Kevin Ziemba
001324999
ziembak

CoE SI4 : Lab 2

Date Started: Jan 26,2016 Half Finished: Jan 28,2016 Completed Date: Jan 29,2016

Code Descriptions

On mergeTo(). Method begins by creating nodes to sequentially advance through *this* and *that*, and a third node *h2* to temporarily hold next nodes of *that* when inserts are done. O(n1+n2) repetitions of the while loop are performed by comparing nodes in *this* and *that* , and carefully advancing 1 or 2 nodes at a time, depending on the case. Comparisons of *this* start at header, and of *that* start after the header, in case an insertion before the first word of *this* is needed.

Insertions are done by holding the next value of a node p2 of *that*, temporarily after the header of *h2* to avoid losing access to *p2.next* . p2 is advanced to the next value of *that*. The node held by h2 is made to link to *p1.next*. The node p1 is then made to link to the node held in h2. Once the size of *this* is incremented by 1, the insertion is complete.

The case that a word needs to be inserted before the end of *this*, and inserted at the end of *this* are separated due to the last node linking to *null*, which cannot be easily compared with. If the word exists in both, the next node is of *that* is then compared, and if the word held by p2 does not yet need to be inserted, the next node of *this* is then compared.

The function ends by setting size of *that* to 0, and linking the header to *null*, emptying *that*.

The function mergeTo() runs in $\Theta(n)$ time, with n corresponding to the size of the two LinkedLists (n1 + n2 is still a constant,so it runs in n time). With each addition in size in *this* or *that*, the number of operations increases by a constant amount. This is because the position of nodes *p1* and p2 in each list is not reset after an insertion. There is also memory allocated for 3 new nodes.

On insert(). Method begins by creating a node *p* to sequentially advance through *this*. Each node's string is compared to *newword*, exiting when *newword* is already found in *this* or is inserted.

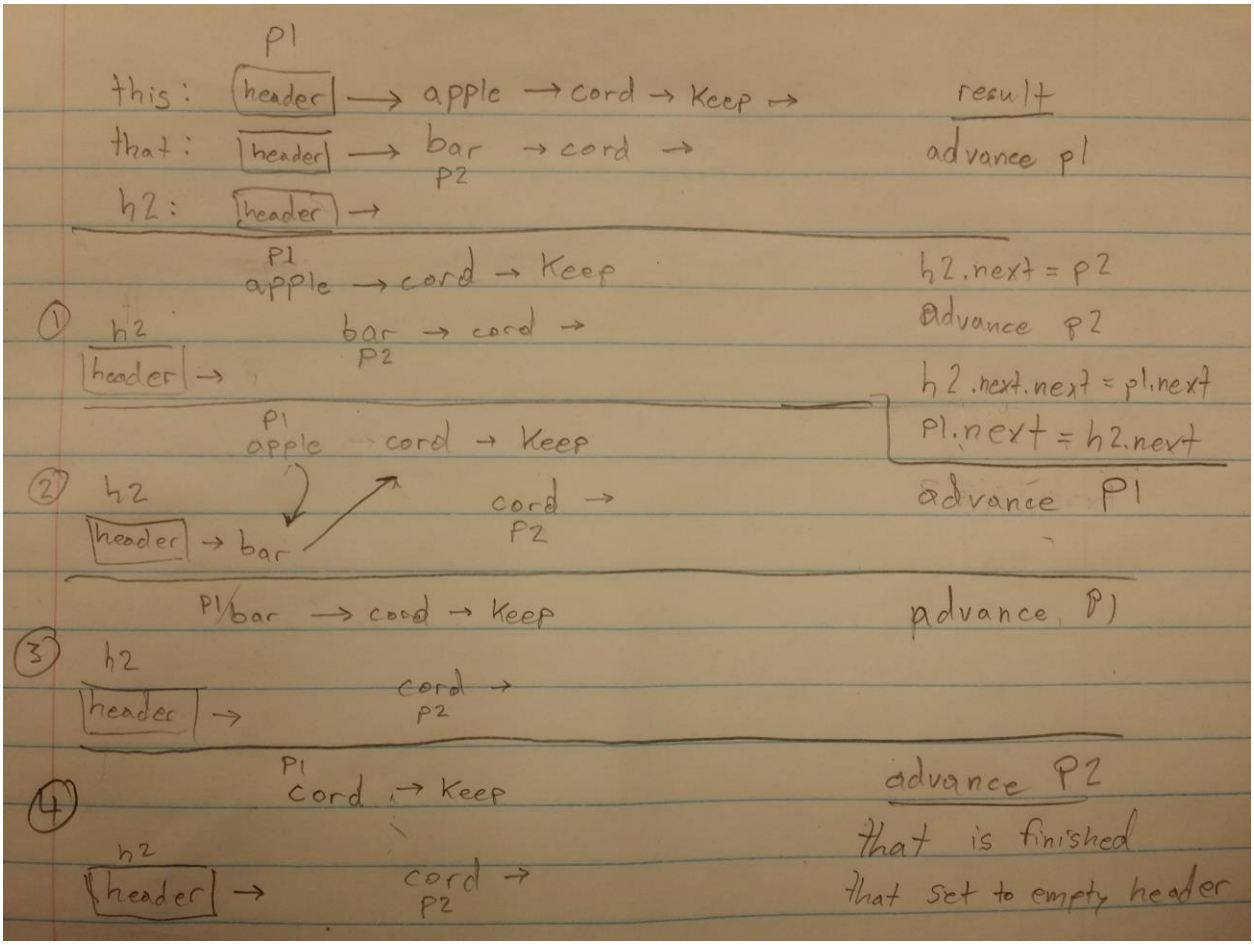
String *newword* is to be inserted when it belongs after the current node *p* , and before the node *p.next* .Insertions are done by creating a new node *h* to hold the string *newword*. The new node *h* is then linked to *p.next*. Node *p* is then linked to node *h*. Once the size of *this* is incremented by 1, the insertion is complete.

The function insert() runs in Θ (n) time, where n is the size of *this* LinkedList. Each node's string is compared against in the worst case. There is memory allocated for 1 new node if no insertion is made, or 2 new nodes if an insertions is made.

On the second constructor. Constructor begins by initializing the size to 0, and the dummy header to hold the empty string "", and to link to *null*. The header has string "" instead of *null* so that the header string can be compared with, in case a word is to be inserted before the first word containing node. The constructor then calls method insert() once for each word in array *arrayOfWords*. Method insert() also handles keeping WordLinkedList *this* in sorted order, and updating the size.

The constructor runs in Θ(n^2) time, where n is the size of array *arrayOfWords*. Each node's string is compared against in the worst case, and each insertion makes the list longer. As the constructor goes on, the number of operations done is increased. There is memory allocated for 1*n nodes in the best case, or 2*n nodes in the worst case.

Sketch of mergeTo()



Sketch of second Constructor