ESTRUCTURA DE DATOS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN Y LA DECISIÓN

MONITOR: ALEJANDRO SALAZAR MEJÍA

alsalazarm@unal.edu.co

2023-2S

TALLER 7: LISTAS ENLAZADAS DOBLES

Este taller está diseñado para afianzar tu comprensión sobre las listas enlazadas dobles, el manejo de apuntadores entre nodos de forma bidireccional, 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 seudocódigo, 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.

1. Consulte el algoritmo **Bubble-Sort** e impleméntelo para ordenar listas doblemente enlazadas, de forma que al comparar dos nodos adyacentes, los nodos son intercambiados en lugar de sólo intercambiar los datos.

2. Dada una lista doblemente enlazada, cree un algoritmo que elimine los nodos con **valores duplicados**, de forma que se mantenga la primera aparición de cada valor y se conserve el orden original de los nodos.

3. Dada una lista doblemente enlazada que **representa** un número entero positivo (cada nodo contiene un dígito), devuelva otra lista doblemente enlazada que represente el doble de dicho número. Por ejemplo, si *L* es 5↔2↔1, 521 * 2 = 1042 y por tanto su algoritmo debe devolver 1↔0↔4↔2.

*(**hint**: La idea es ir calculando el resultado a medida que se recorre la lista, no calcular el número completo que representa y después crear otra lista con su doble).*

4. Dadas dos listas doblemente enlazadas que representan cada una un número entero positivo (cada nodo contiene un dígito), devuelva otra lista doblemente enlazada que represente la suma de ambos números. Por ejemplo, si *L1* es 5↔2↔1 y *L2* es 4↔5↔3, 521 + 453 = 974 y por tanto su algoritmo debe devolver 9↔7↔4.

*(**hint**: La idea es ir calculando el resultado a medida que se recorren ambas listas al tiempo, no calcular los números completos que representan y después crear otra lista con su suma).*

5. Dada una lista doblemente enlazada *L* y dos enteros *n* y *m* tales que *0 <= n <= m* diseñe un algoritmo que invierta la sub-lista de *L L[n:m] in-place*. Por ejemplo, si *n* = 1, *m* = 4 y *L* es A↔B↔C↔D↔E↔F↔G, su algoritmo debe devolver como respuesta A↔E↔D↔C↔B→F→G.

6. Dada una lista doblemente enlazada *L* posiblemente con valores repetidos y un valor *k,* elimine la **última ocurrencia** de *k* en *L.*

7. Dada una lista doblemente enlazada *L* y un número entero positivo *k,* intercambie el *k*-ésimo nodo desde el principio con el *k*-ésimo nodo desde el final.

8. Dada una lista doblemente enlazada ordenada de números enteros positivos distintos, la tarea consiste en **encontrar pares** de dicha lista cuya suma sea igual a un valor dado *x.*

(***hint:*** *es posible hacer este ejercicio en O(n) utilizando dos apuntadores, uno que empiece como el primer nodo de la lista y el otro como el último*).

9. Dada una lista doblemente enlazada que contiene *n* nodos y un número entero positivo *k*, el problema consiste en invertir cada grupo de *k* nodos de la lista. Por ejemplo, si *k* = 3 y tenemos la lista 1↔2↔3↔4↔5↔6↔7↔8, su programa debe devolver 3↔2↔1↔6↔5↔4↔8↔7.

10. Imagine que usted tiene un navegador web de una pestaña donde comienza en la página de inicio y puede visitar otro *URL*, retroceder en el historial un número de pasos o avanzar en el historial un número de pasos. Diseñe un historial de navegador web de acuerdo con las siguientes instrucciones:

- Implemente la clase `BrowserHistory`. `BrowserHistory(string homepage):` Inicializa el objeto con la página de inicio del navegador.
- `void visit(string url):` Visita URL *url* desde la página actual. Borra todo el historial hacia adelante (*forward history*).
- `string back(int steps):` Retrocede una cantidad *steps* de pasos en el historial. Si solo puedes retroceder *x* pasos en el historial y *steps* > *x*, solo retrocederás *x* pasos. Devuelve el URL actual después de retroceder en el historial como máximo la cantidad *steps* de pasos.
- `string forward(int steps):` Avanza una cantidad *steps* de pasos en el historial. Si solo puedes avanzar *x* pasos en el historial y *steps* > *x*, solo avanzarás *x* pasos. Devuelve el URL actual después de avanzar en el historial como máximo la cantidad *steps* de pasos.