LAB 3 : Secondary Data Structures

[CO3]

Instructions for students:

- Complete the following methods.
- You may use Java / Python to complete the tasks.
- DO NOT CREATE a separate folder for each task.
- If you are using **JAVA**, then follow the **Java Template**.
- If you are using **PYTHON**, then follow the **Python Template**.

NOTE:

- **YOU CANNOT USE ANY OTHER DATA STRUCTURE OTHER THAN ARRAY AND LINKED LIST.**
- **YOUR CODE SHOULD WORK FOR ANY VALID INPUTS.**

Python List, Negative indexing and append() is STRICTLY prohibited

TOTAL MARKS: 5*7=35

HashTable

1. Searching in hashtable :

Complete the **`hashFunction()`** and **`searchHashtable()`** methods. **Do not** change the given code; implement only the required methods. Creating and Inserting into a hash table using forward chaining is already done inside of the class. Do not initialize any other instance variable other than the given ones.

a. **search_hashtable()** → this instance method takes a **key value pair** in the parameter then uses the string key to search for the string in the instance variable **ht**. If the **`string s`** is found, this method returns **'Found'**, else returns **'Not Found'**.
[Do not implement sequential search, implement the hash based search]

b. **hashFunction()** → this instance method takes a key-value pair (string, int), calculates the hashed index on key and returns the index. This hash function takes consecutive two letters of the key string, concatenates their ascii values into an integer and sums all the concatenated integers. Then it finds out the modulus of the summation (**think for yourself with which number should we mod the summation**) as the hashed index.
For instance, for a string 'Mortis', the consecutive two letters are Mo, rt, is.
The concatenated integer for
Mo is 77111 (Ascii of M is 77, o is 111);
rt is 114116 (Ascii of r is 114, t is 116);
'is' is 105115 (Ascii of i is 105, s is 115).
The summation is = 77111+114116+105115
Mod the summation with ___ and return the answer as the hashed index.
[fill in the ___ gap with your knowledge]

Note: For an odd length string, add the letter 'N' at the end of it. Thus, 'Morti' becomes 'MortiN' and the consecutive two letters are Mo, rt, iN.

2. Hashtable with Forward Chaining:

Write the **insert()** function and **hash_function()** of the HashTable class which uses an array to store a key-value pair, where the key is a string representing a fruit, and the value is an int representing its respective price.

hash_function(key)

Takes a key which is a string, and calculates its length. If length is even, the sum of the ascii values of the even characters is calculated, otherwise, the sum of the ascii values of odd characters is calculated.
Finally, it returns the sum modded by length of the array

If we call __hash_function(“apple”):
As len(“apple”) is odd, characters in odd indexes (p, l) are taken:

sum = ord(p) + ord(l) = 112 + 108 = 220

If the Hashtable object is initialized with length 5, then:

sum % length of Hashtable array = 220 % 5 = 0
So, the function returns 0

insert(key, value)

Creates a node that contains a tuple which stores the key-value pair as (key, value). Finds index by passing key to hash_function(). If there is no collision, the node is placed in the index.

If there is a collision, forward chaining is applied. Here, the chain should be arranged in **descending order**, i.e, if the node being inserted has the highest value (price), it will be the head of the chain. Otherwise, you should iterate the chain and insert it in the appropriate position.

[If the same key is given twice for insertion, update the value of that particular key. No need to create new Node]

3. Deletion from hashtable :

You are given the hash function, **h(key) = (key + 3) % 6** for a hash-table of length 6. In this hashing, forward chaining is used for resolving conflict and a 6-length array of singly linked lists is used as the hash-table. In the singly linked list, each node has a next pointer, an Integer key and a string value, for example: (4 (key) , “Rafi” (value)). The hash-table stores this key-value pair.

Implement a function **remove(hashTable, key)** that takes a key and a hash-table as parameters. The function removes the key-value pair from the aforementioned hashtable if such a key-value pair (whose key matches the key passed as argument) exists in the hashtable.

Consider,Node class and hashTable are given to you. You just need to complete the remove(hashTable, key) function.

```
class Node:
  def __init__(self, key, value, next=None):
      self.key, self.value, self.next = key, value, next
```

Sample Input and Output:
Some Key-value pairs:
(34, “Abid”) , (4, “Rafi”), (6, “Karim”), (3, “Chitra”), (22, “Nilu”)

HashTable is given in the following:
0: (3, “Chitra”)
1: (22, “Nilu”) → (4, “Rafi”) → (34, “Abid”)
2: None
3: (6, “Karim”)
4: None
5: None

remove(hashTable, key=4) returns the changed hashTable where (4, “Rafi”) is removed.

