12-problem-solving

February 22, 2022

0.1 Problem solving

This lecture is dedicated for you to practice your problem solving skills. All problems below were taken from programming competitions.

Take this opportunity to look back and see how far you have come, and how much you have learned! A few weeks ago, these problems might have looked much more cryptic, and maybe you wouldn't even know how to begin. Hopefully by now you are able to understand them, and start having ideas on how to solve them.

Think about the solutions before coding. Explain the steps on paper as detailed as possible, and convince yourself that your solution works.

0.1.1 Frequently Asked Questions

Lots of sites in the internet have a page called "Frequently Asked Questions" that, as the name suggests, contains the questions that users make frequently.

Jarrin receives lots of questions from students, so she decided that it would be a good idea to add a Frequently Asked Questions page to Scotty. As Jarrin is very busy these days with the class scheduling for next semester, she asked you to help her with this task.

Implement the function `faq(L, n)` that takes as input: - `L`: a list with unique identifiers of the questions asked by the students, and - `n`: the number of times a question needs to be asked to be considered frequent.

This function returns a list with the identifiers of questions that will be added to to Scotty. The list should be sorted in ascending order.

For example:

```
faq([10, 40, 20, 10, 40, 10], 2) == [10, 40]
faq([1, 4, 2, 1, 3], 2) == [1]
faq([1, 1, 3, 5, 4, 6, 3, 4], 3) == []
```

In the first example, questions that were asked 2 or more times will be added to the FAQ. In this case, questions 10 and 40 will be included since they were asked three and two times, respectively.

[1]:
```
def faq(L, n):
    return [42]
```

0.1.2 Detective Watson

John Watson, after years working besides Sherlock Holmes, never understood how he was able to guess who was guilty so easily. On a certain night, however, Sherlock finally told John his secret.

"Elementary, dear Watson", said Sherlock Holmes. "It is never the most suspicious, but the second most suspicious". After he knew the secret, John decided to solve a crime on his own to test Sherlock's method.

Implement the function `guilty(L)` that takes a list with integers representing how much each person is a suspect. The higher the number, the more suspicious the person. The function should return *at which position in the list* the guilty person is, according to Sherlock's method. Since neither Sherlock or Watson are computer scientists, their list indices start from 1, and not 0.

For example:

```
guilty([3,5,2]) == 1
guilty([1,15,3,5,2]) == 4
```

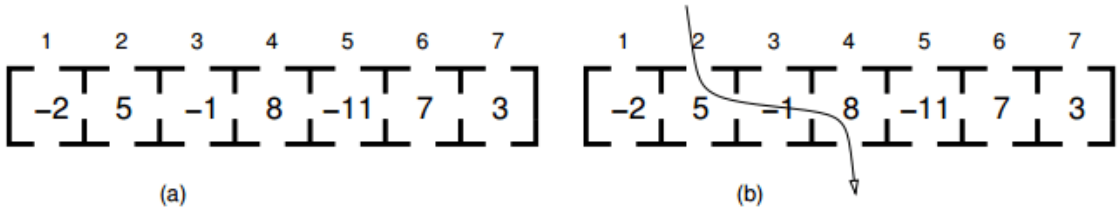
In the first example, the second most suspicious person in the group is the first one in the list.

[2]:
```
def guilty(L):
    return 42
```

0.1.3 Corridor

Aisha is programming a game, and in one of the phases the character must enter a corridor of rooms through one of its side doors, and exit through another. In each room, there is a certain amount of mana that the character will collect. But to make things interesting, Aisha decided that some of the rooms will have negative mana. The goal of the character is to choose the entry and exit point such that the amount of collected mana is maximized.

For example, in the setting of the picture, the character can get at most 12 mana, enterng through door 2 and exiting through 4.



Implement the function `mana(L)` that takes as input a list of manas in each of the rooms, and returns the maximum amount of mana the character can collect.

For example:

```
assert(mana([-2,5,-1,8,-11,7,3]) == 12)
assert(mana([-1,2,-3]) == 2)
```

[3]:
```
def mana(L):
    return 42
```

0.1.4 Door Lock

Omar was coming home one day when he realized he had lost the key to the door. Desperate, he decided to ask his friend Hasan for help, who in a few seconds managed to open the door using his tools.

Amazed at this feat, Omar asked Hasan how he did this. Hasan explained:

"A lock is a set of N pins arranged horizontally which can be moved up or down with the aid of a metal key. When this metal key is inserted in the lock, it can increase or decrease by 1 mm any two consecutive pins, or the last one by itself. When all the pins are at the same pre-determined height H, the door is unlocked."

Omar wanted to know the least number of movements he needed to make to unlock a certain door, where one movement is increasing or decreasing the height of two consecutive pins by 1mm. He also knows what is the height H the pins need to be at.

Help Omar by implementing a function `doorUnlock(L, h)` that takes as input a list `L` with the initial heights of the pins, and the target height `h` for unlocking the door. The function returns the minimum number of movements one needs to make to unlock the door.

For example:

```
doorUnlock([45,45,55,55], 50) == 10
doorUnlock([84,39,17,72,94], 84) == 77
```

[4]:
```
def doorUnlock(L, h):
    return 42
```