



2007-02-13

Lecture #5

(Material on Test 1)

Today's Outline

Linked Lists in C++

Doubly Linked Lists in C++

Multi-Lists in C++

Stack & Queue Linked Lists

Linked Lists / Doubly Linked Lists / Multi-Lists

The textbook has good coverage on these topics

Chapter 3 – Singly Linked Lists

Chapter 3.6 (pg 113) Discusses Doubly Linked Lists

Chapter 3.6 (pg 119) Multi-Linked Lists

Review SORTED Linked Lists

We showed you last week how to ...

- 1. Insert**
- 2. Remove**

We talked about using NULL to represent the end of the list

Alternatives:

- A Circular Linked Lists ... which has the end point back to the head**

How do we test for end of the list in a Circular Linked List?

SORTED Doubly Linked Lists

We showed you last week how to ...

- 1. Insert**
- 2. Remove**

**Now the 1st node points to the last
And the last node points to the first**

So it's circular

How do we test for end of the doubly linked circular list?

Alternatives:

- A “sentinel” head pointer**

SORTED Doubly & Singly Linked List – Insert & Remove

What are their algorithm complexities?

All are $O(n)$

Why?

Notice: What's the Big O if it wasn't sorted?

Insert = $O(k)$

Remove = $O(n)$

Singly & Doubly Linked Lists Complexity

What is the Big O of the IsEmpty function?

$O(k)$

Why?

What is the Big O of the Exists function?

$O(n)$

Why?

The print list function? $O(n)$ also

Singly & Doubly Linked lists in C++

Pointers make writing a linked list easy to understand

**But the pointers also cause for headaches when it comes to
catching memory leaks**

When do you grab new memory on the heap?

- **Insert**
- **(not in the constructor ... cause the list is still empty),
unless of course you have a “sentinel” node**

When do you delete memory off the heap?

- **Remove**
- **Destructor**

Singly & Doubly Linked Lists

The deletes can become tricky ... like in our CD organizer homework 1

For each record / CD ... how many times did we call “new” or how many times did we grab memory from the heap?

The answer is 2

- 1. for the data (music cd)**
- 2. for the linked list node**

That means that for each Remove for each list we better delete twice and the order is going to be important too!!!

**must delete the data first (music cd)
then delete the node
(if you do it the other way around then you've lost the data)**

Sorted Multi-Linked Lists in the book

In the book they describe multi-linked lists this way ...

Singly Linked List Multi-List

- Having a Linked list node that contains MULTIPLE next pointers, one for each “sorting option”

Example:

```
struct MultiListNode{  
    Student * data ;  
    MultiListNode * nextByName ;  
    MultiListNode * nextByAge ;  
}
```

Note: this is different than how we are doing the homework...

Sorted Multi-Linked Lists in the book

**The crucial concept in a Multi-List is ...
the data is stored ONCE ... even though you have multiple lists**

How do we do an insert? What's the Big O?

A delete? What's the Big O?

IsEmpty? What's the Big O?

Exists? What's the Big O?

Print? What's the Big O?

**Note: This book version works great for 2 lists of the same type ... both
Singly Linked Lists that terminate with NULL**

Sorted Multi-Linked Lists in the book

If the book showed a Doubly Linked version it would be like this ...

Example:

```
struct MultiListNode{  
    Student * data ;  
  
    MultiListNode * nextByName ;  
    MultiListNode * prevByName ;  
  
    MultiListNode * nextByAge ;  
    MultiListNode * prevByAge ;  
}
```

Note: this is still different than how we are doing the homework...

Sorted Multi-Linked Lists in the book

**The crucial concept again in a Multi-List is ...
the data is stored ONCE ... even though you have multiple lists**

How do we do an insert? What's the Big O?

A delete? What's the Big O?

IsEmpty? What's the Big O?

Exists? What's the Big O?

Print? What's the Big O?

**Note: This book version works great for 2 lists of the same type ... both
Doubly Linked Circular Lists**

Sorted Multi-List ... for our Homework

**For the homework we're using a modified version of a Multi-List
for our CD Organizer data structure**

The concept is the same in that we only create 1 copy of each Music CD

But it's slightly different than the book because

1. we're using 2 types of Linked Lists

- **Singly Linked Lists that terminates with null**
- **Sorts by Artist Name (Title breaks Ties)**
- **Doubly Linked Circular List**
- **Sorts by Album Year ASC (Artist Name breaks Ties)**

2. we're creating 2 separate nodes for each cd

(instead of creating 1 node with all the pointers inside it)

Sorted Multi-Lists for homework

What this means for us is that ...

Add CD ...

