

**Twelfth Annual
University of Central Florida
High School
Programming Tournament:
Online Edition**

Problems – Division 2

Problem Name	Filename
Arithmetic Sequence	arithmetic
Not Your Serve	badminton
Banana Pancake Perfection	banana
Emojitype Spelling	emoji
Cheating at Hangman	hangman
Dr. Manhattan	manhattan
Playing the Piano	piano
Pizza Picky	picky
Snow Day	snow

Call your program file:
filename.cpp, filename.java, or filename.py

For example, if you are solving Banana Pancake Perfection,

Call your program file:
banana.cpp, banana.java, or banana.py
Call your Java class: banana

Arithmetic Sequence

Filename: arithmetic

One day, Vasya searched left pocket and found mysterious number n . Then, he searched right pocket and found more mysterious number s . He wanted to find an arithmetic sequence of n terms, where the sum of its terms equal s . Can you help him?

Recall that an arithmetic sequence is a sequence of numbers such that the difference between consecutive terms is constant. For example, “3 2 1” and “1 4 7 10 13” are each arithmetic sequences, but “4 6 1 5” and “1 4 9 16 25” are not.

The Problem:

Find an arithmetic sequence of n terms that sums to s .

The Input:

The first line of the input will begin with a single, positive integer, t , representing the number of days. For each day there is a single line containing only two integers, n and s ($1 \leq n \leq 10^5$; $1 \leq s \leq 10^9$), representing the number of terms and the sum, respectively.

The Output:

For each day, output the arithmetic sequence (each number separated by a single space), or the string “IMPOSSIBLE” if there is no answer. Every integer in your answer must fit in a signed 64-bit integer. If there are multiple answers, you may output any of them.

Sample Input:

```
3
3 6
5 55
3 2
```

Sample Output:

```
3 2 1
5 8 11 14 17
IMPOSSIBLE
```

Not Your Serve

Filename: badminton

One pleasant winter, Alice and her friend, Bob, get together to play badminton. Badminton is a racquet and net-based game, similar to tennis. According to the common rules, after a player serves, the players hit the shuttlecock back and forth until someone fails to hit it back to the opponent's side, or if the player fails to land the shuttlecock within the boundaries of the court. A round of each game involves exactly one serve. According to the scoring system, a player can only score on that player's own serve. This means that after a player starts the round by serving and the shuttlecock goes back and forth some number of times, if the same player (server) drops the shuttlecock (or hits it out of bounds), the opponent will not score, but will serve next. If instead, it is the opponent (non-server) who fails to hit the shuttlecock back properly, the server will score a point and serve again the next round. Each game ends at the end of exactly s rounds.

Alice and Bob have a habit of keeping track of how many times they hit the shuttlecock back and forth during each round. So, they would like you, their good friend, to write a program that will report their final score for every game after a given number of rounds. Note that both players are quite proficient at this game and so, they will never hit the shuttlecock out of bounds.

The Problem:

Given the player who serves the first round, the number of total rounds per game, and the number of times the shuttlecock went back and forth in each round, determine the score of the game at the end of the given rounds.

The Input:

The first line of the input will contain a single, positive integer, t , representing the number of games Alice and Bob play. Starting on the next line, for each game, the first line will contain a single character, c , containing either the letter 'A' or 'B', indicating, respectively, whether Alice or Bob serves the first round, and a space separated positive integer, s ($0 < s < 10^6$), representing the number of rounds that game. The second line for each game will contain s space separated positive integers, each indicating the number of times the shuttlecock went back and forth between the two sides of the court during the corresponding round. For example, the integer 2 indicates that the first hit was by that round's server and the second hit was by the non-server. This means on the next move, the server must have been the one to fail to properly hit back the shuttlecock and, therefore, the opponent (currently non-server) will get the next serve. Note that each of these s positive integers will fit in a 32-bit integer variable.

The Output:

The output should contain t lines, with each line having the format " $a-b$ " where a and b are the scores of Alice and Bob, respectively, at the end of the s rounds.

Sample Input:

```
2
A 5
1 1 1 1 1
B 10
5 7 2 7 3 3 10 5 11 5
```

Sample Output:

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

Banana Pancake Perfection

Filename: banana

Jack wants to make some pancakes for breakfast. Of course, his favorite type of pancakes are banana pancakes, but he has made a mistake: He misplaced the bananas on the pancakes!

