



Tecnológico de Monterrey

Act 2.2 - Functionalities of a linear data structure verification

Santiago Vera Espinoza A01641585

06 de Octubre del 2022

Modelación de sistemas mínimos y arquitecturas computacionales

Jorge Enrique González Zapata

ADT Evaluation

Introducción

Las lista ligadas son una de las estructuras de datos más utilizadas en la programación, son muy útiles a la hora de guardar datos y manejar variables en escalas mayores a la unitaria. Sin estas, la programación de datos en cadena sería imposible y múltiples procesos no existirían.

En este documento se explicará el funcionamiento de una lista doblemente ligada, sus casos de prueba, funciones y manejo.

Test cases

Para empezar, el programa parte de ciertos datos dados por un archivo de texto. Los cuales se agregan antes del nodo inicial, posterior a la inicialización de la lista. Dicha información se despliega a continuación.

```
PRODASFA > Homework_4 > ≡ lect.txt
1 14-9-4-6-7-8-11-25-24-
```

Por lo que el árbol inicial sería (24, 25, 11, 8, 7, 6, 4, 9, 14, 1).

```
-----
24
25
11
8
7
6
4
9
14
1
-----
```

Después se realizan las distintas pruebas:

- insertAfter(0, 10), insertAfter(0, 5), insertBefore(1, 7), insertBefore(0, 9), insertBefore(0, 11)
 - Esperado: nueva lista = (11, 9, 24, 7, 5, 10, 25, 11, 8, 7, 6, 4, 9, 14, 1)

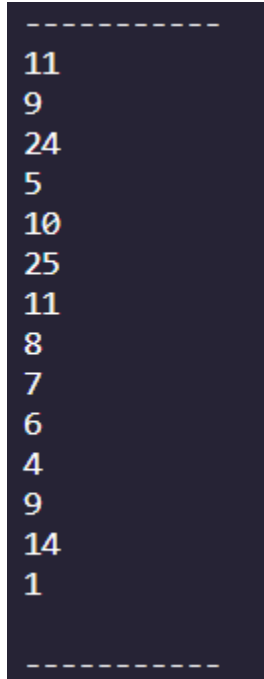
```
-----  
11  
9  
24  
7  
5  
10  
25  
11  
8  
7  
6  
4  
9  
14  
1  
-----
```

-
- searchPos(3)
 - Esperado: 7
 - **7**
- searchValue(7)
 - Esperado: 7
 - **7**
- update(3, 4)
 - Esperado: nueva lista = (11, 9, 24, 4, 5, 10, 25, 11, 8, 7, 6, 4, 9, 14, 1)

```
-----  
11  
9  
24  
4  
5  
10  
25  
11  
8  
7  
6  
4  
9  
14  
1  
-----
```

○

- delByPos(3)
 - Esperado: nueva lista = (11, 9, 24, 5, 10, 25, 11, 8, 7, 6, 4, 9, 14, 1)



```
-----  
11  
9  
24  
5  
10  
25  
11  
8  
7  
6  
4  
9  
14  
1  
-----
```

La razón por la que se eligieron dichos casos prueba fue para verificar que todas las funciones implementadas funcionaran correctamente. Después de cada función se imprime el resultado para verificar. Se agregan nodos antes y después de una posición, y cuando se eliminan se quita el nodo apuntado.

Explicación del software

El software consiste en dos clases principales. La clase Node y la clase DoubleLinkedList. Para generalizar, ambas clases usan plantillas para modificar el tipo de clase que se utilizará.

Clase Node

Contiene tres atributos:

- data: Es del tipo que se pasa al inicializar un objeto de dicha clase, almacena la información del nodo.
- prev: Es un apuntador a un potencial nodo anterior.
- next: Es un apuntador a un potencial nodo siguiente.

La clase tiene 3 constructores:

- Default: El dato no se especifica y los apuntadores son nulos.
- Se pasa data: Se cambia data, pero los apuntadores son nulos.
- Se pasa data y apuntadores: Se cambia data y los apuntadores.

Todas las demás funciones son getters y setters para cada atributo. Todas las funciones de esta clase tienen una complejidad $O(1)$.

Clase DoubleLinkedList

Contiene un solo atributo llamado head, el cual es un apuntador al primer nodo.

Esta clase tiene 2 constructores:

- Default: La cabeza es NULL.
- Se pasa val: La cabeza es un apuntador a un nuevo nodo con valor val.

