ESTRUCTURA DE DATOS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN Y LA DECISIÓN

CONTRIBUCIONES: ALEJANDRO SALAZAR MEJÍA (BECARIO 2023) – CARLOS SEBASTIAN ZAMORA ROSERO (BECARIO 2024) – MARIA ALEJANDRA MUNOZ GONZALEZ (BECARIA 2024)

TALLER 4: LISTAS ENLAZADAS SIMPLES

Este taller está diseñado para afianzar tu comprensión sobre las listas enlazadas simples, el manejo de apuntadores entre nodos de forma unidireccional, fortalecer tu habilidad para resolver algoritmos sobre estructuras de datos y usar los beneficios que traen listas enlazadas para solucionar problemas algorítmicos de forma eficiente. Varios de estos ejercicios están inspirados en problemas de entrevistas técnicas y competencias de programación.

Los talleres son herramientas de estudio, y no tienen asignada una ponderación dentro de las evaluaciones del curso. Se recomienda desarrollar cada problema mediante la presentación de un pseudocódigo, y en el caso de diseño de clase mediante diagramas de clase. Sin embargo, si el estudiante puede, solucionar cada problema empleando un lenguaje de programación de alto nivel.

Nota: El símbolo → se utiliza para representar la conexión en una lista enlazada simple, indicando que cada nodo apunta únicamente al siguiente nodo en la secuencia.

1. Dada una lista enlazada simple *L* de la cual **no conocemos su longitud**, cree un algoritmo que divida 'por la mitad' a *L* y retorne dos listas de longitudes (casi) iguales **usando solamente un ciclo**. Esto último quiere decir que no se vale recorrer toda la lista para calcular su longitud y después volver a iterarla la mitad de su longitud. Asuma que la lista enlazada no tiene atributo *size*. Debe hallar el punto medio de la lista recorriéndola una única vez.

(hint:** en ningún momento es necesario calcular la longitud de la lista. Dentro de un solo ciclo utilice dos apuntadores, y haga que uno avance más rápido que el otro**)

2. Dadas dos listas simples enlazadas **ordenadas**, cree un algoritmo recursivo que las 'fusione' (merge) de forma que la lista resultante contenga todos los elementos de ambas listas y **también esté ordenada**. Las listas pueden tener longitudes diferentes. Su algoritmo debe correr en *O(n+m).*

3. El algoritmo *Merge-Sort* es uno de los algoritmos de ordenamiento más famosos y su ejecución en el peor de los casos tiene complejidad ***O(n log(n))***. Consulte este algoritmo y asegúrese de entenderlo.

Implemente el algoritmo *Merge-Sort* **recursivo** para listas simples enlazadas. Es decir, dada una lista enlazada, ordénela usando la versión recursiva del algoritmo *Merge-Sort*.

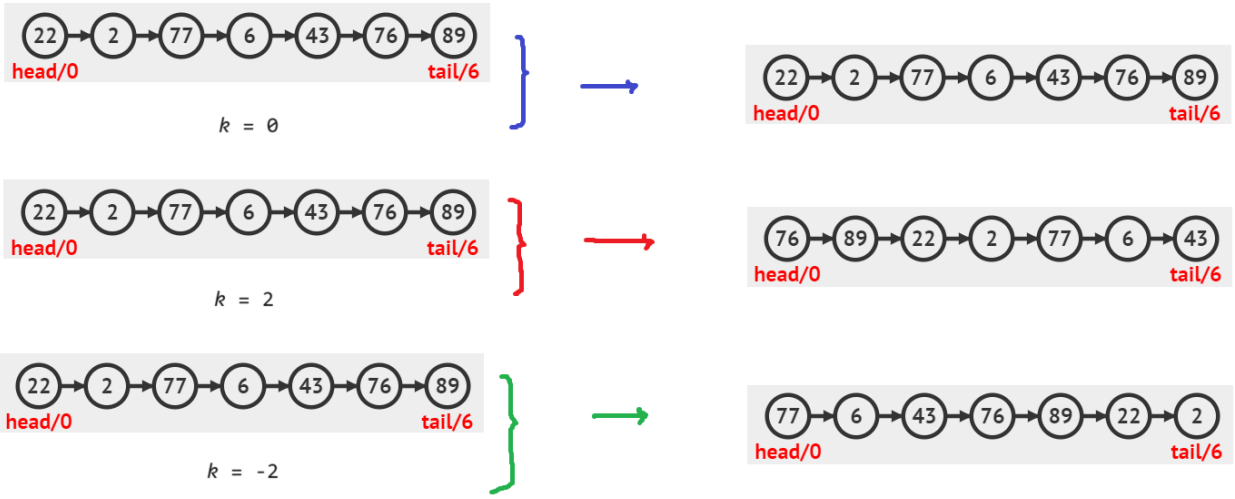
*(**hint**: utilice los algoritmos creados en los puntos **1.** y **2.**)*

4. Dadas dos listas simples enlazadas de números enteros, devuelva otra lista con los elementos pertenecientes a la intersección de ambos conjuntos. Por ejemplo, si la lista 1 contiene los elementos 4, 6, 11, 7, 1 y la lista 2 los elementos 1, 2, 3, 4, 5, 6, 7, la salida de su algoritmo debe ser una lista con los elementos 1, 4, 6, 7 (no importa el orden). Su algoritmo debe correr en tiempo *O(n log(n))*.