Jack has made so many banana pancakes that he has determined the optimal location for each of the banana slices, but he missed the drops and now he has to nudge the banana slices to their correct locations. Nudging, of course, takes time, which is at a premium since pancakes cook so quickly. It should be noted that banana slices are all identical, so any of the banana slices can be used to fulfill any particular banana spot. Jack wants to minimize the total distance he has to move all his banana slices, and needs your help to determine what this distance is.

The Problem:

Given the location of the banana slices, as well as the desired location of bananas, determine the minimum sum of distances Jack would have to nudge the banana slices.

The Input:

The first line will be with a single, positive integer, p , representing the number of pancakes Jack needs to analyze. The first line of each pancake will contain an integer, n ($0 < n < 8$), representing the number of banana mishaps on the pancake. Then, n lines will follow, each containing two integers, b_x and b_y ($0 < b_x < 1,000$; $0 < b_y < 1,000$), where (b_x, b_y) denotes the location of a banana. Following this, n lines will follow, each containing two integers, s_x and s_y ($0 < s_x < 1,000$; $0 < s_y < 1,000$), where (s_x, s_y) denotes a spot where a banana slice needs to lie.

The Output:

For each pancake, output the total minimum distance Jack will have to nudge his banana slices so that each spot has a banana slice. Your answers will be considered correct if they are within absolute or relative error of 0.0001.

Sample Input:


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

Sample Output:

```
3.2360679775
0.0000000000
```

Emojitype Spelling

Filename: emoji

Peter the Panda and Josh the Jiraf (he insists on this spelling of “giraffe”) like to chat on Discord, an application/website for messaging. In Discord, Josh recently found out that he could react to chat messages with emojis of a single letter (such as ). Since Discord has all 26 English letter emojis, he thought that you could spell any word by reacting in a specific order.

However, he quickly found that you can’t spell some words. Apparently, you can’t add the same emoji to a message twice; trying to add it again will remove it instead. Therefore, since there are only emojis for each capital letter in discord (Josh won’t stand for using numbers and other symbols for letters), you can only use each letter once. This means that you can respond with words such as “Josh”, but not “Peter” as you only have one “E” emoji. In addition, you can respond with the *superior* spelling “jiraf” but not “giraffe” as you only have one “F” emoji.

Josh wants to find out what words he can and cannot respond to messages with. Since he has many things he’d like to respond with, could you write a program to help him?

The Problem:

Given a word, determine if you could spell it out in emojis.

The Input:

The first line of the input will be a single, positive integer, n , representing the number of chat messages Josh wants to type in emoji. The following n lines will contain a single word consisting of only uppercase English letters with no spaces. Each word will be between 1 to 26 letters in length.

The Output:

For each word, output “Emote away!” if you can write the word using only emojis, and “Nope” if you cannot.

Sample Input:

```
4
JOSH
PETER
JIRAF
GIRAFFE
```

Sample Output:

```
Emote away!
Nope
Emote away!
Nope
```

Cheating at Hangman

Filename: hangman

Josh the Great Wizard of Woz has been playing hangman (a guess-the-secret-word game) with Josh the Thin, and the Great Wizard thinks Mr. Thin is cheating by using non-existent words. Mr. Thin insists that his secret word is, in fact, a word. To settle this, both Joshes decided to seek the help of Josh the Fair and Wise to see if the current clue given by Mr. Thin corresponds to a real word. Can you help these Joshes sort this out?

Hangman is a game where one player (the “secret keeper”) chooses a secret word that fits a category while the other player (the “guesser”) tries to find the word by guessing a single letter at a time. The game begins with the category revealed and underscores (“_”) drawn for each letter in the secret word, which represent unknown letters. The guesser then chooses a letter, and if the letter is in the word then the guesser reveals all the appearances of that letter. If the letter is not in the word, then the secret keeper begins drawing a piece of a hangman.

For example, if the secret keeper chooses the word “apple” for the category of fruit, they would write five underscores (one for each letter). Then if the guesser guesses the letter “p”, then the secret keeper will reveal all the appearances of p so the clue now looks like “_pp_” (since the guesser hasn’t guessed “a”, “l”, or “e” they are still represented as underscores).

