Problem Set 2 - Data Structures & Algorithms

13/02/2024

Aditya Narayan Rai - 235843

Question 1: The main task in this problem set is to create an extension for the calculator flask app from lab 3. In addition to the basic calculator that we created in the lab, this extension should have a page on which a user can calculate the perimeter of a circle by inputting the radius.

For the calculation of the radius, you should create a $Circle$ class, with one method to calculate the perimeter and one method to calaculate the area of the circle. You should write one unit test for each of these methods. To make this task easier, start by cloning this repository: https://github.com/lenafm/calculator_app to begin with a working version of the calculator app from the lab.

You will submit your code as a 'Pull Request' to the repository https://github.com/lenafm/calculator_app. We will then check whether your tests do what they are supposed to. Please provide a link to the 'Pull Request' in the document you submit on Moodle (along with your answers to questions 2 and 3).

For full marks you should:

- (4 points) Write the $Circle$ class (with the perimeter and area methods) in a separate python module (like helper.py, in the repository) called $circle.py$ and import it into the main flask module.
- (4 points) Create a template similar to the $calculator.html$ template, with an HTML form in which the user can input the radius, and render this template from the main module
- (4 points) Create a $test\_circle.py$ file in the root directory, which tests the two methods in the $circle.py$ module

Solution 1: This is the Pull Request link from GitHub:
https://github.com/lenafm/calculator_app/pull/14

Question 2: (1 point) Examine the following code:

```
def check_array(input):
    for idx in range (len(input)):
        if input[idx][0] == "1":
            input[idx] = None
```

```
        return input

    original_array = ["1_3", "5_2"]

    new_array = check_array(original_array)

    print(original_array)
    print(new_array)
```

What does it output? Why? Use the language we developed in lecture 3, page 30-32 (about pointers).

Solution 2:

In [1]:
```
def check_array(input):
    for idx in range (len(input)):
        if input[idx][0] == "1":
            input[idx] = None
    return input

original_array = ["1_3", "5_2"]

new_array = check_array(original_array)

print(original_array)
print(new_array)
```

```
[None, '5_2']
[None, '5_2']
```

Explanation: Here we are defining a function **'check_array'** which takes a list **'input'** as an argument and iterates over it. Then, for each element in the list, it checks if the first character of the element is **'1'** and if the condition is true, it sets the element at that index to **'None'**. The function then returns the modified list. Once the function has been defined, it is being tested on a list named **'original_array'** and the result is being assigned to a new list named **'new_array'**. The results for both the lists is the same.

The key point to note here is that when a list is created in Python, Python allocates memory for it and assigns a pointer to this memory block to the variable holding the list. And, upon passing the list to a function, Python doesn't create a new list or memory allocation but it passes the reference to the same memory location. Therefore, any changes made to the list within the function affect the original list because both the original variable and the function point to the same list in memory.

Question 3: (2 points) Consider the following algorithm:

```
    max_sum = None
    for i in range(len(array)):
        sum_subarray = 0
        for j in range (i, len(array)):
```

```
            sum_subarray += array[j]
            if max_sum is None or max_sum < sum_subarray:
                max_sum = sum_subarray
    print(max_sum)
```

This algorithm takes an array (assumed to be non-empty) named $array$ and find the subarray (contiguous elements of that array of any size) with the largest sum and outputs that sum.

Is the algorithm defined above "efficient" in the sense we defined in lecture 5(slide 30)? That is, is its runtime polynomial in the size of the array? If not, explain why not. If it is, give its runtime in Big-O notation with the smallest d such that $O(n^d)$ is true.

Solution 3:

Explanation: From the slides, an algorithm is considered 'efficient' if its runtime is polynomial in the size of the input. Now let's check if the given algorithm fits this criteria:

The algorithm has a nested loop structure with the outer loop running once for each element in the array and for each iteration of the outer loop, the inner loop runs through the remaining elements to calculate the subarray sums. What happens here is this: Begin with the first item and consider it alone as a group. Then, we add the next item to this group, evaluating the new group's total value. We keep adding the next item in the set to the group assessing each new group's total value until we have considered a group that includes every item from the starting point to the end of the set. After we have considered all possible groups starting from the first item, we move to the second item and repeat the entire process starting from the second item. We do this until we have considered groups starting from every item in the set. In the meantime, while we are creating the groups we are also noting down the group with the highest total value and as and when we find a group with a total value higher than any we have seen before we make a note of it and store it.

Therefore, since the outer loop runs once for each element in the array, it iterates n times. And, since the inner loop iterates over the remaining elements in the array for each iteration of the outer loop, it is equivalent to n, $n-1$, $n-2$, ..., 1. And, therefore the total number of operations can be estimated as the sum of the first $'n'$ natural numbers which is $\frac{n(n+1)}{2}$. Therefore, in terms of Big-O notation, the runtime of this algorithm is polynoimal: $O(n^2)$. And, since the runtime is polynomial this algorithm can be considered as 'efficient'.

Now let's run and check the algorithm for a made-up array:

In [2]:
```
array = [-2, -4, 6, -3, 4, 7, 0, 1]
max_sum = None
for i in range(len(array)):
    sum_subarray = 0
    for j in range (i, len(array)):
        sum_subarray += array[j]
```

```
        if max_sum is None or max_sum < sum_subarray:
            max_sum = sum_subarray
print(max_sum)
```

```
15
```

Note to the Professor: For the first and last questions, I used ChatGPT and Stackoverflow to understand and implement some parts of the solution.