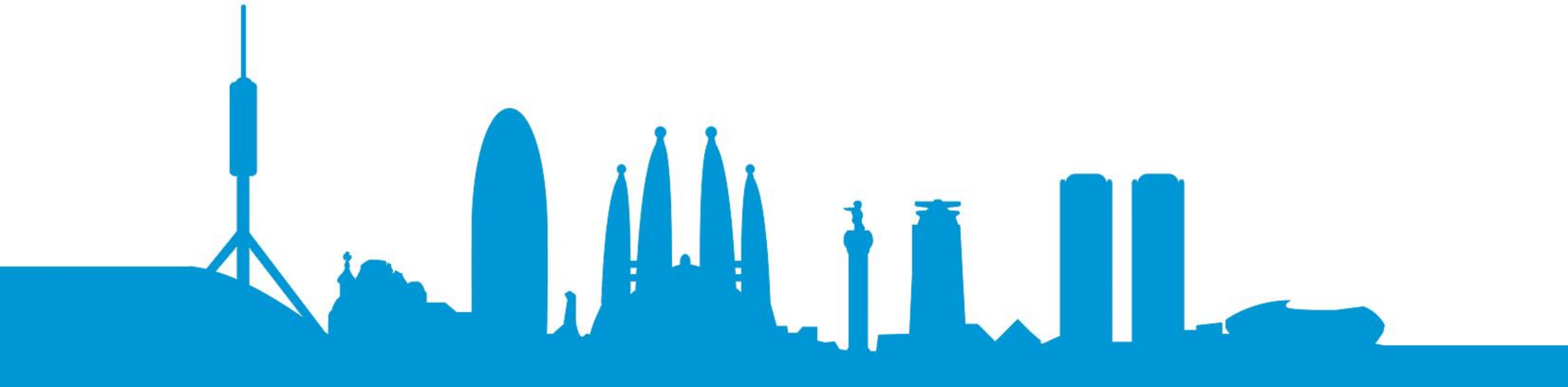


CodeWars 2016
Barcelona

Problems & Solutions

0 Hello, jury!

1 point

NOTE: This is the first of the two problems that can be solved and submitted before the start of the CodeWars competition. Teams are **strongly** encouraged to submit these problems **prior** to the start of the competition – hey, it's basically a free point!

Introduction

Write a program that writes the welcome message.

Input

There is no input for this program.

Output

The program must output the following text.

```
Hello jury!
```

Solution

```
public class Prob00{

   public static void main(String[] args) {

      //output
       System.out.println("Hello jury!");
   }
```

1 Let’s begin introducing ourselves

1 point

NOTE: This is the second of the two problems that can be solved and submitted before the start of the CodeWars competition. Teams are **strongly** encouraged to submit these problems **prior** to the start of the competition – hey, it’s basically a free point!

Introduction

You’ll have no chance to win at HP CodeWars (or life) if you don’t know how to do Input and Output properly. You also won’t do well at CodeWars if you are rude to your judges. Write a program to greet your esteemed judges appropriately. Read in the name of a judge and output your greeting in the appropriate format. If you’re confused at this point, go back and re-read your contest instructions.

Input

The input will be your judge’s first name, a single word with no spaces.

```
Simon
```

Output

Welcome your judge with a friendly greeting.

```
Greetings, Simon the Great! I genuflect in your general direction!
```

Solution

```
public class Prob01 {
   public static void main(String[] args) {

      Scanner read=new Scanner(System.in);
      // input data
      String judge_name;
      judge_name=read.next();
     //output
      System.out.println("Greetings, "+ judge_name+ " the Great! I genuflect in your general direction!");
   }

}
```

2 The cooking library

1 point

Introduction

Peter does not like to show off but he has one of the biggest cooking libraries in town. He loves cooking and, every time he travels to a different country, he brings back cooking books with him. He has even had to pay extra baggage fees for them!

Last month, he travelled to the United States and found that they do not use measuring units Peter understands. They use something called cups and Peter asks himself this question: “it depends on the size of the cup, doesn’t it?” The answer is no. A US cup is defined as 236 milliliters.

Peter needs a program that converts US Cups to milliliters so that he can understand his cooking books.

Input

An amount of US Cups as an integer, no decimals.

2

Output

The equivalent in milliliters.

```
2 US cups are 472 milliliters
```

Solution

```
package prob02;
import java.util.Scanner;

public class Prob02 {


    public static void main(String[] args) {
    int cups, milliliters;
    Scanner leer=new Scanner(System.in);
    cups=leer.nextInt(); //reading input
    milliliters =  cups* 236;
    System.out.println(cups+ " US cups are " + milliliters +" milliliters" );
  }

}
```

3 Internet shoe shopping

2 points

Introduction

Have you ever thought about buying a pair of shoes on your favorite web site? If this is your case, you may know that when you buy on Amazing ™ most of the shoe sizes are stated in US size and you have to be really sure that you choose the correct size to fit your feet.

Of course there is a way to convert from EU size to US size and the rules are quite simple:

- For men shoes: subtract 34.5 from your EU size if your size is smaller than 44 or 35 if your size is 44 or higher
- For ladies shoes: subtract 31.5 from your EU size if your size is smaller than 40 or 32 if your size is 40 or higher

Now it is your time to build an automatic EU to US size converter.

No more returned shoes to Amazing ™!

Input

The input will have two lines, the first one will be a letter, `M` for men shoes and `L` for ladies shoes. The second line will be the EU size.

```
M
43
```

Output

The output will be the equivalent US size.

```
8.5
```

Solution

```
package prob03;

import java.util.Scanner;

public class Prob03 {

    public static void main(String[] args) {

        double pie_eu, pie_us;
        String gender;
        Scanner leer=new Scanner(System.in);

        //input

        //leer.nextLine();
        gender=leer.next(); //read M or L)
        pie_eu=leer.nextDouble(); //read a number
        //operations
        if (gender.equals("M"))
        {
            if (pie_eu<44)
            {
                pie_us = pie_eu - 34.5;
            }
            else
            {
                pie_us = pie_eu - 35;
            }
        }
        else
        {
            if (pie_eu<40)
            {
                pie_us = pie_eu - 31.5;
            }
            else
            {
                pie_us = pie_eu - 32;
            }
        }

        System.out.println(pie_us);
    }
}
```

4 Balanced numbers

2 points

Introduction

A natural number is balanced if the sum of the digits in even positions is equal to the sum of the digits in odd positions. Write a program that tells if a natural number is balanced or not

Input

The input of the program is a set of numbers, ending with a zero.

```
5225
12334
220
48659
917807
0
```

Output

The program must output whether each number is balanced or not.

```
5225 is a balanced number
12334 is not a balanced number
220 is a balanced number
48659 is not a balanced number
917807 is a balanced number
```

Solution

```
#include <iostream>

using namespace std;

int main()
{
    int num;
    cin >> num;
    int last=0;
    int first=0;
    while(num > 9)
    {
        int digit=num%10;
        num/=10;
        last+=digit;
        if(num > 9)
        {
            digit=num%10;
            num/=10;
            first+=digit;
        }
        else
        {
            first+=num;
        }
    }
    if(first==last)
    {
        cout << "OK" << endl;
    }
    else
    {
        cout << "FAIL" << endl;
    }
}
```

5 Speed trap

3 points

Introduction

The cities of Whynot in Mississippi and Zzyzx in California, apart from being in some lists of US towns with the funniest names, compete to have the safest drivers in their roads. There are several speed cameras with radars, each one with its own legal limit. Drivers that pass by them at a higher speed than the limit will be fined. By the end of the day, the Department of Traffic has to award one of the two cities, basing its decision on the amount of fines recorded by the speed cams. You, as an Engineer in charge of the task, are committed to develop a program that automatizes the counting of fines.

Input

The input is a set of records, ending with the # character.
Each record of the speed cameras has 3 values:

- The initial letter of the city name (`W` for Whynot or `Z` for Zzyzx)
- The measured speed, expressed in mph (miles per hour)
- The speed limit

```
W 60 75
Z 61 50
Z 64 38
W 54 75
Z 103 50
Z 47 55
#
```

Output

The output is the amount of fines corresponding to each city (first Whynot and then Zzyzx), followed by a sentence stating which one is the winner of the daily award (see the sample output below). If both cities have the same amount of fines, the script will print “Whynot and Zzyzx inhabitants are equally safe at driving”.

```
0 fines to Whynot
3 fines to Zzyzx
Whynot inhabitants are safer at driving than Zzyzx ones
```

Solution

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

using namespace std;

int main()
{
    float radar, max;
    char city;
    int fines_W = 0, fines_Z = 0;

    while ( true )
    {
        cin >> city;
        cin >> radar;
        cin >> max ;

        if (city == '#')
            break;

        if ( radar > max )
        {
            if (city == 'W')  fines_W++;
            if (city == 'Z')  fines_Z++;
        }

    }

    cout << fines_W << " fines to Whynot" << endl;
    cout << fines_Z << " fines to Zzyzx" << endl;

      if ( fines_W > fines_Z )
            cout << “Whynot inhabitants are safer at driving than Zzyzx ones”
<< endl;

      else if (fines_W < fines_Z )
            cout << “Zzyzx inhabitants are safer at driving than Whynot ones”
<< endl;

      else
            cout << “Whynot and Zzyzx inhabitants are equally safe at driving”
<< endl;

}
```

6 *Ballon d'or*

3 points

Introduction

Every year there is a great controversy when the winner of the FOFA Ballon d'Or is announced. In order to make the voting process as transparent as possible, and avoid complaints, FOFA has asked HP to create an automatic vote counter.

FOFA nominates 15 players from around the world for the final-round voting, and assigns them numbers from 1 to 15. From among these players, each judge will give 3 points to the best player, 2 points to the second best and 1 point to the third. When all the judges have voted, the points for each player are counted and the top three are announced, giving the Ballon d'Or to the player with the most points.

Input

Each line will contain the player numbers a member of the jury awards their points to. The first number is the number of the player with 3 points, the second one is the number of the player with 2 points and the third number is the player with 1 point. A zero will indicate the end of the input.

```
10 7 11        ← Jury gives 3 votes to player 10, 2 to player 7 and 1 to player 11
10 7 1
10 11 7
7 11 4
7 10 11
10 7 11
11 7 10
10 1 7
0
```

Output

The top 3 players ordered by number of points. In case of a draw, the player with lower number will have better ranking. There will be one line for each player containing their player number and their final score.

```
10 18
7 16
11 10
```

Solution

```
//package prob_Balon;

import java.util.*;

public class Prob_balon {

  public static void main(String args[]) {
    // Create a hash map
    Hashtable menu = new Hashtable();
    String str;
    double bal;
    int i,j, player_id,max,max_index;


    int [] players;
    players = new int [15];
    Scanner read=new Scanner(System.in);
    player_id = read.nextInt();

    while (player_id!=0)
    {
     for (i=0; i<3; i++)
     {
        players[player_id] = players[player_id]+3-i;
        player_id = read.nextInt();
     }
    }
    for (j=0; j<3; j++)
    {
     max=0;
     max_index=0;
     for (i=0;i<15; i++)
     {
      if (players[i]>max)
      {
          max=players[i];
          max_index=i;
      }
     }
     System.out.println(max_index + " " + players[max_index]);
     players[max_index]=0;
    }

  }
}
```

7 My binary watch

3 points

Introduction

Your friend is a pain in the neck and, since he does not like wearing a watch, he is constantly grabbing your arm to check the time. You are fed up with him and want to take advantage of you being a programmer (he is not) and have decided to implement your own binary watch so that he quits using yours.

The way a binary watch works is by representing the digits of the time in a binary way, as in the image, where you can see how to code 16:57:19

```
             ///////
             ///////
         ///////////////
2-pow-3  | - - - - - o |
2-pow-2  | - o o o - - |
2-pow-1  | - o - o - - |
2-pow-0  | o - o o o o |
         \\\\\\\\\\\\\\\
             \\\\\\\
             \\\\\\\
           1 6 5 7 1 9
```

Each column of LED's represents the digit of the time. There are six columns, one each for digit. The rows represent the 2-pow value. For instance, 7 in binary is 111 so it is represented as `-ooo` reading from top to bottom.

Input

A time in 24h format

```
23:59:59
```

Output

A graphic representation of what the watch would show

```
- - - o - o
- - o - o -
o o - - - -
- o o o o o
```

Solution

```
import sys

# Take the parameters
hours   = int(sys.argv[1])
minutes = int(sys.argv[2])
seconds = int(sys.argv[3])

# Make an array to store every digit and
# Make a matrix to print at end
binaryHour = range(6)
matrixHour = [[0 for i in range(6)] for j in range(4)]

# Calculate every digit
binaryHour[0] = hours / 10
binaryHour[1] = hours % 10
binaryHour[2] = minutes / 10
binaryHour[3] = minutes % 10
binaryHour[4] = seconds / 10
binaryHour[5] = seconds % 10

# Fill the matrix calculating the binary number
for i in range(6):
        for j in range(4):
                matrixHour[3-j][i] = binaryHour[i] % 2
                binaryHour[i]    = binaryHour[i] / 2

# Print the output
for j in range(4):
        for i in range(6):
                        print 'o' if matrixHour[j][i] == 1 else '-',
        print ' '
```

8 N-ibonacci

3 points

Introduction

One of the most famous series of numbers, besides 4-8-15-16-23-42, is the Fibonacci series. It starts with two ones and the next element is always calculated as the sum of the previous two. That is, the first numbers in the Fibonacci series are:

```
1 1 2 3 5 8 13 21...
```

Tribonacci series is based on the previous one, but instead of starting with two ones and summing the two previous numbers, it starts and sums three:

```
1 1 1 3 5 9 17 31...
```

As you can see, this can be generalized.

Your task is to write a program that is capable of writing any of these n-ibonacci sequences.

Input

The input are two numbers. The first of them indicates the value for *n*, that indicates the amount of ones at the beginning and how many of the previous elements need to be summed. If n is 2, you will get the Fibonacci series, if n is 3 Tribonacci... The second number is the amount of elements of the series the program must output.

```
4 9
```

Output

The first elements of the n-ibonacci series for the provided values on the input.

```
1 1 1 1 4 7 13 25 49
```

Solution

```
package prob08;
import java.util.Scanner;

class Prob08{
  public static void main(String[] args) {

      Scanner read=new Scanner(System.in);

      int n,i,j,numElem;
      int [] serie;
      int suma = 0;

      n=read.nextInt();
      numElem=read.nextInt();

      serie = new int[numElem];

      for (j=0; j<n; j++)
      {
         serie[j]=1;
         System.out.print(serie[j]+" ");
```

```
     }
     for (i=n; i<numElem; i++)
     {
       for (j=1; j<=n; j++)
       {
         suma = suma + serie[i-j];
       }
       serie[i] = suma;
       suma = 0;
       System.out.print(serie[i]+" ");
     }
  }
}
```

9 How far away from our friends?

4 points

Introduction

In the era of social networking, almost everybody is connected to the world using one of the following apps: WhatsApp™, Line™ and so on. But have you thought about how long it takes your messages to travel from your cell phone to any of your friends'?

You have to create a program that calculates the time it takes a message to travel between you and your friends. In order to simplify the calculus, we can assume that we know the distance and the mean speed that the information travels.

You need to write a program to calculate the time to send a message to all your friends and order the result from the smallest to the longest time.

Input

The input will be a sequence of friends with their distance and the mean speed, ending with a dot ('.').

```
Peter 20 1
Jamie 6 3
Eli 20 4
Anna 30 3
.
```

Output

The output of the program is the list of friends and the time it takes for the message to arrive to their cell, ordered from shortest to longest.