The guesser continues to guess until either they complete the word and win the game, or until they guess incorrectly too many times and the secret keeper completes the drawing of the hangman and wins. For now, the Great Wizard isn’t concerned with who won, but rather if there was any cheating.

Josh the Fair and Wise is equipped with dictionaries for each category of words, but he’s far too wise to manually go through the list himself. Thus, he’s enlisted your help!

The Problem:

With these rules in mind, can you help Josh the Fair and Wise judge if there was any cheating? Because the Joshes play a lot of hangman in different categories, you may have to judge multiple games across multiple categories.

The Input:

The first line of the input will begin with a single, positive integer, c , representing the number of categories of hangman to judge. Each category will begin with a single word, s , representing the name of the category. The next line will contain a single, positive integer, n ($1 \leq n \leq 1,000$), representing the number of words in the dictionary for this category. The next n lines will contain a single word, w , consisting of only lowercase letters that represent a word (of length 1 to 10, inclusive) in that dictionary. The following line will then contain a single, positive integer, g ($1 \leq g \leq 100$), representing the number of games to judge for this category. The next g lines will contain a single string, q , consisting of only lowercase letters or underscores that represent the current clue (also of length 1 to 10, inclusive). It is guaranteed that the query string will contain at least one and at most four underscores.

The Output:

For each query, output “That's not a word for a(n) <category>!” replacing “<category>” with the name of the category if Josh the Thin cheated, or instead output the phrase “No cheaters here!” if Josh the Thin did not cheat.

Sample Input:

```
2
fruit
3
apple
banana
cherry
3
ba_a_a
ch_rrry
ap_le
animal
3
duck
penguin
pengwin
4
hum_n
peng_in
-----
---
```

Sample Output:

```
No cheaters here!
That's not a word for a(n) fruit!
That's not a word for a(n) fruit!
That's not a word for a(n) animal!
No cheaters here!
No cheaters here!
That's not a word for a(n) animal!
```


Dr. Manhattan

Filename: manhattan

Dr. Manhattan is known to be one of the greatest heroes of all time. However, his arch nemesis, Professor Euclid is plotting against him once again. Euclid plans to steal the ultimate power for himself and dominate the world. Dr. Manhattan cannot allow Euclid to reach the destination before him but he can only move forward in one of the four cardinal directions (up, down, left, right). Euclid, however, can move in any angle he wants. Assuming that Dr. Manhattan and Euclid move at the same speed, can you determine if the evil Professor Euclid will reach the destination before Dr. Manhattan?

The Problem:

Given a point A where both Dr. Manhattan and Euclid start at, determine if Euclid can reach the destination, point B, faster than Dr. Manhattan.

The Input:

The first line of the input will contain a single, positive n , representing the number of times Euclid schemes for world domination. The following n lines will contain four integers, x_1 , y_1 , x_2 , and y_2 ($-10^9 \leq x_1 \leq 10^9$; $-10^9 \leq y_1 \leq 10^9$; $-10^9 \leq x_2 \leq 10^9$; $-10^9 \leq y_2 \leq 10^9$), representing the starting (x_1, y_1) and ending (x_2, y_2) points on a coordinate plane.

The Output:

For each scheme, output a single line. The line should be "Euclid is too fast!" if Euclid can reach the ultimate power in strictly less time than Dr. Manhattan, or it should be "Dr. Manhattan wins again!" otherwise.

Sample Input:

```
3
0 0 1 1
30 -100 30 2001
-160 88 -160 10000000
```

Sample Output:

```
Euclid is too fast!
Dr. Manhattan wins again!
Dr. Manhattan wins again!
```

Playing the Piano

Filename: piano

Stickman wants to play a piece on the piano. Since he is a stick man, he does not have any fingers and can only play one note with one hand. Nevertheless he found a rather easy piece that he can play. Stickman has a left hand and a right hand and can play any note with any hand.