5. Dada una lista simple enlazada, diseñe un algoritmo que permita decidir si la lista es un **palíndromo**.

6. Cree un algoritmo que solucione el *Problema de Flavio Josefo* utilizando **arreglos circulares.** Estos arreglos circulares pueden ser implementados con listas enlazadas simples.

7. Dada una lista enlazada *L* y un número entero *k* (positivo o negativo), **rote** la lista *L k* veces a la derecha si *k* es positivo o *k* veces a la izquierda si es negativo. Por ejemplo:



8. Dada una lista *L* y un número natural *n,* encuentre el n-ésima elemento desde el final de la lista. Así, por ejemplo, *n* = 1 corresponde al último elemento de *L*, *n* = 2 al penúltimo, n = 3 al trasantepenúltimo, y así sucesivamente. Su algoritmo **no** puede invertir la lista *L* ni utilizar una copia invertida de *L.*

9. Siga las siguientes instrucciones:

- Consulte el algoritmo de ***búsqueda binaria*** para arreglos.
- Responda: ¿por qué **no es eficiente** implementar búsqueda binaria para listas enlazadas?
- Consulte sobre la estructura de datos ***Skip List***.

- Responda: ¿por qué es conveniente utilizar aleatoriedad a la hora de insertar elementos?
- *Opcional*: Intente justificar por qué la Skip List tiene tiempo de búsqueda *esperado* O(log(n)) y espacio usado *esperado* O(n)

10. Implemente el algoritmo *Insertion-Sort* para listas enlazadas simples.

11. Dada una lista enlazada *L,* invierta *L* '**in-place'**. La palabra "in-place" indica que la operación se debe realizar sin utilizar una estructura de datos adicional para almacenar temporalmente los elementos de la lista. En otras palabras, la inversión debe ocurrir directamente en la lista enlazada original, modificando los enlaces entre los nodos sin crear una nueva lista.

12. Consulte el algoritmo de ordenamiento **Counting-Sort,** el cual corre en tiempo *O(n)* y no usa comparaciones (< ó >), sino que cuenta frecuencias. Dada una lista enlazada *L* que contiene solo los valores 0, 1 y 2, ordene la lista **sin usar** Counting-Sort. Es decir, la tarea consiste en reorganizar los enlaces entre los nodos de la lista enlazada de manera que los nodos con valor 0 aparezcan primero, seguidos por los nodos con valor 1 y luego los nodos con valor 2. Esto es importante, pues puede que los valores 0, 1, 2 estén asociados a objetos que tienen más información. Su algoritmo debe correr en ***O(n).***

13. Invertir una Lista Enlazada Simple. Dada una lista enlazada simple, implementa un algoritmo que invierta la lista enlazada en su lugar (sin crear una nueva lista). Esto significa que el primer nodo se convertirá en el último, el segundo en el penúltimo, y así sucesivamente.

Ejemplo de entrada:
Lista: 1 → 2 → 3 → 4 → 5

Ejemplo de salida:
Lista invertida: 5 → 4 → 3 → 2 → 1

14. Encontrar el Nodo en el Medio de una Lista Enlazada. Dada una lista enlazada simple, implementa una función que encuentre el nodo en el medio de la lista. Si la lista tiene un número par de nodos, devuelve el segundo de los dos nodos del medio.

Ejemplo de entrada:
Lista: 1 → 2 → 3 → 4 → 5

Ejemplo de salida:

Nodo medio: 3

Ejemplo de entrada:
Lista: 1 → 2 → 3 → 4

Ejemplo de salida:

Nodo medio: 3 (el segundo nodo en el medio de los dos nodos centrales).

15. Intercambio de Nodos Adyacentes. Dada una lista simplemente enlazada, diseñe un algoritmo para intercambiar nodos adyacentes sin cambiar los datos dentro de los nodos.

Ejemplo de entrada:
Lista: 1 → 2 → 3 → 4 → 5

Ejemplo de salida:
Lista intercambiada: 2 → 1 → 4 → 3 → 5

Restricción: El intercambio debe hacerse únicamente manipulando los apuntadores, sin modificar los valores almacenados en los nodos.

16. Reorganizar Lista en Orden Alterno de Máximos y Mínimos. Dada una lista simplemente enlazada ordenada en orden ascendente, diseñe un algoritmo que reorganice los nodos en un orden alterno de máximos y mínimos.

Ejemplo de entrada:
Lista: 1 → 2 → 3 → 4 → 5 → 6
Ejemplo de salida:
Lista intercambiada: 6 → 1 → 5 → 2 → 4 → 3

Pista: Considere utilizar una lista auxiliar o técnicas avanzadas de manipulación de apuntadores.