Twelfth Annual
University of Central Florida

High School Programming Tournament: Online Edition

Problems – Division 1

Problem Name	**Filename**
Arithmetic Sequence	arithmetic
Banana Pancake Perfection	banana
Emojitype Spelling	emoji
Genome Transformation	genome
Cheating at Hangman	hangman
Minimum Odd Spanning Tree	most
Rook and Pawns	rook
Snow Day	snow
Bestee Ler-Indawest	thief

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* ($1 \le n \le 10^5$; $1 \le s \le 10^9$), 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
```

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 being 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 $\boxed{A}$). 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
```

Genome Transformation

Filename: `genome`

A new form of cancer was sadly recently discovered. It was named "Genometry" because it affects the human genome. While very destructive, it is also predictable. A DNA sequence can be represented as a string of the letters 'A', 'C', 'T', and 'G'. Every year the DNA in an affected organism changes. Every letter in the sequence transforms into a certain pattern.

For example, assume "A" becomes "CT", "C" becomes "GATA", "T" becomes "GGACT", and "G" becomes "T". After one year, the DNA sequence "ACGTA" will then transform into the new sequence "CTGATATGGACTCT".

Scientists have taken a DNA sample affected by the disease and would like to estimate the count of each letter in the string if the disease is allowed to develop for *n* years.

The Problem:

Determine the count of each letter in an affected DNA sequence after *n* years. Since the number may be very large, print it modulo 1,000,000,007.

The Input:

The first line of the input begins with a single, positive integer, *t*, representing the number of DNA sequences. Each sequence consists of six lines. The first line contains the pattern which the letter A transforms into, the second contains the pattern for C, the third contains the pattern for T, and the fourth contains the pattern for G. Then, the fifth line for that sequence contains the starting DNA sequence. Lastly, the sixth line for the sequence contains a single integer, n ($1 \le n \le 10^{18}$), representing the number of years to simulate as described.

All the patterns and the starting DNA sequence consist of only of the characters '`A`', '`C`', '`T`', and '`G`' and will be between 1 and 100 characters in length, inclusive.

The Output:

For each , output a single line saying "`Genome #`*`i`*`:`" where *i* is the number of the DNA sequence, followed by a space and four single space-separated integers: the count of the letters 'A', 'C', 'T', and 'G' in the final string, in that order.

Sample Input:

```
2
ATCG
G
CAT
GG
AT
2
GGG
TTT
CCC
AAA
ACTG
1
```

Sample Output:

```
Genome #1: 4 4 4 6
Genome #2: 3 3 3 3
```

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_i, 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!
```

Minimum Odd Spanning Tree

Filename: `most`

Flavor text is very important. Some might say it is the MOST important. The acronym of this problem title is MOST.

The Problem:

Find the minimum spanning tree that has an odd sum of edge weights. A minimum spanning tree of a graph is a subset of edges that connects all vertices together without any cycles that has minimum possible total edge weight.

The Input:

The input will start with a single integer, *t*, representing the number of graphs. For each of the *t* graphs, a line follows that contains two integers, *n* and *m* ($1 \leq n \leq 10^5$, $1 \leq m \leq 10^5$), representing the number of nodes and the number of edges, respectively. Next, *m* lines follow each containing three integers, *u*, *v* and *w* ($1 \leq u \leq n$; $1 \leq v \leq n$; $1 \leq w \leq 10^9$), denoting that there is an edge between node *u* and node *v* with weight *w*.

The Output:

For each graph, output an integer representing the total weight of the minimum spanning tree that has an odd sum of edge weights. If none exists, print -1.

Sample Input:

```
2
5 5
1 2 11
2 3 4
3 4 10
4 5 2
5 1 4
3 2
1 2 3
1 3 5
```

Sample Output:

```
21
-1
```

Rook and Pawns

Filename: `rook`

Levi is a young programmer prodigy who aspires to reach a chess rating higher than his Codeforces rating (2021). His teacher gave him an infinite chessboard with one white rook, one black king, and *n* black pawns. The king is at position (∞, ∞). In one turn, the rook can move horizontally and vertically by any amount. The pawns can move vertically up by one unit. If the rook takes the position of a pawn, it captures the pawn, and that pawn is removed from the board. All the pieces initially start on the same row.

The two agreed for their first lesson to play by the following algorithm:

- Levi controls the white pieces, and the teacher controls the black pieces. They will take alternating turns, starting with the white pieces.
- On Levi's turn, if there are no pawns on the same row as the rook, he will move the rook to the closest row where a pawn exists. Otherwise, he will capture the closest pawn with the rook. If there are multiple pawns, he will capture the one on the left.
- On the teacher's turn, he will move the pawn that is closest to the rook that is on the same row as the rook. If there are multiple pawns, he will move the one on the left. If there is no such pawn, the teacher will move the king.
- They will stop playing once there is one pawn remaining on the board.

The Problem:

Years later, Levi has surpassed his goal and became a certified chess grandmaster. He looks back on this first lesson fondly. He remembers the exact starting position of every piece, but he forgot the order in which the pawns were captured. Can you figure out the order so that he may complete his memory?

The Input:

The first line of the input will begin with a single, positive integer, *t*, representing the number of alternate realities for which you have to solve this problem. Each of these will consist of two lines. The first consists of a single integer, *n* ($2 \le n \le 10^5$), representing the number of pawns. The next line contains *n*+1 integers: the first is the horizontal position of the rook, and the next *n* are horizontal positions of the pawns. Keep in mind that every piece initially starts on the same row. Each position will fit in a signed 32-bit integer.

The Output:

Output *t* lines, each consisting of *n*-1 integers: the order in which the pawns were captured. The pawns are numbered in the order they were given in the input

Sample Input:

```
3
3
1 2 3 4
2
5 1 6
4
10 14 2 6 18
```

Sample Output:

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

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
RLLRRLLRLR
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
```

Bestee Ler-Indawest

Filename: `thief`

Your name is Bestee Ler-Indawest, and it is your mission to pull off the biggest painting heist in history. The museum you have decided to steal from has paintings with incredible value, but also some pretty tight security. Upon stealing a painting from the wall, the paintings *immediately* to its left and right will become shielded by iron bars, unable to be stolen. You have equipped yourself with a single-use pair of bolt-cutters for exactly this scenario, allowing you to steal an additional single painting from behind bars, per museum.

The Problem:

Given a lineup of paintings and their values, determine the maximum amount of money you can make off your heist.

The Input:

The input starts with a single, positive integer, n, representing the number of museums to make your heist on. This is immediately followed by n lines of museum descriptions. Each museum starts with an integer, p ($1 \leq p \leq 10^4$), indicating the number of paintings in this museum's lineup. This is followed by p integers, v_i ($1 \leq v_i \leq 10^5$), representing each painting's value.

The Output:

For each museum, output a single integer, c , representing the maximum value obtainable.

Sample Input:

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

Sample Output:

```
4
17
```