AI VIET NAM – COURSE 2022

Python Data Structure 2– Exercise

Ngày 6 tháng 6 năm 2022

1. **Thực hiện theo các yêu cầu sau .**

   (a) Viết function trả về một dictionary đếm số lượng chữ xuất hiện trong một từ, với key là chữ cái và value là số lần xuất hiện
   - **Input:** một từ
   - **Output:** dictionary đếm số lần các chữ xuất hiện
   - **Note:** Giả sử các từ nhập vào đều có các chữ cái thuộc [a-z] hoặc [A-Z]

   (b) Viết function đọc các câu trong một file txt, đếm số lượng các từ xuất hiện và trả về một dictionary với key là từ và value là số lần từ đó xuất hiện.
   - **Input:** Đường dẫn đến file txt
   - **Output:** dictionary đếm số lần các từ xuất hiện
   - **Note:**
     – Giả sử các từ trong file txt đều có các chữ cái thuộc [a-z] hoặc [A-Z]
     – Không cần các thao tác xử lý string phức tạp **nhưng cần xử lý các từ đều là viết thường**
     – Các bạn dùng lệnh này để download và tham khảo Code Listing 2
     !gdown https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko

```
# Examples 1(a)
string = 'Happiness'
count_chars(string)
>> {'H': 1, 'a': 1, 'e': 1, 'i': 1, 'n': 1, 'p': 2, 's': 2}

string = 'smiles'
count_chars(string)
>> {'e': 1, 'i': 1, 'l': 1, 'm': 1, 's': 2}
```

Code Listing 1: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

```
# Examples 1(b)
!gdown  https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko
file_path = '/content/P1_data.txt'
word_count(file_path)
>>{'a': 7,
 'again': 1,
 'and': 1,
 'are': 1,
 'at': 1,
 'be': 1,
 'become': 2,
 ...}
```

Code Listing 2: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

2. **Thực hiện xây dựng Stack bằng dictionary và list data type. Viết một function nhận 4 arguments: method_name, stack, capacity, value để mô phỏng lại cách hoạt động của stack**

   - **Input:** method_name, stack, capacity, value
   - **Outcome:** thực hiện theo method_name và ghi kết quả vào stack (input ở trên)
   - Các loại method cần thực hiện trong method_name argument:
     - **"CreateStack"** dùng để khởi tạo stack
     - **"IsEmptyStack"** kiểm tra stack có đang rỗng
     - **"IsFullStack"** kiểm tra stack đã full chưa
     - **"PopStack"** loại bỏ top element và đưa giá trị vào "RetValue" trong stack dictionary
     - **"PushStack"** add thêm value (input argument) vào trong stack
     - **"GetTopStack"** lấy giá trị của top element trong stack đưa vào "TopValue" (không làm thay đổi stack)
   - stack argument: là một dictionary chứa các thông tin cần thiết để tạo thành một stack data. Sau đây là các thông tin cơ bản yêu cầu phải có, các bạn có thể thêm các thông tin tùy ý để thực hiện được stack data cho riêng mình.
     - **"Stack"** (key) : [] (value là kiểu list chứa các element trong stack)
     - **"Capacity"** (key) : int (value là một số nguyên dương thể hiện size tối đa cho phép của stack)
     - **"IsEmptyStack"** (key) : boolean (value là kiểu boolean True nếu stack rỗng và False nếu stack có chứa ít nhất một element)
     - **"IsFullStack"** (key) : boolean (value là kiểu boolean True nếu số lượng element trong stack bằng Capacity, False nếu số lượng element trong stack nhỏ hơn Capacity
     - **"RetValue"** (key) : cùng type với element type (value là giá trị trả về khi thực hiện "PopStack" method)
     - **"TopValue"** (key) : cùng type với top element type (value là giá trị trả về của top element khi thực hiện "GetTopStack" method)

```python
# Examples stack dictionary
stack1 = {
    'Stack' : [1, 2],
    'Capacity' : 5,
    'IsEmptyStack' : False,
    'IsFullStack' : False,
    'RetValue' : 3,
    'TopValue' : 4
}

