ANÁLISIS DEL RETO

Juan Camilo Cancelado, 202410123, j.canceladod@uniandes.edu.co

Gabriela Gomez, 202420506, g.gomezh2@uniandes.edu.co

Pedro Archila, 202421572, p.archila@uniandes.edu.co

Requerimiento 1

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El catálogo con la información de los datos y el año que se quiere filtrar
Salidas	El tiempo que se demoró y una lista que contiene las listas de cada uno de los elementos que cumplen el filtro, la cantidad de elementos que cumplen con el filtro, la cantidad de elementos que cumplen el filtro y son de fuente “SURVEY” y los que son de fuente “CENSUS”.
Implementado (Sí/No)	Si / grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad (O)
Asignación de variables iniciales (start_time, index, count, index_mayor, etc.)	O(1)
Obtener las listas del catálogo (list_load_time, list_year)	O(1)
Iterar sobre los elementos de list_year["elements"]	O(n)
- Convertir el año a entero (int(year_lista.replace(" ", "")))	O(1) por iteración
- Comparar el año con el año objetivo (if int_year == year)	O(1) por iteración
- Obtener y convertir la fecha de carga (dt.strptime(...))	O(1) por iteración
- Comparar fechas (if fecha_mayor is None or fecha_year_dt >= fecha_mayor_dt)	O(1) por iteración
- Actualizar variables (fecha_mayor, fecha_mayor_dt, index_mayor)	O(1) por iteración
Verificar si fecha_mayor es None	O(1)

Obtener las listas adicionales del catálogo (list_source, list_freq, etc.)	O(1)
Crear la lista datos_mayor y agregar elementos	O(1)
Medir el tiempo de ejecución (get_time, delta_time)	O(1)
Retornar los resultados (return general, datos_mayor)	O(1)
TOTAL	O(N)

Pruebas Realizadas

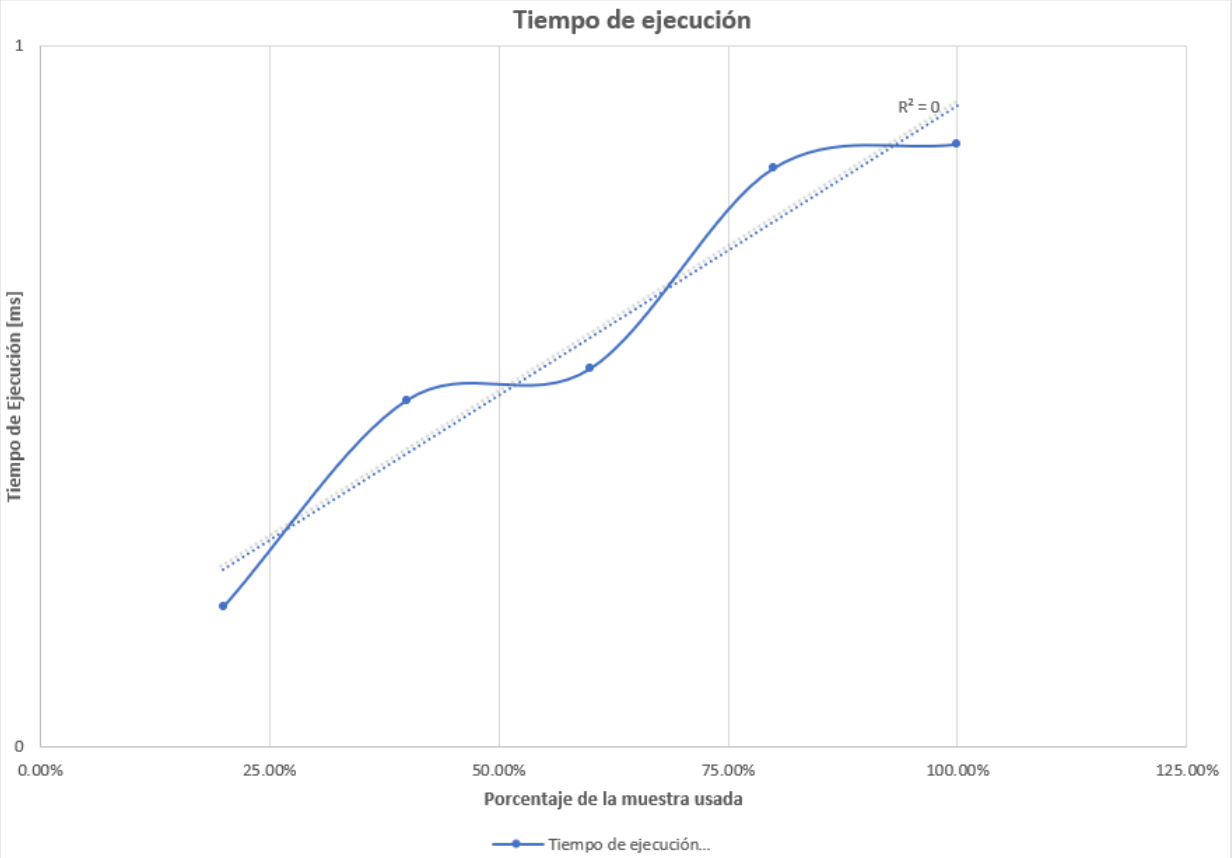
Año = 2007

Procesadores	**Intel core i7 7th gen**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Tablas de datos