```
Jamie 2
Eli 5
Anna 10
Peter 20
```

Solution

```
#include <iostream>
#include <map>

#define MAX_INT (1-(1<<32))

void main()
{
	std::string  name;
	unsigned int  distance;
	unsigned int  meanSpeed;
	unsigned int  timeCall;
	unsigned int  minimumTime = MAX_INT;
	std::map<unsigned int, std::string> friends;
```

```
        //
        // Get all input values until a dot is reached
        //
        std::cin >> name;

        while (name != ".")
        {
                std::cin >> distance;
                std::cin >> meanSpeed;

                //
                // Calculate time
                // takig into account that t = d/v
                // we are asssuming that v = m/s and d= m, so time is always seconds
                // and it is an integer number and there are no collissions
                timeCall = distance/meanSpeed;

                //
                // insert the name of a friend in a map assuming
                // the map is ordering the key from lowest at insertion time
                //
                friends[timeCall] = name;
        }


        for (std::map<unsigned int, std::string>::const_interator it = friends.begin();
            it != friends.end();
    friends++)
        {
                std::count << (*it).second << (*it).first << std::endl;
        }
}
```

10 Don’t be a *bro*!

4 points

Introduction

You have to write in a right way. Sentences start with upper case, and the rest of letters are in lower case. A sentence finishes with a dot and a white space. Afterwards, the same for the next sentences. You have to write a program that converts a text written in a *bro* way to a text written in a right way. You can assume that every sentence will end with a dot and a white space in the input file. Input will never have any word that starts with upper case (such as a person name), except for the pronoun ‘I’, that your program must take into account.

Input

The input of the program is a text with several sentences, ending with a ‘#”.

```
I Used tO tHINK ThaT tHeY WEre thinGS tHE WONderful FolK oF tHE sTorIes WeNt
OUt aNd looKED fOR, bEcaUse tHey wanTEd theM, beCAUse thEy wEre exCitinG aNd
life WAS a Bit DUll, a kINd oF a spORt, As yOU migHt SAy. But ThAt IS not tHe
wAy oF IT wIth tHE tALes tHAt reaLLY mattered, Or The oNEs THat StaY in thE
mINd. foLK SEEM to have beEn JUst lANDed In tHEm, usUAlly theiR pAtHS wERE lAid
That wAY, as yOu PUT it. But I exPEcT tHEy haD lOTs Of CHANces, lIke us, of
tURNinG backK, only tHEy DiD not.
```

Output

The program must output the text correctly written.

```
I used to think that they were things the wonderful folk of the stories went
out and looked for, because they wanted them, because they were exciting and
life was a bit dull, a kind of a sport, as you might say. But that is not the
way of it with the tales that really mattered, or the ones that stay in the
mind. Folk seem to have been just landed in them, usually their paths were laid
that way, as you put it. But I expect they had lots of chances, like us, of
turning back, only they did not.
```

Solution

```
import java.util.Scanner;

public class Main {

   public static void main(String[] args) {

   Scanner read = new Scanner(System.in);
   String text = read.nextLine();
   StringBuilder converted = new StringBuilder(text.toLowerCase());

   converted.setCharAt(0, Character.toUpperCase(converted.charAt(0)));
   for (int i = 2; i < text.length(); i++)
   {
      if (converted.charAt(i - 2) == '.')
      {
         converted.setCharAt(i, Character.toUpperCase(converted.charAt(i)));
      }
   }
   System.out.println(converted);
   }

}
```

11 Spring break! Yahoo!

4 points

Introduction

Easter Sunday is a holiday corresponding to the first Sunday after the first full Moon of the spring. This makes it change from year to year, as you know.
In 1800 the mathematician Carl Friedrich Gauss presented an algorithm to calculate the date of the Gregorian Easter and made corrections until his final proposal in 1816.
Below:

`:=` indicates assignment.
`div` indicates integer division.
`mod` indicates the remainder of the integer division.

Y is the year, M the month and D the day of the Easter Sunday of the given year.

- Compute the following numbers:
```
k := Y div 100
x := Y mod 19
b := Y mod 4
c := Y mod 7
q := k div 4
p := (13  + 8k) div 25
y := (15 – p + k – q) mod 30
z := (19x + y) mod 30
n := (4 + k – q) mod 7
e := (2b + 4c + 6z +n) mod 7
```
- If `z + e ≤ 9`, then `D := 22 + z + e` and `M := 3`.
- Otherwise, if `z = 29` and `e = 6`, then `D := 19` and `M := 4`.
- Otherwise, if `z=28` and `e = 6` and `x > 10`, then `D := 18` and `M := 4`.
- Otherwise, `D := z + e – 9` and `M := 4`.

Write a program to compute the day, month and year (D/M/Y) of the Easter Sunday of every year between the two given in the input.

Input

Two years
```
2010
2016
```

Output

The list of Easter Sundays between those two years, including them.
```
4/4/2010
24/4/2011
8/4/2012
31/3/2013
20/4/2014
5/4/2015
27/3/2016
```

Solution

```
////////////////////////////////////
// X. Easter Sundays
////////////////////////////////////
//
//rule to compile: g++ EasterSundays.cpp -o EasterSundays
//execute example: ./EasterSundays 2010 2016 or ./EasterSundays

#include <iostream>

using namespace std;

int main (int argc, char ** argv)
{

////////////////////////////////////
// Get Data
// Note: It can be simplified as
// much as you want
////////////////////////////////////

   int beginYear = 0, endYear = 0;

   cin >> beginYear;
   cin >> endYear;

////////////////////////////////////
// Compute data
////////////////////////////////////

   int D = 0, M = 0, Y = 0, k = 0, x = 0, b = 0, c = 0, q = 0, p = 0, y = 0, z = 0, n = 0, e = 0;

   for (int i=beginYear; i<=endYear; i++)
   {
      Y = i;
      k = Y / 100;
      x = Y % 19;
      b = Y % 4;
      c = Y % 7;
      q = k / 4;
      p = (13 + (8 * k)) / 25;
      y = (15 - p + k - q) % 30;
      z = ((19 * x) + y) % 30;
      n = (4 + k - q) % 7;
      e = ((2 * b) + (4 * c) + (6 * z) + n) % 7;

      if ((z + e) <= 9)
      {
         D = 22 + z + e;
         M = 3;
      }
      else if ((z == 29) && (e == 6))
      {
         D = 19;
         M = 4;
      }
      else if ((z == 28) && (e == 6) && (x > 10))
      {
         D = 18;
         M = 4;
      }
      else
      {
         D = z + e - 9;
         M = 4;
      }

      //Print results
      cout << D << "/" << M << "/" << Y << endl;

   }

   return 0;
}
```

```
#!/usr/bin/python

import sys

beginYear = int(sys.argv[1])
endYear = int(sys.argv[2])

D = 0
M = 0
Y = 0
k = 0
x = 0
b = 0
c = 0
q = 0
p = 0
y = 0
z = 0
n = 0
e = 0

for i in range(beginYear,endYear+1):
    Y = i
    k = Y / 100
    x = Y % 19
    b = Y % 4
    c = Y % 7
    q = k / 4
    p = (13 + (8 * k)) / 25
    y = (15 - p + k - q) % 30
    z = ((19 * x) + y) % 30
    n = (4 + k - q) % 7
    e = ((2 * b) + (4 * c) + (6 * z) + n) % 7
    if (z+e) <= 9:
        D=22+z+e
        M=3
    elif z==29 and e==6:
        D=19
        M=4
    elif z==28 and e==6 and x>10:
        D=18
        M=4
    else:
        D=z+e-9
        M=4
    print "%d/%d/%d" % (D,M,Y)
```

12 Sometimes, older is better

4 points

Introduction

At HP, a group of engineers in Sant Cugat is collaborating with a group of engineers in San Diego to design and develop a new printing product, completely revolutionary for the world-wide industry. The company CEO, who is located in Silicon Valley, is very interested in this new product and has committed a large part of the company's budget to complete it within the upcoming year.

However, the CEO is concerned about the secrecy of the project, since all the e-mails between Sant Cugat and San Diego are transmitted over the Internet in plain text and, thus, could be intercepted by a third party and used against HP. Since he does not know much about cryptography, he has asked you to implement a simple mechanism to cypher all the communications between Engineers in both places.

After some research, and given the lack of time due to the tight schedule of the project, you have decided to implement one of the first and simplest cypher mechanisms: the Caesar algorithm. This cypher algorithm is named after Roman emperor Julius Caesar, who used it for communicating his military secrets to his generals deployed in the field and, perhaps, also for sending his love letters to Cleopatra.

Your task is to implement a program that is able to decrypt messages that have been cyphered using the Caesar algorithm. The idea behind the Caesar algorithm is simple. Each letter of the original text is substituted by another, following these rules:

- find the letter (which should be encrypted) in the English alphabet
- move K positions further down (the alphabet)
- take the new letter from here
- if "shifting" goes beyond the end of the table, continue from A

For example, if K = 3 (shift value used by Caesar himself), then A becomes D, B becomes E, W becomes Z and Z becomes C and so on, according to the following table:

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

So if the source message was VENI VIDI VICI then after encoding it becomes YHQL YLGL YLFL.

To decrypt a message each letter has to be "shifted back" to decode it - or shifted by 26-K. So if we have encoded message HYHQ BRX EUXWXV, we can apply shift of 26 - K = 26 - 3 = 23 and find the original text to be EVEN YOU BRUTUS.

Input

The first line of the input data will contain two integers separated by a space; the number of lines of encrypted text to be processed and the value of K, that is, the positions that the alphabet has to be shifted to encrypt/decrypt the message.

The following lines will contain encrypted text, consisting of capital Latin letters A ... Z. Each line will be terminated with a dot which should not be decoded.

```
1 3
YHQL YLGL YLFL.
```

Output

Your answer should contain the decrypted message (in a single line, no line splitting needed) and the final dot in the encrypted message.

```
VENI VIDI VICI.
```

Solution

```
#include <vector>
#include <cstdint>
#include <iostream>
#include <string>
#include <sstream>
#include <map>
#include <utility>

typedef std::map<char,char> dictionary_t;

std::vector<uint32_t> split_input(std::string line, char delim) {
        std::vector<uint32_t> output;
        std::stringstream ss(line);

  std::string item;
  while (std::getline(ss, item, delim)) {
    output.push_back(std::stoi(item));
  }

  return output;
}

void create_dictionaries(uint32_t k, dictionary_t& cypher, dictionary_t& uncypher) {
        static const char input [] = {'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'};
        static const uint32_t length = sizeof(input)/sizeof(input[0]);

        for (uint32_t i = 0; i < length; ++i) {
                uint32_t pos = (i + k) % length;
                cypher.insert(std::make_pair(input[i], input[pos]));
                uncypher.insert(std::make_pair(input[pos], input[i]));
        }
}

std::string cypher_text(dictionary_t dictionary, std::string line) {
        std::string output;

        for (std::string::iterator it = line.begin(), end = line.end(); it != end; ++it) {
                char c = *it;
                if (c != ' ' && c != '.') {
                        output.push_back(dictionary.at(c));
                } else {
                        output.push_back(c);
                }
        }

        return output;
}

int main(int argc, char** argv) {
        std::string line;

        // Read number of lines to process and cypher distance
        std::getline(std::cin, line);
        std::vector<uint32_t> data = split_input(line, ' ');
        uint32_t lines = data[0];
        uint32_t k     = data[1];

        // Generate the cypher/uncypher dictionary
        dictionary_t cypher, uncypher;
        create_dictionaries(k, cypher, uncypher);

        // Parse all lines in the message
        for (uint32_t l = 0; l < lines; ++l) {
                std::string message;
                std::getline(std::cin, line);

                message = cypher_text(uncypher, line);

                std::cout << message << std::endl;
        }

        return 0;
}
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  problem12.py
#
#  HP Inc. Copyright 2016
#  Author: Tanausu Ramirez
#

# Caesar Algorithm
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def generate_cypher_alphabet(k):
    """
    Generate a encripted alphabet using Caesar algorithm with shift value k.
    """
    def cypher(char,k):
        pos = (alphabet.find(char) + k) % len(alphabet)
        return alphabet[pos]
    cypher_alph = "".join([cypher(c,k) for c in alphabet])
    return cypher_alph


def encrypt(msg, k):
    """Encrypt a message using Caesar algorithm with shift value k."""
    m = ""
    for c in msg:
        size = len(alphabet)
        pos = alphabet.find(c.upper())
        if pos >= 0:
            m += alphabet[(pos + k) % size]
        else:
            m += c
    return m


def decrypt(msg, k):
    """Decrypt a message using Caesar algorithm with shift value k."""
    m = ""
    for c in msg:
        size = len(alphabet)
        pos = alphabet.find(c.upper())
        if pos >= 0:
            m += alphabet[(pos + (size - k)) % size]
        else:
            m += c
    return m

def read_input():
    """Read the input. The first line of the input data will contain two
    integers separated by a space; the number of lines of encrypted text to be
    processed and the value of K, that is, the positions that the alphabet has
    to be shifted to encrypt/decrypt the message. The following lines will
    contain encrypted text.
    """
    try:
        # Read the number of lines encrypted and the value of K
        line = raw_input()
        nlines, k = [int(n) for n in line.split()]
    except Exception as e:
        emsg = "ERROR Incorrect input conversion: %s" % e.message
        exit(emsg)

    # Read encrypted text
    messages = []
    for line in range(nlines):
        messages.append(raw_input())
    return messages, k


def testing():
    """Some testing..."""
    print "Alphabet size", len(alphabet)

    line ="VENI VIDI VICI."
    cypher_msg = encrypt(line, 3)
    print cypher_msg
    line2 ="YHQL YLGL YLFL."
    if cypher_msg == line2:
        print "Encryption OK!"

    msg = decrypt(cypher_msg, 3)
    print msg
    if msg == line:
        print "Decryption OK!"

def main():

    text, k = read_input()
    #testing()

    # Solution 1.
    for msg in text:
        print decrypt(msg, k)

    # Solution 2.
    cypher_alphabet = generate_cypher_alphabet(k)
    for msg in text:
        m = ""
        for c in msg:
            pos = cypher_alphabet.find(c.upper())
            if pos >= 0:
                m += alphabet[pos]
            else:
                m += c
        print m


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

13 Word wrap

4 points

Introduction

As you know, newspapers print their articles in several columns to help the reader. This way, they do not need to move their eyes from the left to the right side of the page all the time. These columns have a specific width. The process of transforming the text into as many lines as needed but making sure that a line of the text does not have more than the required columns is called wrapping. Wrapping will not break any word, and final blank spaces will not count. But, if original text has a carriage return to start a new line, the wrapped text will respect this.

Input

Input will be a number to specify the column width of resulting output lines, followed by the long text in several lines ending with the symbol '#'.

```
20
It takes a big man to cry, but it takes a bigger man to laugh at that man.
When you're riding in a time machine way far into the future, don't stick your elbow
out the window, or it'll turn into a fossil.
I wish I had a Kryptonite cross, because then you could keep both Dracula AND Superman
away.
I don't think I'm alone when I say I'd like to see more and more planets fall under
the ruthless domination of our solar system.
I hope if dogs ever take over the world, and they chose a king, they don't just go by
size, because I bet there are some Chihuahuas with some good ideas.
The face of a child can say it all, especially the mouth part of the face.#
```

Output

The result of the wrapped text.

```
It takes a big man
to cry, but it
takes a bigger man
to laugh at that
man.
When you're riding
in a time machine
way far into the
future, don't stick
your elbow out the
window, or it'll
turn into a fossil.
I wish I had a
Kryptonite cross,
because then you
could keep both
Dracula AND
Superman away.
I don't think I'm
alone when I say
I'd like to see
more and more
planets fall under
the ruthless
domination of our
solar system.
I hope if dogs ever
take over the
world, and they
chose a king, they
don't just go by
size, because I bet
there are some
Chihuahuas with
some good ideas.
```

Solution

```
import sys

def wordWrap(s,length):
    offset = 0
    while offset+length < len(s):
        if s[offset] in ' \n':
            offset += 1
            continue

        endOfLine = s[offset:offset+length].find('\n')
        if endOfLine < 0:
            endOfLine = s[offset:offset+length].rfind(' ')

        if endOfLine < 0:
            endOfLine = length
            newOffset = offset + endOfLine
        else:
            newOffset = offset + endOfLine + 1

        yield s[offset:offset+endOfLine].rstrip()

        offset = newOffset

    if offset < len(s):
        yield s[offset:].strip()

wrap_size = int(raw_input())
myfollow = True
while myfollow:
    textline = raw_input()
    mylen = len(textline)
    if textline[mylen-1] == '#' :
        myfollow = False
        textline = textline[:-1]

    for l in wordWrap(textline,wrap_size):
        print l
```

14 The assistant and the numbers

5 points

Introduction

Our cell phones are trying to make our lives easier and easier and include assistants that understand our questions and answer in a quite personal way. They get so personal that some guys have fallen in love with their digital assistants! It is very important that they speak like humans do. A lot of research has been done to make machines pronounce words the best way possible, a difficult task in languages such as English that are not read literally (a.k.a. WYSIWYG).

