Southern New Hampshire University
Joshua Wozny
joshua.wozny@snhu.edu
January 13, 2023
CS-300 Week 3: LinkedList

REFLECTION

This week we implemented a linked list data structure in C++. This is the first of several data structures that we will implement in C++ to better understand the advantages and disadvantages of each. Each data structure we will be studying can be used to store data for retrieval later using different techniques.

The linked list uses nodes that store a point of data as well as a pointer the next record in the list - hence the name linked list. Linked lists can grow dynamically and make insertions and deletions more performant than arrays and vectors that must move each record when there is an insertion or deletion because the linkd list need only change the record that a node is pointing.

Developing the code was straightforward, although I hit a few pitfalls along the way. For a few of the methods I would use the current bid instead of the next bid, or vice versa. Debugging code is always challenging, but I find that I learn a lot through the process.

I also created a method to check to see if the key already was used and to prevent another from being added. I also refactored the code to make use of classes to better encapsulate the various objects within the applications, and allow for easier modifications as we progress over the next several weeks. This will enable me to more easily reuse the code to develop the remaining data structures that we'll be studying.

PSEUDOCODE

LinkedList

- Definitions:
    - Node is a structure that contains a data point (in our case a Bid) and a pointer to the next Node
    - head is the first Node in the Linked List
    - tail is the last Node in the Linked List
    - size holds the current number of Nodes in the list
- LinkedList contructor
    - set head and tail equal to null
- LinkedList destructor

    - start at the head, loop over each node, detach from list then delete
- Append(Bid bid)
    - check to see if this bid has a duplicate key, if it does then inform the user and do nothing
    - Create new node
    - if there is nothing at the head, new node becomes the head and the tail
    - otherwise make current tail node point to the new node and tail becomes the new node
    - increment size
- Prepend(Bid bid)
    - check to see if this bid has a duplicate key, if it does then inform the user and do nothing
    - Create new node
    - if there is already something at the head, new node points to current head as its next node
    - head now becomes the new node
    - increment size
- PrintList()
    - start at the head, while loop over each node
    - output current bidID, title, amount and fund
- Remove(string bidId)
    - while loop over each node looking for a match
        - if matching node is the head, make head point to the next node in the list, decrement size
        - otherwise, while loop over each node looking for a match
        - if the next node bidID is equal to the bidID passed
            - hold onto the next node temporarily
            - make current node point beyond the next node
            - now free up memory held by temp
            - decrement size
- Search(string bidId)
    - start at the head of the list, while loop over each node looking for a match
        - if the current node bidID is equal to the bidID passed, return the Node
    - otherwise return an empty Bid
- Size() returns the current value of size