Entrada	**Tiempo (ms)**
Agricultural-20	200.79
Agricultural-40	495.95
Agricultural-60	540.74
Agricultural-80	826.75
Agricultural-100	860.42

Graficas



Análisis

Resulta extraño que, por ejemplo, entre las pruebas del 40% y 60% haya un salto tan corto, esto se puede ver justificado por cierta fortuna en el orden de los datos y en la parametrización de la función. Lo cierto es que, en el largo plazo, el comportamiento de la función es lineal y predecible por las tablas de complejidad

Requerimiento 2

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Catalogo con la información de los datos, N que es el número de registros y el nombre del estado que se quiere filtrar
Salidas	El tiempo que se demoró y una lista que contiene las listas de cada uno de los elementos que cumplen el filtro, la cantidad de

	elementos que cumplen con el filtro, la cantidad de elementos que cumplen el filtro y son de fuente “SURVEY” y los que son de fuente “CENSUS”.
Implementado (Sí/No)	Si / Grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos Generales	Complejidad (O)
Inicialización de variables y obtención de listas	O(1)
Iteración sobre list_state["elements"]	O(N)
Normalización y comparación del estado (replace, upper, if estadom == state)	O(1) por iteración
Creación de mapas y adición a la lista de registros (lp.new_map, ar.add_last)	O(N) por iteración
Verificación de registros (if count == 0)	O(1)
Ajuste del valor de N si es mayor al tamaño de records	O(1)
Ordenamiento de registros con merge_sort	O(MlogM) siendo M la cantidad de elementos en la lista de mapas.
Creación de la lista de índices ordenados	O(N)
Iteración sobre los últimos N registros	O(N)
Construcción de la lista de resultados	O(N)
Iteración sobre índices ordenados y obtención de valores (ar.get_element)	O(1) por iteración
- Adición de valores a la lista de resultados (ar.add_last)	O(1) por iteración
Medición del tiempo de ejecución y retorno de resultados	O(1)
TOTAL	O(N+M+MlogM) pero teniendo en cuenta que N (número de datos del archivo completo) es mucho más grande que M se podría considerar que la complejidad total del algoritmo es O(N)

Pruebas Realizadas

Estado = Iowa

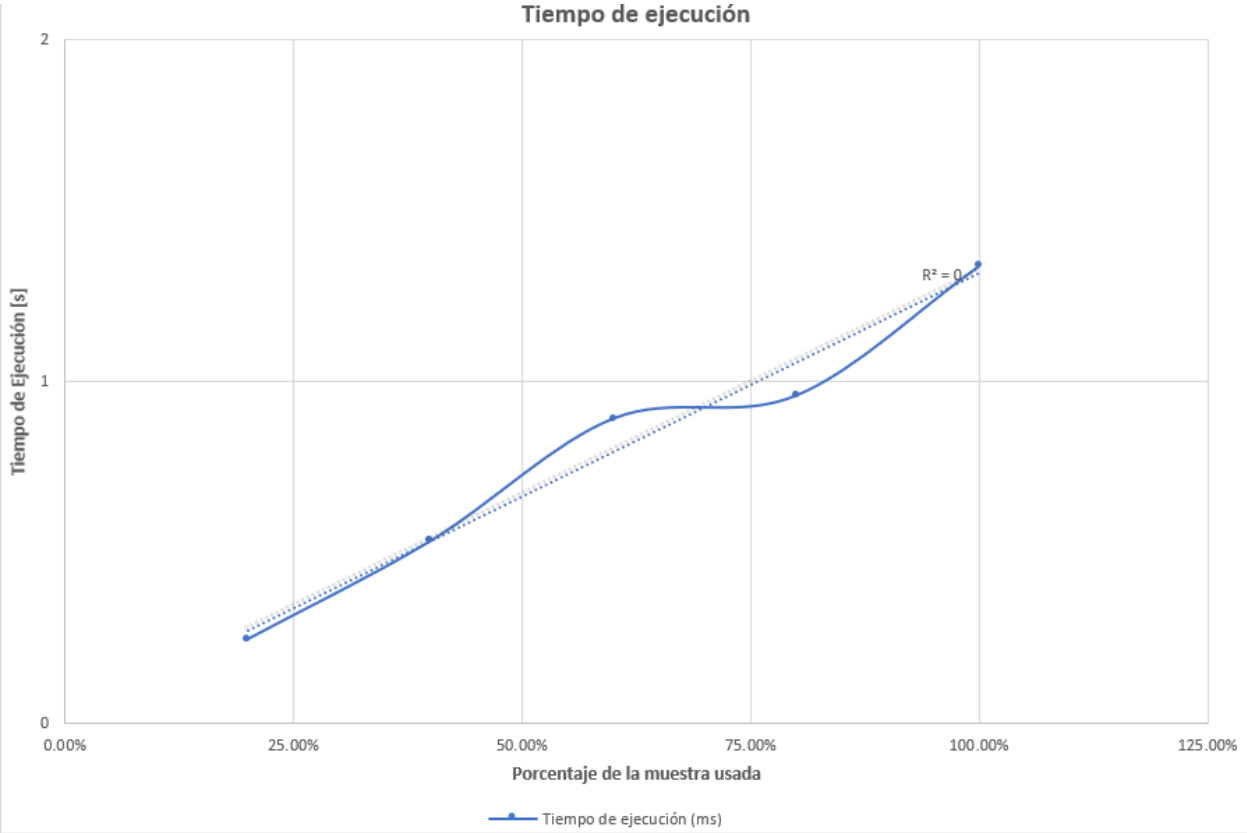
N = 50

Procesadores	**Intel Core i7 7th gen**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Tablas de datos