In daily life not everything is written in plain text, for example, numbers. Your task is to write a program that translates numbers into written English so that the reading module is able to interpret them.

We have simplified the rules and there is no need to include any comma or 'and'.

NOTE: You have to take into account that English wording for numbers is different to Spanish.
In English Billion is 1.000.000.000 while in Spanish it is 1.000.000.000.000

Input

The input of the program is a set of numbers up to twelve digits, ending with a zero.

```
8
14
3672
1234567
1001001001
0
```

Output

Their value in words.

```
8 in words is eight
14 in words is fourteen
3672 in words is three thousand six hundred seventy two
1234567 in words is one million two hundred thirty four thousand five hundred
sixty seven
1001001001 in words is one billion one million one thousand one
```

Solution

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

#define MAX_LEN 12

//
// This function prints the word of a number n bigger than 0 and lower than 1000
//
void convertToWord(int n)
```

```
{
   char *units[] = {"","one","two","three","four","five","six","seven","eight","nine"};
   char *teens[] =
{"ten","eleven","twelve","thirteen","fourteen","fifteen","sixteen","seventeen","eighteen","nineteen"};
   char *tys[] = {"","","twenty","thirty","forty","fifty","sixty","seventy","eighty","ninety"};
        int i = 100, value;

        // This loop analyze the number digit by digit, starting by hundreds
        while (n > 0) {
                value = n / i;
                if ( value > 0)   {
                        if (i == 10) {
                        // It means that tens are being analyzed
                                if (value == 1)  {
                                // It's a number between 10 and 19
                                // It's needed to analyze the next digit before writing
                                        n -= value * i;
                                        i /= 10;
                                        value = n / i;
                                        printf("%s ",teens[value]);
                                }
                                else {
                                        printf("%s ",tys[value]);
                                }
                        }
                        else {
                        // Handling of units and hundreds is the same,
                        // just add the suffix in case of hundreds
                                printf("%s ",units[value]);
                                if (i == 100) {
                                        printf("hundred ");
                                }
                        }
                }
                n -= value * i;
                i /= 10;
        }
}
```

15 My group has money troubles

5 points

Introduction

Your group of friends travels together a lot. During your trips, people pay for others very often. But when it comes to paying back, sometimes it is not easy to settle up. Since you want your group of friends to stick together, and everyone knows that owing money does not help, you have decided to make a program that calculates how much each one needs to pay.

Input

The input starts with the amount of lines that come afterwards. These lines include two names (each of them starting with upper case) that indicate who pays (the first one) and whom (the second one), the third value is the money lent.

```
4
Mary John 2000        ← Mary paid John 2000
John James 3000
Anna Mary 1000
James Anna 300
```

Output

The balance of each person, ordered from who receives the most to who pays the most. In case two people have the same final balance, use alphabetical order.

```
John receives 1000
Mary receives 1000
Anna receives 700
James pays 2700
```

Solution

```
import sys

x = 1

for i in range(1,int(x)+1):
  y = raw_input()
  mymap = dict()
  #print "new map for items #", y
  for j in range(1,int(y)+1):
    line = raw_input()
    args = line.split()
    #print args
    #print args[0], args[1], args[2]
    value = int(args[2])
    #print mymap.get(args[0],0)-value
    mymap.update({args[0]: mymap.get(args[0],0)-value, args[1]:
mymap.get(args[1],0)+value})
  #print mymap
  #for key in mymap:
  #    print key, " ", mymap[key]
  ordered = sorted(mymap.items(), key=lambda x: (x[1],x[0]), reverse=False)
```

```
    for ele in ordered:
        if ele[1] < 0 :
            print ele[0].strip(' \t') + ' receives ' + str(abs(ele[1]))
        else:
            print ele[0].strip(' \t') + ' pays ' + str(ele[1])
    #print ''
    #raw_input()
```

16 Carcassone

5 points

Introduction

Carcassone is a turn-style board game for several players (Game of the year in 2001). The game board is a medieval landscape built by the players as the game progresses. The game starts with a single terrain tile face up and all the others shuffled face down for the players to draw from. Those tiles can have one or more of these different features: city, cloister, fields and roads. On each turn a player draws a new terrain tile and places it adjacent to tiles that are already facing up. The new tile must be placed in a way that extends features on the tiles it touches: roads must be connected to roads, fields to fields, and cities to cities.

During the players' turns, cities, cloisters, and roads (but not fields) are scored. Cities and roads are scored when they are completed (i.e. contain no unfinished edges from which they may be expanded), and cloisters are scored when they are surrounded by eight tiles. At the end of the game, when there are no tiles remaining, all incomplete features are also scored but using different rules.

Some HP co-workers are such big fans that keep playing this lovely game despite they are over 30, but they became so lazy for the math that they need your help to count all the points. To ease this task, only the most basic scoring rules will be considered. The next table represents the points scored for each feature while there are still remaining tiles in game, as well at the end of the game when no tiles remains:

Feature	Scored when completed during players' turns (Main scoring rules)	Scored at the end of the game (Final rules)
City	2 points per tile	1 point per tile
Road	1 points per tile	1 point per tile
Cloister	9 points (it can only score when completely surrounded)	1 point for the cloister + 1 point for each of the surrounding tiles
Field	(Not scored)	3 points for each completed city bordering the field

Input

The input is divided into 3 parts as follows:

1 – The first line of the input data contains a number N with the amount of players.

2 – The following lines represent the main game phase where the features are being completed during the players' turns. These lines are composed by the player number that got the feature, the feature type, and only in case the feature is city or road, the number of tiles to calculate the proper points. In this phase we will be applied the main scoring rules.

At the end of players' turns, when there are no tiles remaining, all the incomplete features are scored. This final phase is indicated with a colon ":" symbol in a new line.

3 - The following lines represent the game's final phase. Now the non-completed features and also the fields gained by players can be scored. These lines contain, as those above, a similar structure, but in this case the cloisters have a third parameter that indicates the surrounding tiles and the third parameter of the fields indicates the amount of bordering cities. Now the final scoring rules are applied instead.

And finally the "." symbol indicates the end of input data lines.

Let's see an example with an explanation:

Input example	Explanation
`4`	4 players.
`1 city 4`	Player 1 completes a city with 4 tiles.
`2 cloister`	Player 2 completes a cloister. *No tiles number are needed in this case.*
`3 road 3`	Player 3 completes a road with 3 tiles.
`:`	*End of players' turns. From now on, final counting rules.*
`1 field 2`	Player 1 has a field with 2 completed cities bordering.
`3 road 6`	Player 3 has a non-completed road with 6.
`4 cloister 1`	Player 4 non-surrounded cloister with only 1 tile beside it.
`.`	*The dot indicates the end of the input data*

Output

The final score of each player, ordered by player number.

```
Player[1] -> 14
Player[2] -> 9
Player[3] -> 9
Player[4] -> 2
```

Solution

```
#include <fstream>
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <stdlib.h>

using namespace std; // We are using cin and cout for input and output

int main(int argc, char ** argv)
{
   //Variable for reading every input line
   string line;

   /** The first we need is to read the number of players,
    *  so we can allocate a vector with the scores
    */
   int numberOfPlayers = 0;
   getline(cin, line);
   /**
    * stringstream splits the input line into single strings
    * it also facilitates the casting for other data types and memory allocation
    */
   stringstream lineStream(line); //lineStream has now as much strings as words in line string.
   lineStream >> numberOfPlayers; //Get the first word and autmatically cast to integer
(numberOfPlayers)

   if (numberOfPlayers <= 0) return 1; //There should be almost 1 player

   /**
    * Initialize a vector where we can store the scores for each player.
    * We also initialize the vector with zeros as initial values.
    *   Player1   Player2   Player3
    * |----0----|----0----|----0----|----
    */
   std::vector<int> scores(numberOfPlayers, 0);

   /** PHASE 1
    * Now we loop over the regular phase of the carcassone's game
    * we reads every line to check the player number, the feature
    * and the possible number of tiles for the feature
    * These lines has this structure:
    * playerNumber feature [numberOfTiles]
    */

   int playerNumber;
   string feature;
   int numberOfTiles;

   getline(cin, line);

   while (line != ":")
   {

      stringstream regularStream(line);
      regularStream >> playerNumber;

      // Checks if the playerNumber is valid
      if ((playerNumber > numberOfPlayers) || (playerNumber < 1)) return EXIT_FAILURE;

      playerNumber = playerNumber - 1; //adapt the playerNumber to the vector range.

      regularStream >> feature;

      /**
       * Apply rules during the game
       */
      if (feature == "city")
      {
         regularStream >> numberOfTiles;
         scores.at(playerNumber) = scores.at(playerNumber) + (numberOfTiles * 2);

      } else if (feature == "cloister")
```

```
  {
    // In this case we don't need to read the next word because it's supposed to be completed
    scores.at(playerNumber) = scores.at(playerNumber) + 9;

  } else if (feature == "road")
  {
    regularStream >> numberOfTiles;
    scores.at(playerNumber) = scores.at(playerNumber) + (numberOfTiles);

  } else if (feature == "field")
  {
    cout << "Fields are not scored during play, only at the end." << endl;

  }else
  {
    cout << "ERROR, feature not permitted" << endl;
    return EXIT_FAILURE;
  }
  // We need to read the new line before entering in the loop.
  getline(cin, line);

}

// A ":" has been read, so we can count the final points.
getline(cin, line);

while (line != ".") // The "." indicates the end of the data
{

  stringstream regularStream(line);
  regularStream >> playerNumber;

  if ((playerNumber > numberOfPlayers) || (playerNumber < 1)) return EXIT_FAILURE;

  playerNumber = playerNumber - 1;

  regularStream >> feature;

  /**
   * Apply the end of the game rules
   */

  if (feature == "city")
  {
    regularStream >> numberOfTiles;
    scores.at(playerNumber) = scores.at(playerNumber) + (numberOfTiles);

  } else if (feature == "cloister")
  {
         regularStream >> numberOfTiles;
         scores.at(playerNumber) = scores.at(playerNumber) + (numberOfTiles) + 1;

      } else if (feature == "road")
      {
         regularStream >> numberOfTiles;
         scores.at(playerNumber) = scores.at(playerNumber) + (numberOfTiles);

      } else if (feature == "field")
      {
         regularStream >> numberOfTiles;
         scores.at(playerNumber) = scores.at(playerNumber) + (numberOfTiles * 3);

      }else
      {
         cout << "ERROR, feature not permitted" << endl;
         return EXIT_FAILURE;
      }
      // We need to read the new line before entering in the loop.
      getline(cin, line);
   }

   /**
    * Display the desired output
    * Player[n] -> score
    */
   for (int player = 0; player < numberOfPlayers; player++){
      cout << "Player[" << player + 1 << "] -> " << scores.at(player) << endl;
   }

   return EXIT_SUCCESS;
}
```

17 CodeWars diet

6 points

Introduction

The World Health Organization (WHO) considers that healthy grown-ups need over 2200 calories a day, while having under 2000 calories is a poor diet, and having more than 2500 calories is known as Hyper-caloric (and non-healthy) diet.
During the days spent preparing HP CodeWars, the organizers just had time to go for a lunch at Burry-King™ once a day and have a delicious four-course meal while thinking in our beloved junior geniuses. Now it is time to give some health advice to the organizers and tell them if their daily diet is poor, enough or hyper-caloric, their hearts will thank your efforts!

Input

Burry-King™ menu in the form of one line for each dish with the name of the dish and the calories contained. A line with a 0 will separate the menu from the selection of dishes for each organizer. Afterwards, the selection for each organizer, their name first and then the list of 4 dishes. A line with just a 1 will mark the end of the input.

```
CheeseBurger 650
BaconBurger 680
PlainBurger 410
ChickenWrap 340
PlainSalad 125
ChickenSalad 470
Nuggets 460
MediumFries 370
LargeFries 680
MediumCoke 350
LargeCoke 720
Water 0
0
Marcus CheeseBurger Nuggets LargeFries LargeCoke
James BaconBurger ChickenSalad MediumFries LargeCoke
Laura PlainSalad PlainBurger MediumFries Water
1
```

Output

A list with the name of each of the organizers and the type of diet they are following (Poor, Enough, Hyper-Caloric).

```
Marcus Hyper-caloric
James Enough
Laura Poor
```

```
import java.util.*;

public class Prob_diet {

  public static void main(String args[]) {
    // Create a hash map
    Hashtable menu = new Hashtable();




    String str;
    double bal;
    int i;

    String meal,name,dish;
    int calories;
    Scanner read=new Scanner(System.in);
    meal=read.next();



    while (!meal.equals("0"))
    {
      calories = read.nextInt();
      menu.put(meal,calories);
      meal = read.next();
    }
    // reading the menu selected

    name = read.next();
    while(!name.equals("1"))
    {
      System.out.print(name+" ");
      calories=0;
      for (i=0; i<4; i++)
      {
         dish = read.next();
         calories = calories + (int)menu.get(dish);
      }
      if (calories<2000)
         System.out.println("Poor");
      else if (calories>2500)
         System.out.println("Hyper-caloric");
      else
         System.out.println("Enough");
      name = read.next();
    }
  }
}
```

18 Intelligence Service

6 points

Introduction

An encrypted message has been received here by the HP Intelligence Service. We need to be able to understand the encrypted messages, and we need a fast way to do it. Our experts have found how sentences are encrypted.

1. Each letter in the sentence is replaced by the following letter in the alphabet. That is, an **A** would be replaced by a **B** (**Z**, then, will be replaced by **A**).
2. Each word in the sentence transformed in the following way:
   a. Divided in two equally long parts (**ABCD → AB CD**)
   b. Each side is reversed (**AB CD → BA DC**)
   c. The two parts are put together again (**BA DC → BADC**)
   d. If the word has an odd number of letters (e.g. 5) then you have to divide the word leaving the middle letter invariable in the process. That is (**ABCDE → AB C DE → BA C ED → BACED**)

Program a decoder that receives the encrypted sentence as an input and returns the original message.

Input

The input of the program will be only one sentence. The sentence will not contain neither punctuation marks, nor symbols different than letters. All letters will be uppercase.

```
MFXDFNP UP IQ
```

Output

The program must output the decrypted message.