```

   - capacity argument: size tối đa của stack và chỉ được truyền vào khi đi cùng "CreateStack" method
   - value argument: là gía trị sẽ được push vào stack và chỉ được truyền vào khi đi cùng "PushStack" method

```python
# Examples
stack1 = {
    'Stack' : [],
    'Capacity' : None,
    'TopIdx' : -1,
    'IsEmptyStack' : None,
    'IsFullStack' : None,
    'RetValue' : None,
    'TopValue' : None
}


simStack(stack=stack1, method_name="CreateStack", capacity=5)

# stack1.push(1)
simStack(stack=stack1, method_name="PushStack", value=1)

# stack1.push(2)
simStack(stack=stack1, method_name="PushStack", value=2)

# print(stack1.IsFullStack())
simStack(stack=stack1, method_name="IsFullStack")
print(stack1["IsFullStack"])
>> False

# print(stack1.top())
simStack(stack=stack1, method_name="GetTopStack")
print(stack1["TopValue"])
>> 2

# print(stack1.pop())
simStack(stack=stack1, method_name="PopStack")
print(stack1["RetValue"])
>> 2

# print(stack1.top())
simStack(stack=stack1, method_name="GetTopStack")
print(stack1["TopValue"])
>> 1

# print(stack1.pop())
simStack(stack=stack1, method_name="PopStack")
print(stack1["RetValue"])
>> 1

# print(stack1.IsEmptyStack())
simStack(stack=stack1, method_name="IsEmptyStack")
print(stack1["IsEmptyStack"])
>> True
```

Code Listing 3: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

3. **Thực hiện xây dựng Queue bằng dictionary và list data type. Viết một function nhận 4 arguments: method_name, queue, capacity, và value để mô phỏng lại cách hoạt động của queue**

   - **Input:** method_name, queue, capacity, value
   - **Outcome:** thực hiện theo method_name và ghi kết quả vào queue (input ở trên)
   - Các loại method cần thực hiện trong method_name argument:
     - **"CreateQueue"** dùng để khởi tạo queue
     - **"IsEmptyQueue"** kiểm tra queue có đang rỗng
     - **"IsFullQueue"** kiểm tra queue đã full chưa
     - **"Dequeue"** loại bỏ first element và đưa giá trị vào "RetValue" trong queue dictionary
     - **"Enqueue"** add thêm value (input argument) vào trong queue
     - **"GetFirstValue"** lấy giá trị của first element trong queue đưa vào "FirstValue" (không làm thây đổi queue)
   - queue argument: là một dictionary chứa các thông tin cần thiết để tạo thành một queue data. Sau đây là các thông tin cơ bản yêu cầu phải có, các bạn có thể thêm các thông tin tùy ý để thực hiện được queue data cho riêng mình.
     - **"Queue"** (key) : [] (value là kiểu list chứa các element trong queue)
     - **"Capacity"** (key) : int (value là một số nguyên dương thể hiện size tối đa cho phép của queue)
     - **"IsEmptyQueue"** (key) : boolean (value là kiểu boolean True nếu queue rỗng và False nếu queue có chứa ít nhất một element)
     - **"IsFullQueue"** (key) : boolean (value là kiểu boolean True nếu số lượng element trong queue bằng Capacity, False nếu số lượng element trong queue nhỏ hơn Capacity
     - **"RetValue"** (key) : cùng type với element type (value là giá trị trả về khi thực hiện "Dequeue" method)
     - **"FirstValue"** (key) : cùng type với first element type (value là gía trị trả về của first element khi thực hiện "GetFirstValue" method)

```
# Examples queue dictionary
queue1 = {
    'Queue' : [],
    'Capacity' : None,
    'IsEmptyQueue' : None,
    'IsFullQueue' : None,
    'RetValue' : None,
    'FirstValue' : None
}

