Recitation Problem: Linked List

CS 2003 – Data Structures

Spring 2023

Overview

In this recitation problem, students will review the following topics:

1. Singly Linked List
2. Doubly Linked List

Problem 1 Create an algorithm that when given the head of a singly linked list it will delete every other node.

```
public void deleteEveryOther(Node head) {
    if (head == null){
        return;
    }

    Node current = head;

    while (current != null && current.next != null) {

        current.next = current.next.next;
        current = current.next;
    }
}
```

Problem 2 Given a singly length list, create an algorithm that will return the length of that list.

```
public int listLength(Node head) {
    int length = 0;
    Node current = head;

    while (current != null) {
        length++;
```

```
            current = current.next;
        }

        return length;
    }
```

Problem 3 Given a singly linked list and a value, create a search algorithm that will return true if the value exists in the list and return false if it does not exist in the list.

```
    public boolean search(Node head, int val) {
        Node current = head;

        while (current != null) {
            if (current.data == val) {
                return true;
            }
            current = current.next;
        }

        return false;
    }
```

Problem 4 Create a recursive algorithm that will remove all nodes with value K from a singly linked list.**Your creating a new list**

```
    public Node removeElements(Node head, int val) {
        if (head == null) {
            return null;
        }

        head.next = removeElements(head.next, val);

        if (head.data == val) {
            return head.next;
        } else {
            return head;
        }
    }
```

Problem 5 Given a singly linked list, create an algorithm that will reverse that linked list and return its new head.

```
public Node reverseLinkedList(Node head) {
    if (head == null || head.next == null) {
        return head;
    }

    Node newHead = reverseLinkedList(head.next);

    head.next.next = head;
    head.next = null;

    return newHead;
}
```

Problem 6 Create a recursive algorithm that will Merge two sorted singly linked lists into one sorted list. Assume that the heads of two sorted list are passed into your method.(*Use the Node class above*)

```
public Node mergeTwoLists(Node list1, Node list2) {
    if (list1 == null){
        return list2;
    }

    if (list2 == null){
        return list1;
    }

    if (list1.data <= list2.data) {
        list1.next = mergeTwoLists(list1.next, list2);
        return list1;
    } else {
        list2.next = mergeTwoLists(list1, list2.next);
        return list2;
    }
}
```

Problem 7 Given two Unsorted Doubly Linked Lists, merge them together into one Linked List.

```
public Node mergeTwoUnsortedLists(Node list1, Node list2) {
    if (list1 == null){
        return list2;
    }
```

```
    if (list2 == null){
        return list1;
    }

    Node current = list1;

    while (current.next != null) {
        current = current.next;
    }

    current.next = list2;
    list2.prev = current;

    return list1;
}
```

Problem 8 A helper Node class is provided to you. For the following list, Create an insert algorithm that inserts into a sorted doubly linked list.

```
public void insert(Node x) {
    if (head == null) {
        head = x;
        return;
    }

    Node current = head;

    if (x.val <= head.val) {
        x.next = head;
        head.prev = x;
        head = x;
        return;
    }

    while (current.next != null && current.next.val < x.val) {
        current = current.next;
    }

    x.next = current.next;
    x.prev = current;

    if (current.next != null) {
        current.next.prev = x;
    }
```

```
        current.next = x;
    }
```

Problem 9 Given the following code, trace the linked list and determine the output.

26

Problem 10 Given a sorted Doubly Linked List, remove all duplicates from that list.

```
public Node removeDuplicatesFromDLL(Node head) {
    if (head == null) return null;

    Node current = head;

    while (current != null && current.next != null) {
        if (current.val == current.next.val) {
            Node duplicate = current.next;
            current.next = duplicate.next;
            if (duplicate.next != null) {
                duplicate.next.prev = current;
            }
        } else {
            current = current.next;
        }
    }

    return head;
}
```

Problem 11 A helper Node class is provided to you.For the following list, Create an insert algorithm that only inserts at the front of the doubly linked list.

```
public void insert(Node x) {
    if (head == null) {
        head = x;
        return;
    }

    x.next = head;
    head.prev = x;
    x.prev = null;
    head = x;
```

```
    }
```

Problem 12 A helper Node class is provided to you.For the following list, Create an insert algorithm that only inserts at the end of the doubly linked list.

```
public void insert(Node x) {
    if (head == null) {
        head = x;
        x.prev = null;
        x.next = null;
        return;
    }

    Node current = head;
    while (current.next != null) {
        current = current.next;
    }

    current.next = x;
    x.prev = current;
    x.next = null;
}
```