uc3m | Universidad Carlos III de Madrid

Data Structure and Algorithms.

Problems - Doubly linked lists.

Given the class DList, implementation of the TAD list based on doubly linked list, create a subclass DList2 and implement the following functions:

- remove(e): receives an element, e, and deletes the first occurrence of e in the list (i.e., removes the first node containing e). The function modifies the list and returns nothing. If the element does not exist in the list, the function must report that it does not exist.
- removeAll(e): receives an element, e, and deletes all occurrences of e in the list (i.e., removes all nodes containing e). The function modifies the list and returns nothing. If the element does not exist in the list, the function must report that it does not exist.
- getAtRev(index): receives an index, index, and returns the element at the index position starting from the end. For example:
  l: 0->1->2->3->4, l.getAtRev(0)=4, l.getAtRev(1)=3, l.getAtRev(2)=2, l.getAtRev(3)=1, l.getAtRev(4)=0.
- getAtEff(index): an efficient version of the getAt function, taking into account whether the index is less than or greater than half of the list, to start the search at the beginning or at the end of the list.
- insertAtEff(index,e): an efficient version of the insertAt function, taking into account whether the index is smaller or larger than half of the list, to start the search at the beginning or at the end of the list.
- removeAtEff(index): an efficient version of the removeAt function, taking into account whether the index is smaller or larger than half of the list, to start the search at the beginning or at the end of the list.
- getMiddle(): returns the element that is in the middle of the list. If the list has an even number of elements, the function will return the element at position len(l)//2 +1. Example: 1->2->3->4->5->6, l.getMiddle()=4.
- count(e): receives an element, e, and returns the number of times it occurs in the list. If the element does not exist in the list, the function returns 0.
- isPalindrome(): checks if the elements contained in the list form a palindrome word (e.g., radar, aba, abba, abcba). If it is palindrome it returns True, and otherwise False.
- isSorted(): checks if the list is sorted in ascending order (in this case it returns True). Otherwise, it must return False.
- removeDuplicatesSorted(): deletes duplicate items in an ordered list. The function modifies the list, it does not return anything.

Example: l: 1->1->2->3->3->3->4->5->5, l: 1->2->3->4->5.

- removeDuplicates(): deletes duplicate elements in a list (does not have to be sorted). The function modifies the list, it does not return anything.
  Example: l: 1->2->1->0->2->2->6->6->4->5->5, l: 1->2->0->6->4->5.
- swapPairwise(): swaps elements that occupy contiguous positions. The function modifies the list, it does not return anything. Examples:
  l: 1->2->2->3->4->5, l:2->1->1->4->3->5.
  l: 1->2->2->3->4->4->5->6, l:2->1->1->4->3->6->5
- moveLast(): moves the last item to the beginning of the list, without using any of the DList class functions. The function modifies the list, it does not return anything. Example: l:1->2->3->4->5->6, l: 6->1->2->2->3->4->5.
- intersection(l2): receives a doubly linked list, l2 and returns a new list containing the intersection of both the calling list and l2. As a precondition, it is required that both lists are sorted in ascending order. Example:
  l:1->2->3->4->5->6, l2: 0->1->2->3, output: 1->2->3.
- segregateOddEven(): modifies the calling list so that all the even elements appear before the odd elements. The function must respect the order of the even elements and the order of the odd elements.
  Example: l: 17->15->8->12->12->10->5->4->4->1->7->6
  l: 8->12->10->4->6->17->15->5->1->7
- reverse(): reverses the list. The function modifies the list, it does not return anything. It implements two different versions of the function: 1) swapping the elements in opposite positions of the list, and 2) swapping the links of the nodes.

Note. Many of these functions can be implemented in a simple way if you use other functions of the DList class. An interesting exercise would be to propose a second solution that does not use functions of the DList class. This exercise will allow you to gain more experience in working with nodes.