

15-110: Principles of Computing

HOMEWORK 06

Due: 13th September, 2022 at 10:00pm

- You need to complete the Python file for this assignment, and submit it to Gradescope.
- There are 100 points.
- You must solve the tasks **individually**, always abiding by the course and CMU's academic integrity policy.

1. (25 points) **Exchanging Cards**

Aliaa and Buthaina collect Yu-Gi-Oh cards, and they have some repeated cards they would like to exchange with cards that they do not have. Aliaa and Buthaina are meeting today and they are both taking *one copy* of each repeated card they have. Moreover, they are taking all the other cards that they have in their collection.

They would like to trade as many cards as possible but they are not interested in cards they already have, of course.

Your job is to write a program that computes the maximum number of cards, among the repeated ones, that can be exchanged. To make your life easier, each card is identified by a unique number.

Implement the function `exchangingCards(ra, oa, rb, ob)` that takes as input four lists: `ra`, Aliaa's repeated cards (all distinct); `oa`, Aliaa's other cards (that she is not interested in trading); `rb`, Buthaina's repeated cards (all distinct); Buthaina's other cards (that she is not interested in trading).

The function returns the maximum number of cards they could exchange.

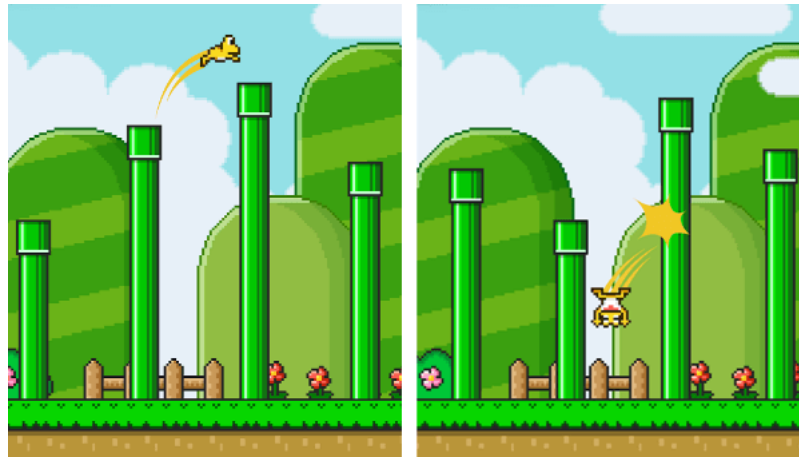
For example:

- `exchangingCards([1,2], [7,8,9], [5,3,4], [6,0,11])` returns 2
- `exchangingCards([1,2], [7,8,9], [5,2,1], [6,0,11])` returns 0
- `exchangingCards([1,2], [7,8,9], [5,3,4], [6,7,1])` returns 1
- `exchangingCards([1,2], [7,8,3], [5,3,4], [6,2,1])` returns 0

2. (25 points) **Jumping Frog**

In each stage of the Jumping Frog game you must safely get your amphibian through a sequence of pipes of different heights to the rightmost pipe. Nevertheless the frog only survives if the height difference of consecutive pipes is at most the frog jump/fall height. If the next pipe height is too high, the frog hits the pipe and falls. If the next pipe height is too low, the frog does not survive the fall. The frog always starts on the top of the leftmost pipe.

In this game the distance of pipes is irrelevant, which means that the frog always can reach the next pipe with a jump.



Implement the function `jumpingFrog(P, h)` that takes as input a list `P` with the pipes' height (from left to right), and the frog's maximum jump/fall height `h`. The function should return `"Frog wins"` if the frog can reach the last pipe safely, and `"Game over"` if it cannot.

For example:

- `jumpingFrog([1,3], 2) == "Frog wins"`
- `jumpingFrog([4,5,2], 2) == "Game over"`

You may assume that `P` will have at least one element.

