Documento de análisis

Estructura de Datos y Algoritmos

Juan Camilo Neira Campos 201922746 j.neira@uniandes.edu.co

Daniel Dorado Toro 201821010

Análisis de complejidad de cada uno de los requerimientos.

1. Requerimiento 1.

Este requerimiento se solucionó en dos partes. Primero, el ordenamiento de la estructura de datos arrayList por medio de un algoritmo de ordenamiento recursivo *mergesort*. Segundo, el empleo de una búsqueda binaría para hallar las posiciones de los extremos del rango de tiempo. Cuando se halla estas posiciones, se le solicita al arrayList que retorne todos los valores que están en la posición inicial hasta la posición final. En ambos casos, se implementó de tal forma que la operación tenga la menor complejidad temporal posible. Por ejemplo, el algoritmo de ordenamiento *mergesort* tiene una complejidad temporal en todos los casos de O(n log(n)). Dado que el reto solicita que la respuesta sea lo más rápido posible, no se consideró la complejidad espacial de este algoritmo que es O(n). Asimismo, la búsqueda binaria es una forma eficiente de buscar un elemento, especialmente en un arrayList, ya que retornar un elemento si tenemos su posición es O(1). La búsqueda binaría tiene una complejidad temporal de O(log(n)). El requerimiento 1 y 2 se solucionó de la misma forma, y la única diferencia es el criterio de ordenamiento y búsqueda que se utilizó, a saber, en el primer caso la fecha de nacimiento del artista y en el segundo caso el año de compra de la obra de arte.

2. Requerimiento 2.

Este requerimiento se solucionó en dos partes. Primero, el ordenamiento de la estructura de datos arrayList por medio de un algoritmo de ordenamiento recursivo *mergesort*. Segundo, el empleo de una búsqueda binaría para hallar las posiciones de los extremos del rango de tiempo. Cuando se halla estas posiciones, se le solicita al arrayList que retorne todos los valores que están en la posición inicial hasta la posición final. En ambos casos, se implementó de tal forma que la operación tenga la menor complejidad temporal posible. Por ejemplo, el algoritmo de ordenamiento *mergesort* tiene una complejidad temporal en todos los casos de O(n log(n)). Dado que el reto solicita que la respuesta sea lo más rápido posible, no se consideró la complejidad espacial de este algoritmo que es O(n). Asimismo, la búsqueda binaria es una forma eficiente de buscar un elemento, especialmente en un arrayList, ya que retornar un elemento si tenemos su posición es O(1). La búsqueda binaría tiene una complejidad temporal de O(log(n)).

3. Requerimiento 3.

Para el requerimiento 3, el catálogo se cargó con lista de artistas, una de obras y otra lista de IDs, todas con estructura de arreglo. Esta lista contiene diccionarios con tres llaves cada uno: ID del artista, obras y nacionalidad. Al llamar la función que implementa el requerimiento, se usa Merge Sort para ordenar la lista de IDs. Merge Sort tiene complejidad O(n log(n)), lo cual suele ser muy bueno. Luego, de la lista de artistas se encuentra la entrada cuyo nombre coincida con el input del usuario y se obtiene su ID. Este procedimiento es O(n). Con busqueda binaria, que es O(log n), se ubica la entrada en la lista de IDs correspondiente al artista buscado. Como la lista es un arreglo, acceder a este elemento es O(1). El valor asociado a la llave de obras es una lista con las obras. Al iterar sobre esta lista se ubican todas las técnicas y se cuenta cuantas obras hay para cada técnica. Esto es O(n) en el peor de los casos. Luego, se itera sobre las técnicas encontradas, buscando la que tenga más obras, lo cual es O(n) también en el peor de los casos y finalmente se itera sobre las obras del artista y se seleccionan únicamente las obras de la técnica más común.

4. Requerimiento 4.

El requerimiento cuatro se solucionó de la siguiente forma. Al momento de crear los datos, se creó una llave llamada 'ConstituentIDs', la cual contiene una lista vacío. En este se almacenó los id de los artistas con su correspondiente nacionalidad y sus obras de arte en un arrayList. Consideré que esta es la mejor forma de almacenar los datos para posteriormente encontrar el TOP 10 de países con más obras de arte. Para ello, utilicé un diccionario como estructura de datos auxiliar, con el fin de crear llaves que son las nacionalidad y el valor es el número de veces que dicha nacionalidad aparece en una obra de arte. La lógica es la siguiente: si el artista tiene una determinada nacionalidad, retorna el tamaño del arrayList de sus obras y se suma a la correspondiente llave en el diccionario auxiliar. Retornar el tamaño del arrayList es O(1), lo cual es mejor que un O(n) en el que se recorre está lista para encontrar su tamaño o peor aún un O(n^2) en el que se busca entre todas las obras dónde se encuentra el artista de esta nacionalidad. Por último, dado que ya tengo el TOP1 de nacionalidad, comparo en la llave 'id' si ese artista es de esa nacionalidad y si lo es, retorna una arrayList que contiene sus obras. Este es de complejidad temporal O(n).

5. Requerimiento 5.

El requerimiento numero 5 se solucionó de la siguiente forma. Se recorre la lista artworks, es decir que es O(n) y se compara si esa obra de arte pertenece al departamento en cuestión. Si es así, se calcula el costo de transporte en la función costArtwork en el model. Posteriormente se incluye una llave en el diccionario de esta obra que es la pareja (costo, costo en dólares) y cada una de estas obras se almacena en un arrayList. Después se utiliza dos algoritmos de ordenamiento recursivos mergesort, los cuales tienen una complejidad temporal O(log(n)). Con estos algoritmos se ordenó el arrayList por criterio de fecha y de costo, y se imprimió los primeros cinco.

Pruebas de tiempos de ejecución para cada uno de los requerimientos.

	Máquina
Procesadores	Intel® Core™ i5 Dual-Core @ 2.5 GHz
Memoria RAM (GB)	8.0 GB
Sistema Operativo	macOS Catalina

Tabla 1. Especificaciones de la máquina para ejecutar las pruebas de rendimiento.

Porcentaje de la muestra [pct]	small	10	30	80	large
Tiempo					

Tabla 2. Tiempos de ejecución del requerimiento 1.

Porcentaje de la muestra [pct]	small	10	30	80	large
Tiempo					

Tabla 3. Tiempos de ejecución del requerimiento 2.

Porcentaje de la muestra [pct]	small	10	30	80	large
Tiempo					

Tabla 4. Tiempos de ejecución del requerimiento 3.

Porcentaje de la muestra [pct]	small	10	30	80	large
Tiempo					

Tabla 5. Tiempos de ejecución del requerimiento 4.

Porcentaje de la muestra [pct]	small	10	30	80	large
Tiempo					

Tabla 6. Tiempos de ejecución del requerimiento 5.

En caso de los requerimientos individuales, indicar quién lo implementó.

Requerimiento 3: Daniel Dorado.

Requerimiento 4: Juan Camilo Neira.