RETO 3- ESTRUCTURA DE DATOS Y ALGORITMOS

Integrantes:

- **Nombre:** Juan Sebastián Sánchez Delgado
- **Correo**: js.sanchezd1@uniandes.edu.co
- **Código:** 202013577

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

ANALISIS DE COMPLEJIDAD Y PRUEBAS DE EJECUCION:

REQUISITO 1:

El requisito en cuestión fue implementado en la función avistamientosCiudad. En ella, primero se crea un map ordenado de tipo RBT, en este se guardarán parejas cuyas llaves serán las fechas y los valores la información de los avistamientos. Luego se crea un map no ordenado denominado mapCuidades, en él se almacenarán las diferentes ciudades que se identifiquen a lo largo del recorrido.

Tras haber definido las estructuras, se inicia un ciclo for en el cual se recorrerá la lista info[“UFOS] una sola vez. El ciclo consta de dos condicionales: en el primero se verifica si la ciudad del avistamiento ya está en mapCuidades, en caso de no estarlo se agrega a este map. El segundo por su parte se encarga de verificar si la ciudad del avistamiento es la misma que se pasó por parámetro, de ser así, se agrega al map avistamientoCiudad una pareja que tenga como llave la fecha del avistamiento y como valor la información del mismo.

Como solo se recorre la lista una sola vez, se puede concluir que la complejidad temporal es O(N), en donde N es el número de elementos de info[“UFOS]. Cuando se verifica si cierta ciudad ya está en mapCuidades, como la búsqueda del elemento por medio de esta estructura es constante, entonces dicha operación no afectara la complejidad establecida anteriormente.

En cuanto a la memoria extra utilizada, al sumar ambas estructuras de datos se obtiene k + C, en donde k es el número de elementos dentro de avistamientoCiudad y C el número de ciudades donde se han reportado avistamientos. Teniendo en cuenta lo anterior, se puede decir que la memoria extra utilizada es aproximadamente k + C.

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:

Ciudad: “las vegas”

Tiempo de ejecución aproximado: 1,24s

Prueba 2:

Ciudad: “los angeles”

Tiempo de ejecución aproximado: 1,02s

Prueba 3:

Ciudad: “melbourne”

Tiempo de ejecución aproximado: 1,08s

Considerando que las pruebas fueron hechas con el archivo de extensión “large”, se puede decir que los tiempos de respuesta fueron muy buenos. Además, todos los tiempos de ejecución obtenidos son congruentes con la complejidad temporal O(N). Aparentemente, la memoria extra utilizada no afecto de sobremanera el comportamiento de la implicación, aun así, como solo se pide el número de las ciudades, cabe la posibilidad que un rediseño con contador ofrezca un rendimiento ligeramente mejor.

REQUISITO 2 (Juan Sebastián Sánchez Delgado):

El requisito 2 se implementó por medio de la función avistamientosDuracion. En primer lugar, se crea un map denominado AvistamientosPorDuracion de tipo RBT con el objetivo de guardar en este todos aquellos avistamientos que se encuentren dentro del intervalo dado. Posteriormente se define la variable duracionMaxima, la cual almacenará la duración más larga entre todos los avistamientos.

La función está formada por dos ciclos for. Cada uno de estos solo recorre la lista info[“UFOS] una sola vez, es decir, que sin importar el caso siempre se recorrerá dos veces la lista. En el primer ciclo existen dos condicionales, en el primero se verifica si la duración del avistamiento es mayor que la duracionMaxima, si esto se cumple se reasigna la variable duracionMaxima

con la duración dada. Por otra parte, en el segundo condicional se comprueba si la duración se encuentra en el rango que se dio por parámetro, si es así, se agrega la información del avistamiento como valor, mientras que, la ciudad, el país y la fecha son agregados a un arreglo, el cual se agregará como llave al map AvistamientosPorDuracion.

El segundo ciclo for se encarga de contar el número de avistamientos con la duración máxima. Para tal fin, se verifica si la duración del avistamiento concuerda con la duracionMaxima, si tal condición se cumple, se agrega una unidad a la variable contador.

Como ya se mencionó anteriormente, sin importar el escenario siempre se recorrerá la lista info[“UFOS], lo que implica que la complejidad temporal será aproximadamente 2N. Debido a que ninguna otra instrucción, ya sea de comparación o de búsqueda tiene una complejidad variable, se puede concluir que la complejidad temporal de todo el requisito es de O(N).

