CIND830W21_Assignment_3_updated

April 12, 2021

0.1 CIND830 - Python Programming for Data Science

0.1.1 Assignment 3 (10% of the final grade)

0.1.2 Due on April 12th, 2021 11:59 PM

This is a Jupyter Notebook document that extends a simple formatting syntax for authoring HTML and PDF. Review this website for more details on using Jupyter Notebook.

Use the JupyterHub server on the Google Cloud Platform, provided by your designated instructor, for this assignment. Ensure using **Python 3.7.6** release then complete the assignment by inserting your Python code wherever seeing the string "###CODE HERE ###"

When you click the `File` button, from the top navigation bar, then select `Export Notebook As ...`, a document (PDF or HTML format) will be generated that includes both the assignment content and the output of any embedded Python code chunks.

Using these guidelines, submit **both** the IPYNB and the exported file (PDF or HTML). Failing to submit both files will be subject to mark deduction.

0.1.3 Question 1 [30 pts]:

a) Complete the methods of the following `Stack` class according to their description

[1]:
```
class Stack:
    def __init__(self):
        """ Initialize a new stack """
        self.elements = []
    def push(self, new_item):
        """ Append the new item to the stack """
        self.elements.append(new_item)
    def pop(self):
        """ Remove and return the last item from the stack """
        return self.elements.pop()
    def size(self):
        """ Return the total number of elements in the stack """
        return len(self.elements)
```

```
    def is_empty(self):
        """ Return True if the stack is empty and False if it is not empty """
        return len(self.elements) == 0
    def peek(self):
        """ Return the element at the top of the stack or return None if the␣
↪stack is empty """
        if len(self.elements) > 0:
            return self.elements[-1]
        return None
```

Use the Stack class you defined in `Q1a` to solve the following problems.

b) Write a function to detect whether the orders of brackets is correct using stacks. Some examples are given below:

[2]:
```
exp1 = "(2+3)+(1-5)" # True
exp2 = "((3*2))*(7/3))" # False
exp3 = "(3*5))]" # False
```

[3]:
```
def is_valid(exp):
    opening = ['(', '[', '{']
    closing = [')', ']', '}']
    s = Stack()
    for character in exp:
        if character in opening:
            s.push(character)
        elif character in closing:
            idx = closing.index(character)
            if not s.is_empty() and s.peek() == opening[idx]:
                s.pop()
            else:
                return False
    return s.size() == 0
```

b) Write a function to detect whether the orders of brackets is correct using stacks. Some examples are given below:

[4]:
```
print(is_valid(exp1))
```

```
True
```

[5]:
```
print(is_valid(exp2))
```

```
False
```

[6]:
```
print(is_valid(exp3))
```

```
False
```

c) Write a simple spell checking function using stacks that removes consecutive similar strings from a sequence of words. Some examples are given below

HINT: You can use the `peek` function of the Stack

Input: hello world world \ Output: hello world

Input: Such an an amazing time to be alive! \ Output: Such an amazing time to be alive!

[7]:
```
text1 = "hello world world"
text2 = "Such an an amazing time to be alive!"
```

[8]:
```
def remove_consecutive_repeats(text: str):
    words = text.split()
    s = Stack()
    final_string = ''
    for word in words:
        if not s.is_empty() and s.peek() == word:
            continue
        else:
            s.push(word)
            final_string += word + ' '
    final_string = final_string.strip()
    return final_string
```

[9]:
```
remove_consecutive_repeats(text1)
```

[9]:
```
'hello world'
```

[10]:
```
remove_consecutive_repeats(text2)
```

[10]:
```
'Such an amazing time to be alive!'
```

0.1.4 Question 2 [30 pts]:

Implement Round Robin Scheduling algorithm using queues in Python. It is the most commonly used algorithm for CPU scheduling. - Each process takes an equal share of CPU time. For this question we choose a time quantum of 2 units. - After being processed for a predefined time, if the process still requires more computation, it is passed to a waiting queue.

