RETO 2- ESTRUCTURA DE DATOS Y ALGORITMOS

Integrantes:

- **Nombre:** Juan Sebastián Sánchez Delgado
- **Correo**: js.sanchezd1@uniandes.edu.co
- **Código:** 2020135577

- **Nombre:** Nicolás Alexander rodríguez Pinilla
- **Correo:** na.rodriguezp1@uniandes.edu.co
- **Código:** 20202250

ANALISIS DE COMPLEJIDAD Y PRUEBAS DE EJECUCION:

REQUISITO 1:

La función listarCronologicamente crea un map denominado mapEnRango y agrega a este último parejas que tiene como llave el nombre del artista y como valor la información del mismo. Para ello se debió haber recorrido la lista con la información de los artistas una vez, por lo que la complejidad hasta este punto es O(N), siendo N el número de elementos de la lista original. Hasta este punto, la complejidad temporal concuerda con la implementación hecha con listas.

Luego, se crea un segundo map llamado mapFinal, donde se agregará la información de las parejas del mapEnRango pero de manera ordenada. Es en este punto en el que se evidencia la primera diferencia clara con respecto a la implementación con listas. Debido a que, no existe una noción de orden sin importar que las parejas se agreguen de forma ordenada, para saber la posición que ocupa cada pareja en el intervalo se utilizó como llave para cada una un numero único que identificara este parámetro. En el peor caso, todos los elementos de la lista estarán situados también en el primer map que fue creado, así que la complejidad se seguirá expresando en términos de N. El programa recorrerá el map mapFinal en su totalidad, el número de veces en que lo hará está definido por la siguiente expresión: añoFinal - añoInicial.

En conclusión, la complejidad temporal del programa es N + (añoFinal - añoInicial)N, que en notación O vendría siendo: O(N). Si bien la complejidad en notación O es la misma que se obtuvo para la otra implementación, el factor diferenciador que hace de esta segunda solución como una opción menos viable es la memoria.

En teoría, la memoria extra utilizada es de aproximadamente 2N, pero en la práctica eso no resulta ser así. Cuando se crea un map se debe especificar la cantidad de elementos que aproximadamente se guardaran, como existen varios tipos de archivos y muchas posibles variantes de intervalos, es imposible determinar con exactitud el tamaño del map. Aunque la librería cuenta con una opción para ampliar el tamaño de este, aun así, en la gran mayoría de casos siguen existiendo varios espacios de la tabla sin ocupar.

Todo lo anterior provoca que los tamaños de los maps casi siempre sean mayores a las listas creadas para la implementación del reto 1, por lo que los tiempos obtenidos para el reto 2 resultan ser considerablemente mayores que los del reto 1. A continuación se muestran algunas pruebas hechas acerca del funcionamiento de esta implementación.

PRUEBA

* Las pruebas mostradas a continuación se hicieron con los archivos de extensión “large”.

Sistema operativo: Mac OS

Procesador: 1,6 GHz Intel Core i5 de dos núcleos

Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1(con maps):

Año inicial: 1920

Año final: 1985

Tiempo de ejecución aproximado: 4,03s

Prueba 1 (con listas):

Año inicial: 1920

Año final: 1985

Tiempo de ejecución aproximado: 0,93s

Prueba 2 (con maps):

Año inicial: 1850

Año final: 2015

Tiempo de ejecución aproximado: 15,13s

Prueba 2 (con listas):

Año inicial: 1850

Año final: 2015

Tiempo de ejecucion aproximado: 1,58s

Prueba 3 (con maps):

Año inicial: 1700

Año final: 2021

Tiempo de ejecución aproximado: 28,12s

Prueba 3 (con listas):

Año inicial: 1700

Año final: 2021

Tiempo de ejecucion aproximado: 3,22s

Los resultados muestran tiempos más o menos acordes con la complejidad temporal obtenida y los problemas de memoria extra que fueron planteados anteriormente. Sin embargo, teniendo en cuenta la cantidad de información, los tiempos que se obtuvieron fueron óptimos. Como el tamaño de las estructuras es mayor, la cantidad de iteraciones aumenta significativamente

Es bastante claro que la implementación del reto 1 es mucho más eficiente que la obtenida con el uso de maps, esto sumado al hecho de que los maps desordenados no son muy eficientes a la hora de buscar en intervalos, hace de la implementación con listas la solución más eficiente hasta el momento. En conclusión, lo resultados de esta implementación por sí solo no es malo, pero queda bastante atrás comparada con la del reto 1.

