.real = a_scalar * a_complex.real;
        tmpRes.i    = a_scalar * a_complex.i;

        return tmpRes;
}

/*
    | a b |   | A |   | Aa + Bb |
    |     | * |   | = |         |
```

```
    | c d |   | B |   | Ac + Bd |
*/
Qubit ProductQuantumGateQubit(QuantumGate &a_quantumGate, Qubit &a_qubit)
{
        Qubit res;
        ComplexNumber aux1, aux2;

        aux1 = ProductComplexs(a_quantumGate.matrix[0][0], a_qubit.alfa);
        aux2 = ProductComplexs(a_quantumGate.matrix[0][1], a_qubit.beta);
        res.alfa = SumComplexs(aux1, aux2);

        aux1 = ProductComplexs(a_quantumGate.matrix[1][0], a_qubit.alfa);
        aux2 = ProductComplexs(a_quantumGate.matrix[1][1], a_qubit.beta);
        res.beta = SumComplexs(aux1, aux2);

        res.alfa = ProductScalarComplex(a_quantumGate.scalar, res.alfa);
        res.beta = ProductScalarComplex(a_quantumGate.scalar, res.beta);

        return res;
}

void ShowComplexNumber(ComplexNumber &a_complex)
{
        float tmpRounded = round(a_complex.real * 1000.0) / 1000.0;
        std::cout << std::fixed;
        std::cout << std::setprecision(3) << tmpRounded;

        tmpRounded = round(a_complex.i * 1000.0) / 1000.0;
        if (tmpRounded >= 0)
                std::cout << "+" ;

        std::cout << std::setprecision(3) << tmpRounded << "i";
}

int main()
{
        Qubit qIn, qRes;
        std::string quantumCircuit;
        std::string aux;

        initMatrices();

        // Read first complex number for the |0>  coeficient
        std::cin >> aux;
        qIn.alfa.real = atof(aux.c_str());
        std::cin >> aux;
        qIn.alfa.i = atof(aux.c_str());

        // Read second complex number for the |1> coeficient
        std::cin >> aux;
        qIn.beta.real = atof(aux.c_str());
        std::cin >> aux;
        qIn.beta.i = atof(aux.c_str());

        // Read quantum circuit
        std::cin >> quantumCircuit;

        // Initialize qbit result as  initial qubit
        qRes.alfa.real = qIn.alfa.real;
        qRes.alfa.i = qIn.alfa.i;
        qRes.beta.real = qIn.beta.real;
        qRes.beta.i = qIn.beta.i;
```



```
        // Perfom operations of quantum gates one by one
        for (uint8_t iCnt = 0; iCnt < quantumCircuit.length(); iCnt++)
        {
                switch(quantumCircuit[iCnt])
                {
                        case 'H':
                                qRes = ProductQuantumGateQubit(H, qRes);
                                break;
                        case 'X':
                                qRes = ProductQuantumGateQubit(X, qRes);
                                break;
                        case 'Y':
                                qRes = ProductQuantumGateQubit(Y, qRes);
                                break;
                        case 'Z':
                                qRes = ProductQuantumGateQubit(Z, qRes);
                                break;
                        default:
                                break;
                }
        }

        // Write output in terms of (a0 + b0i) |0> + (a1 + b2i) |1> with three decimals precision
        std::cout << "(" ;
        ShowComplexNumber(qRes.alfa);
        std::cout << ")|0> + (" ;
        ShowComplexNumber(qRes.beta);
        std::cout << ")|1>" ;

        std::cout << std::endl;

