Lista Doblemente Enlazada

Realizado Por:

Diego Cardenas

Zayra Gutiérrez

Felipe Calvache

Presentado A:

Sebastián Camilo Martínez Reyes

Escuela Colombiana De Ingeniería Julio Garavito

Algoritmos Y Estructuras De Datos

Programa De Ingeniería De Sistemas

Bogotá D.C.

1. Contexto

Realizar la implementación de la lista doblemente enlazada usando como referencia el código adjunto (DoubleLinkedList) extendiendo la definición de Nodo a tener un apuntador a aun nodo previo

2. Diseño

Para implementar la lista doblemente enlazada, agregamos un nuevo atributo al constructor el cual es el nodo previo, y se agregaron dos nuevos métodos los cuales tienen como objetivo obtener el valor previo (get), y asignar el valor previo (set).

2.1. Casos de Prueba

Creamos una lista doblemente enlazada mediante código.

3. Análisis

3.1. Temporal

- Los métodos get y set de nuestra implementación tiene complejidad O(1).
- El método delete tiene complejidad O(n)
- El método para eliminar duplicados tiene complejidad O(n).
- El método para combinar dos DoubleLinkedList, tiene complejidad O(1).
- El método length tiene complejidad O(n).

Pila y Cola

Realizado Por:

Diego Cardenas

Zayra Gutiérrez

Felipe Calvache

Presentado A:

Sebastián Camilo Martínez Reyes

Escuela Colombiana De Ingeniería Julio Garavito

Algoritmos Y Estructuras De Datos

Programa De Ingeniería De Sistemas

Bogotá D.C.

1. **Contexto**

   Realizar la implementación de Pila y Cola (Queue, Stack) en donde en su definición estructural se reuse la definición realizada en el punto 1. En las operaciones CRUD recuerde mantener la política de cada estructura (LIFO - FIFO)

2. **Diseño**

   A. Para implementar Stack, esta clase de pila hereda todas las propiedades y métodos de DoubleLinkedList, los cambios son la forma de agregar y eliminar valores porque Stack usa LIFO como estrategia para agregar nuevos elementos y eliminar LIFO.

   B. Para implementar Queue, esta clase Queue hereda todas las propiedades y métodos de DoubleLinkedList, lo que cambia la forma en que se agregan y eliminan los valores, ya que las Colas tienen una estrategia de último en entrar, primero en salir para agregar nuevos elementos y FIFO para eliminar a ellos.

2.1. Casos de Prueba

A.

Como caso de prueba se utilizó la simulación de una pila de libros donde ingresan 6 libros, se prestan 3 de estos y devuelven uno.

```
def mainStack():
    pila = Stack()
    pila.push("Unit 1")
    pila.push("Unit 2")
    pila.push("Unit 3")
    pila.push("Unit 4")
    pila.push("Unit 5")
    pila.push("Unit 6")
    print(pila)
    pila.pop()
    pila.pop()
    pila.pop()
    print(pila)
    pila.push("Unit 7")
    print(pila)
mainStack()
```

```
Unit 1 Unit 2 Unit 3 Unit 4 Unit 5 Unit 6
Unit 1 Unit 2 Unit 3
Unit 1 Unit 2 Unit 3 Unit 7
```

B.

Como caso de prueba se utilizó la simulación la cola de un banco en donde hay 3 personas en fila y llega otra a ser atendida.

```
def mainQueue():
    cola = Queue()
    cola.enqueue("Antonio")
    cola.enqueue("Julian")
    cola.enqueue("Camilo")
    print(cola)
    cola.dequeue()
    cola.dequeue()
    print(cola)
    cola.enqueue("Ignacio")
    print(cola)
```

```
Antonio Julian Camilo
Camilo
Camilo Ignacio
```

3. Análisis

3.1.

- El método pop tiene complejidad O(n).
- El método push tiene complejidad O(n).

3.2

- El método dequeue tiene complejidad O(1).
- El método enqueue tiene complejidad O(n).