Entrada	Tiempo (ms)
Agricultural-20	245.40
Agricultural-40	533.28
Agricultural-60	891.84
Agricultural-80	959.13
Agricultural-100	1338.22

Graficas



Análisis

Tiene un comportamiento bastante acorde con lo esperado, un comportamiento casi lineal que crece con pendiente constante

Requerimiento 3

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Año inicial del periodo de tiempo que se quiere revisar, año final del periodo de tiempo que iodo de tiempo que se quiere revisar, el estado por el cual se quiere filtrar la información y el catálogo donde está toda la información
Salidas	Una lista que contiene las listas de cada uno de los elementos que cumplen el filtro, la cantidad de elementos que cumplen con el filtro, la cantidad de elementos que cumplen el filtro y son de fuente “SURVEY” y los elementos que cumplen el filtro y son de fuente “CENSUS”.
Implementado (Sí/No)	Si, Pedro Archiila

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignar variables y diferentes listas a un elemento del mapa del catálogo	O(1)
Pasar por cada uno de los elementos revisando el año de colección y el estado para ver si cumple o no con el filtro dado	O(N)
Crear un nuevo mapa donde se pondrá el tiempo de carga de cada elemento que cumple el filtro y su índice dentro del archivo	O(N) porque como cada mapa tiene una capacidad tan pequeña es insignificante comparado a la cantidad de datos que cumplen un solo filtro
Suma uno al contador de la cantidad de elementos que cumplen el filtro y luego revisa si su fuente es “SOURCE” o “CENSUS” para agregarle uno al contador respectivo	O(N)
Pone la fecha de carga y el índice dentro del mapa creado en el paso 3	O(N)
Mete el mapa de un solo dato al final de una lista que contendrá las tablas de hash de todos los elementos que cumplen el filtro	O(N)
Si la lista que contiene los mapas de cada elemento NO está vacía organiza esa lista con respecto a la fecha de carga usando merge_sort	O(MlogM) siendo M la cantidad de elementos en la lista de mapas.
Entra a la lista de mapas ya ordenada y mete cada índice y lo pone en una lista de índices ordenados con respecto a la fecha de carga	O(M)
Por cada uno de los índices organizados busca el valor en dicha posición en cada una de las listas nombradas en el paso 1 y mete cada valor dentro de una lista de python y luego mete esa lista de python al final de un	O(M)

array_list que tendrá todos los valores, con todos los datos necesarios en el orden deseado	
TOTAL	***O(N+M+MlogM) pero teniendo en cuenta que N (número de datos del archivo completo) es mucho más grande que M se podría considerar que la complejidad total del algoritmo es O(N)***

Pruebas Realizadas

Las pruebas se hacen con los siguientes datos:

- Año inicial = 1950
- Año final = 2024
- Estado = Nebraska

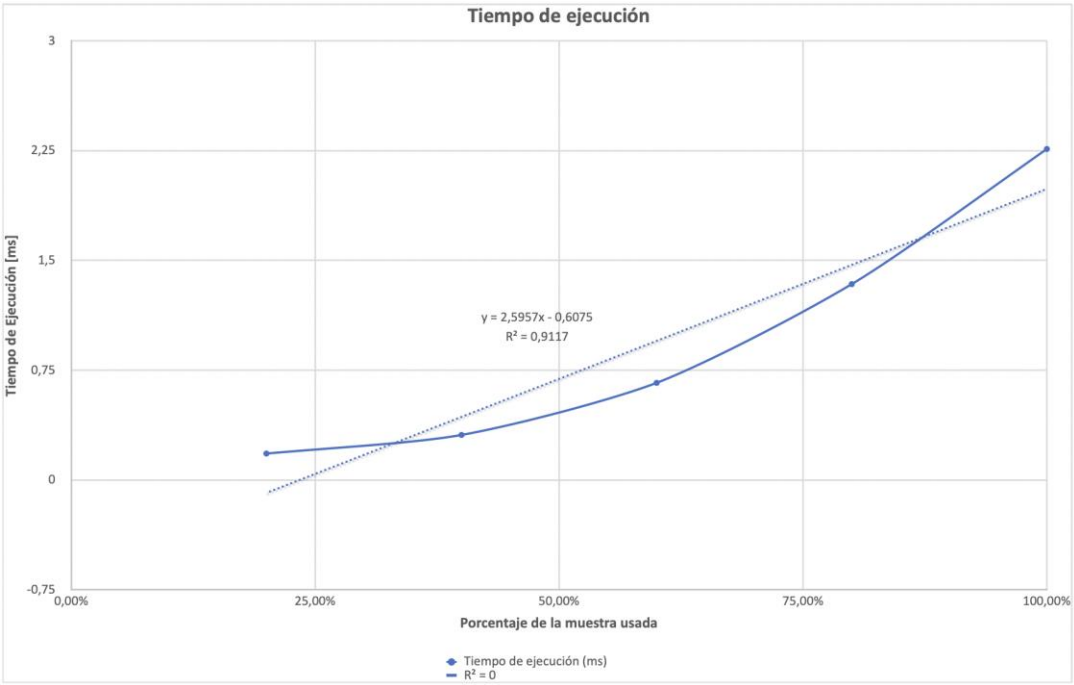
Se utilizan mapas, array_list, merge_sort, y librerías como datetime para poder comparar las fechas de carga y tabulate para imprimir los datos

