CS211 - Data Structures and Algorithms

Lab 06

Instructor: Dr. Sharaf Hussain
E-mail: shhussain@uit.edu

Semester: Fall, 2021

1 Objective

The purpose of this lab session is to implement **Linked List** data structure.

2 Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

3 How to Submit

- Submit lab work in a single .py file on Google Classroom. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW04.py)
- Submit home work in a single .py file on Google Classroom. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_HW04.py)

4 Implementing Linked List

Implementing **node** in a linked list..

```
class ListNode :
def __init__( self, data ) :
self.data = data
self.next = None

```

Traversing a linked list..

```
def traversal( head ):
curNode = head
while curNode is not None :
print curNode.data
curNode = curNode.next

```

Searching a linked list..

```
def unorderedSearch( head, target ):
curNode = head
while curNode is not None and curNode.data != target :
curNode= curNode.next
return curNode is not None

```

Prepending a node to the linked list..

```
# Given the head pointer, prepend an item to an unsorted linked list.
newNode = ListNode( item )
newNode.next = head
head = newNode
```

Removing a **node** from a linked list..

```
# Given the head reference, remove a target from a linked list.
predNode = None
curNode = head
while curNode is not None and curNode.data != target :
predNode = curNode
curNode = curNode.next
if curNode is not None :
if curNode is head :
head = curNode.next
else :
predNode.next = curNode.next
```

Implementing the Bag ADT using a singly linked list.

```
# Implements the Bag ADT using a singly linked list.
class Bag :
# Constructs an empty bag.
def __init__( self ):
self._head = None
self._size = 0
# Returns the number of items in the bag.
def __len__( self ):
return self._size
# Determines if an item is contained in the bag.
def __contains__( self, target ):
curNode = self._head
while curNode is not None and curNode.item != target :
curNode = curNode.next
return curNode is not None
# Adds a new item to the bag.
def add( self, item ):
newNode = _BagListNode( item )
newNode.next = self._head
self._head = newNode
self._size += 1
# Removes an instance of the item from the bag.
def remove( self, item ):
predNode = None
curNode = self._head
while curNode is not None and curNode.item != item :
predNode = curNode
curNode = curNode.next
# The item has to be in the bag to remove it.
assert curNode is not None, "The item must be in the bag."
# Unlink the node and return the item.
self._size -= 1
if curNode is head :
self._head = curNode.next
else :
predNode.next = curNode.next
return curNode.item
# Returns an iterator for traversing the list of items.
def __iter__( self ):
return _BagIterator( self._head )
# Defines a private storage class for creating list nodes.
class _BagListNode( object ):
def __init__( self, item ) :
self.item = item
self.next = None
```

5 Exercises

Task 1:

The **removeAll(head)** function, which accepts a head reference to a singly linked list, unlinks and remove every node individually from the list.

Task 2:

The splitInHalf(head) function, which accepts a head reference to a singly linked list, splits the list in half and returns the head reference to the head node of the second half of the list. If the original list contains a single node, None should be returned.

Task 3:

Implement a new version of the Set ADT using an unsorted linked list.

Task 4:

Implement a new version of the Set ADT using a sorted linked list.

Task 5:

Evaluate your new implementations to determine the worst case run time of each operation.

Task 6:

Compare the run times of your new versions of the Set ADT to those from the previous implementations.