**ANÁLISIS DEL RETO**

Iker Barbosa, 202424135, i.barbosac@uniandes.edu.co

Camilo Castro, 202412178, email 2

Melisa Molina, 202312232, email 3

# **Requerimiento 1**

## **Descripción**

El punto del requerimiento es identificar el último registro recopilado según un año de interés. Por este motivo, usamos la librería datetime para poder comparar las fechas de la columna load\_time. Primero, revisamos que la lista no este vacia, si lo está, retorna null. En caso de que no esté vacía, recorremos la lista para hallar el último registro añadido por medio de comparaciones y guardarlo en una variable. Por último, recorremos la lista otra vez para hallar la fecha mínima encontrada y así agregar los datos al diccionario de retorno. Al encontrar el primero forzamos la salida del bucle con return().

|  |  |
| --- | --- |
| **Entrada** | agro, year:str |
| **Salidas** | retorno\_final = {"numero\_registros": 0, "registro": {}} |
| **Implementado (Sí/No)** | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(1), consultar el size de la lista para verificar que no esté vacía. |
|  | O(n), recorrer la lista para hallar el último registro añadido en el año pasado por parámetro. |
|  | O(n), recorrer la lista otra vez para hallar la fecha mínima encontrada y así agregar los datos al diccionario de retorno. |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el año 2024.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 100000 datos |  | 34.431 |
| 200000 datos |  | 58.343 |
| 300000 datos |  | 91.552 |
| 400000 datos |  | 120.980 |
| 500000 datos |  | 136.652 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Al tratarse de recorridos de lista completos para realizar comparaciones, por temas de velocidad decidimos usar Single Linked List, la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar para recorrer la lista a su totalidad, ya que debemos comparar todos los elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por su forma y su pendiente es una función lineal. Por lo que cumple con el comportamiento lineal esperado.

# **Requerimiento 2**

## **Descripción**

El punto del requerimiento es identificar el último registro cargado dado un departamento de interés. Similar al requerimiento anterior, usamos la librería datetime para poder comparar las fechas de la columna load\_time y así identificar el último registro cargado. En este caso, simplemente recorremos la lista y con un condicional verificamos los datos correspondan al departamento que recibimos por parámetro y guardamos las fechas en variables para compararlas. Por último, recorremos la lista otra vez para hallar la fecha mínima encontrada y así agregar los datos al diccionario de retorno. Al encontrar el primero forzamos la salida del bucle con return().

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** | retorno\_final = {"numero\_registros": 0, "registro": {}} |
| **Implementado (Sí/No)** | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(n), recorrer la lista para hallar el último registro añadido dependiendo del departamento pasado por parámetro. |
|  | O(n), recorrer la lista para hallar el último registro añadido en el departamento pasado por parámetro. |
| ***TOTAL*** | ***O(n)*** |

## **Pruebas Realizadas**

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el catálogo de datos y el estado de CALIFORNIA.

|  |  |
| --- | --- |
| Procesadores | AMD Ryzen 7 5700G |
| Memoria RAM | 22 GB |
| Sistema Operativo | Windows 10 |

|  |  |
| --- | --- |
| **Entrada** | **Tiempo (s)** |
| 100000 datos |  |
| 200000 datos |  |
| 300000 datos |  |
| 400000 datos |  |
| 500000 datos |  |

### **Tablas de datos**

Las tablas con la recopilación de datos de las pruebas.

|  |  |  |
| --- | --- | --- |
| **Muestra** | **Salida** | **Tiempo (ms)** |
| 100000 datos |  | 29.489 |
| 200000 datos |  | 86.088 |
| 300000 datos |  | 120.451 |
| 400000 datos |  | 145.154 |
| 500000 datos |  | 166.005 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Al tratarse de recorridos de lista completos para realizar comparaciones, por temas de velocidad decidimos usar Single Linked List, la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar para recorrer la lista a su totalidad, ya que debemos comparar todos los elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Al inicio tiene una pendiente más inclinada, pero a medida que tenemos más datos, adopta su forma lineal y una pendiente más fija. Por lo que cumple con el comportamiento lineal esperado.