Procesadores	**Chip M1**
Memoria RAM	8 GB
Sistema Operativo	MacOS sequoia 15.1

Tablas de datos

Entrada	**Tiempo (s)**
Agricultural-20	0,18027
Agricultural-40	0,30743
Agricultural-60	0,66327
Agricultural-80	1,33777
Agricultural-100	2,26084

Graficas



Análisis

Según la tabla el algoritmo tiene un comportamiento cuadrático lo cual no corresponde con lo que se puede ver en el código ni con lo relacionado en la tabla teórica anterior. La gráfica según el código debería tener un comportamiento logarítmico (gracias al merge_sort) o lineal, pues en lo que tarda más tiempo es en recorrer todos los datos revisando si cumplen con el filtro o no. Sin embargo, este comportamiento también deja ver el efecto que tiene el tamaño de M en el comportamiento del algoritmo.

Requerimiento 4

Descripción

Entrada	El catálogo con la información de los datos, el tipo de producto que se quiere filtrar y el rango de años en el que se quiere filtrar
Salidas	Una lista con el tiempo que se demoró, el número total de registros filtrados, y la cantidad de elementos que cumplen el filtro y son de fuente “SURVEY” y los que son de fuente “CENSUS”. También una lista con listas adentro (de los elementos que cumplen el filtro) las cuales contienen el tipo de fuente, el año, el tiempo de carga, la frecuencia, el departamento y la unidad.
Implementado (Sí/No)	Si / Juan Camilo Cancelado

Análisis de complejidad

Pasos Generales	Complejidad (O)
Inicialización de variables y obtención de listas	O(1)
Iteración sobre productos para filtrar por producto y rango de años	O(N)
- Creación de mapas y adición a la lista de registros logic.py), logic.py))	O(N) por iteración
Verificación de registros encontrados ([`if count == logic.py))	O(1)
Ordenamiento de registros con logic.py)	O(MlogM) siendo M la cantidad de elementos en la lista de mapas.
Creación de la lista de índices ordenados	O(N)
Construcción de la lista de resultados	O(N)
- Iteración sobre índices ordenados y obtención de valores logic.py))	O(1) por iteración
- Adición de valores a la lista de resultados logic.py))	O(1) por iteración
Medición del tiempo de ejecución y retorno de resultados	O(1)
TOTAL	O(N+M+MlogM) pero teniendo en cuenta que N (número de datos del archivo completo) es mucho más grande que M se podría considerar que la complejidad total del algoritmo es O(N)

Pruebas Realizadas

Producto = Cattle

Inicial = 1985

Final = 2020

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Tablas de datos

Entrada	**Tiempo (ms)**
Agricultural-20	2590.62

Agricultural-40	5489.97
Agricultural-60	8721.07
Agricultural-80	21,250
Agricultural-100	41,151

Graficas

Análisis

El comportamiento casi cuadratico que se observa es extraño considerando la aproximación mediante el análisis. Lo que se puede considerar es que en este caso, como tomamos un rango tan grande de tiempo, el valor de M se acerca al valor de N mucho más que en otras pruebas, esto hace que la complejidad esté más cerca de ser NlogN. Como se puede ver en la gráfica, este es el comportamiento que adopta nuestra función.

Requerimiento 5

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El catálogo con la información de los datos, la categoría que se quiere filtrar y el rango de años en el que se va a filtrar
Salidas	Una lista con el tiempo que se demoró, el número total de registros filtrados, y la cantidad de elementos que cumplen el filtro y son de fuente “SURVEY” y los que son de fuente “CENSUS”. También una lista con listas adentro (de los elementos que cumplen el filtro) las cuales contienen el tipo de fuente, el año, el tiempo de carga, la frecuencia, el departamento, la unidad y el tipo de producto.
Implementado (Sí/No)	Si / Gabriela Gomez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignación de variables y listas con elementos del catálogo usando	O(1)
Recorrer la lista de categorías y evaluar si la categoría y el año cumplen el filtro	O(N), donde N es la cantidad de registros
Obtener valores específicos como `load_time`, `state_name`, y `index` para los registros que cumplen el filtro	O(N)
Crear un nuevo mapa para cada registro filtrado y agregarlo a la lista	O(N)
Contar elementos SURVEY y CENSUS dentro del ciclo principal	O(N)
Ordenar la lista `registros` usando `merge_sort`	O(M log M), donde M es la cantidad de elementos que cumplen el filtro
Extraer los índices de los registros ordenados y agregarlos a `registros_finales`	O(M)
Crear la lista `resultados` con la información final recorriendo los índices ordenados	O(M)
Calcular el tiempo de ejecución y crear la lista `general` con los resultados finales	O(1)
TOTAL	***O(N + M logM) pero debido a que N es generalmente mayor que M. Por lo que la complejidad dominante es O(N)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**Intel core i5**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