Hasta aquí todas las funciones tienen una complejidad $O(1)$. Todas las siguientes tienen una complejidad $O(n)$. Desglosando función por función, su funcionamiento es el siguiente:

- insertAfter: Inserta un nuevo nodo a la derecha de una posición dada.
 - Primero se genera un apuntador a la cabeza para iterar la lista.
 - Se crea un contador y este aumenta mientras el siguiente nodo sea válido y la posición ingresada sea mayor al contador.
 - Si el nodo es NULL, sale de la función.
 - Se crea un apuntador a un nuevo nodo con los datos de entrada.
 - El nuevo nodo se conecta al nodo actual y a su siguiente elemento.
 - El nodo actual se conecta al nuevo nodo y el nodo posterior al actual también.
 - Se regresa al inicio de la lista y se establece que la cabeza apunte a dicho nodo.
- insertBefore: Inserta un nuevo nodo a la izquierda de una posición dada.
 - Primero se genera un apuntador a la cabeza para iterar la lista.
 - Se crea un contador y este aumenta mientras el siguiente nodo sea válido y la posición ingresada sea mayor al contador.
 - Si el nodo es NULL, sale de la función.
 - Se crea un apuntador a un nuevo nodo con los datos de entrada.
 - El nuevo nodo se conecta al nodo actual y a su anterior elemento.
 - El nodo actual se conecta al nuevo nodo y el nodo anterior al actual también.
 - Se regresa al inicio de la lista y se establece que la cabeza apunte a dicho nodo.
- searchPos: Busca un elemento por posición.
 - Primero se genera un apuntador a la cabeza para iterar la lista.
 - Se crea un contador y este aumenta mientras el siguiente nodo sea válido y la posición ingresada sea mayor al contador.
 - Si el nodo es NULL, regresa un error.
 - Regresa el nodo encontrado.
- searchValue: Busca un elemento por valor.
 - Primero se genera un apuntador a la cabeza para iterar la lista.
 - Se itera al siguiente nodo mientras este sea válido y el valor ingresado sea diferente al del nodo iterado.
 - Si el nodo es NULL, regresa un error.
 - Regresa el nodo encontrado.
- update: Cambia el valor de un nodo dada una posición.
 - Primero se genera un apuntador a la cabeza para iterar la lista.
 - Se crea un contador y este aumenta mientras el siguiente nodo sea válido y la posición ingresada sea mayor al contador.
 - Si el nodo es NULL, regresa un error.
 - Cambia el valor del nodo encontrado.
 - Se regresa al inicio de la lista y se establece que la cabeza apunte a dicho nodo.
- delByPos: Borra un nodo dada una posición.

- Primero se genera un apuntador a la cabeza para iterar la lista.
- Se crea un contador y este aumenta mientras el siguiente nodo sea válido y la posición ingresada sea mayor al contador.
- Si el nodo es NULL, regresa un error.
- Crea dos apuntadores llamados left y right y les asigna NULL como valor.
- Si el valor siguiente al encontrado no es NULL, right apuntará a él.
- Si el valor anterior al encontrado no es NULL, left apuntará a él.
- Si el valor siguiente al encontrado no es NULL, el apuntador siguiente de left será right.
- Si el valor anterior al encontrado no es NULL, el apuntador anterior de right será left.
- Si el valor anterior al actual es inválido, la cabeza apuntará a este.
- Se borra el nodo encontrado.
- print: Imprime la lista.
 - Primero se genera un apuntador a la cabeza para iterar la lista.
 - Se itera por la lista hasta que el valor actual sea nulo y en cada iteración se imprime el valor seguido por un salto de línea.

Para leer el archivo de texto se crea una variable infile, la cual se abre y se obtiene su primera línea para guardarla en una string. Se cierra el archivo y se crea la DoubleLinkedList y se inicializa con 1. Se itera por la línea y todo lo que esté antes del carácter “-” se agrega a un string, se convierte a int y se agrega a la lista doblemente ligada.

Conclusión

En esta actividad aprendí cómo crear una lista doblemente ligada y cómo manejar de mejor manera los apuntadores de un código. Pudo observar la utilidad de una lista doblemente ligada y la estructura más interna de una. Esto me ayuda en gran manera a entender las bases de la programación y cómo estas se enlazan como bloques de LEGO.