}
```

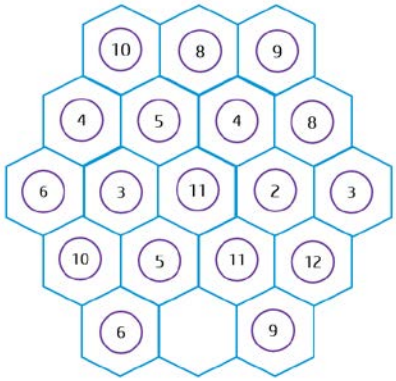
31 Settlers of CodeWars

24 points

Introduction

Do you know the multiplayer board game Settlers of Catan? With more than 22 million copies sold in 30 different languages, it is one of the most popular board games in the world.

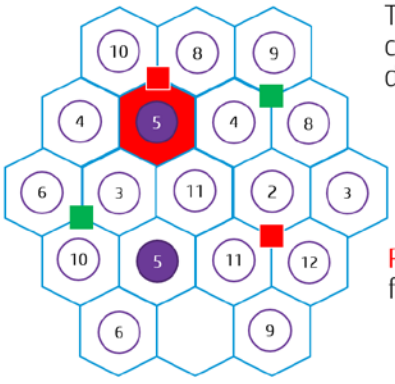
A Catan board is made up of hexagonal shaped tiles that fit together. Each tile is assigned a number between 2-12 (not including 7), and a resource. In addition to these, there will be one blank line called a desert, which we will refer to as the number 0. For the purpose of this problem, we will only focus on the number of the tile and ignore the resource.



Catan always starts the same way. Each player needs to choose two crossroads of the board. A crossroads is the point where three tiles intersect.

Like many board games, Catan is played with by rolling two dices, each die having six faces. The result of a roll determines which players can get resources. Thus, you can increase your chances of getting at least one resource per roll if your chosen crossroads target the major variety of numbers. Follow the examples to get a better understanding about the results of a roll and how players can get resources.

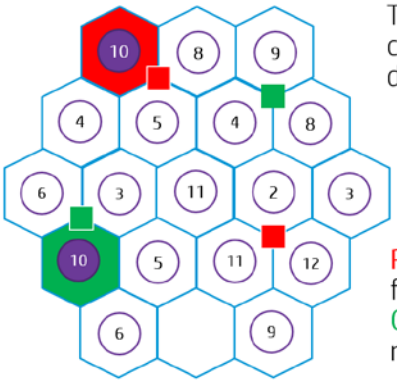
Example 1



Two players choose two crossroad each. The roll of dice is:

Red Player can get a resource from tile "5"

Example 2



Two players choose two crossroad each. The roll of dice is:

Red Player can get a resource from tile "10".
Green Player can also get a resource form another tile "10".

But, what are the two best crossroads to choose?

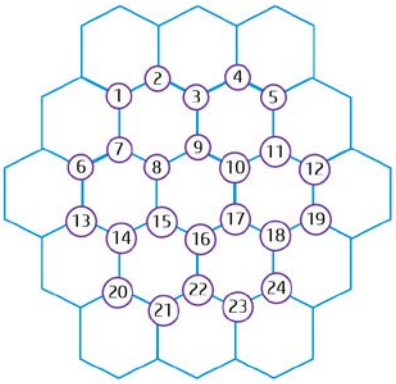
Your objective is to build a program that, given a Catan board, computes which two crossroads, combined, maximizes the chances of getting at least one resource per roll. These must be inner crossroads (ones where 3 tiles intersect).

Input

The input of your program is a sequence of numbers, in different lines, which make up the board according to the numbers of each tile. Each number is separated by a single token. The first line has 3 numbers, the second 4 numbers, the third 5 numbers, the fourth 4 numbers and last line 3 numbers again.

Output

The output must be two numbers, separated by a single token and sorted in ascending order, each one referring to a crossroad identifier. Those two crossroads are the best initial choice. Check the figure below to see the map of crossroad identifiers.



If there is more than one best choice, separate them with a space. Sort the results in ascending order with respect to the first number of each choice (see Example 2).

Example 1

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

Output
```
20 24
```

Example 2

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

Output
```
6 16
6 17
6 23
```

Example 3

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

Output
```
9 19
```

Python3