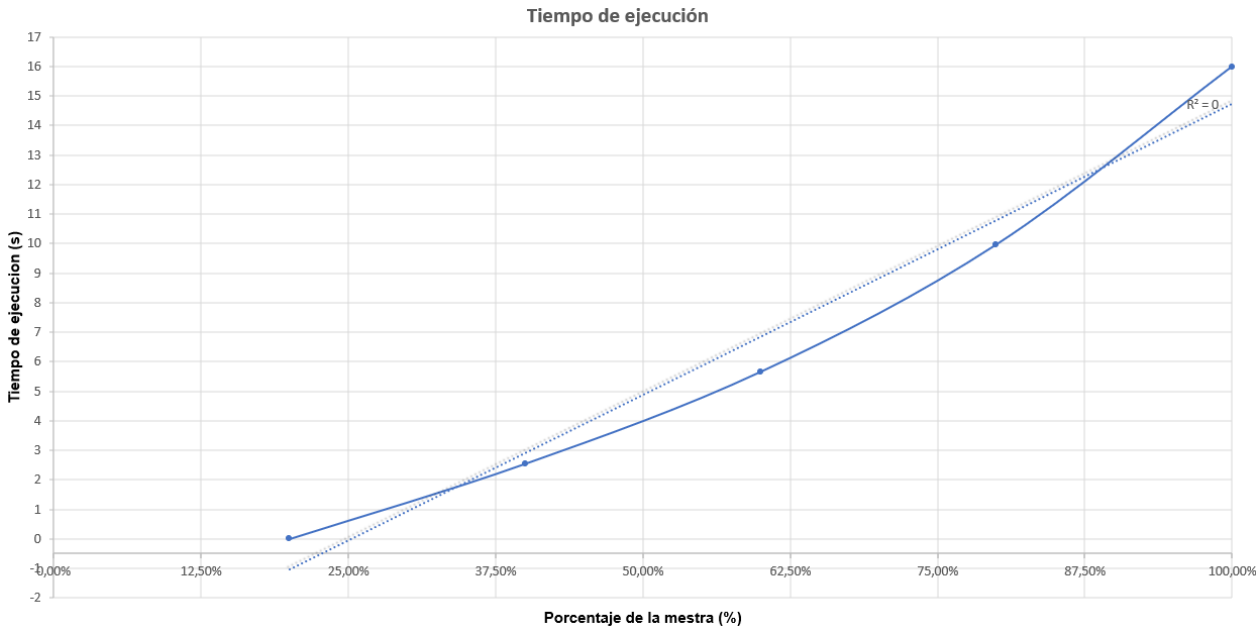
Tablas de datos

Entrada	**Tiempo (s)**
Agricultural-20	0,84799 s
Agricultural-40	2,54348 s
Agricultural-60	5,65968 s
Agricultural-80	9,96461 s
Agricultural-100	15,98791 s

- Categoria : Slaughtered
- Fecha inicio: 1980
- Fecha final : 2020

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Segun la grafica, se puede ver como el tiempo de ejecución del requerimiento 5 sigue una tendencia casi lineal debido a la complejidad O(N + M log M). La mayor parte del tiempo se debe al recorrido inicial de O(N), mientras que la ordenación de los registros filtrados en O(M log M) tiene un impacto creciente a medida que M aumenta. Además, la gráfica confirma este comportamiento, mostrando un crecimiento lineal con una ligera curvatura cuando M es significativo. Esto indica que la implementación es eficiente y se alinea con la complejidad teórica esperada.

.

Requerimiento 6

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Catálogo que contiene todos los datos de todo el archivo que se carga, estado que se quiere filtrar, fecha inicial en formato YYYY-MM-DD del rango de fechas de carga que se quieren filtrar y la fecha final en formato YYYY-MM-DD del rango de fechas de carga que se quieren filtrar.
Salidas	Una lista que contiene fuente, año de recolección, fecha de carga, frecuencia de recolección, estado, unidad de medida y producto. Un número entero que representa el número total de elementos que completaron el filtro. Otro número entero que representa la