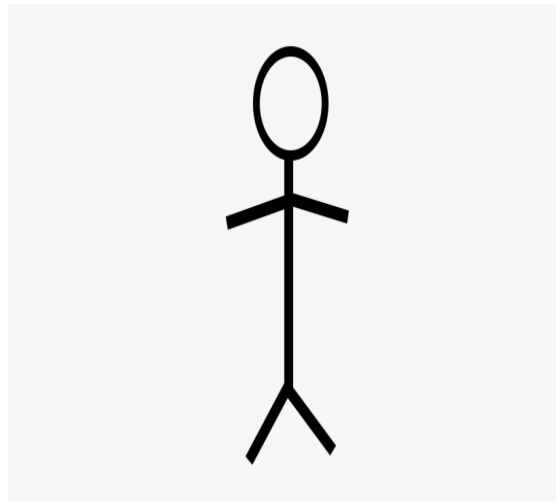
However, there is one critical condition: if Stickman's hands ever cross then they will form an X and he will die.

Formally, if Stickman plays a note with his left hand that is strictly to the right of the last note he played with his right hand, or plays a note with his right hand that is strictly to the left of the last note he played with his left hand, he will die. Note that Stickman hands begin off of the piano for each piece he plays.

Piano for reference:



Stickman for reference:



The Problem:

You will be given the notes of a piece. On your piano you have the notes in this order (from left to right): CDEFGAB. For each note, print L or R - depending on whether it should be played with the left hand or the right hand.

The Input:

The first line of the input will begin with a single, positive integer, t , representing the number of pieces to play. For each piece, there is a single line containing a string, p ($1 \leq |p| \leq 1,000$), of uppercase English characters from A to G denoting the notes of the piece.

The Output:

For each piece, output a single line containing characters L and/or R, denoting which hand was used to play its corresponding note. If there are multiple answers, print any of them.

Sample Input:

```
3
EEFGGFEDCCDEEDDEEFGGFEDCCDEDCC
EDCDEEEDDDEGGEDCDEEEDDEDC
CAB
```

Sample Output:

```
LLRRRRLLLLLLLLLLLLLRRRRLLLLLLLLLL
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
LRR
```

Pizza Picky

Filename: picky

Meghan and Natalie are twins who love going to their favorite restaurant, Pizza Plane! Pizza Plane is known for serving strange shapes of pizza. Each pizza is a convex polygon on the coordinate plane. Also, pizza is not complete without toppings! Each topping is defined by a point on the coordinate grid lying strictly within the polygon.

Meghan and Natalie want to slice the pizza into two pieces so they can share. When it comes to sharing pizza, Meghan is extremely picky. She defines a valid slice as a line segment fulfilling the following four criteria:

- Both endpoints of the segment must coincide with vertices of the polygon
- The segment splits the pizza into exactly two pieces, each with positive area
- No toppings lie directly on the segment
- After the slice, the same number of toppings lie within each piece of pizza

Meghan needs your help to determine how to slice the pizza in a valid way. If there are multiple valid slices, Meghan will choose the slice such that she and Natalie get as close to the same area of pizza as possible.

The Problem:

Given a convex polygon representing the pizza and the coordinates of the toppings, determine whether a valid slice exists, and the minimum absolute difference between the areas of the pieces of pizza after a valid slice.

The Input:

The first line of the input will contain a single, positive integer, t , representing the number of pizzas.

The first line of each pizza will contain two integers, n and m ($4 \leq n \leq 100$, $1 \leq m \leq 10$), representing the number of vertices of the pizza and the number of toppings the pizza has, respectively.

The following n lines then each contain two integers, x and y ($0 \leq x \leq 1,000$; $0 \leq y \leq 1,000$), representing the coordinates of each vertex. The vertices are given in counter-clockwise order and guaranteed to represent a convex polygon.

The following m lines each contain two integers, x and y ($0 \leq x \leq 1,000$; $0 \leq y \leq 1,000$), representing the coordinates of each topping. Each topping will lie strictly within the pizza.

The Output:

For each pizza, if no valid slice exists, output "No". Otherwise, output "Yes", followed by the minimum absolute difference between the two pieces of pizza after a valid slice. The answer will be accepted if it is within 0.000001 absolute error or 0.0001% relative error of the right answer. Print a blank line after the output for each pizza.