```python
import sys
import itertools

# 1. Get all board lines
fields    = 3  # (x,y,tile number), 3 fields
hex_list  = [] # define the list that will contain tiles information
tile_row  = 0  # index to travel through "x" dimension of tiles
row       = 0  # index that point to each tile
num_tiles = 0  # define variable num_tiles
offset    = 0  # define offset postion to be used to "y" dimension of tiles

# Find each tile information, x, y and tile number
for line in sys.stdin:
  line_parse = line.split(" ") # Parse line and remove space

  if num_tiles < len(line_parse):
    offset = 0
  else:
    offset = offset + 1

  num_tiles  = len(line_parse)

  cont = 0
  for row in range(row,num_tiles+row):
    hex_list.append([0]*fields) # generate new "tile" array
    hex_list[row][0] = tile_row      # "x" of tile
    hex_list[row][1] = cont+offset   # "y" of tile
    hex_list[row][2] = int(line_parse[cont]) # "tile number" field
    cont = cont + 1

  row = row +1
  tile_row = tile_row + 1

# Algorithm to find each addressable crossroad
crossroads  = []
n_tiles     = len(hex_list)
coordinates = [x[0:2] for x in hex_list]
sequence    = [[1,1],[1,0],[0,-1],[-1,-1],[-1,0],[0,1],[1,1]]
#sequence    = [[-1,-1],[-1,0],[0,1],[1,1],[1,0],[0,-1],[-1,-1]]
```

```
for n in range(n_tiles):
  tile = coordinates[n]

  for k in range(6):
    # Follow sequence to find each tile neighbor
    operator_1   = sequence[k]
    operator_2   = sequence[k+1]

    neighbor_1   = [x+y for x,y in zip(tile,operator_1)]
    neighbor_2   = [x+y for x,y in zip(tile,operator_2)]

    # Check if found neighbor is a defined coordinate or not
    if (neighbor_1 in coordinates) and (neighbor_2 in coordinates):
      n1_index = coordinates.index(neighbor_1)
      n2_index = coordinates.index(neighbor_2)

      scores = [hex_list[n1_index][2],hex_list[n2_index][2],hex_list[n][2]]
      scores.sort()

      crssrd = [neighbor_1, neighbor_2, tile]
      crssrd.sort()
      crssrd.append(scores)

      # Chek if crossroad found has been already found
      if crssrd in crossroads:
        continue
      else:
        crossroads.append(crssrd)
    else:
      #print "Tile not defined"
      continue

## FIND WINNER COMBINATIONS ##
# 1. Find crossroads adjacent to "null" tile to be skipped later
# search     = 0
# score_list = [x[-1] for x in crossroads]
# skip_index  = []

# for sublist in score_list:
#   for k in range(len(sublist)):
#     if sublist[k] == search:
#       skip_index.append(score_list.index(sublist))
#       continue

# 2. Compute all combinations. Check for score condition
settlements = []
p_lut = {2: 1./36, 3: 2./36, 4: 3./36, 5: 4./36, 6: 5./36, 8: 5./36, 9: 4./36, 10: 3./36,
11:2./36, 12:1./36, 0: 0}

settle_index = list(itertools.combinations(range(len(crossroads)),2))

for ind_tuple in settle_index:
  ind     = list(ind_tuple)
  score_1 = crossroads[ind[0]][-1]
  score_2 = crossroads[ind[1]][-1]
  score   = list(set(score_1 + score_2))
  p_score = round(sum([p_lut[x] for x in score]),2)

  ind.append(p_score)
  settlements.append(ind)

scores      = [x[-1] for x in settlements]
```

```
best_choice_index  = [x for x, value in enumerate(scores) if value==max(scores)]
best_choice = [settlements[k] for k in best_choice_index]
best_choice.sort()

for k in best_choice:
  val_1 = k[0]+1
  val_2 = k[1]+1
  #val_3 = k[2]