	cantidad de elementos que cumplen el filtro y tienen como fuente “SURVEY”. Otro número entero que representa la cantidad de elementos que cumplen el filtro y tienen como fuente ”CENSUS”. Y finalmente el tiempo de ejecución del algoritmo
Implementado (Sí/No)	Si/ grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Asignar variables y diferentes listas a un elemento del mapa del catálogo	O(1)
Pasar por cada uno de los elementos revisando la fecha de carga y el estado para ver si cumple o no con el filtro dado	O(N)
Crear un nuevo mapa donde se pondrá el tiempo de carga de cada elemento que cumple el filtro y su índice dentro del archivo	O(N) porque como cada mapa tiene una capacidad tan pequeña es insignificante comparado a la cantidad de datos que cumplen un solo filtro
Pone la fecha de carga y el índice dentro del mapa creado en el paso 3	O(N)
Mete el mapa de un solo dato al final de una lista que contendrá las tablas de hash de todos los elementos que cumplen el filtro	O(N)
Suma uno al contador de la cantidad de elementos que cumplen el filtro y luego revisa si su fuente es “SOURCE” o “CENSUS” para agregarle uno al contador respectivo	O(N)
Si la lista que contiene los mapas de cada elemento NO está vacía organiza esa lista con respecto a la fecha de carga usando merge_sort	O(MlogM) siendo M la cantidad de elementos en la lista de mapas.
Entra a la lista de mapas ya ordenada y mete cada índice y lo pone en una lista de índices ordenados con respecto a la fecha de carga	O(M)
Por cada uno de los índices organizados busca el valor en dicha posición en cada una de las listas nombradas en el paso 1 y mete cada valor dentro de una lista de python y luego mete esa lista de python al final de un array_list que tendrá todos los valores, con todos los datos necesarios en el orden deseado	O(M)
TOTAL	***O(N+M+MlogM) pero teniendo en cuenta que N (número de datos del archivo completo) es mucho más grande que M se podría considerar que la complejidad total del algoritmo es O(N)***

Pruebas Realizadas

Las pruebas se hacen con los siguientes datos:

- Fecha inicial = 1950-01-01
- Fecha final = 2024-12-31
- Estado = Michigan

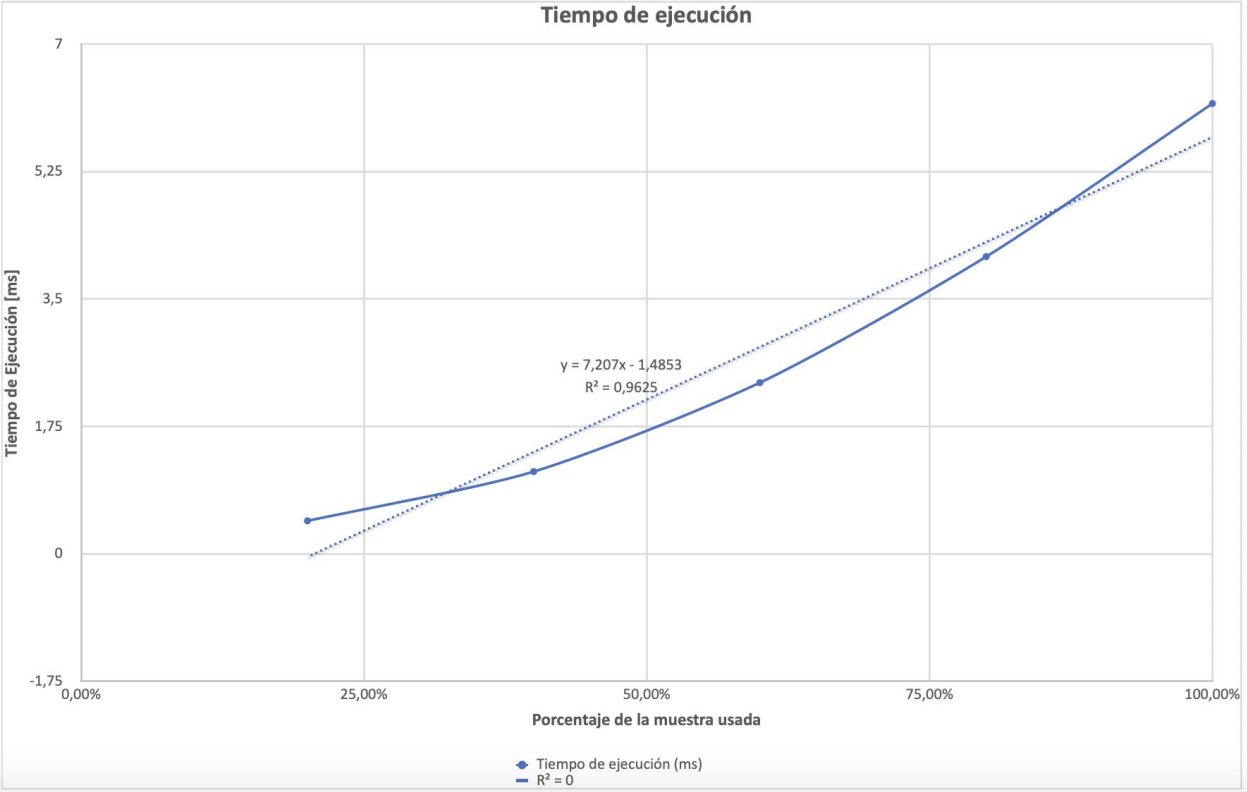
Se utilizan mapas, array_list, merge_sort, y librerías como datetime para poder comparar las fechas de carga y tabulate para imprimir los datos

