CS 367 Homework 2

Alex Pizzuto

---

1 Question One

We are starting with a `LinkedList` class implemented as a doubly-linked chain of nodes with a dummy header node. The class uses the class `DblListnode` and has the following fields:

```
private DblListnode<E> items;
private int numItems;
```

We have been given the following code to reverse the order of the objects in a specified sublist:

```
public void reverse(int pos1, int pos2){
  // If pos1 == pos2, reversing a single list element does nothing
  // If pos1 > pos2, reversing an expty sublist does nothing
  if (pos1 >= pos2)
     return;

  // We swap the 1st and last items in the sublist,
  //  then recursively reverse the remaining sublist
  // We stop when the remaining sublist has size 0 or 1

  // Swap list items at pos1 and pos2
  E temp = remove(pos2);
  add(pos2, get(pos1));
  remove(pos1);
  add(pos1, temp);

  // Now recursively reverse remainer of sublist (if any)
  // The remaining sublist is from pos1+1 to pos2-1
  reverse(pos1+1, pos2-1);
}
```

And we will now complete a second version of this method that directly changes the chain of nodes by unlinking the nodes to be swapped and re-links them into the chain in the appropriate way using only DblListnode methods.

```
/**
     * Reverses the order of the items from pos1 to pos2
     * NOTE: The Double Linked List will have a header node that is null.
     *
     * @param pos1 The start position of the reversal group
     * @param pos2 The end position of the reversal group
     */
    public void reverse(int pos1, int pos2){
```

```
if(pos1 < 0 || pos2 < 0 || pos1 > numItems - 1 || pos2> numItems - 1) {
   throw new IndexOutOfBoundsException();
}
   if (pos1 >= pos2 ) {
      return; //return if input doesn't make sense
   }

   //Access the node at pos1
   DblListnode<E> pos1node = items.getNext();
   // items is the header node so this is the first node in the list
    for (int k = 0; k < pos1; k++) {
         pos1node = pos1node.getNext();
    }

    //Access the node at pos2
    DblListnode<E> pos2node = pos1node.getNext();
    //To get the second node, start from the first
    for (int k = pos1 + 1; k < pos2; k++ ) {
         pos2node = pos2node.getNext();
    }

    //If pos1 is the beginning of the list, it still has a next and prev
    //If pos2 is at the end of the list, it only has a prev
    DblListnode<E> node1Prev = pos1node.getPrev();
    DblListnode<E> node1Next = pos1node.getNext();
    DblListnode<E> node2Prev = pos2node.getPrev();

    //treat pos2 at the end of the list as a special case
    if (pos2 == numItems - 1) {
         if(pos2-pos1 == 1) {
            //deal with nodes next to each other
            pos1node.setNext(null);
            pos1node.setPrev(pos2node);
            node1Prev.setNext(pos2node);
            pos2node.setNext(pos1node);
            pos2node.setPrev(node1Prev);
         } else {
            pos1node.setNext(null);
            pos1node.setPrev(node2Prev);
            node1Prev.setNext(pos2node);
            node1Next.setPrev(pos2node);
            pos2node.setNext(node1Next);
            pos2node.setPrev(node1Prev);
            node2Prev.setNext(pos1node);
         }
    }

    //Deal with all other cases
    else {
         DblListnode<E> node2Next = pos2node.getNext();
         if(pos2-pos1 == 1) {
            //deal with nodes next to each other
            pos1node.setNext(node2Next);
            pos1node.setPrev(pos2node);
```

```
                node1Prev.setNext(pos2node);
                pos2node.setNext(pos1node);
                pos2node.setPrev(node1Prev);
                node2Next.setPrev(pos1node);
            } else {
                pos1node.setNext(node2Next);
                pos1node.setPrev(node2Prev);
                pos2node.setNext(node1Next);
                pos2node.setPrev(node1Prev);
                node1Next.setPrev(pos2node);
                node1Prev.setNext(pos2node);
                node2Next.setPrev(pos1node);
                node2Prev.setNext(pos1node);
            }
        }
        reverse(pos1+1, pos2-1);

    }
```

2 Question Two

We can find the worst-case time complexity for the method from the previous question in terms of `N`, the list size.

All of the test statements for positions in the beginning are clearly `O(1)`. When we obtain pos1node and pos2node, we could potentially have to traverse the entire list, so this step is `O(N)`. From there, all we are doing is constant time operations. However, that is only one of the recurisve iterations. Worst case, we could have `pos1` reference the first node in the list and `pos2` reference the last node in the list. In this case, we must perform `N/2` iterations. Although each successive iteration requires 2 less operations to obtain the original nodes, we can still bound the time complexity by `N` iterations of `O(N)` operations, thus it is quadratic in time, or $O(N^2)$.