  print(str(val_1) + " " + str(val_2))
  #print k
```

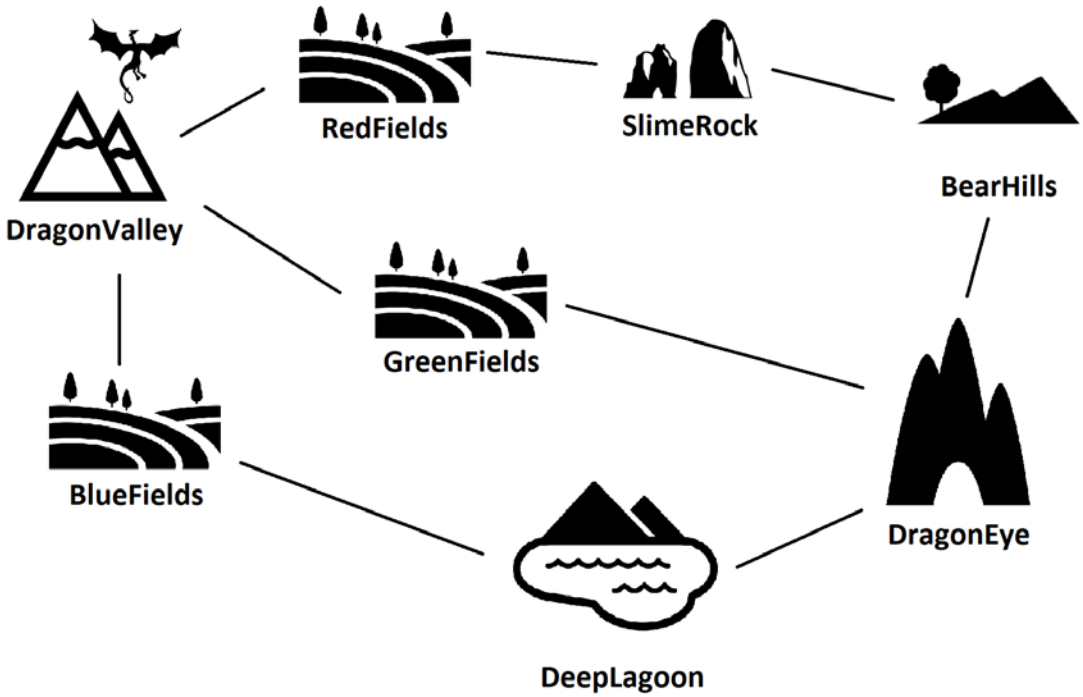
32 The Dragon King

31 points

Introduction

In a remote part of this world there is a place called the DragonValley, land of dragons. The last Dragon King died recently and, according to the ancient tradition, the king's successors must prove their bravery to become the next Dragon King. They do this by competing to see who is first to reach the Dragons' Gem, which is stored at the top of the highest mountain in the land: the DragonEye.

However, there are a lot of different paths that link DragonValley to the DragonEye, and these paths have a peculiarity: the direction of the wind changes every 30 minutes. Therefore, the time it takes a dragon to fly from one point to another varies from the half hour to the next.



Please, help our friend Spyro find the quickest path from DragonValley to DragonEye and become the next Dragon King!

HINT: Note that if a point is reached during the first half hour of the journey, the dragon can either take the next path immediately, or wait until the next half hour and only then take the second path. For example, if there is a connection "A B 50 15" and the dragon reaches A at minute 20, the dragon can then continue on to B right away, which will take, a total of 20 + 50 = 70 minutes, or alternatively, it can wait 10 minutes in node A and then leave for B, which will take a total of 20 + 10 + 15 = 45 minutes.

Input

The input describes the connections between different points of the dragon lands. It starts with a number N that indicates the amount of connections in the sky map, followed by the N connections.

Each connection is represented by 4 words: - The origin point O. - The destination point D. - The time (in minutes) required to go from the O to D during the first half of an hour. - The time (in minutes) required to go from the O to D during the second half of an hour.

The input must always contain the points DragonValley and DragonEye, which indicate the beginning and the ending respectively.

Output

The output is the time of the quickest path from the DragonValley to the DragonEye, and the stops along that path.

If there are 2 paths with an identical time, the path with fewer stops wins. If there are 2 paths with the same time and the same number of stops, the first path in alphabetical order wins. For example: if the two quickest paths are "DragonValley A DragonEye" and "DragonValley B DragonEye", the first path wins because A goes before B.

Example 1

Input

