CS2040C Semester 1 2018/2019

Data Structures and Algorithms

Tutorial 03 - Linked List, Stack, Queue, Deque

For Week 5 (Week Starting 10 September 2018)

Document is last modified on: September 5, 2018

1 Introduction and Objective

For this tutorial, you will need to review `https://visualgo.net/en/list?slide=1` (to last slide 9-5) about Linked List and all its variations as they will be the focus of today's tutorial.

2 Tutorial 03 Questions

Linked List, Mini Experiment

Q1). Please use the 'Exploration Mode' of `https://visualgo.net/en/list` to complete the following table (some cells are already filled as illustration). You can use use the mode selector at the top to change between (Singly) Linked List (LL), Stack, Queue, Doubly Linked List (DLL), and Deque mode. You can use 'Create' menu to create input array of various types. Stack, Queue and Deque are ADTs with a subset of list features. Hence, they can be implemented using a Linked List.

Mode → ↓ Action	Singly Linked List	Stack	Queue	Doubly Linked List	Deque
search(any-v)	$O(N)$	not allowed	not allowed	$O(N)$	not allowed
peek-front()	$O(1)$				
peek-back()					$O(1)$
insert(0, new-v)				$O(1)$	
insert(N, new-v)					$O(1)$
insert(i, new-v), $i \in$[1..N-1]		not allowed			
remove(0)			not allowed		
remove(N-1)		not allowed			
remove(i), $i \in$[1..N-2]				$O(N)$	

You will need to fully understand the individual strengths and weaknesses of each Linked List variations discussed in class in order to be able to complete this mini experiment properly. You can assume that all Linked List implementations have head and tail pointers, have next pointers, and only for DLL and Deque: have prev pointers.

Points to be discussed in class:

- The comparison of these 9 possible actions in 5 modes of Linked List
- The fact that the specialized ADTs: stack, queue, and deque take advantage of the fastest $O(1)$ operations of the underlying data structure:
  - stack uses `insert(0, new-v)/remove(0)` of Singly Linked list for `push(new-v)/pop()` respectively,
  - queue uses `insert(N, new-v)/remove(0)` of Singly Linked List for `enqueue(new-v)/dequeue()` respectively,
  - deque uses `insert(0, new-v)/insert(N, new-v)/remove(0)/remove(N-1)` of Doubly Linked List for `push-front(new-v)/push-back(new-v)/pop-front()/pop-back()`, respectively.

Mode → ↓ Action	Singly Linked List	Stack	Queue	Doubly Linked List	Deque
search(any-v)	$O(N)$	not allowed	not allowed	$O(N)$	not allowed
peek-front()	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
peek-back()	$O(1)$	not allowed	$O(1)$	$O(1)$	$O(1)$
insert(0, new-v)	$O(1)$	$O(1)$	not allowed	$O(1)$	$O(1)$
insert(N, new-v)	$O(1)$	not allowed	$O(1)$	$O(1)$	$O(1)$
insert(i, new-v), $i \in$[1..N-1]	$O(N)$	not allowed	not allowed	$O(N)$	not allowed
remove(0)	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
remove(N-1)	$O(N)$	not allowed	not allowed	$O(1)$	$O(1)$
remove(i), $i \in$[1..N-2]	$O(N)$	not allowed	not allowed	$O(N)$	not allowed

Q2). Assuming that we have a List ADT that is implemented using a Singly Linked List with both head and tail pointers. Show how to implement an additional operation `reverseList()` that takes in the current list of N items $\{a_0, a_1, ..., a_{N-2}, a_{N-1}\}$ and reverse it so that we have the reverse content $\{a_{N-1}, a_{N-2}, ..., a_1, a_0\}$. What is the time complexity of your implementation? Can you do this faster than $O(N)$?

We can use either one of these possible answers. Note to tutor: If a student answers one version in class, the tutor will challenge him/her to think of the other way:

Option 1: We iterate through the N items of the list in $O(N)$ time and insert them into the head of a **new** list (initially empty), $O(1)$ each time. Finally we return the new list, which is the reverse of the original list(, and erasing the original list). This is $O(N)$ overall.

Option 2: We can also use recursion to go deep from the head element to the tail element in $O(N)$. Then, as the recursion unwinds, we can reverse the link from $u \rightarrow v$ into $v \rightarrow u$. Lastly, we swap the head and tail pointers.

It is not possible to do better than $O(N)$ as there are N items that will need to change their (next) pointers during the `reverseList()` operation.

Online Judge Exercises

Q3) How would you solve the Online Judge exercises mentioned in `https://visualgo.net/en/list?slide=9-5` using a Linked List (or other data structures). What methods of a Linked List would be used? What is the total time complexity of your algorithm?

- Kattis - Backspace (`https://open.kattis.com/problems/backspace`)
- UVA - Broken Keyboard (`https://uva.onlinejudge.org/external/119/11988.pdf`)
- Kattis - Integer Lists (`https://open.kattis.com/problems/integerlists`)

For Backspace, it is just a simple simulation using any resize-able data structure that supports efficient insertion at the back and deletion at the back, like `vector` or for the sake of this Tutorial topic: `deque` (essentially a Doubly Linked List). We can actually use `stack` too but we have to reverse the final output at the end. Note that we may not be able to use `queue` as it does not have efficient deletion from the back. For Broken Keyboard, we need (either Singly or Doubly) Linked List as we need its

ability to do insertion at the middle of the list in $O(1)$, (for this, we have to keep the reference pointer instead of iterating from head at all times). For Integer Lists, the key observation is that it is possible

to iterate a (doubly-linked) list in reversed order without actually reversing it. To do so, we can start from the tail and repeatedly advance towards the head using the pointer to the previous element. Also, notice that reversing a list twice in succession is effectively not reversing the list at all. Hence, we can store whether the list is in 'forward state' or 'reversed state' using a boolean. When a list is to be reversed, we toggle between the two states. This can be achieved in O(1) time. In the 'forward state', we will drop the first element of the list. In the 'reversed state', we will drop the last element of the list. Both drop operations are O(1) time as well.

Problem Set 2

We will end the tutorial with discussion of PS2.

This part is left to tutor after seeing the latest situation of PS2 as of the day of tutorial.