```

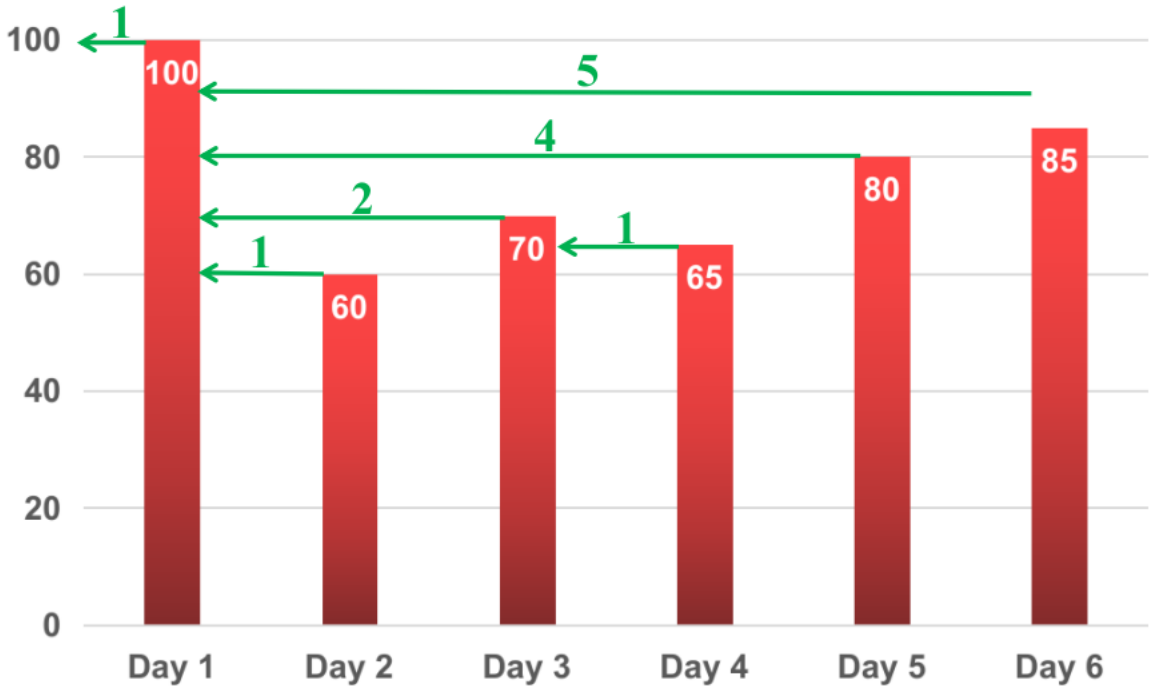
   - capacity argument: size tối đa của queue và chỉ được truyền vào khi đi cùng "CreateQueue" method
   - value argument: là gía trị sẽ được enqueue vào queue và chỉ được truyền vào khi đi cùng "Enqueue" method

```
# Example
queue1 = {
    'Queue' : [],
    'Capacity' : None,
    'IsEmptyQueue' : None,
    'IsFullQueue' : None,
    'RetValue' : None,
    'FirstValue' : None
}

simQueue(queue=queue1, method_name="CreateQueue", capacity=5)

# queue1.enqueue(1)
simQueue(queue=queue1, method_name="Enqueue", value=1)

# queue1.enqueue(2)
simQueue(queue=queue1, method_name="Enqueue", value=2)

# print(queue1.IsFullQueue())
simQueue(queue=queue1, method_name="IsFullQueue")
print(queue1["IsFullQueue"])
>> False

# print(queue1.front())
simQueue(queue=queue1, method_name="GetFirstValue")
print(queue1["FirstValue"])
>> 1

# print(queue1.dequeue())
simQueue(queue=queue1, method_name="Dequeue")
print(queue1["RetValue"])
>> 1

# print(queue1.front())
simQueue(queue=queue1, method_name="GetFirstValue")
print(queue1["FirstValue"])
>> 2

# print(queue1.dequeue())
simQueue(queue=queue1, method_name="Dequeue")
print(queue1["RetValue"])
>> 2

# print(queue1.IsEmptyQueue())
simQueue(queue=queue1, method_name="IsEmptyQueue")
print(queue1["IsEmptyQueue"])
>> True
```

Code Listing 4: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

4. **Stock Span Problem**: Cho trước một list giá của một loại cổ phiếu trong N ngày. Viết function đi tìm stock span cho mỗi ngày trong list trên.