```
WELCOME TO HP
```

Solution

```
#Problem HP CodeWars 2016

import sys
import string

sentence = str(sys.stdin.readline()) #reads the encrypted sentence
words = sentence.split() #splits the sentence in the different words

alphabet = list(string.ascii_uppercase) #list containing the alphabet (uppercase). It would obviously be
correct if done like: alphabet= ["A", "B", "C", ...]

decrypted_words =[] #list of reversed words
for word in words:
   w_length = len(word) #length of the word
   if w_length%2 == 0: #if the word has an even number of letters
      f = word[0:w_length/2] #first half of the word
      s = word[w_length/2:] #second half of the hord
```

```
        f_rev = f[::-1] #first half of the word reversed
        s_rev = s[::-1] #second half of the word reversed
        word_rev = f_rev + s_rev #modified word

    else: #if the word has an odd number of letters
        f = word[0:w_length/2] #first half of the word
        m = word[w_length/2] #middle letter
        s = word[w_length/2+1:] #second half of the word

        f_rev = f[::-1] #first half of the word reversed
        s_rev = s[::-1] #second half of the word reversed
        word_rev = f_rev + m + s_rev #modified word

    decr_word = []
    for letter in word_rev:
        alphabet_index = alphabet.index(letter) #position of the letter in the alphabet
        decr_word.append(alphabet[alphabet_index-1]) #we add the correct letter to the list
    decr_word = "".join(decr_word)

    decrypted_words.append(decr_word) #we add the decrypted word to the list

decrypted_sentence = " ".join(decrypted_words) #we merge the words in the list in one string

print decrypted_sentence #print the decrypted sentence
```

19 Bet-n-win

6 points

Introduction

With my friends, we like to watch soccer matches and bet for the winner. A bet consists on playing 3 coins per match and specifying the result of the match, for example 3-1 if locals wins scoring 3 goals and visitors score 1 goal. We only allow the same bet to happen twice. Each of us can only make a maximum of two bets. All coins collected by bets are for the winner of the bet, or split in two in case there are two winning bets. Your friends have requested you, as the programmer of the group, to write a program that helps compute the bet results.

Input

The first line of the input is the number of bets, followed by a sequence of bets in the form of the name of a friend and their bet in the format local-visitant. The last line is the actual result of the match.

```
6
John 3-1
Peter 1-2
Laura 3-0
John 3-0
Mark 3-1
Peter 0-3
3-1
```

Output

The output of the program is the total amount of coins collected, number of different players and the list of players ordered by the amount of money won from most to least. In case of a tie, order alphabetically the players.

```
Total 18
Players 4
John 9
Mark 9
Laura 0
Peter 0
```

Solution

```
import sys

bet=3
numbets = int(raw_input())
total_amount=numbets*bet

mymap = set()
mybets = dict()
for i in range(1,numbets+1):
  line = raw_input()
  #print "new map for items #", y
  args = line.split()
  #print args
  #print args[0], args[1]
  mymap.update({args[0]: args[1]})
  #print mymap
  mybets.update({args[1]: (mybets.get(args[1],'')+ ' ' + args[0]).strip()})
  #print mybets
result = raw_input()
winners = mybets.get(result,'')
print "Total "+ str(total_amount)
print "Players " + str(len(mymap))
if winners == '' :
  mymap = sorted(mymap)
  for player in mymap:
    print player + " 3"
  sys.exit()
winners=winners.split()
if len(winners) == 2 :
  winners = sorted(winners)
  print winners[0] + ' ' + str(total_amount/2)
  print winners[1] + ' ' + str(total_amount/2)
if len(winners) == 1 :
  print winners[0] + ' ' + str(total_amount)
for winner in winners :
  mymap.remove(winner)
mymap = sorted(mymap)
for player in mymap:
  print player + " 0"
```

20 Happy numbers

8 points

Introduction

We all want to be happy and numbers are no different. But, mathematically, a happy number complies with the following:

In order to know if a number is happy, you have to replace the number by the sum of the squares of its digits and repeat the process until the number equals 1 (where it will stay). In this case, it is a happy number. If it loops endlessly in a cycle, never reaching 1, then it is an unhappy number (or sad number).

For example, 19 is happy, as the associated sequence is:

$1^2 + 9^2 = 82$

$8^2 + 2^2 = 68$

$6^2 + 8^2 = 100$

$1^2 + 0^2 + 0^2 = 1$

Write a program that, given a number *n*, finds all happy numbers smaller than *n*.

Input

The input of the program is a positive integer.

```
20
```

Output

The program must find all happy numbers smaller than the provided one.

```
1
7
10
13
19
```

Solution

```
Import java.util.HashSet;
public class Happy{
  public static boolean happy(long number){
     long m = 0;
     int digit = 0;
     HashSet<Long> cycle = new HashSet<Long>();
     while(number != 1 && cycle.add(number)){
        m = 0;
        while(number > 0){
           digit = (int)(number % 10);
           m += digit*digit;
           number /= 10;
        }
        number = m;
     }
     return number == 1;
```

```
 }

 public static void main(String[] args){
        int total = 0;
        if (args.length > 0) {
            try {
                total = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                System.err.println("Argument" + args[0] + " must be an integer.");
                System.exit(1);
            }
        }

             for(int num = 1,count = 0;count<total;num++){
                 if(happy(num)){
                     System.out.println(num);
                     count++;
                 }
             }
        }
}
```

21 Earthcraft prototype

8 points

Introduction

You work in the upcoming amazing sandbox game called Earthcraft. The game will be played from an overhead perspective in a 2D map and will consist on the recollection of materials to build all kind of stuff while you avoid enemies that may attack you.

The head designer has requested to develop a small prototype to show the mechanics of the game to some investors.

In this prototype the objective will be to collect all materials, preventing the player's death.

Input

The input of your game is:

- The size of the map given as the amount of rows and columns.
- Several positions in the map, each of them with only one of the following elements:
  - Empty: represented as '_', by default all map is empty.
  - Player: represented as 'p'. Only one can exist.
  - Material: represented as 'm'
  - Enemy: represented as 'e'
- A dot that separates map setup from movements.
- A sequence of movements, given as a pair (±1, ±1), indicating the rows and columns to advance each round.
  - The player can only walk one position at a time in any direction.
  - If the player walks into a material position, it will be collected. When all materials are collected, the game ends.
  - If the player walks into an enemy position, they will die and the game ends.

```
3 3          ← this is a 3x3 map
0 1 p        ← row 0 and column 1 is the player position
1 1 e        ← at row 1 and column 1 there is an enemy
1 2 m        ← at row 1 and column 2 there is a material
2 0 e        ← at row 2 and column 0 there is an enemy
.            ← From now on all inputs are movements!
1 1          ← Go down one row and right one column
```

Output

The output will be a matrix, showing each element and the evolution of the movements.

```
Initial state
 _  p  _
 _  e  m
 e  _  _
Materials collected 0/1
 _  _  _
 _  e  p
 e  _  _
Materials collected 1/1
You have collected all the materials! Congratulations!
```

Solution

```
import java.util.*;

public class Main {

        static final char EMPTY = '_';
        static final char PLAYER = 'p';
        static final char MATERIAL = 'm';
        static final char ENEMY = 'e';
        static char[][] map;
        static int currentMaterials;
        static int totalMaterials;
        static boolean dead;
        static int playerRow= -1, playerCol= -1;

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        StringTokenizer st;
        String line="";
        totalMaterials = currentMaterials = 0;
        int maxRows,maxCols=0;

        //Create map
        st = new StringTokenizer(sc.nextLine());
        maxRows = Integer.parseInt(st.nextToken());
        maxCols = Integer.parseInt(st.nextToken());
        map = new char[maxRows][maxCols];
        //populate empty map
        for (int i=0; i < maxRows ; i++ ) {
            for (int j=0; j < maxCols ; j++ ) {
                map[i][j]=EMPTY;
            }
        }

        //Populate map with elements
        line = sc.nextLine();
        while(!line.equals("."))
        {
            st = new StringTokenizer(line);
            int row = Integer.parseInt(st.nextToken());
            int col = Integer.parseInt(st.nextToken());
            map[row][col]=st.nextToken().charAt(0);
            if(checkPosition(MATERIAL,row,col))
                totalMaterials++;
            else if(checkPosition(PLAYER,row,col)) {
                if(playerRow !=-1) {
                    System.err.println("Error: It can't be several players!");
                    System.exit(-1);
                } else {
                    playerRow=row;
                    playerCol=col;
                }
            }

            line = sc.nextLine();
        }

        if(playerRow==-1) {
            System.err.println("Error: It has to be a player!");
            System.exit(-1);
        }

        System.out.println("Initial state");
        //Read and execute movements
        while(true)
        {
            printMap();
            printStatus();
            if(isFinished())
                break;

            st = new StringTokenizer(sc.nextLine());
            int despRow = Integer.parseInt(st.nextToken());
            int despCol = Integer.parseInt(st.nextToken());
            execMove(despRow,despCol);
        }

    }
    static boolean isFinished() {
        if(currentMaterials==totalMaterials){
            System.out.println("You have collected all the materials! Congratulations!");
            return true;
        }else if(dead) {
            System.out.println("You died!");
            return true;
        }
        return false;
    }
    static void execMove(int despRow, int despCol)
    {
        if(despRow > 1 || despCol > 1 || despRow < -1 || despCol < -1) {
            System.out.println("Can't move that distance! ("+despRow+","+despCol+")");
        }
        else {
            int newRow = playerRow+despRow;
            int newCol = playerCol+despCol;
```

```
                if(checkPosition(MATERIAL,newRow,newCol)) {
                    currentMaterials++;
                }
                else if(checkPosition(ENEMY,newRow,newCol)) {
                    dead = true;
                }
                map[playerRow][playerCol] = EMPTY;
                map[newRow][newCol] = PLAYER;
                playerRow=newRow;
                playerCol=newCol;
                //System.out.println("Last movement ("+despRow+","+despCol+")");
            }
    }

    static boolean checkPosition(char value,int row, int col) {
        return map[row][col]==value? true:false;
    }

    static void printStatus() {
        System.out.println("Materials collected "+currentMaterials+"/"+totalMaterials);
    }

    static void printMap() {
        for (int i=0; i < map.length ; i++ ) {
            for (int j=0; j < map[0].length ; j++ ) {
                System.out.print(map[i][j]+" ");
            }
            System.out.println();
        }
    }

}
```

22 Simply perfect

8 points

Introduction

In number theory a positive integer is called “perfect” if it is equal to the sum of its proper positive divisors, excluding the number itself (also known as its *aliquot sum*), that is:

$$sum\big(div(n)\big) = n$$

If a number is not perfect, it can be deficient ($sum\big(div(n)\big) < n$) or abundant ($sum\big(div(n)\big) > n$).
You have to write a program that determines if a number is perfect, deficient of abundant.

Input

The input of the program is a list of positive integers, ending with a 0.

```
3
6
28
412
198
0
```

Output

The program must output whether each integer is perfect, deficient or abundant.

```
3 is deficient
6 is perfect
28 is perfect
412 is deficient
198 is abundant
```

Solution

```
import sys
import getopt

def getSumOfDivisors(number):
   sumOfDivisors = 0
   for i in range (1, number):
      if(number % i == 0):
         sumOfDivisors = sumOfDivisors + i

   return sumOfDivisors

def getNumberType(number):
   typeOfNumber = ""
   sumOfDivisors = getSumOfDivisors(number)

   if sumOfDivisors > number:
      typeOfNumber = "abundant"
```

```
    elif sumOfDivisors < number:
        typeOfNumber = "deficient"
    else:
        typeOfNumber = "perfect"

    return typeOfNumber

def main():
    finish = False
    numbers = []

    while finish == False:
        number = int(raw_input())
        if number != 0:
            numbers.append(number)
        else:
            finish = True

    for val in numbers:
        print (str(val) + ' is ' + str(getNumberType(val)))

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

23 Get revenge!

10 points

Introduction

The imperial Stormtrooper TR-8R wants to get revenge of his ex-friend and traitor FN-2187. Since the accuracy of the Stormtroopers is quite poor, he is going to the Imperial Stormtrooper Marksmanship Academy to improve it.

In the training, you have to shoot static targets that are in different spots but in each round the visible targets that can be hit change. For example, in round 1 you may see the targets at the position zero and three, but in the next round the visible targets are the ones at positions two and four.

Each target can withstand a certain amount of hits and afterwards the target is destroyed and never shown up again.

You are very bad at shooting so when you aim at a determined position, sometimes the bullet may be deviated to another position and the desired target will not be hit, but you may hit another one if it is also visible!!. For example you aim at the position one but the bullet deviates one position and goes to the position two and hits the target at that position (only hits because it is visible, the hidden targets can never be hit)

Input

- The first line has 3 numbers.
  - 1st number: number of targets in the round (bigger than 0)
  - 2nd number: number of hits that the target can withstand (bigger than 0)
  - 3rd number: number of rounds (positive number)

- Afterwards, the rounds information. Each of them is made up of two lines:
  - 1st Line: The first number determines the number of targets that will show up (bigger than 0). The succeeding quantity of numbers must match the value of the first number, and those numbers are the targets that will show up in the specific round (from 0 to n - 1).
  - 2nd Line: The first number is the position of the target that you are aiming to shoot (from 0 to n - 1)and the second one is the deviation of the shot (can be negative)

```
6 2 5
3 1 2 3
1 1
2 2 5
5 1
2 4 5
0 -1
2 0 3
2 1
1 3
3 0
```

Output

The program must output the total number of hits and destroyed targets

```
TR-8R hit 3 targets and destroyed 1
```

Solution

```
// Example program
#include <iostream>
#include <vector>

int main()
{
   int numTargets, life, rounds, hits, destroys;
   hits = 0;
   hits = destroys = 0;
   std::cin >> numTargets >> life >> rounds;
   std::vector<int> targetsLife =  std::vector<int> (numTargets, life);
   std::vector<bool> targetsShowed = std::vector<bool> (numTargets, false);
   for( int i = 0; i < rounds; ++ i )
   {
      int roundTargets, aim, deviation, target;

      // Reset shown targets for the round
      for ( int j = 0; j < numTargets; ++j)
         targetsShowed[j] = false;

      //Read number of shown targets
      std::cin >> roundTargets;
      //Set the corresponing targets as shown
      for ( int j = 0; j < roundTargets; ++j )
      {
         int t;
         std::cin >> t;
         targetsShowed[t] = true;
      }

      //Read aimed position and deviation
      std::cin >> aim >> deviation;

      target = aim + deviation;

      //Check that the target that will be hit is in range, is shown and is not destroyed.
      //  If so -> increase hits, decrease life and check if destroyed.
      if ( target >= 0 && target < numTargets && targetsShowed[target] && targetsLife[target] > 0)
      {
         ++hits;
         --targetsLife[target];
         if ( targetsLife[target] == 0 )
            ++destroys;
      }
   }

  std::cout <<  "TR-8R hit " << hits << " targets and destroyed " << destroys << std::endl;
}
```

24 Gamma Ray and his collection of art

10 points

Introduction

Gamma Ray is an art collector that is worried about the security at his home, where he keeps his priceless collection of pictures. That is why he has designed a mirror-based laser circuitry to detect if somebody makes it into his house.
Until now, his method was too hard, since he used to generate the circuits by hand and test them by hand too, moving the mirrors one by one.
In the last conference he attended, he met a programmer who let him know what a computer could do for him.
Gamma has hired you to make his work easier, because he is not really sure how to do it.
Note that Gamma's house is 100 tiles height per 200 tiles width and, therefore, that is the maximum area he can cover.