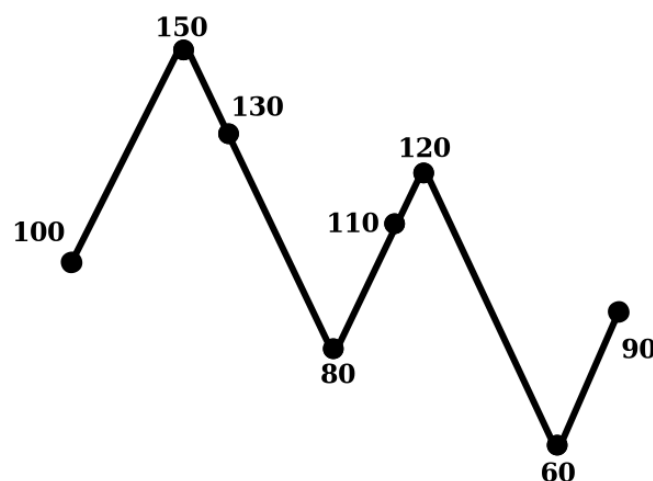
3. (25 points) Musical Loop

A musical loop is a small section of music composed to be played continuously (that is, the section is played again when it reaches the end), in a seamless way. Loops are used in many styles of popular music (hip hop, techno, etc), as well in computer games, especially casual games on the internet.

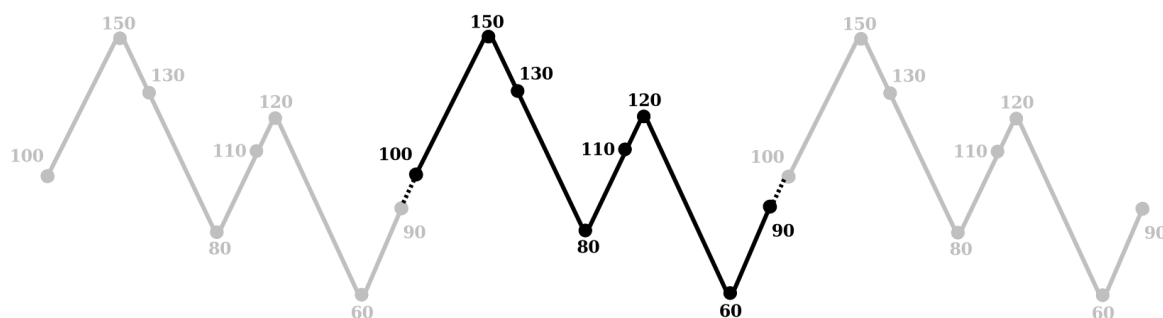
Loops may be digitalized for example using PCM (Pulse Code Modulation), a technique for representing analog signals used extensively in digital audio. In PCM, the magnitude of the signal is sampled at regular intervals, and the values sampled are stored in sequence. To produce the sound for the sampled data, the procedure is applied in reverse (demodulation).

Sideeg works for a game software house, and composed a beautiful musical loop, coded in PCM. Analyzing the waveform of his loop in audio editing software, Sideeg became curious when he noticed a number of "peaks". A peak in a waveform is a *local maximum or minimum*.

For example, suppose the samples in a musical loop were `[100, 150, 130, 80, 110, 120, 60, 90]`. Its waveform can be draw as:



Since it is a loop, the wave form repeats itself infinitely often. Here is a cut of the waveform above repeated three times. The dashed lines indicate when a repetition starts and ends:



This sample has two local maximums, at 150 and 120, and two local minimums, at 80 and 60.

Note that the endings (100 and 90) may or may not be a local maximum or minimum.

Given a sequence of samples in a musical loop, Sideeg wants to know how many peaks exist in the loop.

Implement the function `musicalLoop(L)` that takes as input a list of samples of the music composed by Sideeg, and return the total number of peaks that exist in that musical loop.

For example:

- `musicalLoop([100, 150, 130, 80, 110, 120, 60, 90]) == 4` (two local maximums at 150 and 120, and two local minimums at 80 and 60)
- `musicalLoop([1, -3]) == 2` (one local maximum and one local minimum)
- `musicalLoop([40, 0, -41, 0, 41, 42]) == 2` (one local minimum at -41, and one local maximum at 42)

You may assume that there are no consecutive samples of the same magnitude.

4. (25 points) **Prime Factors**

The fundamental theorem of arithmetic states that every integer greater than 1 is the product of a unique combination of prime numbers.

Implement the function `primeFactors(n)` that returns a list with all *distinct* prime factors of `n` in increasing order. If the number is less than 1, the function returns the empty list.

For example:

- `primeFactors(18) == [2, 3]`
- `primeFactors(5) == [5]`
- `primeFactors(100) == [2, 5]`