```
5
DragonValley GreenFields 10 5
DragonValley BlueFields 15 5
DragonValley RedFields 20 5
RedFields DragonEye 5 10
GreenFields DragonEye 16 20
```

Output

```
Minimum time: 25
Minimum path: DragonValley RedFields DragonEye
```

Example 2

Input

```
5
DragonValley GreenFields 10 5
DragonValley BlueFields 15 5
DragonValley RedFields 20 5
RedFields DragonEye 5 10
GreenFields DragonEye 15 20
```

Output

```
Minimum time: 25
Minimum path: DragonValley GreenFields DragonEye
```

Example 3

Input

```
9
DragonValley GreenFields 10 5
DragonValley BlueFields 15 5
DragonValley RedFields 20 5
RedFields SlimeRock 20 8
GreenFields DragonEye 210 184
SlimeRock BearHills 35 10
BearHills DragonEye 22 22
BlueFields DeepLagoon 34 10
DeepLagoon DragonEye 43 27
```

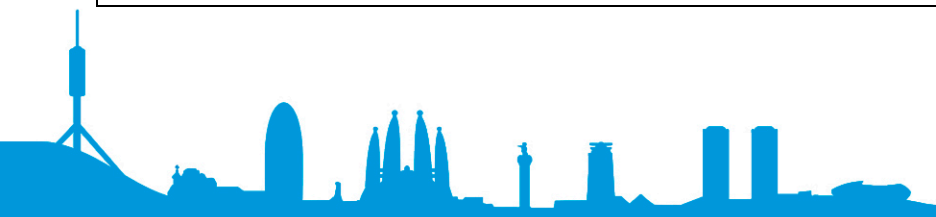
Output

```
Minimum time: 67
Minimum path: DragonValley BlueFields DeepLagoon DragonEye
```

Java

```java
import java.util.Scanner;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.HashMap;
import java.util.TreeSet;

// Problem 32: The Dragon King
public class Main {

	// Adjacency list. Edges info ((dest, firstHalf, secondHalf)) by origin node name.
	static HashMap<String, LinkedList<Edge> > edges = new HashMap<String, LinkedList<Edge>
>();

	// All nodes discovered so far by node name (visited or reachable).
	static HashMap<String, Node > nodes = new HashMap<String, Node >();

	// Priority queue with reachable nodes that have not been visited. TreeSet used as a
priority queue because we will need to update nodes in any position within the queue (log(n)).
	static TreeSet<Node> queue = new TreeSet<Node>();

	public static void main(String[] args) throws Exception {
		// Parse edges information from input:
		//Scanner reader=new Scanner(new FileInputStream("Prob32a.in"));  // Read from
sample test file
		Scanner reader=new Scanner(System.in);  // Read from standard input (required
for the Judge)

		int N = reader.nextInt();
		while (N-- > 0) {
			String orig = reader.next();
			Edge e = new Edge();
			e.dest = reader.next();
			e.firstHalf = reader.nextInt();
			e.secondHalf = reader.nextInt();
			if (!edges.containsKey(orig))
```

```
                                edges.put(orig, new LinkedList<Edge>());
                        edges.get(orig).add(e);
                }

                // Set the start node as the first reachable node at time 0:
                Node n =  new Node();
                n.name = "DragonValley";
                n.time = 0;
                n.path = new ArrayList<String>();
                n.path.add(n.name);
                nodes.put(n.name, n);
                queue.add(n);

                while (selectNext());  // Iterate until DragonEye is reached.

