Computer Science 136

Data Structures

Lecture #11 (October 6, 2021)

1. Announcements:
   (a) Lab 1 returned. Lab 2 in. Lab 3 out.
   (b) Questions?
2. Recall: The `Node<T>` class: two logical fields, a `value` and `next`, a link to another `Node<T>`. It is `public`, so users outside the `structure` package could use it for whatever purposes they desire.
3. Recall: The `SinglyLinkedList<T>` class, our first `structure`-specific object.
   (a) Many methods keep track of a "finger" that directs the focus of the method at hand.
   (b) Think about recursive approaches: many require helper methods.
      i. Be prepared to write `add(i,v)` or `remove(v)` recursively.
         A. Recursive variants often need helper functions to smooth over the *edge cases* that have typically caused us to write head-of-list-checking `if` statements. In the future, we may be able to eliminate both.
         B. You must be very careful to make sure your method works for (0) empty lists or (1) lists with one element. Only then will it work for larger lists.
4. Doubly-linked lists.
   (a) Every nodes has two links—one to previous node, the other to the next node.
   (b) Insertion and deletion are a bit more complex and must handle special cases (empty list, or list with one element, or element at one end of list or other).
   (c) But, typically, we keep two pointers in the list: a pointer to the head, and one to the tail.
   (d) Adding a bit more space overhead increases the speed. Obviously, operations at the tail of the list will work faster for `DoublyLinkedLists`.
   (e) If you're insecure about big-O notation and analysis, lists and vectors are a good source of practice material.
5. Lab this week: Potential improvement in speed and beauty: Using a dummy node.
   (a) Some of the complexity of handling the base case in linked lists can be avoided by having head (and tail) reference a *dummy node*.
   (b) The dummy node does not hold data, but is a *sentinel* for an end of the list. It avoids always having to check for a null reference.
   (c) Consider the code for removing a node from the middle of a doubly linked list.
   (d) How complex is it to write a recursive solution for remove from a doubly linked list with dummy nodes?
6. Make sure you read about: `CircularList`, singly linked, but has quick access to tail.
7. Next: Sorting.

Notes: