Homeweok2

Mingren Shen

mshen32@wisc.edu

Q1

Assume that a `LinkedList` class has been implemented as a doubly-linked chain of nodes **with** a (dummy) header node. The `LinkedList` class has the following fields:

```
private DblListnode<E> items;  // pointer to the header node`

private int numItems;
```

LinkedList uses the class `DblListnode` that includes the public methods `getData`, `setData`, `getPrev`, `setPrev`, `getNext`, and `setNext`.

Below is an implementation of a new method, `reverse`, for the `LinkedList` class. This method reverses the order the objects in a specified sublist. Consider this list: `[1, 2, 3, 4, 5]`. The call `reverse(1,3)` reverses the sublist `[2, 3, 4]`, returning `[1, 4, 3, 2, 5]`. Note that the call `reverse(0,size()-1)` reverses the entire list.

The `reverse` method operates in a recursive manner. Given a list `L`, it first swaps the very leftmost and rightmost items. It then recursively reverses the remaining list (excluding the two items that have been swapped.) Given `[1, 2, 3, 4, 5]` it swaps `1` and `5` to obtain `[5, 2, 3, 4, 1]`. Then the sublist `[2, 3, 4]` is reversed in a recursive call. `4` and `2` are swapped to obtain `[4, 3, 2]`. The remaining sublist, `[3]`, is trivially reversed, and our final list, after all recursive calls return, is: `[5, 4, 3, 2, 1]`

```
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);
  }
```

Recall that if a position is invalid, the `get` and `remove` methods throw an `IndexOutOfBoundsException`.

For this homework, **complete a second version of reverse** that functions that same as the code above. However, your version must **directly change the chain of nodes by unlinking the nodes to be swapped and re-linking them into the chain in the appropriate way**. To receive full credit, your `reverse` method must use only `DblListnode` methods (and cannot use any `LinkedList` methods).

Be sure that your code works for all cases such as when the list is empty, has just one item, has just two items, or has more than two items. Also consider when the positions of items to be swapped are next to each other, or one of the items to be swapped is at the front or the end of the chain.

Answer for Q1

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

            // Now pos1 must be smaller than pos2, then
            // check whether pos1 and pos2 in the allowed range of index (0 ~
numItems-1)
            if (pos1 >= numItems || pos1 < 0 || pos2 >= numItems || pos2 < 0 )
                return;

            // edge cases
            // (1) empty lists
            if (numItems == 0) {
                return;
            }
            // (2) single nodes list
            //     reversing a single list element does nothing
            if ( numItems == 1) {
                return;
```

```
            }

            // (3) double nodes list
            //     only need to adjust two nodes
            if ( numItems == 2) {
                // use two pointer points to two nodes
                DblListnode<E> tmp1 = items.getNext();
                DblListnode<E> tmp2 = tmp1.getNext();
                // from dummy head node, adjust the two nodes
                items.setNext( tmp2 );
                tmp2.setNext( tmp1 );
                tmp2.setPrev(items);
                tmp1.setPrev( tmp2 );
                tmp1.setNext( null );
                return;
            }

            // (3) more than 2 items list
            // move through the list and adjust the pointers accordingly
            // we will process the last node of sublist individually to avoid
null
            // problems of next nodes

            // pointer points to the starting nodes of the sublist to be
reversed
            // also the last nodes of the sublist after reversing
            DblListnode<E> sublistHeadNode = items;

            // Internal counter
            int i = 0;
            // get head node of sublist
            while( i <= pos1 )
            {
                sublistHeadNode = sublistHeadNode.getNext();
                i++;
            }

            // reverse sublist
            int j = pos1;
            int k = pos2;
            // the ending nodes of the lists that do not needed to be reversed
            DblListnode<E> origialListHeadTail = sublistHeadNode.getPrev();
            // pointer to current node that needed to be reversed
            // now is the nodes after headed nodes of sublist
            DblListnode<E> reverseIter = sublistHeadNode.getNext();
```

```
            // reverse head nodes of sublist to the nodes after it before
reversing
            sublistHeadNode.setPrev( reverseIter );

            while ( j < k - 1) // reverseIter points to next nodes of head
nodes of sublist
            {
                // prev and next nodes of current nodes in sublist
                DblListnode<E> tmp_prev = reverseIter.getPrev();
                DblListnode<E> tmp_next = reverseIter.getNext();
                // reverse pointers of current nodes
                reverseIter.setPrev( tmp_next );
                reverseIter.setNext( tmp_prev );
                // move current nodes to the next nodes of sublist
                reverseIter = tmp_next;
                // increase counter of the number of nodes processed
                j++;
            }
            // reverseIter points to last nodes of sublist now and nothing done
on it now
            // now linking the ending node of first part of main list to
sublist
            origialListHeadTail.setNext( reverseIter );
            // now linking the beginning node of second part of main list to
sublist
            sublistHeadNode.setNext( reverseIter.getNext() );
            // reverse pointers of last nodes of sublist,
            // also the beginning node of the reversed sublist
            reverseIter.setNext( reverseIter.getPrev() );
            reverseIter.setPrev( origialListHeadTail );
        }
```

Q2

Give the worst-case time complexity for your method above in terms of N, the list size. Identify what aspect of the method characterizes the problem size. Write a brief justification for the time complexity you give. Include in your justification any assumptions you make about the complexity of any methods that are called by your implementation.

Answer for Q2

The the worst-case time complexity is $\mathcal{O}(N)$.

Most operations of `reverse` should be spent on the two `while` loop of code and we can ignore the time spent outside the two `while` loop since they are assignment,arithmetic, comparison, etc. which take constant time, $\mathcal{O}(1)$, that can be ignored.

The first `while` loop is to get the head node of sublist, the `while` loop contains 2 operations

```
            // get head node of sublist
            while( i <= pos1 )
            {
                sublistHeadNode = sublistHeadNode.getNext();
                i++;
            }
```

Another `while` loop is to reverse nodes in sublist which contains 5 operations

```
            while ( j < k - 1) // reverseIter points to next nodes of head
nodes of sublist
            {
                // prev and next nodes of current nodes in sublist
                DblListnode<E> tmp_prev = reverseIter.getPrev();
                DblListnode<E> tmp_next = reverseIter.getNext();
                // reverse pointers of current nodes
                reverseIter.setPrev( tmp_next );
                reverseIter.setNext( tmp_prev );
                // move current nodes to the next nodes of sublist
                reverseIter = tmp_next;
                // increase counter of the number of nodes processed
                j++;
            }
```

If we assume all operations will use equal time a , then for a reverse call from position N_a to position N_b, the time used for the whole method should be[1] :

$$2 \times a \times N_a + 5 \times a \times (N_b - N_a) = 5aN_b - 3aN_a$$

And because $N_a > 0$, so $5aN_b - 3aN_a < 5aN_b$, so if ignoring constants and coefficients, we know the over-all worst-case time complexity is $\mathcal{O}(N)$.

---

1. notice that our code handles that last node of sublist outside the `while` loop.↩