                reader.close();
        }

        /***
         * All visited nodes have the minimum time and the most optimal path (number of steps,
then alpha order) from
         * "DragonValley" set.
         * In each selectNext() invocation, we select the non-visited node that is closest to
"DragonValley" through
         * any of the visited nodes (there is no better way to reach it from "DragonValley") and
we mark it as visited
         * (so we do not re-visit it later).
         * Then, we consider all the outgoing edges from this node to find new reachable nodes
and potentially improve
         * access to known reachable nodes.
         * @return true if we need to keep searching for the solution.
         */
        @SuppressWarnings("unchecked")
        private static boolean selectNext() {

                if (queue.size()<1) {
                        System.err.println("Unexpected situation: There is no way to reach the
DragonEye");
                        return false;
                }

                Node n = queue.first(); // Pick the closest node from the priority queue.
                //System.err.println("Processing "+n.name);
                queue.remove(n);
                n.visited = true;

                if (n.name.equals("DragonEye")) {  //  Found the optimal way to reach
"DragonEye"
                        StringBuffer fullPath = new StringBuffer();
                        for (String place : n.path)
                                fullPath.append(" "+place);
                        System.out.println("Minimum time: "+n.time);
                        System.out.println("Minimum path:"+fullPath.toString());
                        return false;
                }

                // Check what nodes can we reach from this node (if any).
                if (edges.containsKey(n.name)) {
                        for (Edge e : edges.get(n.name)) {
                                // Build a tentative representation of the destination node
going through this edge (e).
                                Node tentative = new Node();
```



```
                                tentative.name = e.dest;
                                tentative.time = n.time +
Math.min(delayFirstHalf(n.time)+e.firstHalf, delaySecondHalf(n.time)+e.secondHalf);  // Consider
the best option including delays
                                tentative.path =  (ArrayList<String>) n.path.clone();
                                tentative.path.add(tentative.name);

                                Node dest = nodes.get(e.dest);
                                if (dest == null) {  // First time we can reach this node.
                                        nodes.put(tentative.name, tentative);  // Add to
nodes.
                                        queue.add(tentative);                  // Add to
priority queue.
                                }
                                else if ( !dest.visited && tentative.compareTo(dest)<0) { //
Tentative is better than previous node info (less time, steps or lexicographical order)
                                        nodes.put(tentative.name, tentative); // Replace the
node info.

                                        // Update the priority queue details.
                                        queue.remove(dest);
                                        queue.add(tentative);
                                }
                        }
                }

                return true;
        }

        private static int delayFirstHalf(int minutes) {
                return isFirstHalf(minutes) ? 0 : 60-(minutes%60);
        }
        private static int delaySecondHalf(int minutes) {
                return isFirstHalf(minutes) ? 30-(minutes%60) : 0;
        }
        private static boolean isFirstHalf(int minutes) {
                return minutes%60 < 30;
        }

        static class Edge{
                public String dest;
                public int firstHalf;
                public int secondHalf;
        }
        static class Node implements Comparable<Node> {
                public String name;
                public int time;                // Optimal time to get to this node
(Integer.MAX_VALUE)
                public ArrayList<String> path;  // Optimal path to get to this node
                public boolean visited = false;

                @Override
                public int compareTo(Node other) {
                        if (this.time-other.time != 0) // Sort by time first.
                                return this.time-other.time;
                        if (this.path.size()-other.path.size() != 0) // Sort by path length
second.
                                return this.path.size()-other.path.size();
                        for (int i=0;i<this.path.size();i++) // Sort by lexicographically based
on path points.
                                if (this.path.get(i).compareTo(other.path.get(i)) != 0)
                                        return this.path.get(i).compareTo(other.path.get(i));
```

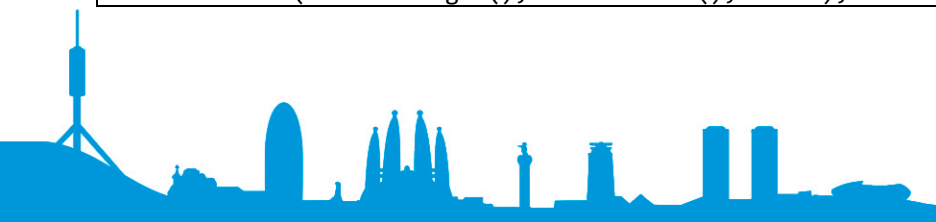
```
                        return 0;
                }
        }

