**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.

# **Requerimiento 4**

## **Descripción**

El punto del requerimiento es consultar los registros recopilados para un tipo de producto de interés dado un rango de años de recopilación dado. Por este motivo, use una función auxiliar para que me dé un Array List filtrado entre años de carga. Desde acá es más fácil recorrer de nuevo un Array List más pequeño y filtrarlo por tipo de producto. En caso de que el numero de registros sea menor o igual a 20 devuelvo una Pila, en caso contrario, devuelvo un Array List con acceso a elementos.

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** | 1. Tupla con pila, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. 2. Tupla con Array List, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. |
|  | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(n), la función auxiliar buscar\_entre\_anios\_al() es una función que recorre todo el Array List para filtrar entre los años recibidos por parametro. |
|  | O(n), recorrer la lista filtrada entre años para hallar los registros que coincidan con el tipo de producto que se recibió por parámetro. |
|  | O(n), si el tamaño de lista es menor o igual a 20, recorremos la lista para añadir sus elementos a una pila y retornarla. |
|  | O(n), en caso de que el tamaño de la lista sea mayor a 20. Creamos una nueva Array List con new\_list(), e iteramos la lista para añadir los primeros 5 elementos y los últimos 5 con get.element() y retornamos el Array List. |
| ***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 tipo de producto EGGS, entre 1980 y 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 |  | 72.495 |
| 200000 datos |  | 142.293 |
| 300000 datos |  | 249.642 |
| 400000 datos |  | 293.690 |
| 500000 datos |  | 359.728 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Por temas de retorno decidimos usar Array List, ya que nos facilita el acceso a elementos con get\_element() y es de complejidad O(1), la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar los Array List varias veces poder realizar filtros, añadir elementos y obtener elementos.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Por poco es una recta completa, indicando su crecimiento lineal.

# **Requerimiento 6**

## **Descripción**

El punto del requerimiento es conocer la estadística del departamento de mi interés para un rango de fechas de carga de los registros dado. Por este motivo, use una función auxiliar para que me dé un Array List filtrado entre fechas de carga que funciona con la librería datetime. Desde acá es más fácil recorrer de nuevo un Array List más pequeño y filtrarlo por departamento. En caso de que el número de registros sea menor o igual a 20 devuelvo una Pila, en caso contrario, devuelvo un Array List con acceso a elementos.

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** | 1. Tupla con pila, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. 2. Tupla con Array List, boolean, size, numero de registros tipo celsus, numero de registros tipo survey. |
|  | Implementado, Iker Barbosa |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
|  | O(n), la función auxiliar buscar\_entre\_fechas\_al() es una función que recorre todo el Array List para filtrar entre fechas de carga recibidas por parámetro. |
|  | O(n), recorrer la lista filtrada entre fechas de carga para hallar los registros que coincidan con el tipo de departamento que se recibió por parámetro. |
|  | O(n), si el tamaño de lista es menor o igual a 20, recorremos la lista para añadir sus elementos a una pila y retornarla. |
|  | O(n), en caso de que el tamaño de la lista sea mayor a 20. Creamos una nueva Array List con new\_list(), e iteramos la lista para añadir los primeros 5 elementos y los últimos 5 con get.element() y retornamos el Array List. |
| ***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, el estado de CALIFORNIA, la fecha de carga 2012-01-01 00:00:00 y la fecha de carga 2024-11-07 12:00:00 .

|  |  |
| --- | --- |
| 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 |  | 571.472 |
| 200000 datos |  | 1167.862 |
| 300000 datos |  | 1743.198 |
| 400000 datos |  | 2334.033 |
| 500000 datos |  | 2891.148 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

Similar al requerimiento 4, decidimos usar Array List, ya que nos facilita el acceso a elementos con get\_element() y es de complejidad O(1), la implementación de este requerimiento tiene un orden lineal O(n). Esto se debe a que debemos iterar los Array List varias veces poder realizar filtros, añadir elementos y obtener elementos.

Este comportamiento se puede evidenciar perfectamente en la gráfica. Una recta completa, lo que indica su crecimiento lineal.

# **Requerimiento 8**

## **Descripción**

El punto del requerimiento es conocer el cuál es el departamento con mayor diferencia promedio de tiempo entre la recopilación de los registros y su carga a la plataforma de entre todos los registros. Para este requerimiento decidimos crear un diccionario de listas enlazadas en donde la llave del diccionario representa un departamento y la lista enlazada contiene la información necesaria para saber la mayor diferencia. Para el cálculo de diferencias hicimos uso de la librería Datetime.

|  |  |
| --- | --- |
| **Entrada** |  |
| **Salidas** | ret = (estado, promedio, año\_max, año\_min,diferencia\_max,  diferencia\_min, census, survey) |
| **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 así crear un diccionario donde las llaves son cada departamento. Si no existe, crea la llave y crea una nueva Single Linked List con sl.new\_list(). En caso de que existe, crea un nodo con información, para añadirlo en la Single Linked List con sl.add\_last(). |
|  | O(i), el diccionario tiene un numero de estados fijo. Por lo tanto, se recorre i veces, siendo i una constante. |
|  | O(N), recorre la lista enlazada hasta el final para realizar las comparaciones y sumatorias necesarias. |
| ***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.

|  |  |
| --- | --- |
| 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 |  | 893.817 |
| 200000 datos |  | 1843.170 |
| 300000 datos |  | 2699.180 |
| 400000 datos |  | 3639.811 |
| 500000 datos |  | 4628.467 |

### **Graficas**

Las gráficas con la representación de las pruebas realizadas.

## **Análisis**

El requerimiento 8 presenta un crecimiento lineal. Simplemente se recorre por completo una primera vez para crear diccionarios de listas enlazadas y luego se recorre el diccionario completo para poder hacer los cálculos necesarios.

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.