CPE 212 - Fundamentals of Software Engineering

Sorted Lists

Outline

- Sorted List: Array Based
- Binary Search
- Sorted List: Linked List
- Coding Examples

Array based List

A common issue of the old array based list is the list does not maintain ”order” of the list.

Reasons: Speed! Inserts are O(1) and deletes are O(1) at best.

However: Deletes can be up to O(N), and with this, the lack of maintaining order can cause further issues with your data.

Our solution is to go with a “Sorted List”

Sorted Array Functions

- MakeEmpty() -> void
- IsFull() const -> bool
- GetLength() const -> unsigned
- GetItem(bool&) const -> Type
- Insert(Type) -> void
- Delete(Type) -> bool
- NextItem() -> Type
- ResetIterator() -> void
- AtEnd() const -> bool

Sorted Array Functions

- MakeEmpty() -> void
- IsFull() const -> bool
- GetLength() const -> unsigned
- GetItem(bool&) const -> Type
- Insert(Type) -> void
- Delete(Type) -> bool

- NextItem() -> Type
- ResetIterator() -> void
- AtEnd() const -> bool

These will be completely modified because of the “sorted” nature of the list

Sorted Array Functions

- MakeEmpty() -> void
- IsFull() const -> bool
- GetLength() const -> unsigned
- GetItem(bool&) const -> Type
- Insert(Type) -> void
- Delete(Type) -> bool
- NextItem() -> Type
- ResetIterator() -> void
- AtEnd() const -> bool

This will be become “faster” or changed because of the sorted implementation

Insert

Goals: Maintain a list that is sorted from “Lowest” to ”Highest”

With our operator overloads and templates, we can make this completely generic, and rely on our type to implement the “< > == !=“ operators.

As items get inserted, the items after that get shifted down in the list.

Algorithm
Insert(Type Item) -> bool

```
Start at first position
While index < Size():
  If(item == currentItem): Item exists in list already, return
     (false if return type is bool)
  Else If(item > currentItem): Continue searching, index++
  Else: break;
For i in Size() to index (going down):
     Swap(data[i], data[i-1])
Set data[index] = item;
Return (true if bool retval)
```

Algorithm
Delete(Type Item) -> bool

We will remove in more or less the same way as we inserted, except we will shift the new items up the list instead of down.

```
Found=false
Start at first position
While(not at end && not found):
  If currentitem > item: continue to next item
  If currentItem < item: return false (item does not exist)
  If currentItem == item: Found = true. Break from loop
For indexes(currentIndex->End()):
     Swap(data[currentIndex], data[currentIndex +1])
Decrement Length
Return (Found)
```

Order Analysis

How fast are the following algorithms?

Insert

Delete

GetItem(Item &)

Order Analysis

How fast are the following algorithms?

Insert - O(1)

Delete - O(N) worst and O(1) at best

GetItem(Item &) - O(N)

Can we make any of these faster?

Binary Search

Binary Search

Binary Search can return the item, an Index of the item, or whatever the preferred implementation requires.

```
Found = false, first = 0, last = Size()
While(index < Size() && !Found)
  midpoint = (first + last) + 2
  If(data[midpoint] < Item):
      last = midpoint - 1,
      if(first >=   last)
            return (Item not found)
  If(data[midpoint] > item):
      first = midpoint + 1;
      if(first >= last)
            return (Item not found)
  If(data[index] == item):
      found = true
Return appropriate information (either them Item or the index:
midpoint)
```

Question: What is the speed of this algorithm?

Order Analysis

How fast are the following algorithms?

Insert - O(1)

Delete - O(N) worst and O(1) at best

GetItem(Item &) - O(N)

Using Binary Search:

GetItem(Item &)

Order Analysis

How fast are the following algorithms?

Insert - O(1)

Delete - O(N) worst and O(1) at best

GetItem(Item &) - O(N)

Using Binary Search:

GetItem(Item &) - O(logn)

Linked List Functions

- MakeEmpty() -> void
- IsFull() const -> bool
- GetLength() const -> unsigned
- GetItem(bool&) const -> Type
- Insert(Type) -> void
- Delete(Type) -> bool
- NextItem() -> Type
- ResetIterator() -> void
- AtEnd() const -> bool

Linked List: Insert

We will use a ”Crawler” similar to how we have done for “Find items” in the past.

Note: These examples are with a forward list.

Algorithm (LL)

Insert(Type Item) -> bool

```
CurrentItem = begin, previousItem = null,
moreToSearch = (Length() > 0)
While(moreToSearch)
  If(currentItem == item):
    Return false (Cannot insert this item)
  If(currentItem < item):
    set prevItem = currentItem,
    currentITem = currentItem->next,
    moreToSearch = (currentItem != null)
  If(currentItem > item):
    location for item found!
    moreToSearch = false
Now insert the item at our currentItem location:
  Create a New Node
  newNode-> next = currentItem
  previousItem->next = currentItem
```

Compare the Data Structures

Algorithm	**Array List**	**Linked List**
MakeEmpty() -> void		
IsFull() -> bool		
GetLength() -> size		
ResetList() -> void		
GetNextItem() -> Item		
GetItem(Item) -> index / Item		
PutItem(Item) -> Bool		
DeleteItem(Item) -> Bool		