Input

The input of the program will be a sequence like this:

```
'>4/3\6/3\1' 2 4 8 5
```

Where in the string:
> represents the position where he puts the laser-emitter and the laser-detector of the circuit. The laser-light always goes to the right.
/ represents a right-sided mirror. The laser is always refracted 90 degrees at the point where it reaches the mirror.
\ represents a left-sided mirror.
`<number>` represents the number of straight segments the light has to travel

And the numbers mean:

1st	position X to begin
2nd	position Y to begin
3rd	maximum X capacity of the circuit
4th	maximum Y capacity of the circuit

Output

The output of the program must represent in two dimensions the tour of the light between all mirrors system. And the only way laser can reach the detector is in the same direction it has begun. For the input above the output has to be:

```
/------\
|      |
|      |
|      |
\->----/
```

For the output represent the straight segments with symbols:
'-' if horizontal and
'|' if vertical

Solution

```
import sys, string, os
import shlex

def getList(string):
    lexer = shlex.shlex(string)
    stringList = []
    for token in lexer:
    stringList.append(str(token))

    return stringList

def calculateDirection(piece, direction):
    if piece.isdigit():
        n = int(piece)
    else:
        n = -1
        if piece == '\\':
            direction = 3 - direction
        elif piece == '/':
            direction = ((direction + 1) if direction % 2 == 0 else direction - 1)

    return n, direction

# Get the circuit parameter and save it like an array to access easily
circuit = str(sys.argv[1])

circuitList = getList(circuit)
circuitLength = len(circuitList)

# Get the first position and the initial direction
position = circuitList.index('>')
lastDirection = 0
lastPosition = position

posY = 0
posX = 1
n = 0
offsetX = 1
offsetY = 1
maxX = 0
maxY = 0
position += 1

while position != lastPosition:
    piece = circuitList[position]
    n, direction = calculateDirection(piece, lastDirection)
    if direction == 0:
        posX += (n + 1)
    elif direction == 1:
        if (posY - (n + 1)) < 0 :
            offsetY += (n + 1 - posY)
            posY = 0
        else:
            posY -= (n + 1)
    elif direction == 2:
        if (posX - (n + 1)) < 0 :
            offsetX += (n + 1 -posX)
            posX = 0
        else:
            posX -= (n + 1)
    elif direction == 3:
        posY += (n + 1)

    if maxX < posX: maxX = posX
    if maxY < posY: maxY = posY

    position = (position + 1) % circuitLength;
    lastDirection = direction

print posX, posY, offsetX+maxX, offsetY+maxY
```

25 Where is Luke?

10 points

Introduction

Luke Skywalker has vanished. In these difficult times of galactic wars it is critical to find him and count on him to help fight the sinister First Order. Three known droids, R2-D2, C-3PO and BB-8, which are at the base of the brave Resistance in the planet D'Qar, are responsible for locating Luke.
The astromech droids R2-D2 and BB-8 stored in their memory two sets of data with the galactic Cartesian coordinates of different planets of the galaxy. An old ally of the Resistance discovered a clue to Luke's whereabouts: Luke is at the midpoint of the pair of the closest planets stored in the droids memories.
C-3PO, as a good protocol droid, must write a program that quickly finds the closest pair of Cartesian coordinates and calculate the intermediate coordinates where Luis is supposed to be hiding.
May the force (or the 4th) be with you!

The distance between two points is the length of the path connecting them. In the plane, the distance between points (x_1,y_1) and (x_2,y_2) is given by the Pythagorean theorem,

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Input

The input of the program includes the content of the memory of the droids following this structure twice (one per each droid):
Droid name; Number of pair of galactic Cartesian coordinates
First pair of galactic Cartesian coordinates separated by space
Second pair of galactic Cartesian coordinates separated by space
...
Nth pair of galactic Cartesian coordinates separated by space

```
R2-D2; 4
5.0 9.0
9.0 3.0
2.0 0.0
8.0 4.0
BB-8; 6
7.0 4.0
9.0 10.0
1.0 9.0
8.0 2.0
10.0 10.0
9.0 6.0
```

Output

Just the pair of galactic Cartesian coordinates where Luke can be found, accurate up to one decimal.

```
7.5 4.0
```

Solution

```
# 25 Where is Luke?

# Input example:
# The following lines without the first character '#'
#R2-D2; 4
#5.0 9.0
#9.0 3.0
#2.0 0.0
#8.0 4.0
#BB-8; 6
#7.0 4.0
#9.0 10.0
#1.0 9.0
#8.0 2.0
#10.0 10.0
#9.0 6.0

# Output:
# A single line with coordinates with one decimal accuracy 7.5 4.0

import sys
import math

# Variables

lineCounter = 0
numPoints = 0
numDroids = 0
listPoints = []
infinity = float('inf')

# Auxiliar functions

def bruteForceFindClosestPair(point):
 # Set minimum distance to infinity
 minDist = infinity
 numPoints = len(point)
 # Compare all points with all points to get the pair with minimum distance
 i = 0
 while (i < (numPoints-1)):
  j = i + 1
  while (j < numPoints):
   xi = point[i][0]
   yi = point[i][1]
   xj = point[j][0]
   yj = point[j][1]
   distance = math.sqrt( math.pow(xi - xj,2) + math.pow(yi - yj,2))
   #print "(", xi, ",", yi, ")", ", (", xj, ",", yj, ") -> distance: ",  distance
   if distance < minDist:
    closestPair = [point[i], point[j]]
    minDist = distance
   j = j + 1
  i = i + 1
 return closestPair

# Main program

# 1. Parse data from input file and collect all the coordinates in list
for line in sys.stdin:
 # 1.1. Read header robot name and total number of coordinates.
 if lineCounter == 0 and numDroids < 2:
  robotName, numPoints = line.split(";")
  numPoints = int(numPoints)
  #print robotName, numPoints
 else:
  # 1.2. Store coordinates in the list
  if numDroids < 2:
   x, y = line.split()
   listPoints.append([float(x),float(y)])
   #print x, y
 # 1.3. Reset the line counter to process another set of coordinates.
 if lineCounter == numPoints:
  lineCounter = 0
  numDroids += 1
 else:
  lineCounter +=1

# 2. Find the closest pair

coordinates = bruteForceFindClosestPair(listPoints)

# 3. Find middle point coordinates where Luke is hidding

x = (coordinates[0][0] + coordinates[1][0]) / 2
y = (coordinates[0][1] + coordinates[1][1]) / 2

# 4. Print results

print("%.1f %.1f" % (x, y))
```

26 The smart winemaker

12 points

Introduction

An Engineer from HP in Sant Cugat is starting up a small winemaking enterprise in his spare time.
One of the interesting issues he has discovered while researching the winemaking industry is that the price of the wine is not constant; some years customers are ready to buy more wine barrels and pay higher, while some other years they are reluctant and prices need to be lowered to get rid of wine in stores.
He has found that the reason for such behavior is simple; after rainy years grapes yield wine of better quality and people are more eager to purchase it.
This gives the engineer an idea; he needs to find the formula for calculating expected wine price is, depending on weather records from the preceding year (the wine sold in an specific year is prepared with the grapes picked the year before), so that price in the current year is set to the optimal and sales run more smoothly.
Your task is to help the Engineer create a program that calculates wine price for the current year. To keep things simple we will try approximating the dependency between rainfall and wine price with a linear function in a linear form:

$Y = K + X * B$, where X is the amount of rainy days and Y is the price.
For this task you will be given a list of records, each containing the number of rainy days during previous year along with the average price for which the wine was sold during that year.
We will use the simple, linear regression and the ordinary least squares criteria to find the parameters of the linear function which can approximate the dependence between price and amount of rainy days as follows:

$$B = \frac{Cov[x,y]}{Var[x]} = \frac{\sum x_i y_i - \frac{1}{n}\sum x \sum y}{\sum x_i^2 - \frac{1}{n}(\sum x_i)^2} \text{ and } K = \overline{y} - B\overline{x},$$

where $\overline{x}, \overline{y}$ is the mean of vector x and y respectively.

Remember that the mean $\overline{x}$ of a vector x can be calculated as $\overline{x} = \frac{\sum x_i}{n}$, where n is the number of elements in the vector.

Input

The input data will contain starting A and ending B year in the first line.
Then lines follow for each year in form YYYY: D P where YYYY is the mark of year, D is the number of rainy days (in previous season) and P is the wine price in euros per barrel.

```
1925 1947
1925: 89 257
1926: 75 226
1927: 83 235
1928: 52 173
1929: 148 332
1930: 109 268
1931: 129 306
1932: 115 289
1933: 102 265
1934: 99 269
```

```
1935: 50 228
1936: 102 265
1937: 91 256
1938: 79 238
1939: 118 298
1940: 134 311
1941: 61 155
1942: 146 340
1943: 108 274
1944: 96 242
1945: 89 232
1946: 143 328
1947: 133 303
```

Output

Output should contain values for K and B with an accuracy of 0.001 or better.

```
1.541 107.313
```

Solution

```
# 26 The smart winemaker

import sys

# Auxiliar functions

def basic_linear_regression(x, y):
    # Basic computations to save a little time.
    length = len(x)
    sum_x = sum(x)
    sum_y = sum(y)

    sum_x_squared = sum(map(lambda a: a * a, x))
    sum_of_products = sum([x[i] * y[i] for i in range(length)])

    # Magic formulae!
    a = float(sum_of_products - float((sum_x * sum_y) / float(length))) /
(sum_x_squared - float(((sum_x ** 2) / float(length))))
    b = (sum_y/float(length)) - a * (sum_x /float(length))

    return a, b


# Main program

lineCounter = 0
x=[]
y=[]
```

```
# 1. Parse data from input file
for line in sys.stdin:
    #print line
    if lineCounter == 0:
        firstYear, lastYear = line.split()
        numLines = int(lastYear) - int(firstYear) + 1
    else:
        if numLines > 0:
            data = line.split()
            #print data
            x.append(int(data[1]))
            y.append(int(data[2]))
            numLines -= 1
    lineCounter += 1
#print x
#print y
# 2. Perform the linear regression
a, b = basic_linear_regression(x, y)

print("%.3f %.3f" % (a, b))
```

27 Playing Tetris^{TM} with the printer

12 points

Introduction

You are working in a photo edition app and one of the main features you plan to offer is tilling. Tilling consists in placing photos in the rectangular sheet of paper you want to print. These photos are square-shaped and all have the same size with a maximum length of each side. All together have to cover the whole surface.

For example:

- If your algorithm receives as input that the sheet is 12 x 12, the algorithm will return that it can print 1 photo of 12 x 12.
- Now you get a 5 x 10 paper sheet. In this case the maximum square is 5 x 5 and you can print 2 photos in it
- If you get a 3 x 5 paper, in this case the only option to avoid wasting any paper is to print 15 pictures of length 1.

Input

The first line will contain the number of test cases. Following lines will be the width and height of the paper sheets.

```
3
12 12
5 10
3 5
```

Output

For each paper sheet print the following sentence:

```
I will print 1 picture(s) of length 12
I will print 2 picture(s) of length 5
I will print 15 picture(s) of length 1
```

Solution

```
#include <iostream>

unsigned GCD(unsigned u, unsigned v)
{
   while (v != 0)
   {
      unsigned r = u % v;
      u = v;
      v = r;
   }
   return u;
}

int main()
{
   unsigned tc;
```

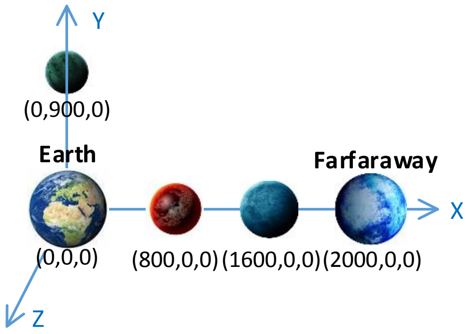
```
    unsigned w, h;

    std::cin >> tc;

    for (; tc; tc--)
    {
        std::cin >> w >> h;
        unsigned gcd = GCD(w, h);
        std::cout << "I will print " << w / gcd * h / gcd << " picture(s) of length " << gcd << std::endl;
    }

    return 0;
}
```

28 Family dinner in Farfaraway

14 points

Introduction

We are planning a trip with our brand new spaceship to visit our cousins in a planet named Farfaraway. The engine is fueled with high pressure radioactive gas and it allows us to go as far as 1000 light-years before refueling again.



While 1000 light-years is a significant distance, we can further extend this range by making stops in safe planets with gas stations. Our Micheleen guide contains a list with all such planets where we may stop for refueling.

Our guide indicates the 3D orthogonal coordinates (x,y,z) for each planet expressed in light-years. The Earth is always located at coordinates (0,0,0).

The distance between 2 points *p* and *q* in the 3D space is computed with the following formula:

$$distance = \sqrt{(q_x - p_x)^2 + \left(q_y - p_y\right)^2 + (q_z - p_z)^2}$$

Input

The program will receive the following information:

- The first line indicates the 3D coordinates of the Farfaraway planet separated by spaces.
- The second line indicates the number of additional planets (up to 50) where we may stop for refueling. For each additional planet, we will have a line in the input with their 3D coordinates separated by spaces.

```
2000 0 0
3
0 900 0
1600 0 0
800 0 0
```

Output

We need you to code a program that indicates whether it is possible to travel from the Earth to Farfaraway with the spaceship. The output will consist of one line indicating "`yes`" if it is possible to reach Farfaraway with any required refueling stops, or "`no`" otherwise.

```
Yes
```

Solution

```
import java.util.ArrayList;
import java.util.Scanner;

// java Main < prob10.txt
```

```
public class Main {


    public static void main(String[] args) {
        ArrayList<Planet> unvisited = new ArrayList<Planet>();
        ArrayList<Planet> reached = new ArrayList<Planet>();

        Scanner sc = new Scanner(System.in);
        // The Earth is the starting point:
        Planet e = new Planet();
        e.x = 0;
        e.y = 0;
        e.z = 0;
        reached.add(e);

        // The Farfaraway planet is our final destination:
        Planet f = new Planet();
        f.x = sc.nextFloat();
        f.y = sc.nextFloat();
        f.z = sc.nextFloat();
        unvisited.add(f);

        // Read all the N other planets that we may visit in our journey.
        int N = sc.nextInt();
        for (int i=0;i<N;i++)
        {
            Planet p = new Planet();
            p.x = sc.nextFloat();
            p.y = sc.nextFloat();
            p.z = sc.nextFloat();
            unvisited.add(p);
        }
        sc.close();

        // Breadth First Search traversal algorithm to see if we can reach the Farfaraway planet.
        for(int i=0; i<reached.size(); i++)  // The 'reached' array can grow in size, all elements in a
position lower than 'i' have been fully processed.
        {
            for (int j=unvisited.size()-1;j>=0;j--)  // The 'reached' array can shrink in size by
removing elements after 'j' if reached.
            {
                if (reached.get(i).distance(unvisited.get(j))<1000d)
                {
                    if (unvisited.get(j)==f)
                    {
                        System.out.println("yes");  // We just reached the Farfaraway
planet :)
                        return;
                    }
                    reached.add(unvisited.get(j));  // Move this planet to the reached list
and remove it from the unvisited list.
                    unvisited.remove(j);
                }
            }
        }
        System.out.println("no");
    }

    static class Planet
    {
        public double x;
        public double y;
        public double z;
        public double distance(Planet other)  // Returns the distance from 'this' planet to the 'other'
planet.
        {
            return Math.sqrt(
                    (other.x-x)*(other.x-x)+
                    (other.y-y)*(other.y-y)+
                    (other.z-z)*(other.z-z)
                    );
        }
    }
}
```

29 Quipus (the birth of writing numbers)

15 points

Introduction

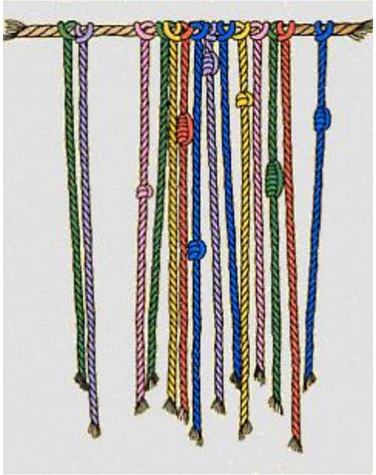
Over the course of evolution, Homo sapiens has moved from living in small groups of hunter-gatherer, to develop large social structures, Kingdoms and Empires. But our brain has not evolved to be able to store huge quantity of numerical information, like how many chickens are needed to feed my Kingdom, or the list of who has paid their taxes, or how much I need to maintain the army that prevents the neighbor King from becoming the King of my Empire. That is why, to overcome our brain limitation, a new need appeared: the need of storing Empire-sized mathematical data.

So, more or less 5,500-5,000 years ago, in Mesopotamia, the Sumerians were the first to solve the problem: they invented a system to store information through signs pressed in clay tablets. This newfangled system was called “writing”.

