Introduction to Computer Science II

CS112-2012S-20

Linked Lists IV

David Galles

Department of Computer Science
University of San Francisco
Doubly Linked List

- Deleting from (and inserting into!) a linked list can be challenging because you need to find the node before the node you are looking for.
- Once you’ve found the node, it’s too late – can’t follow pointers backwards to get to the previous node.
  - ... unless you keep a pointer to the previous node in the list, too!
public class DLLNode
{
    Object data;
    DLLNode next;
    DLLNode previous;
    public DLLNode (Object data)
    {
        this.data = data;
        this.next = null;
        this.previous = null;
    }
    public DLLNode (Object data, DLLNode next)
    {
        this.data = data;
        this.next = next;
        this.previous = null;
    }
    public DLLNode (Object data, DLLNode next, DLLNode previous)  
    {
        this.data = data;
        this.next = next;
        this.previous = previous;
    }
}
20-2: Doubly Linked List

head

- Object
  - Next
  - Prev

- Object
  - Next
  - Prev

- Object
  - Next
  - Prev

- Object
  - Next
  - Prev
Many of the functions for Double-Linked Lists are similar (or even the same) as Singly-Linked versions.
public class LinkedList
{
    private DLLNode head;

    public LinkedList()
    {
        head = null;
    }

    public boolean find(Object o)
    {
        ...
    }
}
public class LinkedList
{
    private DLLNode head;

    public LinkedList()
    {
        head = null;
    }

    public boolean find(Object o)
    {
        DLLNode tmp = head;
        while (tmp != null)
        {
            if (tmp.data.equals(o))
            {
                return true;
            }
        }
        return false;
    }
}
public class LinkedList {
    private DLLNode head;

    public LinkedList() {
        head = null;
    }

    public Object removeAt(int index) {
        ...
    }
}
20-7: Double Linked List removeAt

- Find the element that you want to remove
  - Don’t need to be “one-off”, can find the actual element

- Rearrange pointers
  - Removing from middle of list
  - Removing from end
20-8: Doubly Linked List

head

A → B → C → D
20-9: Doubly Linked List

A → B → C → D

head → A → B → C → D → temp
20-10: Doubly Linked List

```
temp.previous.next = temp.next
```
20-11: Doubly Linked List

```
temp.previous.next = temp.next

temp.next.previous = temp.previous
```
20-12: Doubly Linked List

head

\[
\text{temp.previous.next} = \text{temp.next} \\
\text{temp.next.previous} = \text{temp.previous}
\]
public class LinkedList
{
    private DLLNode head;

    public Object removeAt(int index)
    {
        DLLNode tmp = head;
        for (int i = 0; i < index; i++)
        {
            tmp = tmp.next;
        }
        // Removing from the middle of list
        // (won't work for removing 1st/last element
        tmp.prev.next = tmp.next;
        tmp.next.prev = tmp.prev;
        return tmp.data;
    }
}

20-14: Doubly Linked List
20-15: Doubly Linked List

```
head = temp.next;
```
20-16: Doubly Linked List

head = temp.next;
head.previous = null
20-17: **Doubly Linked List**

```
head = temp.next;
head.previous = null
```
Doubly Linked List

head

A <-> B <-> C <-> D

temp
temp.previous.next = null
20-20: Doubly Linked List

temp.previous.next = null
public Object removeAt(int index)
{
    DLLNode tmp = head;
    for (int i = 0; i < index; i++)
    {
        tmp = tmp.next;
    }
    if (tmp.prev != null)
    {
        tmp.prev.next = tmp.next;
    }
    else
    {
        head = tmp.next;
    }
    if (tmp.next != null)
    {
        tmp.next.prev = tmp.prev;
    }
    return tmp.data;
}
Back to single linked lists

How could we reverse a linked list?

Function reverse
  - Takes as input parameter a list to reverse
  - Returns a reversed version of the list
  - OK to destroy original list
• Go through how to reverse on board, with diagrams
```java
LinkedListNode reverse(LinkedListNode l) {
    LinkedListNode newFront = null;
    LinkedListNode tmp = null;
    while (l != null) {
        tmp = l.next;
        l.next = newFront;
        newFront = l;
        l = tmp;
    }
    return newFront;
}
```
Recursive Reverse

- Let’s look at doing reverse recursively
- This one is a little tricky ...
**Recursive Reverse**

- **Base case:**
  - Easy List to reverse

- **Recursive Case:**
  - Make list smaller, by removing first element
  - Recursively reverse smaller list
  - Add first element back into correct location
What is an easy list to reverse?

There are actually 2 lists that are easy to reverse ...
private StringLinkedListNode reverse(LinkedListNode l) {
    if (l == null || l.next == null) {
        return l;
    }
    ...
private StringListNode reverse(ListNode l) {
    if (l == null || l.next == null) {
        return l;
    }
    ListNode reversed = reverse(l.next);

    // What does reversed look like now?
    // How should we modify it?
}
private StringLinkedListNode reverse(LinkedListNode l) {
    if (l == null || l.next == null) {
        return l;
    }
    LinkedListNode reversed = reverse(l.next);
   .appendToEnd(reversed, l);
    return reversed;
}
Recursive Reverse

void appendToEnd(LinkedListNode list, LinkedListNode elem) {
    while (list.next != null) {
        list = list.next;
    }
    elem.next = null;
    list.next = elem;
}

private StringLinkedListNode reverse(LinkedListNode l) {
    if (l == null || l.next == null) {
        return l;
    }
    LinkedListNode reversed = reverse(l.next);
    appendToEnd(reversed, l);
    return reversed;
}
Previous reverse was *very* inefficient.

Why?
Previous reverse was very inefficient.
- Need to traverse entire list to append an element to the end
- We will do this traversal over and over again (examples)
Problem is that we need to traverse the entire list to get to the last element.

If only there was some way of getting to that last element without traversing the list ...
20-35: **Recursive Reverse**

- Problem is that we need to traverse the entire list to get to the last element.
- If only there was some way of getting to that last element without traversing the list ...
  - Right before we reverse the list, the first element in the smaller list is the last element in the reversed list!
Doubly Linked List
Doubly Linked List

list

A → B → C → D

second
Doubly Linked List

```
list
```

```
A -- B -- C -- D
```

second
20-39: Doubly Linked List

list

A \rightarrow B \rightarrow C \rightarrow D

second

reversed
20-40: Doubly Linked List
private LinkedListNode reverse(LinkedListNode list)
{
    if (list == null || list.next == null)
    {
        return list;
    }
    LinkedListNode second = list.next;
    list.next = null;
    StringLinkedListNode reversed = reverse(second);
    second.next = list;
    return reversed;
}
public class LinkedList
{
    private LinkedListNode head;

    public void reverse()
    {
        head = reverse(head);
    }
}

20-42: Recursive Reverse
20-43: Hands-On

- Work on lab/project