(1)grab memory on the heap for the 1 Music Cd

(2)grab memory on the heap for the Singly Linked List Node

- node contains: ptr to data, ptr to next

(3)grab memory on the heap for the Doubly Linked List Node

- node contains: ptr to data, ptr to next, ptr to prev

(4)Both nodes will point their data ptrs to the SAME data in #1 above

What is the Big O? $O(2n)$ or $O(n)$

Sorted Multi-Lists for homework

What this means for us is that ...

Remove CD ...

(1) find the node in the singly linked list & delete the NODE from heap

(2) find the node in the doubly linked list & delete the DATA and the NODE from the heap

What is the Big O? $O(2n)$ or $O(n)$

IsEmpty

**(1) Notice that the 2 lists will ALWAYS have the same data in them ...
just sorted in a different order ... so we just have to check 1 !!!**

What is the Big O? $O(k)$

Sorted Multi-Lists for homework

Exists

(1) This is also interesting because since the linked lists always both have the SAME data ... we only need to search 1 !!!!

What is the Big O? $O(n)$

Print by Artist Name

(1) This is simple ... we need to use the Singly Linked List to traverse once through and print all the nodes

What is the Big O? $O(n)$

Sorted Multi-lists in homework

Print by Album Year ASC

(1) Also easy because we just print that doubly linked list once straight forward and we're done

What is the Big O? $O(n)$

Print by Album Year DESC

(2) A bit trickier, but not much ... remember it's a doubly linked list ... so we just have to traverse backwards in order to get DESC

What is the Big O? $O(n)$

Sorted Multi-Lists for our homework

Destructor

- When the cd organizer is trashed ... we need to clean up the memory in the destructor ... this means that both lists could have nodes in them ... the good news is that the lists all have the same data in it ... but the bad news is that each lists has it's separate node on the heap too
- That means we have to traverse both lists, for example
 - traverse the singly linked lists deleting each node from the heap
 - traverse the doubly linked lists deleting each data & node

What is the Big O? $O(2n)$ or $O(n)$

Sorted Multi-Lists in homework vs Mutli-Lists in book Advantages & Disadvantages

- **The book version only has 1 node ... so the remove / destructor is simpler ... when it comes to deleting the data**
- **The homework version is perhaps more readable because it identifies each lists separately and you won't get confused while writing your code as to which sorting you're working with**
- **The homework version uses up slightly more memory because we actually have 2 pointers to the same data ... where as in the book we'd only have 1 pointer to the data**
 - **NOTE: This extra memory for 1 pointer is typically irrelevant**
 - **Example:**

If each data object was 500bytes and there is 100 objects

$500 * 100 = 50000$ bytes for data

$100 * 4 = 400$ extra bytes for pointers

50000bytes vs 400bytes???

Sorted Multi-Lists in Homework

Be careful ... memory leaks will be a big points loser in this (and all future homeworks) ... so make sure if you call “new” you call “delete”

Also be careful you don't delete the same memory twice ... that will lead to run-time errors and Segmentation faults

Also make sure you're only storing the data / music cd ONCE in your cd organizer ... if you're storing it twice ... that's first of all a big waste of space (which defeats the purpose of a multi-list) and second you'll lose lots of points

If it helps ... get out a pencil & paper and draw pictures of exactly what you're doing in your program (with pointers, next, prev, stack, heap)

Linked Lists for other things

So we all know what a linked list is ...

we know that there are many different types

- singly linked list that terminates in null**
- singly linked circular list**
- singly linked circular list with sentinel node**
- doubly linked circular list**
- doubly linked circular list with sentinel node**
- multi-lists (in book & homework)**

The interesting thing is that we can use these lists for many different things (not just for a list as you may expect) ...

Other implementations of Linked Lists

- 1.) Stacks (Chapter 4 of your book)**
 - i. push & pop (FILO – first in last out)**
- 2.) Queue (Chapter 5 of your book)**
 - i. enqueue & dequeue (FIFO – first in first out) ... like printers**
- 3.) C++ Strings**
 - i. we had just an array of characters, but we could have easily created a linked list where the data type was a character (notice this would probably be overkill though)**

Linked List implementation of a Stack

Specs of a stack linked list? TopPtr, push, pop functions

How would a push & pop function be written for a linked list?

What is the Big O of a push? $O(k)$

What is the Big O of a pop? $O(k)$

Thus ... notice that you can't just say that the insert on a Linked List is $O(n)$ or $O(k)$... etc. ...

You need to know how the list is organized ... if it's sorted, the insert is $O(n)$... if it's not sorted (like a stack) ... then the $O(k)$

Linked List implementation of a Queue

Specs of a stack linked list? FrontPtr, EndPtr, enqueue, dequeue funcs

How would a enqueue & dequeu function be written for a linked list?

What is the Big O of a enqueue? $O(k)$

What is the Big O of a dequeue? $O(k)$