}
```

C++

```
#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <cassert>
#include <memory>
#include <queue>
#include <limits>
#include <functional>
#include <algorithm>

struct SkyLink
{
    int skyPointIdx;
    int time0;
    int time1;

    SkyLink(int sp, int t0, int t1)
        : skyPointIdx(sp)
        , time0(t0)
        , time1(t1)
    {}
};

struct SkyPoint
{
    std::string name;
    std::vector<SkyLink> links;
    int minPathTime;
    int minPathNumSkyPoints;
    std::vector<int> minPathPreviousSkyPointIdx;
    std::vector<int> minPathNextSkyPointIdx;

    static const std::string INI;
    static const std::string END;

    explicit SkyPoint(const std::string& n)
        : name(n)
        , minPathTime(std::numeric_limits<int>::max())
        , minPathNumSkyPoints(std::numeric_limits<int>::max())
    {}
};
const std::string SkyPoint::INI = "DragonValley";
const std::string SkyPoint::END = "DragonEye";

using SkyMap = std::vector<SkyPoint>;

int getSkyPointIdx(std::map<std::string, int>& dict, SkyMap& skyMap, const std::string& key)
{
    auto it = dict.find(key);
    if (it == dict.end())
    {
        int newIdx = skyMap.size();
        skyMap.emplace_back(key);
        dict[key] = newIdx;
        return newIdx;
    }
    else
    {
```

```
        return it->second;
    }
}

void readSkyMap(SkyMap& skyMap, std::map<std::string, int>& dict)
{
    dict.clear();
    skyMap.clear();

    int numLinks = 0;
    std::cin >> numLinks;
    for (int i = 0; i < numLinks; ++i)
    {
        std::string orig, dest;
        int time0 = 0, time1 = 0;
        std::cin >> orig >> dest >> time0 >> time1;

        int origIdx = getSkyPointIdx(dict, skyMap, orig);
        int destIdx = getSkyPointIdx(dict, skyMap, dest);
        skyMap[origIdx].links.emplace_back(destIdx, time0, time1);
        skyMap[destIdx].links.emplace_back(origIdx, time0, time1);
    }
}

void findMinPath(SkyMap& skyMap, int ini, int end)
{
    auto compareSkyPoints = [&skyMap](int idx0, int idx1)
    {
        return skyMap[idx0].minPathTime > skyMap[idx1].minPathTime;
    };

    std::priority_queue<int, std::vector<int>, std::function<bool(int, int)>>
idxsQueue(compareSkyPoints);

    // Set to 0 the min dist of the origin node and add it to the priority queue
    skyMap[ini].minPathTime = 0;
    idxsQueue.push(ini);

    // Start the algorithm to find the minimum path from SkyPoint::INI to SkyPoint::END
    std::vector<bool> visited(skyMap.size(), false);
    while (!idxsQueue.empty())
    {
        // Get the SkyPoint with the minimum path until now
        int idx = idxsQueue.top();
        idxsQueue.pop();

        // If the node is the END, we are done
        if (idx == end)
        {
            break;
        }

        // If the node has been already visited, go to next one
        if (visited[idx])
        {
            continue;
        }

        // Mark the current sky point as visited
        visited[idx] = true;

        // Update all the linked SkyPoint with the minimum distance from the current node
```

```
        int currTime = skyMap[idx].minPathTime;
        for (const SkyLink& link : skyMap[idx].links)
        {
            // Ignore the link if the sky point has been already visited
            if (visited[link.skyPointIdx])
            {
                continue;
            }