Another peculiar script that survived until the Spanish conquest of South America was the system used by the Inca Empire (that ruled up to 12 million people) to record large amounts of mathematical data: the Quipus.

Each quipu consists of many cords of different colors, with several knots tied in different places.

A single quipu can contain hundreds of cords and thousands of knots. By combining different knots in different cords with different colors, it is possible to record large amounts of mathematical data.



Quipu coding

Although the investigations have not clarified completely its interpretation, a simplification of a quipu consists in:

Each knot is a digit. A group of knots form a number.

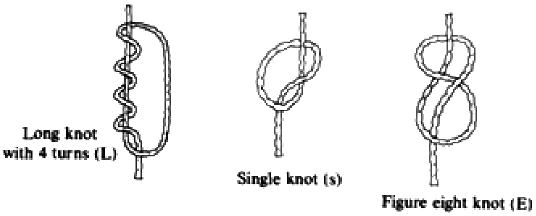
There are three different types of knots:

Single: it is a single knot, represented with an s

Long: it is a knot with one or more additional turns, represented with a L

Eight shaped: represented with an E

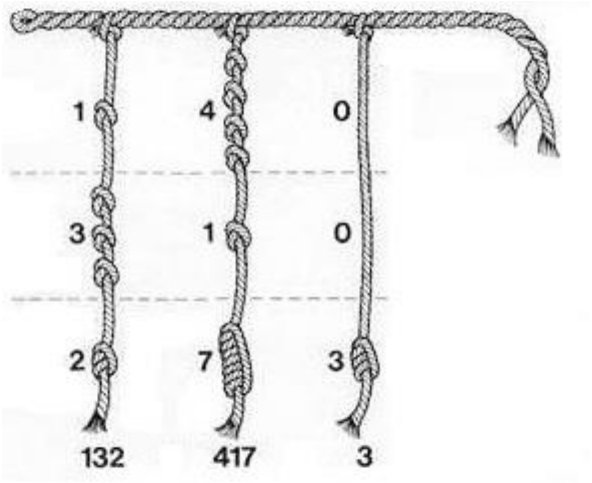
The absence of knot is represented with a X



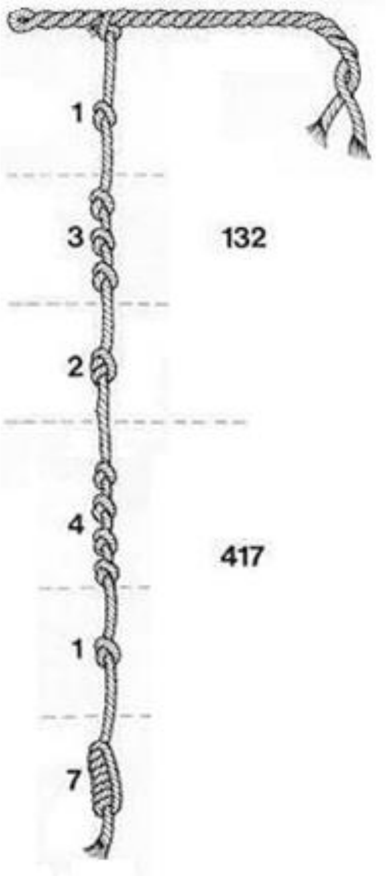
A number is represented by a sequence of knots in decimal base:

- Powers of ten are shown by position along the cord, and this position is aligned between successive cords.
- Digits in positions for 10^1 and higher powers are represented by clusters of simple knots (e.g., 40 is four simple knots in a row in the "tens" position).
- Digits in the 10^0 position are represented by long knots (e.g., 4 is a knot with four turns). Because of the way knots are tied, the digit 1 cannot be shown this way and is represented in this position by a figure-of-eight knot.
- Zero is represented by the absence of a knot in the appropriate position.
- Because the 10^0 digit is shown in a distinctive way, it is clear where a number ends. One strand on a quipu can therefore contain several numbers.

In example:
132 == 1s 3s 2L
417 == 4s 1s 7L
3 == X X 3L



In this case, there are two numbers in a single string:
132, 417 == 1s 3s 2L 4s 1s 7L



Input

Your program should read an input Quipu-matrix and help the quipucamayocs ("quipu specialists") to do some arithmetic.

It should calculate the sum of the numbers in the rows and columns, and return the Quipu-matrix with the original numbers, adding a final column with the rows' sums, and a final row with the columns' sums.

In example, for an input matrix (converted to decimal) as:

132	417	3
43	265	732

the program should calculate

```
 132  +  417  +    3    =   552
   +        +        +
  43  +  265  +  732  =  1040
   ll        ll       ll
 175      682      735
```

and return the matrix

132	417	3	552
43	265	732	1040
175	682	735	0

In this sample the numbers are in decimal just for clarification process, but the program must read and write the matrix in Quipu, of course, otherwise Incas will not understand it.
As the input Quipu-matrix size is unknown, it will end with the # character.

```
1s  X  X
 X  X 3s
 X 4s 2s
 E 3L  E
 #
```

Output

Corresponding to the numbers 1001, 43 , 321
...so **1001+43+321=1365** ...

```
1s  X  X  1s
 X  X 3s  3s
 X 4s 2s  6s
 E 3L 1L  5L
```

Solution

```
import copy
def quipu_to_int (str_quipu):
    # Coding:
    # X == 0
    # ns == n (n is single digit)
    # nL == n
    # E == 1
    if str_quipu == 'X' :
        return 0
    elif str_quipu == 'E' :
        return 1
    else:
        return int(str_quipu[0])

def int_to_quipu (i_quipu, isUnit):
    # Coding:
    # X == 0
    # ns == n (n is single digit)
    # nL == n
    # E == 1
    str_quipu=""
    if i_quipu == 0:
        return 'X'
    if isUnit==1:
        if i_quipu == 1:
            return 'E'
        else:
            return str(i_quipu)+'L'
    else:
        return str(i_quipu)+'s'

##############################################################################
# Read the input: a Quipu matrix with and unknown number or rows/colums
# It will read each line, until get EOL (if from a file, or an empty line) or # character
bEOF = 0
input_quipu_matrix=[]
while( bEOF == 0):
    line=[]
    # The try/except will allow us reading a input file without # (cleaner ;-)
    try:
        line=input().split()
    except:
        bEOF=1
    if line == [] or line == ["#"]:
        bEOF=1
    else:
        input_quipu_matrix.append(line)

# Converts the Quipu matrix to integers. In order to do this:
# 1-It already knows the number of colums, as its the same as the input one
# 2-Look for the units knots: from here it can count how many number (so rows) are in each string
inumbers_per_string=0
iclusters_weight=[]
# loops trough the knots clustes (so, each group of knots in the same horizontal, for each string)
for r in range(len(input_quipu_matrix)):
    bIsUnit=0
    # loops though all the strings
    for string in input_quipu_matrix[r]:
        # Check if any of the  clusters is unitary (as there could be 'X')
        if(string.find('E') != -1 or string.find('L') != -1 ):
            bIsUnit=1
    if(bIsUnit):
        inumbers_per_string+=1
        iclusters_weight.append(1)
    else:
        iclusters_weight.append(0)
# Here we know: how many numbers are in each string (inumbers_per_string )
#            have a vector that point where the units are (iclusters_weight)
# Now, we'll go from bottom to top in the weight vector, putting there the correct power of ten
ipower=1
for index in range(len(iclusters_weight)-1,0-1,-1):
    if iclusters_weight[index]==1:
        ipower=1
    else:
        ipower=10*ipower
        iclusters_weight[index]=ipower
# And we are ready to construct the input_int_matrix :-)
#  rows: inumbers_per_string
#  cols: same as any component of input_quipu_matrix]
input_int_matrix=[]
input_int_matrix.append([])
for col in range(len(input_quipu_matrix[0]) ):
    #for rows in range(inumbers_per_string):
    row=0
    iaux=0
    for cluster in range(len(input_quipu_matrix)):
        iaux = iaux + iclusters_weight[cluster] * quipu_to_int( input_quipu_matrix[cluster][col] )
        if iclusters_weight[cluster]==1:
            input_int_matrix[row].append(iaux)
            iaux=0
            row+=1
            if col==0 and cluster!=len(input_quipu_matrix)-1 :
                input_int_matrix.append([])

# Now, it's simple: we just should do the math...
# Output matrix will be
#  rows: inumbers_per_string+1 (if inumbers_per_string>1 )
#  cols: same as any component of input_quipu_matrix+1  (if input_quipu_matrix_cols>1 )
output_int_matrix = copy.deepcopy(input_int_matrix)

if len(input_int_matrix[0]) >1:
    for row in range(len(input_int_matrix)):
        # Add the final column sums
        iaux=0
        for col in range( len(input_int_matrix[row]) ):
            iaux = iaux + input_int_matrix[row][col]
        output_int_matrix[row].append(iaux)

if len(input_int_matrix)>1:
    output_int_matrix.append([]) # We should add a final row
    for col in range(len(input_int_matrix[0])):
        # Add the final row sums
        iaux=0
        for row in range( len(input_int_matrix) ):
            iaux = iaux + input_int_matrix[row][col]
        output_int_matrix[len(input_int_matrix)].append(iaux)
    if len(input_int_matrix[0]) >1:
        output_int_matrix[len(input_int_matrix)].append(0)

# And we just should help our Inca friends to convert from int to quipu
# ...as the tag quipu() doesn't seem to work yet in Python 3.x, we should do some work...

# This time we'll do slightlty different:
# - Check the max number of digits in the numbers, horizontally: this will determine the number of knot clusters
# - Creacte the output_int2str_matix: the int matrix where we'll add 0 to the left up to the max num of digits
# - Convert this matrix to quipu, digit by digit -> we need to know where the units are
# - Print it (we could do this in the previous steep...by the more programing, the more fun)
inum_output_digits = [0]
for row in range(len(output_int_matrix)):
    for col in range(len(output_int_matrix[row])):
        if len(str(output_int_matrix[row][col])) > inum_output_digits[row]:
            inum_output_digits[row] = len(str(output_int_matrix[row][col]))
    if row < (len(output_int_matrix)-1):
        inum_output_digits.append(0)

output_int2str_matix=copy.deepcopy(output_int_matrix)
for row in range( len(output_int2str_matix)  ):
    for col in range( len(output_int2str_matix[row]) ):
        format = "%%0%dd" % inum_output_digits[row]
        output_int2str_matix[row][col]= format % output_int2str_matix[row][col]

output_quipu_matrix=[]
output_index=-1
#output_quipu_matrix.append([])
for num in range( len(output_int2str_matix) ):
    for digit in range( len(output_int2str_matix[num][0]) ):
        output_index+=1
        output_quipu_matrix.append([])
        for col in range( len(output_int2str_matix[num]) ):
            if digit == (len(output_int2str_matix[num][0])-1) :
                isunit=1
            else:
                isunit=0
            output_quipu_matrix[output_index].append( int_to_quipu(
int(output_int2str_matix[num][col][digit]) ,isunit))

for cluster in range(len(output_quipu_matrix)):
    line=""
    for string in range( len(output_quipu_matrix[cluster]) ):
            line = line + output_quipu_matrix[cluster][string]
            if string != len(output_quipu_matrix[cluster]):
                line = line + " "
    print(line)
```

30 The hardest to understand in college

17 points

Introduction

Your elder sister has just started her first year of university, and during the beginning of the course welcome talks, she has been overwhelmed with the so called *compensable* subjects. She has always been a good student and has confidence in exceeding the subjects, but, as a good Engineer, just in case, she wants to have a backup plan. So, in order to understand this option she decided to ask your father for advice. He excused saying "when I was young, we just passed or failed a subject; what the #$*# does it mean to compensate!?!"
As she knows about your advanced skills in programing (or so she thinks), she asked you to write a program that based in her subjects grades, helps her to know if she has passed the course.
The program will read as input the list of ten subject grades (as in the list below showing the credits for each subject)
After computing the possible compensations (if any needed), the program must output the list of final scores after the compensation, and the Pass/Fail for the course. In case of Fail, no grades should be modified: the output final scores will be the same as input.
This is how compensation works:

5. Qualifications of the subjects.

Will be specified in regulations issued by the Governing Council of the University, that is, with a resolution of 0.1 points and the descriptive grade in accordance to the following table:
0.0 to 4.9: Fail
5.0 to 6.9: Pass
7.0 to 8.9: Remarkable
9.0 to 10: Outstanding

7. Criteria for automatic compensation.

The University can establish the criteria to overcome a curricular block, with a mechanism that may include passing failed subjects with numerical rating not smaller than 4.0. According to this regulation, it states:

7.1 Only subjects with grades between 4.0 and 4.9 can be changed from fail to pass 5.0 by the procedure of compensation.

7.2 The maximum total number of credits for the subjects that can change its qualification as compensation procedure are 12. To compensate these credits automatically, it is necessary that the average of the grades, weighted by the credits of curricular Initial Phase, is equal to or greater than 5.0.

8. Procedure to determinate the final grades of the subjects after the initial phase compensation procedure.

If, as a result of the curricular evaluation, some suspense subjects become pass, the procedure will keep constant the value for the average of the grades, weighted by the credits, whenever possible, and in no case will change the descriptive rating of a subject. For this reason, the procedure consists in:

8.1 The increase in the grade of the subjects failed, that according to the compensation procedure change their rating to 5.0 Approved, will be done increasing the weighted by the credits grade in these subjects until the numerical mark becomes 5.0, and decreasing properly the weighted grade of the qualifications of the subjects already passed, giving priority to those in which the student has obtained the lowest grades.