Procesadores	**Chip M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sequoia 15.1

Tablas de datos

Entrada	**Tiempo (s)**
Agricultural-20	0,452205
Agricultural-40	1,138619
Agricultural-60	2,350124
Agricultural-80	4,080318
Agricultural-100	6,183355

Graficas



Análisis

En esta gráfica se puede ver un comportamiento parabólico. Sin embargo, esta gráfica parece estarse acercando a algo lineal. Al igual que en el requerimiento 3, la gráfica no muestra lo que se puede ver en el análisis del código en cuanto a su complejidad pues según este, la gráfica debería mostrar casi una recta con un comportamiento medianamente logarítmico, cosa que no ocurre. Finalmente, este comportamiento parabólico demuestra el efecto que tiene M sobre el algoritmo, no obstante, este no es el efecto esperado. # Requerimiento 7

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	El catálogo con la información de los datos, el departamento a filtrar, el orden en el que se desea organizar y el rango de años para filtrar
Salidas	Se retorna una lista con el número de registros totales que cumplen con el filtro y el tiempo que dura la funcion. Despues otra lista con diccionarios los cuales están ordenados por el valor de ingresos sino por el numero de registros. Estas lista tiene diccionarios que tienen

	el año, si es mayor o menor, el numero total de ingresos con el filtro, el número total de registros inválidos, y por ultimos la cantidad de registros con survey y de census.
Implementado (Sí/No)	Si / grupal

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear el mapa	O(1)
Inicialización de variables	O(1)
Acceso a listas desde el catalogo	O(1)
Iteración sobre la lista de estados	O(N)
Acceso a elementos dentro del bucle	O(N)
Condicional de validación de valores	O(N)
Verificar y añadir diccionario en el mapa	O(N)
Actualización de valores en el mapa, específicamente en el diccionario correspondiente	O(1)
Contador de los tipos de fuentes	O(N)
Creación de la lista_datos desde los valores del mapa	O(M)
Ordenamiento con merge_sort	O(M log M)
Inversión de la lista dado de que si el orden fuera ascendente	O(M)
Encontrar el primero y ultimo elemento dependiendo de cómo esten ordenados	O(1)
Creacion de las las tres listas (primer_lista,ultimo_lista,lista_intermedia)	O(M)
Retorno de los resultados empaquetados con los valores finales	O(1)
TOTAL	***O(N+M+MlogM) pero teniendo en cuenta que N (número de datos del archivo completo) es mucho más grande que M se podría considerar que la complejidad total del algoritmo es O(N)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Procesadores	**Intel Core i5**
Memoria RAM	8 GB
Sistema Operativo	Windows 11