Sample Input:

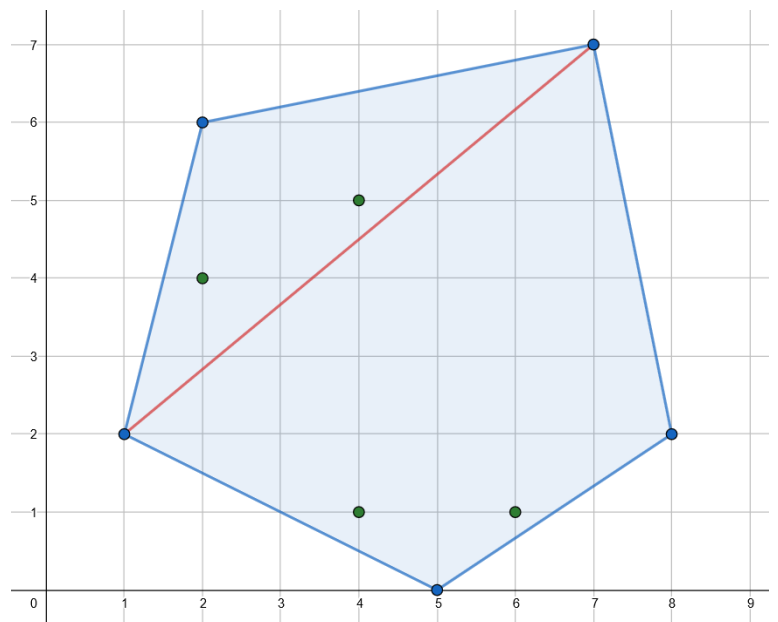
2
5 4
5 0
8 2
7 7
2 6
1 2
2 4
4 5
6 1
4 1
4 4
0 0
5 0
5 5
0 5
1 1
4 1
1 4
4 4

Sample Output:

Yes
15

No

This diagram corresponds to the first sample test case.



Snow Day

Filename: snow

Jake, Natasha, and Vick are enjoying the freezing weather outside and the snow pouring down from the sky. They decide to build some snowmen but because of the amount of snow falling they decide they can make snowmen bigger than the traditional three layers. The three realize that they used up all the snow to make a circle of snowmen of various heights numbered 1 to n .

Not wanting their fun to end, they end up playing a game. Jake, Natasha, and Vick take turns pushing the tallest remaining snowman over to the left or to the right thus destroying it. If there are multiple tallest snowmen, the one with the smallest index is chosen. Whenever a snowman of height x is pushed over, the top $x - 1$ layers destroy the neighboring $x - 1$ snowmen either to the left or right of it. If a neighbor is already destroyed, the next intact neighboring snowman will be destroyed. If there are not enough intact neighboring snowmen, the game simply ends.

Since they want to destroy all the snowmen quickly, they want to know how many turns it will take to destroy all the snowmen. Note: the snowman directly to the right of the last snowman is the first snowman, and likewise the snowman left of the first snowman is the last snowman.

The Problem:

Given a circular list of integers of size n denoting the height of the i th snowman and list of directions denoting which direction the tallest remaining snowman must be pushed in on the j th move, determine how many total turns Jake, Natasha, and Vick take in total to destroy all the snowmen.

The Input:

The first line of input will contain a single, positive integer, t , representing the number of snow days. For each snow day there will be three lines of input. The first line will contain a single integer, n ($1 \leq n \leq 10^5$), which is the number of snowmen. The second line will contain a list of n integers denoting the height, h_i ($1 \leq h_i \leq n$), of the i th snowman. The third line will contain a single string, s , of length n . This string, s , will only contain the characters 'L' and 'R'. The j th character in s indicates which direction the snowman on the j th turn should be pushed. Note that not all n turns may end up being used so s may end up having more directions than the amount of turns taken; in this case ignore the remaining directions.

The Output:

For each snow day, output a line containing a single integer representing the total number of turns taken to destroy all the snowmen on the that snow day.

Sample Input:

```
5
2
2 2
RL
3
2 2 1
RLR
10
1 1 1 1 1 1 1 1 1 1
RLLRRLRLR
10
1 1 1 1 3 2 3 2 4 1
RLLLRLLLRR
7
1 2 3 7 6 5 4
RLLRLLR
```

Sample Output:

```
1
2
10
3
1
```