Homework 7 : Algorithms and Data Structures

Digdarshan Kunwar

April 2019

Problem 7.1

Stacks Queues

(a) Go inside the folder Stack
It is implemented in teststack.cpp and Stack.h
$: make all
$: make run

(b) Go inside the folder Queue
It is implemented in testqueue.cpp and Queue.h
$: make all
$: make run

Problem 7.2

Linked Lists Rooted Trees

!! NOTE !!
Implementation of different algorithms are in different folders.

(a) The pseudo code for an in-situ algorithm that reverses a linked list of n elements in Θ(n) is given below:

Algorithm 1 Pseudo code for reversing the the Linked List

```
procedure REVERSE_LIST(LINKED LIST A)                    ▷ Procedure Declaration
    if A.first == NULL then                              ▷ Check if list is empty
        return

    Node *prev=NULL:
    Node *next=NULL:
    Node *current=A.first:

    while current!=NULL do        ▷ Reverse the pointer in nodes till the last elements
        next=current->next
        current->next=prev
        prev=current
        current=next

    A.first=prev
```

This algorithm reverse the list of the elements in a Linked List. The function reverses the pointers within the Nodes of the list from the front to the end.And finally sets the start pointer to the end element.

Here the function uses three pointers for the Node and moves throughout the list from the first to the last of the list until the current pointer is NULL.
So if there are n elements we iterate through each element that is Θ(n) changing the direction of the pointer of the Node.
As for every iteration the memory required is constant (that are just the three pointers) it is therefore a in-situ algorithm.

(b) Go inside the folder toLinkedList.
It is implemented within tolist.cpp, BST.h,LinkedList.h
$: make all
$: make run

Asymptotic time complexity:
As we have in the algorithm a recursive approach to traverse through the elements.As the we would traverse through the BTS such that it send every time a new element at the last of the list.So every insertion in the list would take a constant time to add at the back thus for n elements taking each element inoder would take we have a linear asymptotic behaviour.
Here the time complexity of the algorithm is :$\Theta(n)$.

(c) Go inside the folder toBST.
It is implemented within toBTS.cpp, BST.h,LinkedList.h
$: make all
$: make run

Asymptotic time complexity:
Here in this case we take the element from the list and insert it to the Binary Search Tree so for inserting an element in the tree we have a complexity of O(lg(n)) or O(h) where h is the height of the tree.So for the n elements in the list we insert it n times. Therefore we have the time complexity of O(nlg(n)).