New HashTable Output:
0: (3, “Chitra”)
1: (22, “Nilu”) → (34, “Abid”)
2: None
3: (6, “Karim”)
4: None
5: None

remove(hashTable, key=9) returns the same given hashTable since 9 doesn’t exist in the table.

Stack

YOUR CODE SHOULD WORK FOR ANY VALID SAMPLES

[FOR ALL THE FOLLOWING STACK TASKS YOUR METHODS/TASKS SHOULD BE OUTSIDE OF THE STACK CLASS]

4. Diamond Count:

Faisal is working in a diamond mine, trying to extract the highest number of diamonds "< >". A "<" followed by ">" forms one new diamond. He must exclude all the sand particles found (denoted by ".") and **unpaired** "<", ">" in this process to extract new diamonds.

You need to solve the above problem using **Stack class**. You cannot use other methods than pop(), peek(), push(), isEmpty() methods of Stack.

Complete the function **count_diamond** which will take an object of Stack and a string and then return the number of diamonds that can be extracted using the mentioned process.

Sample Input String	Sample Output
.<...<<..>>....>....>>>.	3
<<<..<.....<<<<....>	1
<..><.<..>>	3

Explanation:
For an input ".<...<<..>>....>....>>>." three diamonds are formed. The first is taken from <..> resulting ".<...<>....>....>>>." The second diamond <> is then removed, leaving ".<.....>....>>>." The third diamond <> is then removed, leaving at the end ".....>>>." without the possibility of extracting new diamonds. Hence, 3 diamonds have been extracted.

5. Tower of Blocks:

A kid is trying to make a tower using blocks with numbers or characters written on them. He puts a block on the top of another to make the tower. As for removing the blocks, he removes the topmost block if needed. He is afraid that removing other blocks from the middle in one go will result in the collapse of his tower.

Now his friend wants your nth block from the top of your tower. He is willing to give him the nth block from the top, preserving the relative position of others.

Following his tower build process, solve this scenario using **Stack** class. You can only use the stack methods.
Hints: You can create a temporary object of Stack class.

In the following examples, consider the rightmost element to be the topmost element of the stack. **(Rightmost item is the top)**

Sample Tower	Sample Output	Explanation
Stack: 4 19 23 17 5 n: 2	Stack: 4 19 23 5	In this tower, the 2nd block from the top is the block with 17. After removing the block the stack looks like 4 19 23 5
Stack: 73 85 15 41 n: 3	Stack: 73 15 41	In this tower, the 3rd block from the top is the block with 85. After removing the block the stack looks like 73 15 41

6. Stack Reverse:

Consider that a Stack class has been implemented containing the push(element), pop(), peek() and isEmpty() functions.

Complete the function **conditional_reverse()** which will take an object of Stack class as an input that contains some integer values. **The function returns a new stack** which will contain the values in reverse order from the given stack **except the consecutive one's.** You cannot use any other data structure except Stack.

Note: The Stack class implements a singly linked list-based Stack hence overflow is not possible. The pop() and peek() functions return None in case of the underflow.

In the following example, consider the rightmost element to be the topmost element of the stack.

Sample Input Stack (Rightmost is the top)	Output Reversed Stack (Rightmost is the top)	Explanation
Stack: 10, 10, 20, 20, 30, 10, 50 Top = 50	New Stack: 50, 10, 30, 20, 10 Top = 10	Consecutive 20 and 10 are not present in the output reversed stack

Queue

7. Customer Service Call Center:

You are developing a **Customer Service Call Center System** where customers call for assistance and are placed in a queue based on a **first-come-first-served** basis. However, there are two types of customers:

i. **Regular Customers**
ii. **VIP Customers** (who should be served before regular customers but in the order they arrive among VIPs).

Your task is to implement a **call handling system** using a queue that follows these rules:

i. **VIP customers get priority** over regular customers.
ii. Among VIP customers, they are served in the order they are called (FIFO).
iii. Among regular customers, they are also served in the order they are called (FIFO).
iv. When a call is handled, remove the customer from the queue and process them.

Implement the **`CallQueue`** class using a queue data structure that supports the following methods:

- **enqueue_call(customer_id, is_vip) →**
  - ➢ Adds a customer to the queue. If is_vip = True, they should be placed in the VIP queue; otherwise, in the regular queue.
- **dequeue_call() →**
  - ➢ Processes the next call. It should first check for VIP customers, and if none exist, then process regular customers.
- **display_queue() →**
  - ➢ Prints the current order of the queue (VIPs first, followed by regular customers).

[NOTE: A LinkedListQueue class is already given in the driver code, you just need to utilize it in your CallQueue class to complete the tasks]