            // Get the times to next link
            int time0 = link.time0;
            int time1 = link.time1;

            // Apply the time offset to the link that needs it
            if ((currTime/30)%2 == 0)
            {
                // Current time is in the first half of an hour, offset the time1
                time1 += 30 - currTime%30;
            }
            else
            {
                // Current time is in the second half of an hour, offset the time0
                time0 += 30 - currTime%30;
            }

            // Get the time for the linked SkyPoint
            int bestTime = currTime + ((time0 < time1)? time0 : time1);

            int numSkyPoints = skyMap[idx].minPathNumSkyPoints + 1;

            // Update the linked SkyPoint with the best path from the current SkyPoint
            if (bestTime < skyMap[link.skyPointIdx].minPathTime)
            {
                skyMap[link.skyPointIdx].minPathTime = bestTime;
                skyMap[link.skyPointIdx].minPathNumSkyPoints = numSkyPoints;
                skyMap[link.skyPointIdx].minPathPreviousSkyPointIdx.clear();
                skyMap[link.skyPointIdx].minPathPreviousSkyPointIdx.push_back(idx);
                idxsQueue.push(link.skyPointIdx);
            }
            else if (bestTime == skyMap[link.skyPointIdx].minPathTime)
            {
                if (numSkyPoints < skyMap[link.skyPointIdx].minPathNumSkyPoints)
                {
                    skyMap[link.skyPointIdx].minPathTime = bestTime;
                    skyMap[link.skyPointIdx].minPathNumSkyPoints = numSkyPoints;
                    skyMap[link.skyPointIdx].minPathPreviousSkyPointIdx.clear();
                    skyMap[link.skyPointIdx].minPathPreviousSkyPointIdx.push_back(idx);
                    idxsQueue.push(link.skyPointIdx);
                }
                else if (numSkyPoints == skyMap[link.skyPointIdx].minPathNumSkyPoints)
                {
                    skyMap[link.skyPointIdx].minPathPreviousSkyPointIdx.push_back(idx);
                }
            }
        }
    }

    // Show the minimum time
    std::cout << "Minimum time: " << skyMap[end].minPathTime << std::endl;

    // Here we have the min path from END to INI. Build the min paths from INI to END.
    std::fill(visited.begin(), visited.end(), false);
```



```
    std::queue<int> idxsQ;
    idxsQ.push(end);
    visited[end] = true;
    while (!idxsQ.empty())
    {
        int idx = idxsQ.front();
        idxsQ.pop();
        for (int prevIdx : skyMap[idx].minPathPreviousSkyPointIdx)
        {
            skyMap[prevIdx].minPathNextSkyPointIdx.push_back(idx);
            if (!visited[prevIdx])
            {
                idxsQ.push(prevIdx);
                visited[prevIdx] = true;
            }
        }
    }

    // Show the minimum path
    {
        std::cout << "Minimum path:";
        int idx = ini;
        std::cout << " " << skyMap[idx].name;
        while (idx != end)
        {
        assert(skyMap[idx].minPathNextSkyPointIdx.size() > 0);
            int nextIdx = skyMap[idx].minPathNextSkyPointIdx[0];
            for (int i = 1; i < skyMap[idx].minPathNextSkyPointIdx.size(); ++i)
            {
                int nextIdxCandidate = skyMap[idx].minPathNextSkyPointIdx[i];
                if (skyMap[nextIdxCandidate].name < skyMap[nextIdx].name)
                {
                    nextIdx = nextIdxCandidate;
                }
            }
            idx = nextIdx;
            std::cout << " " << skyMap[idx].name;
        }
        std::cout << std::endl;
    }
}

int main()
{
    std::map<std::string, int> dict;
    SkyMap skyMap;
    readSkyMap(skyMap, dict);
    findMinPath(skyMap, dict[SkyPoint::INI], dict[SkyPoint::END]);
}
```


Anniversary
5th
Anniversary
CodeWars 2019
Barcelona