CS2040C Semester 2 2021/2022

Data Structures and Algorithms

Tutorial+Lab 03
Linked List, Stack, Queue, Deque

For Week 05

Document is last modified on: February 3, 2022

1 Introduction and Objective

For this tutorial, you will need to (re-)review `https://visualgo.net/en/list?slide=1` (to last slide 9-6) about List ADT and all its variations (compact array (or vector-)based List - from tut01.pdf, SLL, Stack, Queue, DLL, Deque) 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), or Deque mode. You can use 'Create' menu to create input list 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()	$O(1)$	O(1) not allowed	O(1) O(1)	O(1) O(1)	O(1) $O(1)$
peek-back()	O(1)				
insert(0, new-v)	O(1) O(1) O(N) / if you have iterator its o(1)	O(1) not allowed	not allowed O(1) not allowed	$O(1)$	O(1)
insert(N, new-v)				O(1) O(N)	$O(1)$ not allowed
insert(i, new-v), $i \in$[1..N-1]		not allowed			
remove(0)	O(1) O(N) O(N)	O(1)	not allowed O(1) not allowed not allowed	O(1) O(1)	O(1) O(1) not allowed
remove(N-1)		not allowed		$O(N)$	
remove(i), $i \in$[1..N-2]		not allowed			

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 two additional operation:

1. `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 $O(N)$?

   O(N) best case for recursive and iterative, can't go any faster than that even if sorted.

2. `sortList()` that takes in the current list of N items and sort them so that $a_0 \leq a_1 \leq ... \leq a_{N-2} \leq a_{N-1}$. What is the time complexity of your implementation? Can you do this faster than $O(N \log N)$?

   2) The best time complexity is O(N log N) because this is the fastest sort for best case scenario of merge sort (not quick sort due to O(n^2) issue)

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.

Hands-on 3

TA will run the second half of this session with a few to do list:

- Do a short debrief of PS2 (after grading),
- Very quick review of C++ STL std::list (std::forward_list is optional), std::stack, std::queue, and/or std::deque,
- Do a sample speed run of VisuAlgo online quiz that are applicable so far, e.g., `https://visualgo.net/training?diff=Medium&n=7&tl=3&module=list`.
- Finally, live solve another chosen Kattis problem involving list ADT.

Problem Set 3

We will end the tutorial with **algorithmic** discussion of PS3.