REQUISITO 2:

La función listarAdqusiciones crea un map denominado mapFechas y agrega a este último parejas que tiene como llave el título de la obra y como valor la información de la misma. Para ello se debió haber recorrido la lista con la información de las obras una vez, por lo que la complejidad hasta este punto es O(N), siendo N el número de elementos de la lista original.

Luego, se crea un segundo map llamado Adquisiciones, donde se agregará la información de las parejas del mapFechas pero de manera ordenada. Al igual que con el requisito anterior, se nota una clara diferencia con las listas, por el mismo problema de cómo se agregan las parejas.

La complejidad temporal del requisito es N + (FechaFinal - FechaInicial)N, que en notación O vendría siendo: O(N). Siendo la memoria de nuevo una razón del porqué es menos viable, ya que los maps por lo general sean mayores que las listas y con los tiempos considerablemente mayores.

A continuación se muestran algunas pruebas hechas acerca del funcionamiento de esta implementación.

PRUEBA

* Las pruebas mostradas a continuación se hicieron con los archivos de extensión “large”

Sistema operativo: Windows

Procesador: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz

Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1(con maps):

Fecha inicial: 1980-09-10

Fecha final: 1985-10-09

Tiempo de ejecución aproximado: 2,53s

Prueba 1 (con listas):

Fecha inicial: 1980-09-10

Fecha final: 1985-10-09

Tiempo de ejecución aproximado: 1,02s

Prueba 2 (con maps):

Fecha inicial: 1990-08-09

Fecha final: 2015-09-10

Tiempo de ejecución aproximado: 25,13s

Prueba 2 (con listas):

Fecha inicial: 1990-08-09

Fecha final: 2015-09-10

Tiempo de ejecucion aproximado: 3,58s

Prueba 3 (con maps):

Fecha inicial: 1700-10-09

Fecha final: 2020-07-05

Tiempo de ejecución aproximado: 57,32s

Prueba 3 (con listas):

Fecha inicial: 1700-10-09

Fecha final: 2020-07-05

Tiempo de ejecucion aproximado: 5,22s

REQUISITO 3 (Nicolás Alexander rodríguez Pinilla):

El requisito fue solucionado usando dos funciones en el model.py. La primera se llama catalog_id la cual recibe el catálogo de los artistas y el nombre del artista a buscar, en esta función se compara el nombre dado por el usuario con el objetivo de ver si coincide con el "DisplayName" del artista, de ser así, retorna la id del artista para así más tarde usarse, tiene la complejidad de O(N).

Dentro de la función clasificar_tecnica se recibe la id obtenida en la función anterior y el catálogo de las obras, en donde a través del ConstituientID de los artistas, el cual coincide con un elemento del mismo nombre en las obras, se asigna el artista a las obras, guardándose esa información en un map, también con complejidad de O(N).

Desde el map se cuenta el medio con mayor repetición y se retorna, y finalmente desde el print se guarda el tamaño en una variable y se recuperan los primeros 3 y últimos tres elementos de la lista, al recorrerse una vez, está también tiene la complejidad de O(N).