   - **Input:** list chứa giá cổ phiếu của N ngày (một ngày là một giá)
   - **Output:** trả về một list là stock span của từng ngày
   - **Khuyến khích các bạn sử dụng stack, và nếu có thể thì sử dụng stack của các bạn đã thực hiện ở bài 2**
   - **Stock Span:** của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có giá cổ phiếu <= giá cổ phiếu của ngày hiện tại
   - **Ví dụ:**
     - Input: [100, 60, 70, 65, 80, 85]
     - Output: [1, 1, 2, 1, 4, 5]




```
prices = [100, 60, 70, 65, 80, 85]
stock_span(prices)
>> [1, 1, 2, 1, 4, 5]

prices = [100, 80, 60, 70, 60, 75, 85]
stock_span(prices)
>> [1, 1, 1, 2, 1, 4, 6]

prices = [31, 100, 14, 20, 50, 22]
stock_span(prices)
>> [1, 2, 1, 2, 3, 1]
```

Code Listing 5: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

5. **Cho trước một range với N số nguyên dương (từ 1 đến N). Viết function biến đổi các số thập phân trong range sang dạng binary string và trả về một dictionary với key là số thập phân và value là binary dạng string**

   - **Input** N là số nguyên dương thể hiện range từ 1 đến N
   - **Output**: một dictionary với key là số thập phân và value là binary dạng string tương ứng
   - **Khuyến khích các bạn sử dụng queue, và nếu có thể thì sử dụng queue của các bạn đã thực hiện ở bài 3**
   - Các bạn có thể tìm hiểu thêm binary number system tại link

```
# Examples
generateBinaryString(8)
>> {1: '1',
2: '10',
3: '11',
4: '100',
5: '101',
6: '110',
7: '111',
8: '1000'}

generateBinaryString(16)
>> {1: '1',
 2: '10',
 3: '11',
 4: '100',
 5: '101',
 6: '110',
 7: '111',
 8: '1000',
 9: '1001',
 10: '1010',
 11: '1011',
 12: '1100',
 13: '1101',
 14: '1110',
 15: '1111',
 16: '10000'}
```

Code Listing 6: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

6. **(OPTIONAL) Cho một chuỗi DNA, được tạo bởi các nucleotides 'A', 'C', 'G' và 'T'. Khi nghiên cứu về DNA cần phải nhận dạng các chuỗi được lặp lại trong DNA. Bài tóan cho một chuỗi s là DNA, hãy trả về các chuỗi có 10 chữ cái được lặp lại nhiều hơn một lần dựa vào s (Thứ tự trả về không quan trọng).**

   - **Input:** s là chuỗi DNA dạng string chỉ có các ký tự 'A', 'C', 'G' và 'T'
   - **Outnput:** list các chuỗi có 10 ký tự được lặp lại hơn một lần
   - **Hint**: các bạn có thể sử dụng deque

```
s = "TTTTTTTTTTTTTTTTTTT"
findRepeatedDnaSequences(s)
>> ['TTTTTTTTTT']

s = "TTTTTGGGGGTTTTTGGGGGGTTTTTAAACCC"
findRepeatedDnaSequences(s)
>> ['TTTTTGGGGG', 'GGGGGTTTTT']

s = "TTTTTGGGGGTTTTTGGGGG"
findRepeatedDnaSequences(s)
>> ['TTTTTGGGGG']
```

Code Listing 7: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