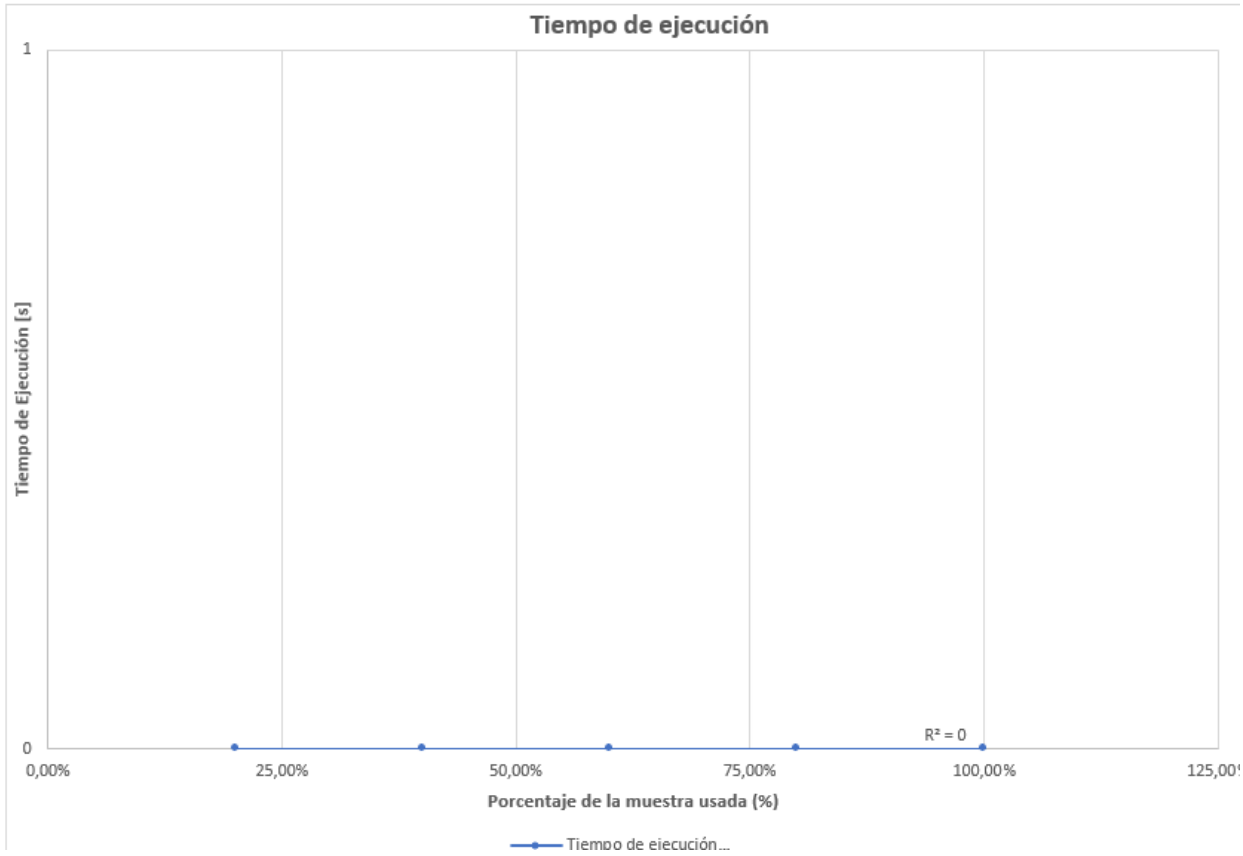
Tablas de datos

Entrada	Tiempo (s)
Agricultural-20	0,05943 s
Agricultural-40	0,16384 s
Agricultural-60	0,45213 s
Agricultural-80	0,65361 s
Agricultural-100	0.82998 s

- Estado = Arkansas
- Año inicio = 1990
- Año final = 2022
- Orden = Ascendente

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Esta gráfica muestra que el tiempo de ejecución de este requerimiento crece de forma lineal con el tamaño de los datos, lo que confirma una complejidad O(N) en la mayoría de las operaciones. Sin embargo, el uso de `merge_sort` añade una complejidad de O(M log M) en el ordenamiento de los resultados. Aunque el uso de mapas optimiza el acceso y almacenamiento, en volúmenes grandes de datos, el tiempo de ordenamiento podría afectar el rendimiento general. Sin embargo, dado que N (cantidad total de datos) es mucho mayor que M (cantidad de datos filtrados), la complejidad general se considera O(N).

Requerimiento Ejemplo

Descripción

```python
def get_data(data_structs, id):
    """
    Retorna un dato a partir de su ID
    """
    pos_data = lt.isPresent(data_structs["data"], id)
    if pos_data > 0:
        data = lt.getElement(data_structs["data"], pos_data)
        return data
    return None
```

Este requerimiento se encarga de retornar un dato de una lista dado su ID. Lo primero que hace es verificar si el elemento existe. Dado el caso que exista, retorna su posición, lo busca en la lista y lo retorna. De lo contrario, retorna None.

Entrada	Estructuras de datos del modelo, ID.
Salidas	El elemento con el ID dado, si no existe se retorna None
Implementado (Sí/No)	Si. Implementado por Juan Andrés Ariza

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar si el elemento existe (isPresent)	O(n)
Obtener el elemento (getElement)	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms)**
small	0.05
5 pct	0.33
10 pct	1.28
20 pct	2.54
30 pct	4.98
50 pct	7.51
80 pct	13.81
large	25.97

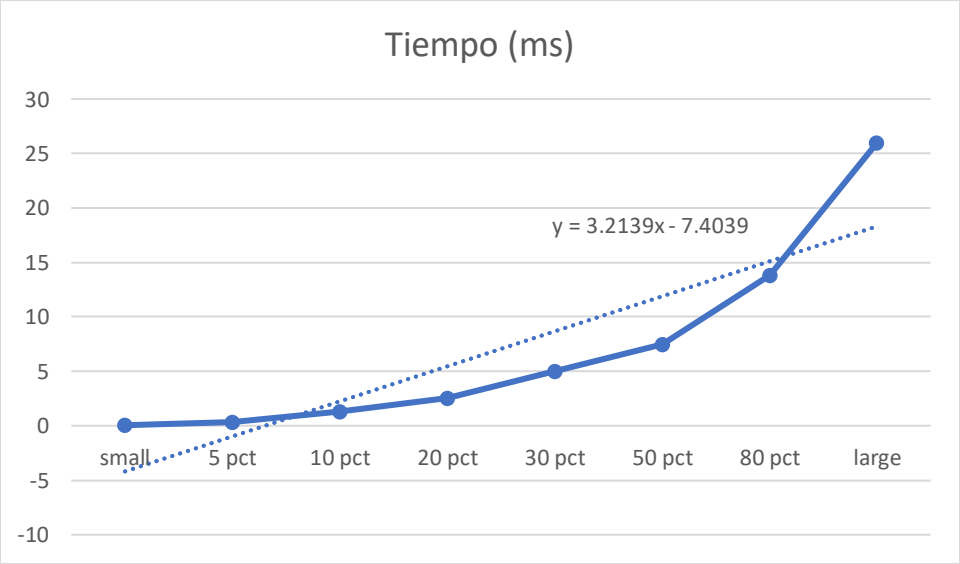
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	0.05
5 pct	Dato2	0.33
10 pct	Dato3	1.28
20 pct	Dato4	2.54
30 pct	Dato5	4.98
50 pct	Dato6	7.51
80 pct	Dato7	13.81
large	Dato8	25.97

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.