**Análisis de Complejidad Temporal y Espacial.**

*Juan Andrés Cano R && Juan David Quintero P.*

**Complejidad Temporal:**

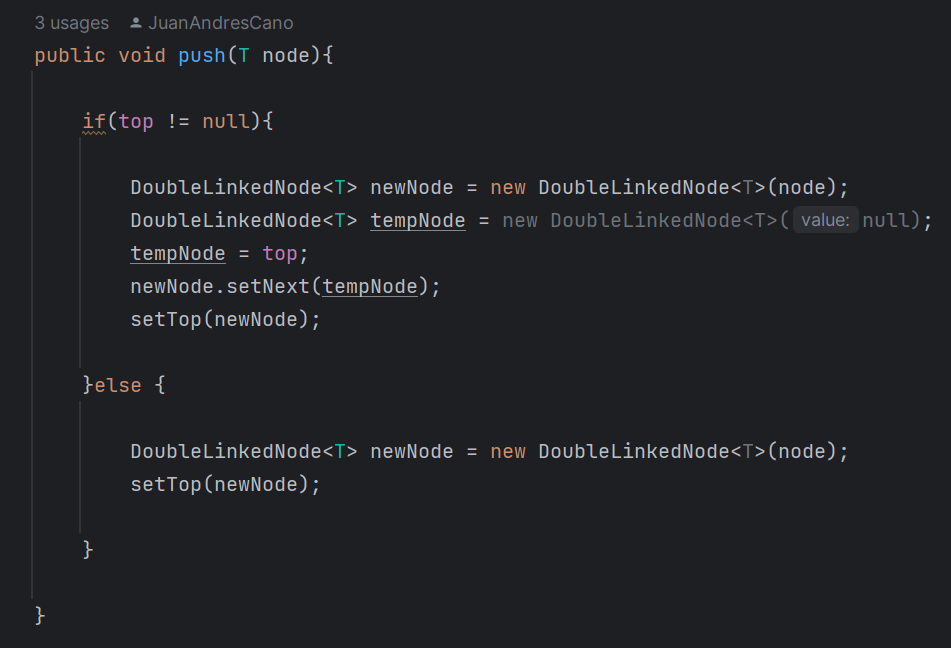
Texto

Descripción generada automáticamenteLa complejidad temporal de este algoritmo es O(n). Esto porque lo que hace el algoritmo, es recorrer una DoubleLinkedList genérica para encontrar un nodo especifico y removerlo. El algoritmo no tiene ningún ciclo como un “while” o “for”, y la mayoría de métodos que se llaman tienen una complejidad de O(1). No obstante, este método tiene un llamado recursivo a sí mismo, y se hará tantas veces como sea necesario para encontrar al nodo a eliminar o hasta recorrer toda la lista, entonces el algoritmo tiene una complejidad de O(n) siendo n la cantidad de nodos en la lista.

Texto

Descripción generada automáticamenteLa complejidad temporal de este algoritmo es O(n). Esto porque lo que hace el algoritmo, es recorrer una DoubleLinkedList genérica para encontrar un nodo especifico y removerlo. EL algoritmo no tiene ningún ciclo como un “while” o un “for”, y la mayoría de métodos como “getNext()”, “getId()” y “setValue()” tienen una complejidad temporal de O(n). No obstante, este método tiene un llamado recursivo a sí mismo para recorrer los nodos de la lista. Este llamado se hará tantas veces como número de nodos en la lista o hasta que se encuentre el nodo a modificar, lo que pase primero. Entonces como en el peor de los casos se tendrá que recorrer toda la lista, la complejidad temporal es O(n) siendo n el número de nodos en la lista.

**Complejidad Espacial:**

Este algoritmo tiene una complejidad espacial de O(1). Para argumentar esto verifiquemos los objetos que intervienen en la complejidad espacial de este algoritmo. “top” es el nodo en la cima del Stack, este ocupa una cantidad constante de memoria que varía proporcionalmente según el objeto que se introduzca como “top”. “newNode” es un nuevo nodo para almacenar al nodo que se está dando como parámetro, el espacio ocupado por este objeto siempre será directamente proporcional al tamaño de la entrada (Si la entrada es el doble de grande, entonces el espacio será el doble). Por último “tempNode” es un objeto que inicialmente tiene un valor nulo, y luego se utiliza para manipular al objeto que estaba asociado al puntero “top”, por lo cual el espacio en memoria que utilice será directamente proporcional al espacio en memoria que use “top”. Por todas estas razones, y que mientras más grande la entrada, la cantidad de objetos instanciados y creados son los mismos, entonces la complejidad temporal de este algoritmo es O(1).

Texto

Descripción generada automáticamente

Este algoritmo tiene una complejidad espacial de O(1). Para justificar esto, verifiquemos los objetos que intervienen en la complejidad espacial de este algoritmo. “top” es una referencia al nodo en la cima de la pila. Ocupa una cantidad constante de memoria, independientemente del tamaño de entrada. “node” es una variable que almacena el valor del nodo que se va a retirar de la pila. Ocupa una cantidad constante de memoria. “size” es una variable que lleva un registro del tamaño de la pila. Ocupa una cantidad constante de memoria. La función “getTop()” devuelve una referencia al nodo en la cima de la pila, pero esta referencia no agrega memoria adicional, ya que simplemente apunta al nodo existente. La operación “top.getNext()” se utiliza para actualizar la referencia top al siguiente nodo en la pila, lo que no agrega memoria adicional. En este algoritmo, la cantidad de memoria utilizada es principalmente constante y no depende del tamaño de entrada. No se están creando estructuras de datos adicionales que crezcan con el tamaño de la entrada. Por lo tanto, la complejidad espacial del algoritmo es O(1).