PRUEBA

* Las pruebas mostradas a continuación se hicieron con los archivos de extensión "large"

Sistema operativo: Windows

Procesador: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz

Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1(con maps):

Nombre: Joseph Schwartz

Tiempo de ejecución aproximado: 18,53s

Prueba 1 (con listas):

Nombre: Joseph Schwartz

Tiempo de ejecución aproximado: 1,02s

Prueba 2 (con maps):

Nombre: Mary Callery

Tiempo de ejecución aproximado: 14,13s

Prueba 2 (con listas):

Nombre: Mary Callery

Tiempo de ejecucion aproximado: 3,58s

Prueba 3 (con maps):

Nombre: Lawrence Lam

Tiempo de ejecución aproximado: 17,32s

Prueba 3 (con listas):

Nombre: Lawrence Lam

Tiempo de ejecucion aproximado: 5,22s

REQUISITO 4 (Juan Sebastián Sánchez Delgado):

El requisito en cuestión fue implementado utilizando en total 3 funciones en el model.py. La primera y principal se denomina nacionalidadCreadores, en ella primero se recorre la lista de los artistas una sola vez, esto con el objetivo de crear un map dentro que contenga como llaves los identificadores de los artistas y como valores unos arreglos con el nombre del artista y su nacionalidad. Este map se denomina info.

Dentro del primer for también se identifican las diferentes nacionalidades existentes y se agregan al map llamado catalog[“Nationality”], esto con él objetivo de realizar el conteo de las obras de cada nacionalidad. Como el primer ciclo solo recorrerá la lista una vez independientemente del caso, se puede afirmar que tiene una complejidad de O(N), siendo N el número de elementos de la lista.

En la misma función, existe un segundo ciclo for que recorre toda la lista de obras una sola vez. Este se encarga de evaluar el ConstituentID de cada una de las obras, para posteriormente consultar en el map info la nacionalidad del artista que se identifica por dicho id. Finalmente dependiendo de la nacionalidad, se sumará 1 a su respectivo semejante en el map catalog[“Nationality”] de los países. Este bloque de código también cuenta con una complejidad de O(N).

Se puede concluir que la complejidad temporal para toda la función será O(N). Suponiendo que cada artista tenga una nacionalidad distinta y que una pareja llave-valor ocupa 1 espacio en memoria, en el peor caso, se estaría utilizando aproximadamente N + 3N. A pesar de que la memoria extra que en teoría se usa es igual a la implementación con listas, al igual que como se mencionó en el requisito 1, en realidad, la memoria usada será casi siempre mayor, puesto

que al no saber de antemano el tamaño exacto del map siempre quedaran espacios vacíos en la tabla, que igual deben ser recorridos.

La función sortPaises recorre 10 veces map (que viene siendo catalog[“Nationality”]) para encontrar las nacionalidades con el mayor número de obras. Por ello su complejidad temporal aproximada seria 10N sin importar el caso o O(N). La memoria extra que usa siempre es 10, aunque como el elemento que agrega a la lista lo va removiendo del map, se puede decir que la función en realidad no ocupa memoria adicional. Para este caso en concreto, como se sabe con anterioridad el número de elementos, no debería quedar ningún espacio en la tabla vacío utilizando “CHAINING” con un factor de carga de 1. Al final se obtiene un nuevo map cuyas llaves son las posiciones de los países y sus valores son arreglos con la nacionalidad y el número de obras.

Por último, la función obrasPais primero averigua la nacionalidad con más obras a partir de la llave “0”, luego se recorre por medio de un ciclo for la lista de obras también solo una vez. En este recorrido se verifica si el artista de la obra tiene como nacionalidad aquella que más obras tiene, de ser así se agrega una pareja a mapFinal cuya llave es la posición en la que fue identificada y su valor es un arreglo con la información de la obra. Para saber la nacionalidad de un artista se busca en el map info la nacionalidad de artista que tenga el mismo ConstituentID de la obra. Por otra parte, la posición se deduce a partir de un contador.

Teniendo en cuenta que esta última función solo recorre una sola vez la lista con las obras, se puede decir que la complejidad temporal es de O(N) para esta función y para el requisito en general. Al igual que para el caso del requisito 1, es esperable un menor desempeño de la variante con maps, debido a que existe la posibilidad de que haya espacios vacíos en la tabla, lo que en ultimas aumente el número de iteraciones en algunos casos.

