CSCI-SHU 210 Data Structures

Homework Assignment 5 Linked List

Singly linked lists and doubly linked lists are widely useful data structures. In this homework, we will be studying:

- Single linked list
    - Singly linked, **single ended**, no sentinel.
- Double linked list
    - Doubly linked, double ended, with sentinel.

Guidelines & general hints:
1. Please keep other methods unchanged when you write new methods.
2. Please make sure the test code provided runs without any modification to the test code.
3. Drawing how references change can help a lot when you code with linked lists.
4. Watch out for special case! They can hurt your assignment grade if you forget to handle.

I. Singly Linked List:

Question 1 (get item in a Singly Linked List):

1. __getitem__(self, k)
    - Return the element (not the node) stored at kth indexed node.
    - We will only test with valid k. (0 ~ len(self) – 1)

Question 2 (list reversal of a Singly Linked List):

2. list_reverse(self)
    - Reverses the self Singly linked list.
    - Runtime requirement: O(n)
    - Space requirement: in-place, so O(1) memory allowed

Question 3: search for prior reference

3. search_node_pair(self, target)

   When traversing a singly linked list, it is usually very useful to keep a reference to the previous node. The search_node_pair function asks user to return a pair of node objects: the first one being the previous node, and the second one the targeted node.
    - target is the **element** stored in the targeted node
    - Runtime requirement in the worst case: O(n)
    - Return the two **nodes** if found
        - Return None, None, if not found
        - Return None, node if the target is the first node
    - Must be recursive

Question 4: Favorite Items List with Move-to-front Heuristics

A favorite items list records webpages and how many clicks on each webpage in a Singly-ended Singly Linked List. The idea is to keep the most frequently accessed items towards the head of the list, using a move-to-front heuristic. (Textbook 7.6.2)

We implement a new class FavSinglyLinkedList, using the Singly Linked List as a data storage. The constructor has been written for you. Elements stored inside each node is a python list [url, count], where url is a string of your item, and count is how many times you clicked on this item. You should not change the constructor.

When accessing an item, we first look for this url in the existing Singly Linked List. If it does not exist, then we add it to the **tail** of the Singly Linked List, and set the count = 1. If it does exist, we shift to the front and increase the count by 1. The general idea is that we use the most recently accessed item as a heuristic for the most frequently used item.

Implement the function access(self, url).

Requirement:

- Methods for previous questions can be re-used when solving later questions
- The count must be increased
- You can write other methods in the class FavSinglyLinkedList if you need

Question 5: Top-k item generator

This question writes another method in the FavSinglyLinkedList of Question 4.

In order to find the top-k favorite item, we iterate through the singly linked list for k times. Write a generator that yield the urls of the topk items one by one (from most frequently visited).

Notice, the FavSinglyLinkedList you created should not be altered.

II． Doubly Linked List:

Question 6: Doubly Linked List Shuffling

Given 2n elements stored inside a Doubly Linked List self (n>1), implement a method shuffle(self) which changes the order of the elements.

A shuffle is a permutation where a list L is cut into two lists L1 and L2, where L1 is the first half of L and L2 is the second half of L. Then these two lists are merged into one by taking the first element in L1, then the last element in L2, then the second element in L1, then the second last element in L1, and so on.

Requirement:

- In-place operation, i.e. you should not use any other data structures (python lists, etc.). Only O(1) temporary variables are allowed. You should return self.
- Do not modify the element stored in any nodes
- It is the original nodes that you should permute. In other words, do not delete the original node and insert a new clone of it.