CS2040C Semester 2 2018/2019

Data Structures and Algorithms

Tutorial 03 - Linked List, Stack, Queue, Deque

For Week 05

Document is last modified on: February 4, 2019

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. Note: Due to yet another lost week 04 due to CNY, tutorial groups on Tuesday will be mostly lost (at first) with this tutorial content.

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.

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.

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 $O(N)$?

Stack, Queue, or Deque

Q3). In the Lisp programming language, each of the four basic arithmetic operators appears before an arbitrary number of operands, which are separated by spaces. The resulting expressions are enclosed in parentheses. There is only one operator in a pair of parentheses. The operators behave as follows:

- (+ a b c) returns the sum of all the operands, and (+) returns 0.

- (- a b c) returns a - b - c - and (- a) returns 0 - a,
  i.e. the minus operator must have at least one operand.

- (* a b c) returns the product of all the operands, and (*) returns 1.

- (/ a b c) returns a / b / c / and (/ a) returns 1 / a, using double division.
  The divide operator must have at least one operand.

You can form larger arithmetic expressions by combining these basic expressions using a fully parenthesized prefix notation, e.g. the following is a valid Lisp expression: (+ (- 6) (* 2 3 4)). The expression is evaluated successively as follows: (+ -6.0 (* 2.0 3.0 4.0)), then we have (+ -6.0 24.0), and we finally have 18.0.

Design and implement an algorithm that uses up to 2 stacks to evaluate a legal Lisp expression composed of the four basic operators, integer operands, and parentheses. The expression is well formed (i.e. no syntax error), there will always be a space between 2 tokens, and we will not divide by zero. Output the result, which will be a double-precision floating point number.

Q4). How would you solve the Online Judge exercises mentioned in https://visualgo.net/en/list?slide=9-5?

Problem Set 2

We will end the tutorial with discussion of PS2.