PRUEBA

* Las pruebas mostradas a continuación se hicieron con los archivos de extensión “large”.

Sistema operativo: Mac OS

Procesador: 1,6 GHz Intel Core i5 de dos núcleos

Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1 (con maps):

Tiempo de ejecución aproximado: 6,53s

Prueba 1 (con listas):

Tiempo de ejecución aproximado: 1,33s

Prueba 2 (con maps):

Tiempo de ejecución aproximado: 6,61s

Prueba 2 (con listas):

Tiempo de ejecución aproximado: 1,18s

Prueba 3 (con maps):

Tiempo de ejecución aproximado: 6,64s

Prueba 3 (con listas):

Tiempo de ejecución aproximado: 1,21s

REQUISITO 5:

Este requisito en concreto se divide en tres funciones:transportar_obras, obras_mas_antiguas y obras_mas_caras. La primera de estas recorre una sola vez toda la lista de las obras, durante este ejecucionde este ciclo for se van contando las obras que hacen parte del departamento, sumando los pesos de las obras y calculando el costo total de transportarlas. Antes de iniciar el ciclo se crean dos maps denominados: Fechas_obras y Precio_obras. En el primero se guarda una pareja por cada obra del departamento, la cual tiene como llave el "ObjectID" y como valor la obra. Por otra parte, el segundo map albergara una pareja que tenga como llave el "ObjectID" y como valor un arreglo con el precio y la obra.

Para el cálculo de precio se utilizaron la altura, la profundidad y el ancho, en lugar del parámetro de dimensiones. Cuando alguno de los datos es 0, el programa ignora dicho valor y realiza el cálculo a partir de las otras dimensiones. Como el recorrido se hace solo una vez, la complejidad temporal de esta función es de O(n).

La segunda función se encarga de hallar las 5 obras más antiguas del departamento, para ello recorre 5 veces el map Fechas_obras y al final de cada uno de ellos arroja la obra más antigua que se halló y almacena su id en la lista llamada id_mas_antiguas . Para evitar que se agregue varias veces la misma obra, aparte de verificar que su año sea el menor al que se encuentra en la variable MasAntigua, también el programa se cerciora que el id de la obra no esté ya en id_mas_antiguas. La información de las obras y los precios de cada una de ellas residen en las

listas ObrasMasAntiguas y Precios_obras, respectivamente. Como el recorrido se hace solo 5 veces, la complejidad temporal de esta función es de O(n).

Finalmente, la cercera función se encarga de hallar las 5 obras más caras del departamento. Al igual que la función anterior, en esta se recorren 5 veces el map Precio_obras y en cada recorrido se encuentra aquella que es más costosa. Para evitar que siempre se agregue la misma se agregan los precios exactos a una lista, si el id de una obra concuerda con alguno de esta, la obra en cuestión será omitida. Mientras que la información de la obra se guarda en la lista ObrasMasCaras, los precios de las obras se ubican en la lista idMacCara. Como el recorrido se hace solo 5 veces, la complejidad temporal de esta función es de O(n).La complejidad de todo el requerimiento es de O(n).

PRUEBA

* Las pruebas mostradas a continuación se hicieron con los archivos de extensión “large”

Sistema operativo: Windows

Procesador: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz

Memoria Ram: 8 GB 2133 MHz LPDDR3

Prueba 1 (con maps):

Departamento: Drawings & Prints

Tiempo de ejecución aproximado: 7,03s

Prueba 1 (con listas):

Departamento: Drawings & Prints

Tiempo de ejecución aproximado: 2,53s

Prueba 2 (con con maps):

Departamento: Photograph

Tiempo de ejecución aproximado: 3,21s

Prueba 2 (con listas):

Departamento: Photograph

Tiempo de ejecución aproximado: 2,68s

Prueba 3 (con maps):

Departamento: Architecture and Design

Tiempo de ejecución aproximado: 9.51s

Prueba 3 (con listas):

Departamento: Architecture and Design

Tiempo de ejecución aproximado: 2,57s