uc3m | Universidad Carlos III de Madrid

Estructura de Datos y Algoritmos.

Problemas - Listas doblemente enlazadas.

Dada la clase DList, implementación del TAD lista basada en lista doblemente enlazada, crea una subclase DList2 e implementa los siguientes métodos:

- remove(e): método que recibe un elemento, e, y borra la primera ocurrencia de e en la lista (es decir, elimina el primer nodo que contiene a e). El método modifica la lista y no devuelve nada. Si el elemento no existe en la lista, el método debe informar que no existe.
- removeAll(e): método que recibe un elemento, e, y borra todas las ocurrencias de e en la lista (es decir, elimina todos los nodos que contienen a e). El método modifica la lista y no devuelve nada. Si el elemento no existe en la lista, el método debe informar que no existe.
- getAtRev(index): método que recibe un índice, index, y devuelve el elemento en la posición index empezando por el final. Por ejemplo:
  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): versión eficiente de El método getAt, teniendo en cuenta si el index es menor o mayor que la mitad de la lista, para comenzar la búsqueda por el principio o por el final de la lista.
- insertAtEff(index,e): versión eficiente de El método insertAt, teniendo en cuenta si el index es menor o mayor que la mitad de la lista, para comenzar la búsqueda por el principio o por el final de la lista.
- removeAtEff(index): versión eficiente de El método removeAt, teniendo en cuenta si el index es menor o mayor que la mitad de la lista, para comenzar la búsqueda por el principio o por el final de la lista.
- get_middle(): método que devuelve el elemento que está en la mitad de la lista. Si la lista tiene un número par de elementos, El método devolverá el elemento en la posición len(l)//2 +1. Ejemplo: 1->2->3->4->5->6, l.get_middle()=4
- count(e): método que recibe un elemento, e, y devuelve el número de veces que ocurre en la lista. Si el elemento no existe en la lista, el método devuelve 0.
- is_sorted(): método que comprueba si la lista está ordenada de forma ascendente (en este caso devuelve True). En caso contrario, debe devolver False.
- remove_duplicates_sorted(): método que borra los elementos duplicados en una lista ordenada. El método modifica la lista, no devuelve nada.

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

- remove_duplicates(): método que borra los elementos duplicados en una lista (no tiene que estar ordenada). El método modifica la lista, no devuelve nada.
  Ejemplo: l: 1->2->1->0->2->6->6->4->5->5, l: 1->2->0->6->4->5.
- swap_pairwise(): método que intercambia los elementos que ocupan posiciones contiguas. El método modifica la lista, no devuelve nada. Ejemplos:
  l: 1->2->3->4->5, l:2->1->4->3->5
  l: 1->2->3->4->5->6, l:2->1->4->3->6->5
- move_last(): método que mueve el último elemento al principio de la lista, sin usar ninguna de los métodos de la clase DList. El método modifica la lista, no devuelve nada. Ejemplo: l:1->2->3->4->5->6, l: 6->1->2->3->4->5
- intersection(l2): método que recibe una lista doblemente enlazada, l2 y devuelve una nueva lista que contenga la intersección de ambas listas, la invocante y l2. Como precondición, se exige que ambas listas estén ordenadas de forma ascendente. Ejemplo: l:1->2->3->4->5->6, l2: 0->1->2->3, output: 1->2->3.
- segregate_odd_even(): método que modifica la lista invocante para que todos los elementos pares aparezcan antes que los elementos impares. El método debe respetar el orden de los elementos pares y el orden de los elementos impares.
  Ejemplo: **l:** 17->15->8->12->10->5->4->1->7->6
  l: 8->12->10->4->6->17->15->5->1->7

Nota. Muchas de estas métodos pueden implementarse de forma sencilla si usas otras métodos de la clase DList. Sin embargo, en algunas ocasiones el uso de estos métodos aumenta la complejidad temporal del método final. Recuerda que tus soluciones siempre deberían ser lo más eficientes posibles en términos de complejidad temporal.