8.2 In the event that all numerical grades of the subjects that should decrease are at the limits of descriptive rating (ie. pass 5.0, remarkable 7.0, outstanding 9.0) subjects can be overcome, but in no case there will be a decrease in the numerical grades of the subjects at the limits of the descriptive rating. (The average of the notes will change in these cases)

She is studying Aeronautics, and her first course credits are:

1A

Code	**Subject**	**Credits**
220001	Linear Algebra	6
220002	Calculus I	6
220003	Economics	6
220004	Physics I	6
220005	Computers	6
	Sum of credits	**30**

1B

Code	Subject	Credits
220006	Chemistry	6
220007	Calculus II	6
220008	Aerial space, navigation and infrastructure	4.5
220009	Physics II	6
220010	Drawing	7.5
	Sum of credits	**30**

Input

The input will be the list of the 10 grades corresponding to the subjects.

```
4.5
5
5
5
5
5
5
5
5
5.4
```

Output

In this case, the first subject has been *compensated* with +0.5, and the last one reduced -0.4 (do not forget to consider the different number of credits weight)

The weighted average has not changed in this case.

```
5
5
5
5
5
5
5
5
5
5
PASS
```

Solution

```
##################################
##################################
##################################
# Read the ten subjects grades
# Calculate the weighted grades# Checks if:
    # Is compensation needed?
    # Is compensation possible?
        # Num of credits to compensate should be < 12
        # Average weighted grades >= 5.0
        # Grade of the suspended should be >= 4.0
# If everything is correct...
# Orders the list from smaller to bigger. It should keep track of the reordening
 #, to later on, reverse (or not?)
# Increment the weighted grade of the failed assignatures, so they become approved.
# by doing so, calculate the needed weighted increments
# Now, it tryes to decrease the approved grades, until has compensated the needed, or
can't
# modify any other grade. The algotihm to do so is:
 # For all the subjects
  # Calculate the maximum decrease that does'n modify the descriptive qualification
  # Reduce the weighted grade by the minimum of 'the weighted maximum decrease'
and 'the remaining to decrease'
  # If the remaining is zero, stops
# Reorder and print, of just print acordind a reference order
##################################

subject_names=["Linear Algebra", "Calculus I", "Economics", "Physics I", "Computers",
"Chemistry", "Calculus II", "Aerial space, navigation and infrastructure", "Physics II",
"Drawing"]
subject_credits=[6,6,6,6,6,6,6,4.5,6,7.5]
# Read the ten subjects grades
# Reads a hash table, indexed by the subject name
subject_grades={};
index=0
for x in subject_names:
    # So, here we:
        # Read the input grade
        # Add to hash table, indexed by the subject
        # At the same time, add the the credits, and calculate the weighted grade
    grade=float(input())
    subject_grades[x]= [grade,subject_credits[index],grade*subject_credits[index]/6]
    index+=1

# Checks if:
    # Is compensation needed?
    # Is compensation possible?
        # Num of credits to compensate should be < 12
        # Average weighted grades >= 5.0
        # Grade of the suspended should be >= 4.0
bcancompensate=1
f_weighted_credits_to_compensate=0
f_weighted_grade_to_compensate=0
for k in subject_grades.keys():
    if subject_grades[k][0] < 4.0:
      bcancompensate=0;   # There is one subject with grade <4.0. This is a direct FAIL
   if subject_grades[k][0] < 5.0 and subject_grades[k][0] >= 4.0:
      f_needed_compensation = (5.0 - subject_grades[k][0])*subject_grades[k][1]/6
      f_weighted_credits_to_compensate += subject_grades[k][1]
   else:
      f_needed_compensation = 0
   f_weighted_grade_to_compensate += f_needed_compensation
   subject_grades[k].append(f_needed_compensation)
if f_weighted_credits_to_compensate > 12:
   bcancompensate=0

# If the compensation is not possible, or not needed, it just print the output and finish
if bcancompensate==0 or  f_weighted_grade_to_compensate==0:
   for index in range(len(subject_names)):
      if( 10*subject_grades[subject_names[index]][0] % 10 == 0):
         str = "%.0f" % float(subject_grades[subject_names[index]][0])
      else:
         str = "%.1f" % float(subject_grades[subject_names[index]][0])
      print(str)
   if bcancompensate==1 and f_weighted_grade_to_compensate==0:
      print("PASS")
   else:
      print("FAIL")
   quit() # This way of treminating the program could be nicer...

# So...if we have reach this point of the program, means that compensation is needed
and possible
#...lets calculte it
# the point is that we must try to descrese the subject starting by the smallest grade.
# Lets sort them by their grades ('final grades' -> grade+compensation that we'll apply)
for k in subject_grades.keys():
   compensate_grade = subject_grades[k][0] +
subject_grades[k][3]*6/subject_grades[k][1]
   subject_grades[k].append(compensate_grade)

# Now, it tryes to decrease the approved grades, until has compensated the needed, or
can't
# modify any other grade.
for k in sorted(subject_grades, key=subject_grades.get):
   if(f_weighted_grade_to_compensate>0):
      f_possible_compensation=0
      if subject_grades[k][4] >= 9.0:
         f_possible_compensation = subject_grades[k][4] - 9.0
      elif subject_grades[k][4] >= 7.0:
         f_possible_compensation = subject_grades[k][4] - 7.0
      elif subject_grades[k][4] >= 5.0: # This allways will be true, but I write if for
conceptual reasons ;-)
         f_possible_compensation = subject_grades[k][4] - 5.0
```

```
        f_real_weighted_compensation = min
(f_possible_compensation*subject_grades[k][1]/6,f_weighted_grade_to_compensate)
        f_real_compensation = f_real_weighted_compensation*6/subject_grades[k][1]
        subject_grades[k][4] = subject_grades[k][4] - f_real_compensation
        f_weighted_grade_to_compensate = f_weighted_grade_to_compensate -
f_real_weighted_compensation

# Print acordind a reference order (of course, if we are here, it's a PASS)
for index in range(len(subject_names)):
    if( 10*subject_grades[subject_names[index]][4] % 10 == 0):
        str = "%.0f" % float(subject_grades[subject_names[index]][4])
    else:
        str = "%.1f" % float(subject_grades[subject_names[index]][4])
    print(str)
print("PASS")
```

31 Latin squares

17 points

Introduction

A latin square is an n×n array filled with n different symbols, each occurring exactly once in each row and exactly once in each column. Here is an example:

A	B	C
C	A	B
B	C	A

A common representation of a Latin square is as an array of triple (r,c,s), where r is the row, c is the column, and s is the symbol. For example, the representation of the following Latin square is:

A	B	C
B	C	A
C	A	B

{ (1,1,A),(1,2,B),(1,3,C),(2,1,B),(2,2,C),(2,3,A),(3,1,C),(3,2,A),(3,3,B) } where for example the triple (2,3,A) means that the cell in row 2 and column 3 contains the symbol A.

Two Latin squares of the same order n called L_1 and L_2 are orthogonal if, for each ordered pair of symbols (k,k') there is one and only one position (i,j) where $L_1(i,j) = k$ and $L_2(i,j)=k'$

For example, the following two Latin squares are orthogonal as each of the pairs (A,A), (A,B),…,(D,D) just appears in one of the 16 positions.

A	B	C	D
B	A	D	C
C	D	A	B
D	C	B	A

A	B	C	D
C	D	A	B
D	C	B	A
B	A	D	C

Write a program that reads in the first line an array representation of a Latin square of arbitrary order n and that for the rest of the lines gets array representations of *n*×*n* matrices with the same set of symbols of the first line Latin squares. The program must output those arrays being orthogonal Latin squares to the first one.

Assumptions:

- Symbols are alphanumeric characters and format can have spare spaces.
- Indexes start at 1.
- Lines can be empty (just carriage return), in that case, the row will be considered as a non-Latin square.
- Any of the rows can have a format error, in that case, the row will be considered as a non-Latin square.
- If the first line is not a Latin square none could be orthogonal to it.
- The input contains at least two lines.

Input

The first line is an array representation of a Latin square of order n, followed by other lines that represent nxn matrices with the same set of symbols.

```
{ (1,1,A),(1,2,B),(1,3,C),(2,1,C),(2,2,A),(2,3,B),(3,1,B),(3,2,C),(3,3,A) }
```

```
{ (1,1,A),(1,2,B),(1,3,C),(2,1,B),(2,2,C),(2,3,A),(3,1,C),(3,2,A),(3,3,B) }
{ (1,1,B),(1,2,C),(1,3,A),(2,1,C),(2,2,A),(2,3,B),(3,1,A),(3,2,B),(3,3,C) }
{ (1,1,A),(1,2,A),(1,3,A),(2,1,B),(2,2,C),(2,3,A),(3,1,C),(3,2,A),(3,3,B) }
{ (1,1,B),(1,2,A),(1,3,C),(2,1,C),(2,2,B),(2,3,A),(3,1,A),(3,2,C),(3,3,B) }
```

Output

The output is the list of arrays being orthogonal Latin squares for the provided in the first line of the input. In this case, those in lines 2 and 3, since 4 is not a Latin square and 5 is not orthogonal.

```
{ (1,1,A),(1,2,B),(1,3,C),(2,1,B),(2,2,C),(2,3,A),(3,1,C),(3,2,A),(3,3,B) }
{ (1,1,B),(1,2,C),(1,3,A),(2,1,C),(2,2,A),(2,3,B),(3,1,A),(3,2,B),(3,3,C) }
```

Solution

```
#include <iostream>
#include <sstream>
#include <string>
#include <map>
#include <vector>
#include <utility>
#include <algorithm>
#include <math.h>

class LatinSquare
{
public:

   typedef std::map<std::pair<int,int> ,char>  ReadElems;
   typedef std::map<char ,int>  SymbolCount;

   LatinSquare(  std::string & line )
   {
      line_ = line;
      squareMatrixOrder_ = 0;
      isLatinSquare_ = false;

      std::istringstream ssin(line);
      bool end = false;

      std::vector<char> markers;
      markers.push_back(',');
      markers.push_back('}');
      ReadElems elems;

      // jump first clause
      //
      if ( ! jumpToMark('{', ssin) ) return;

      // read elems
      //
   while ( ! end )
   {
      int x, y;
      char v;

      // read elem
      //
      if ( readElem(ssin, x, y, v ) )
      {
         // normalize x, y
         //
         x--;
         y--;
         elems[std::make_pair(x,y)] = v;
         // elems.insert(std::make_pair(std::make_pair(x,y),v));
         //std::pair<SymbolCount::iterator,bool> ret;
         auto ret = symbols_.insert (std::make_pair<char,int>(v,1));
         if (ret.second == false )
         {
            ret.first->second++;
         }

         //std::cout << " elem " <<x << " " << y << " " << v << std::endl;

         char nextMarker;

         // find if final elem or we continue with a ,
         //
         if ( ! jumpToMarks(markers, ssin, nextMarker ) ) return;

         // if final elem
         //
         if ( nextMarker == '}' )
         {
            int orderInt = isSquareMatrix(elems);

            //std::cout << orderInt<< std::endl;
            if ( orderInt  )
            {
               allocateData(orderInt);
               squareMatrixOrder_ = populateDataAndComputeOrder( orderInt, elems );
               isLatinSquare_ = checkForLatin();
            }
         }
      }
      else
      {
         end = true;
      }
   }
}

bool isLatinSquare()
{
   return isLatinSquare_;
}

int getOrder()
{
   return squareMatrixOrder_;
}

char getElemAt(int i, int j)
{
   if ( data_ && isLatinSquare_ )
   {
      return data_[i][j];
   }

   return ' ';
}

std::string getLine() const
{
   return line_;
}

bool isOrthogonal( LatinSquare & other )
{
   if ( ! ( isLatinSquare_  && other.isLatinSquare()) )
   {
      return false;
   }

   if ( other.getOrder() != squareMatrixOrder_ )
   {
      return false;
   }

   for( auto symbol = symbols_.begin(); symbol != symbols_.end(); symbol++)
   {
       for( auto symbol2 = symbols_.begin(); symbol2 != symbols_.end(); symbol2++)
       {
          int pairFoundCount = 0;

          //std::cout << symbol->first << " " << symbol->first
```

```
            for (unsigned int i = 0; i <  squareMatrixOrder_; i++ )
            {
               for (unsigned int j = 0; j <  squareMatrixOrder_; j++ )
               {
                  if ( ( getElemAt(i,j) == symbol->first ) && ( other.getElemAt(i,j) == symbol2->first ) )
                  {
                     pairFoundCount++;
                  }
               }
            }
            if ( pairFoundCount != 1 )
            {
               return false;
            }
         }
      }
      return true;
   }

private:

   bool jumpToMark(char mark, std::istringstream & data)
   {
      char readChar;
      do
      {
         if ( data >> std::skipws >> readChar )
         {
            if ( readChar == mark )
            {
               return true;
            }
         }
         else
         {
            return false;
         }
      }
      while (  1  );
   }

   bool jumpToMarks(std::vector<char> marks, std::istringstream & data, char & foundMark)
   {
      char readChar;
      do
      {
         if ( data >> std::skipws >> readChar )
         {
            auto found = std::find(marks.begin(),marks.end(), readChar);
            if ( found != marks.end() )
            {
               foundMark = *found;
               return true;
            }
         }
         else
         {
            return false;
         }
      }
      while (  1  );
   }

   bool readElem( std::istringstream & data, int & x, int & y ,char & v)
   {
      if ( ! jumpToMark('(', data) ) return false;
      if ( !( data  >> std::skipws >> x ) ) return false;
      if ( ! jumpToMark(',', data) ) return false;
      if ( !( data  >> std::skipws >> y ) ) return false;
      if ( ! jumpToMark(',', data) ) return false;
      if ( !( data  >> std::skipws >> v ) ) return false;
      if ( ! jumpToMark(')', data) ) return false;
      return true;
   }

   int populateDataAndComputeOrder(int orderInt, const ReadElems & elems)
   {
      for (unsigned int i = 0; i <  orderInt; i++ )
      {
         for (unsigned int j = 0; j <  orderInt; j++ )
         {
            auto found = elems.find(std::make_pair(i,j));
            if ( found  == elems.end() )
            {
               //std::cout<<"Missing " << i <<" " <<j <<std::endl;
               return 0;
            }
            else
            {
               //std::cout<<"found " << i <<" " <<j << " " << found->second <<std::endl;
               data_[i][j]=found->second;
            }
         }
      }
      return orderInt;
   }

   void allocateData(int orderInt)
```

```
   {
      data_ = new char*[ orderInt];
      for (unsigned int i = 0; i <  orderInt; ++i)
      {
         data_[i] = new char[ orderInt];
      }
   }

   int isSquareMatrix(ReadElems & elems)
   {
      int orderInt = (int) sqrt(elems.size());;

      if ( (orderInt * orderInt) != (float) elems.size())
      {
         return 0;
      }

      return orderInt;
   }

   bool checkForLatin()
   {
      for( auto symbol = symbols_.begin(); symbol != symbols_.end(); symbol++)
      {
         //std::cout << symbol->first << " " << symbol->second<<std::endl;

         if ( symbol->second != squareMatrixOrder_ )
         {
            return false;
         }
      }

      return true;
   }

private:

   char **data_;
   int squareMatrixOrder_;
   std::string line_;
   bool isLatinSquare_;

   SymbolCount symbols_;
};

int main(int argc, char **argv)
{
   std::string line;
   std::vector<LatinSquare> squares;
   while (std::getline(std::cin, line))
   {
      LatinSquare sq(line);

      squares.push_back(sq);
      /*
      if ( sq.isLatinSquare() )
      {
         std::cout<< line << " isLatin " << sq.getOrder()<< std::endl;
      }
      else
      {
         std::cout<<line << " not" << std::endl;
      }
      */
   }

   if ( squares.size() )
   {
      LatinSquare & master = squares[0];

      if ( master.isLatinSquare() )
      {
         for ( auto it = ++(squares.begin()); it != squares.end(); it++ )
         {
            if (  master.isOrthogonal(*it))
            {
               std::cout << it->getLine() << std::endl;
            }
         }
      }
   }
   return 0;
}
```

32 Fast and cheap

20 points

Introduction

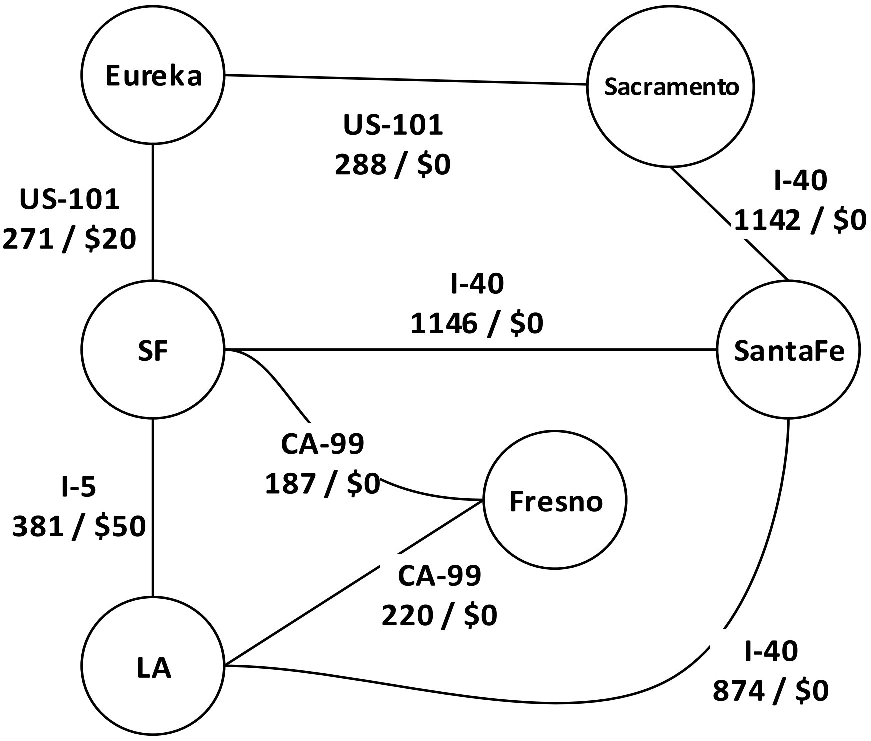
A transportation planner is interested in knowing the cheapest way (considering both fuel and the toll costs) to get between any two cities and wants you to develop a program to do so. The cost to travel between two cities is computed by the following formula:

$$Travel\ Cost = \frac{Distance}{Fuel\ Efficiency} * Gas + Toll$$

Where:

- Travel Cost is the final travel cost in US dollars.
- Distance is the distance between the places in miles.
- Fuel Efficiency is the car efficiency in miles per gallon.
- Gas is the cost is the cost of the gasoline in dollars per gallon.
- Toll is the cost in US dollars of the possible tolls the road may have.