Answer the following questions: - Report the time each process is completed - Report wait times of each process in the queue

$$\text{Wait time} = \text{End time} - \text{Arrival Time} - \text{Required Time}$$

Process ID	Arrival Time	Required Time
P1	0	4
P2	1	3

Process ID	Arrival Time	Required Time
P3	2	2
P4	3	1

[19]:
```
from collections import deque
time_quantum = 2
```

[20]:
```
class Process:
  def __init__(self, name, arrival_time, required_time):
    self.name = name
    self.arrival_time = arrival_time
    self.required_time = required_time
    self.time_processed = 0
  def __repr__(self):
    return self.name
```

[21]:
```
p0 = Process('P1', 0, 4)
p1 = Process('P2', 1, 3)
p2 = Process('P3', 2, 2)
p3 = Process('P4', 3, 1)
processes = [p0, p1, p2, p3]
```

[14]:
```
end_times = {process.name:0 for process in processes}
wait_times = {process.name:0 for process in processes}
```

[15]:
```
queue = deque()
running_proc = None # Tracks running process in the CPU
running_proc_time = 0 # Tracks the time running process spent in the CPU
for t in range(11):
    if t == 0:
        for p in processes:
            if p.arrival_time == 0:
                queue.append(p)

    if running_proc == None:
        if len(queue) > 0:
            running_proc = queue.popleft()
            running_proc_time = 0

    for p in processes:
        if p.arrival_time == t + 1:
            queue.append(p)

    if running_proc is not None:
```

```
        running_proc.time_processed += 1
        running_proc_time += 1
        if running_proc.required_time == running_proc.time_processed:
            end_times[str(running_proc)] = t + 1
            wait_times[str(running_proc)] = end_times[str(running_proc)] -␣
↪running_proc.arrival_time - running_proc.required_time
            running_proc = None
            running_proc_time = 0
        elif running_proc_time >= time_quantum:
            queue.append(running_proc)
            running_proc = None
            running_proc_time = 0
```

[16]:
```
print(end_times) # End times for each process
print(wait_times) # Wait times for each process in the queue
```

```
{'P1': 8, 'P2': 10, 'P3': 6, 'P4': 9}
{'P1': 4, 'P2': 6, 'P3': 2, 'P4': 5}
```

0.1.5 Question 3 [60 pts]:

Write a funciton to find the maximum length possible by cutting N given sticks into at least K pieces.
Given an array `stick[]` of size **N**, representing the length of N pieces of stick and an integer K, at least **K** pieces of the same length need to be cut from the given stick pieces. The task is to find the maximum possible length of these K stick pieces that can be obtained.

Examples:

Input: stick[] = {5, 9, 7}, K = 3
Output: 5

Explanation:
Cut arr[0] = 5 = 5
Cut arr[1] = 9 = 5 + 4
Cut arr[2] = 7 = 5 + 2
Therefore, the maximum length that can be obtained by cutting the sticks into 3 pieces is 5.

Input: stick[] = {5, 9, 7}, K = 4
Output: 4

Explanation:
Cut arr[0] = 5 = 4 + 1
Cut arr[1] = 9 = 2 * 4 + 1

Cut arr[2] = 7 = 4 + 3
Therefore, the maximum length that can be obtained by cutting the sticks into 4 pieces is 4.

[17]:
```
def isValid(stick, N, len, K):
    count = 0
    for i in range(N):
        count += stick[i]//len
    return (count >= K)
def maxlen(stick, N, K):
    left = 1
    right = max(stick)
    while (left <= right):
        mid = left + (right - left) // 2
        if (isValid(stick, N, mid, K)):
            left = mid + 1
        else:
            right = mid - 1
    return right

if __name__ == '__main__':
    stick = [ 5, 9, 7 ]
    N = len(stick)
    K = 4
    print(maxlen(stick, N, K))
```

```
4
```

0.1.6 Question 4 [20 pts]:

Write a function to move all negative numbers to the beginning and positives to the end.
An array contains both positive and negative numbers in random order.
Rearrange the array elements so that all negative numbers appear before all positive numbers.

Example :

Input: -12, 11, -13, -5, 6, -7, 5, -3, -6
Output: -12 -13 -5 -7 -3 -6 11 6 5
Note: Order of elements is **not** important.

[18]:
```
def rearrange(arr, n ) :
    j = 0
    for i in range(0, n) :
        if (arr[i] < 0) :
            temp = arr[i]
            arr[i] = arr[j]
            arr[j]= temp
            j = j + 1
```

```
    print(arr)

arr = [-12, 11, -13, -5, 6, -7, 5, -3, -6]
n = len(arr)
rearrange(arr, n)
```

```
[-12, -13, -5, -7, -3, -6, 5, 6, 11]
```