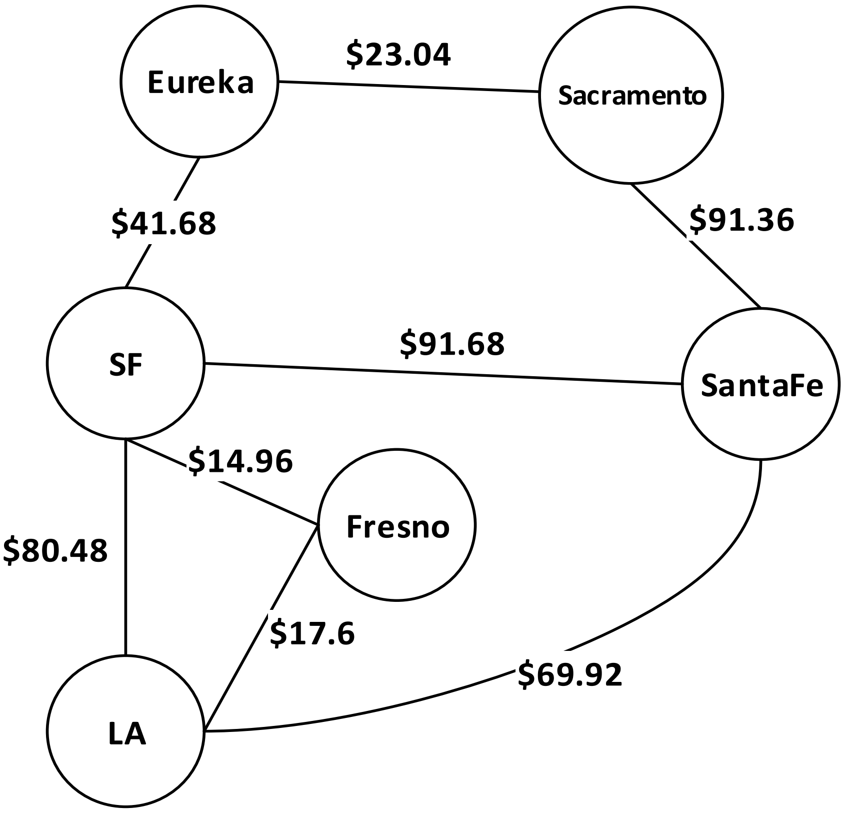
For example, consider this simple road system where the nodes represent a city and the edges joining them represent a bidirectional road. In each road segment you have, there is a distance in miles and a toll cost in dollars.



Assuming that a car has a fuel economy of 25 miles per gallon and the cost of the fuel is $2 per gallon, we can compute the travel cost to travel from San Francisco (SF) to Los Angeles (LA) as:

$$Travel\ Cost = \frac{381}{25} * 2 + 50 = 80.48\ dollars$$

Now we can compute the cost of each segment in the previous road system that will be



The minimum cost to travel from Eureka to LA would be 74.24 dollars travelling through the following roads:

1. $41.68 using US-101 from Eureka to SF.
2. $14.96 using CA-99 from SF to Fresno.
3. $17.6- using CA-99 from Fresno to LA.

Important Considerations:

- If there are two solutions that have the same exact cost you need to choose the one according to the following rule:
    i. Pick first the trip that has the minimal distance travelled.
    ii. If the distance travelled is the same, pick the one that visits less cities.
    iii. If the number of cities visited are the same, pick the trip that has a total lower toll cost.
- All the inputs and outputs to the programs will be correct, so there is no need to check that cities are defined or duplicated, check for white spaces or strange characters in the city names, negative mileages or toll costs…

Input

The input to your program will consist of one data set of in the following format:

1. A floating point number that indicates the fuel efficiency in miles per gallon.
2. A floating point number that indicates the gas cost in dollars per gallon.
3. An integer `N` that indicates the number of cities in the system, followed by the `N` names of the cities, each of them placed in a different line.
4. An integer `M` that indicates the number of roads in the system follow by `M` road definitions containing 5 items:
    i. The road city of origin as string.
    ii. The road city of destination as a string.
    iii. The road name as a string.
    iv. The road mileage as a floating point number.
    v. The road toll as a floating point number.
5. An integer `T` that indicates the number of trips to compute, followed by the `T` names of the origin and destination cities placed in a different line each.

```
25.0                                   ← fuel efficiency in miles per gallon.
2.0                                    ← fuel cost dollars per gallon.
6                                      ← 6 cites followed its names.
Eureka
Sacramento
SF
SantaFe
Fresno
LA
8                                      ← 8 roads segments.
Eureka Sacramento US-101 288.0 0.0     ← origin destination name mileage toll
Eureka SF US-101 271.0 20.0
Sacramento SantaFe I-40 1142.0 0.0
SF LA I-5 381.0 50.0
SF SantaFe I-40 1146.0 0.0
SF Fresno CA-99 187.0 0.0
Fresno LA CA-99 220.0 0.0
LA SantaFe I-40 874.0 0.0
3                                      ← number of trips.
Eureka LA                              ← first trip.
Eureka SF                              ← second trip.
```

```
Fresno SantaFe                          🡐 third trip.
```

Output

1. The total trip cost in US dollars and the total mileage travelled using a precision of 2 decimals.
2. The roads visited during the trip containing:
   i. The road name.
   ii. The road mileage using a precision of 2 decimals.
   iii. The road segment cost in dollars using a precision of 2 decimals.
   iv. The city of origin.
   v. The city of destination.

```
74.24 678.00                                 🡐 first trip cost.
US-101 41.68 Eureka SF                       🡐 the 3 trip steps.
CA-99 187.00 29.92 SF Fresno
CA-99 220.00 17.60 Fresno LA
41.68 271.00                                 🡐 second trip cost.
US-101 4271.00 1.68 Eureka SF                🡐 the single trip steps.
87.52 1094.00                                🡐 third trip cost.
CA-99 220.00 17.60 Fresno LA                 🡐 the 2 trip steps.
I-40 874.00 69.92 LA SantaFe
```

Solution

```
#include <iostream>
#include <string>
#include <algorithm>
#include <limits>
#include <utility>
#include <deque>
#include <set>
#include <list>
#include <stack>
#include <iomanip>

class City;

class Road
{
public:
   Road (std::set<City>::const_iterator origin, std::set<City>::const_iterator destination, const std::string
& name, float mileage, float toll, float efficiency, float gas) :
         origin_(origin), destination_(destination), name_(name), mileage_(mileage), toll_(toll)
   {
      // compue the cost of the trip using this road.
      //
      cost_ = ((mileage / efficiency) * gas) + toll;
   }

   ~Road()
   {
   }

   /** get the road name.
    *  \return the road name as a string.
    */
   const std::string & name() const
   {
      return name_;
   }

   /** get the road mileage.
    *  \return the road milage.
    */
   float mileage() const
   {
      return mileage_;
   }

   /** get the road toll cost.
    *  \return the road toll cost.
    */
   float toll() const
   {
      return toll_;
   }

   /** get the road cost in dollars.
    *  \return the road cost in dollars.
    */
   float cost() const
   {
      return cost_;
   }

   /** get the road's city of origin.
    *  \return the iterator that points to the city of origin.
    */
   std::set<City>::const_iterator origin() const
   {
      return origin_;
   }

   /** get the road's city of destination
    *  \return the iterator that points to the city of destination.
    */
```

```
   std::set<City>::const_iterator destination() const
   {
      return destination_;
   }

private:
   std::set<City>::const_iterator origin_;
   std::set<City>::const_iterator destination_;
   std::string name_;
   float    mileage_;
   float    toll_;
   float    cost_;
};

class City
{
public:

   City(const std::string & name) : name_(name)
   {
   }

   ~City()
   {
   }

   /** get the city name.
   *  \return the city name as a string.
   */
   const std::string & name() const
   {
      return name_;
   }

   /** get the list of outbound roads from \e this city.
   *  \return a reference to the list of outbound roads from \e this city.
   */
   const std::list<Road> & roads() const
   {
      return roads_;
   }

   /** add an outbound road to the city.
      \param[in] road the road to be added.
   */
   void add_road (const Road & road) const
   {
      roads_.push_back(road);
   }
   /** overloaded equal operator.
      \param[in] rhs the right-hand-side of the comparison.
      \return \e true if both cities are equal, \false otherwise.
   */
   bool operator== (const City & rhs) const
   {
      return name_ == rhs.name_;
   }

   /** overloaded less-than operator.
      \param[in] rhs the right-hand-side of the comparison.
      \return \e true if \e this city name is strictly less than the right-hand-side.
   */
   bool operator< (const City & rhs) const
   {
      return name_ < rhs.name_;
   }

private:
   std::string          name_;
   mutable std::list<Road> roads_;
};

float toll_cost(std::deque<Road> trip)
{
   float toll_cost = 0.0;
   while (!trip.empty())
   {
      toll_cost += trip.front().toll();
      trip.pop_front();
   }
   return toll_cost;
}

void find_cheapeast_path_r(std::set<City>::const_iterator it_origin, std::set<City>::const_iterator
final_destination, std::set<City> & visited, std::deque<Road> & current_trip, float current_mileage, float
current_cost, std::deque<Road> & cheapest_trip, float & min_mileage, float & min_cost)
{
   if (*it_origin == *final_destination)
   {
      // check whether this trip is better than the current one.
      //
      bool cheaper;
      if (current_cost < min_cost)
      {
         // the trip is cheaper in absolute dollars.
         //
         cheaper = true;
      }
      else if (current_cost == min_cost)
      {
         // the trip has the same dollar cost, check the mileage.
         //
         if (current_mileage < min_mileage)
         {
            // the trip is shorter.
            //
            cheaper = true;
         }
         else if (current_mileage == min_mileage)
         {
            // the trip has the same mileage, check the number of cities visited.
            //
            if (current_trip.size() < cheapest_trip.size())
            {
               // the trip will visit less cities.
               //
               cheaper = true;
            }
            else if (current_trip.size() == cheapest_trip.size())
            {
               // the trip will visit the same number of cities, check the total toll costs.
               //
               float current_tool_cost  = toll_cost (current_trip);
               float cheapest_tool_cost = toll_cost (cheapest_trip);
               if (current_tool_cost < cheapest_tool_cost)
               {
                  // the trip has a lower toll cost.
                  //
                  cheaper = true;
               }
               else
               {
                  // same cost, miles and number of cities visited but higher toll costs, forget about it.
                  //
                  cheaper = false;
               }
            }
            else
            {
               // same cost and miles but more cities visited, forget about it.
               //
               cheaper = false;
            }
         }
         else
         {
            // same cost but more miles, forget about it.
            //
            cheaper = false;
         }
      }
      else
      {
         // more expensive, forget about it.
         //
         cheaper = false;
      }

      if (cheaper)
      {
         // we did find a cheaper, shorter way that visits less cities.
         //
         cheapest_trip = current_trip;
         min_cost = current_cost;
         min_mileage = current_mileage;
      }
   }
   else
   {
      // we are not there, traverse all outbound roads from the origin city.
      //
      for (auto road = it_origin->roads().cbegin(); road != it_origin->roads().cend(); ++road)
      {
         // check that we haven't been there.
         //
         if (visited.find(*(road->destination())) == visited.end())
         {
            // mark as visited.
            //
            visited.insert(*(road->destination()));

            // add the road into the current trip and increment the current milage and cost.
            //
            current_trip.push_back(*road);
            current_mileage = current_mileage + road->mileage();
            current_cost    = current_cost + road->cost();

            // recursively visit the next city.
            //
            find_cheapeast_path_r(road->destination(), final_destination, visited, current_trip,
current_mileage, current_cost, cheapest_trip, min_mileage, min_cost);

            // remove the road from the current trip and decrement its milage and cost from it.
            //
            current_mileage = current_mileage - road->mileage();
            current_cost    = current_cost - road->cost();
            current_trip.pop_back();

            // unmark it from the visited set.
            //
            visited.erase(*(road->destination()));
         }
      }
   }
}

void find_cheapeast_path_recursive(std::set<City>::const_iterator it_origin,
std::set<City>::const_iterator it_destination, std::deque<Road> & cheapest_trip, float & min_mileage,
float & min_cost)
{
   std::deque<Road> current_trip;

   // initialize the visited set with the origin city.
   //
   std::set<City>   visited;
   visited.insert(*it_origin);

   // initialize the minimum mileage and cost with the maximum floating point number.
   //
   min_mileage = std::numeric_limits<float>::max();
   min_cost = std::numeric_limits<float>::max();

   // reset the current milage and cost.
   //
   float current_mileage = 0.0;
   float current_cost = 0.0;

   // recursively find the cheapest path.
   //
   find_cheapeast_path_r(it_origin, it_destination, visited, current_trip, current_mileage, current_cost,
cheapest_trip, min_mileage, min_cost);
}

int main()
{

   std::set<City> cities;

   // read the efficiency & gas cost and pre-compute the road cost.
   //
   float efficiency, gas;
   std::cin >> efficiency >> gas;

   // read the cities.
   //
   int num_cities;
   std::cin >> num_cities;

   for (int i = 0; i < num_cities; ++i)
   {
      std::string city_name;
      std::cin >> city_name;
      cities.insert (City(city_name));
   }

   // read and create the roads.
   //
   int num_roads;
   std::cin >> num_roads;

   for (int i = 0; i < num_roads; ++i)
   {
      std::string city_origin;
      std::string city_destination;
      std::string road_name;
      float       road_mileage;
      float       road_toll;

      std::cin >> city_origin >> city_destination >> road_name >> road_mileage >> road_toll;

      // search the origin and destination cities.
      //
      std::set<City>::iterator it_origin      = cities.find(City(city_origin));
      std::set<City>::iterator it_destination = cities.find(City(city_destination));

      // create one road in each cities going in both directions.
      //
      it_origin->add_road (Road(it_origin, it_destination, road_name, road_mileage, road_toll, efficiency,
gas));
      it_destination->add_road(Road(it_destination, it_origin, road_name, road_mileage, road_toll,
efficiency, gas));
   }

   // now read the number of trips.
   //
   int num_trips;
   std::cin >> num_trips;

   for (int trip = 0; trip < num_trips; ++trip)
   {
      // now read the origin and destination cities of the trip.
      //
      std::string city_origin;
```

```
    std::string city_destination;
    std::cin >> city_origin >> city_destination;
    std::set<City>::const_iterator it_origin = cities.find(City(city_origin));
    std::set<City>::const_iterator it_destination = cities.find(City(city_destination));

    // now find the cheapest path using a recursive algorithm.
    //
    float min_mileage;
    float min_cost;
    std::deque<Road> cheapest_trip;
    find_cheapeast_path_recursive(it_origin, it_destination, cheapest_trip, min_mileage, min_cost);

    // genererate the output.
    //
    std::cout << std::fixed << std::setprecision(2) << min_cost << " " << min_mileage << std::endl;
    while (!cheapest_trip.empty())
    {
      Road road = cheapest_trip.front();
      std::cout << road.name() << ' ' << std::fixed << std::setprecision(2) << road.mileage() << ' ' <<
road.cost() << ' ' << road.origin()->name() << ' ' << road.destination()->name() << std::endl;
      cheapest_trip.pop_front();
    }
  }

  return 0
```

CodeWars 2016

Barcelona