Con respecto a la memoria extra empleada, si se considera únicamente el map AvistamientosPorDuracion, la memoria extra usada sería igual al número total de avistamientos dentro del rango (k). Como las variables ocupan memoria constante que no resulta ser tan representativa, se puede concluir que la memoria extra utilizada fue aproximadamente igual a k.

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:

Duración mínima: 30.0s

Duración máxima: 150.0s

Tiempo de ejecución aproximado: 1,07s

Prueba 2:

Duración mínima: 0s

Duración máxima: 200.0s

Tiempo de ejecución aproximado: 2,02s

Prueba 3:

Duración mínima: 0

Duración máxima: 500.0s

Tiempo de ejecución aproximado: 2,58s

Es evidente que los tiempos de ejecución registrados tienen sentido, ya que el crecimiento de estos concuerda con él comportamiento de una complejidad O(N). Los tiempos de respuesta son excelentes, por lo que sin lugar a dudas la función es muy eficiente. A pesar de los múltiples intentos realizados, no se encontró una forma efectiva de cumplir el requisito recorriendo solo una vez la lista. Esto se debe a que, para saber cuál es la máxima duración entre todos lo avistamientos, es necesario haber recorrido toda la lista una vez. Aun así, el segundo ciclo for, aparentemente, no influyó significativamente en el funcionamiento del requisito.

REQUISITO 3 (Nicolás Alexander rodríguez Pinilla):

En el requisito 3 se usa la función avistamientos_tiempo. En está función primero se crea un map llamado Avistamientos tipo RBT, este almacenará los datos finales de manera ordenada, también se asigna una lista llamada Tardía la cual se usa para asignar y finalmente devolver el valor más tardío encontrado entre los datos.

Se toman los datos, haciendo un split para así compararlos con el rango de tiempo ingresado por el usuario, a través de esto se retorna la lista ya con los datos que se encuentren en el rango de manera organizada.

Para finalizar, se retorna además de la lista y el dato almacenado en la variable Tardía un contador el cuál guarda cuántas veces se repite Tardía con el objetivo de devolver el número de elementos con este. La complejidad temporal sería de 2N, aunque debido a que no se presentan más instrucciones de búsqueda o comparación, se puede concluir que la complejidad total es de O(N).

Ya que la memoria usada es igual a la lista que se entrega al final del requisito, se puede concluir que la memoria utilizada es igual a K.

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:

Tiempo mínimo: 20:45

Tiempo máximo: 23:15

Tiempo de ejecución aproximado: 0,7s

Prueba 2:

Tiempo mínimo: 11:30

Tiempo máximo: 20:30

Tiempo de ejecución aproximado: 1,3s

Prueba 3:

Tiempo mínimo: 00:00

Tiempo máximo: 14:45

Tiempo de ejecución aproximado: 1,5s

REQUISITO 4:

En el requisito 4 se usa la función avistaminetosRango En está función primero se crea un map llamado avistamientosRango tipo RBT, este almacenará los datos finales de manera ordenada, también se asigna una lista llamada antigua la cual se usa para asignar y finalmente devolver el valor más tardío encontrado entre los datos.

Se toman los datos, haciendo un split para así compararlos con el rango de tiempo ingresado por el usuario, a través de esto se retorna la lista ya con los datos que se encuentren en el rango de manera organizada.

Para finalizar, se retorna además de la lista y el dato almacenado en la variable antigua un contador el cuál guarda cuántas veces se repite antigua con el objetivo de devolver el número de elementos con este. La complejidad temporal sería de 2N, aunque debido a que no se presentan más instrucciones de búsqueda o comparación, se puede concluir que la complejidad total es de O(N).

Ya que la memoria usada es igual a la lista que se entrega al final del requisito, se puede concluir que la memoria utilizada es igual a K.

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:

Fecha mínima: 1945-08-06

Fecha máxima: 1984-11-15

Tiempo de ejecución aproximado: 1,1s

Prueba 2:

Fecha mínima: 1920-09-10

Fecha máxima: 2000-10-09

Tiempo de ejecución aproximado: 1,62s

Prueba 3:

Fecha mínima: 1870-09-10

Fecha máxima: 1915-09-10

Tiempo de ejecución aproximado: 2,21s

REQUISITO 5:

La función que se encarga de la implementación del requisito se denomina se denomina avistamientos_zona. Antes que nada, se crea un map ordenado de tipo RBT llamado info_avistamiento, en el cual se introducirán todos aquellos avistamientos que estén dentro del rango. Posteriormente, se inicia un ciclo for con el cual se recorrerá la lista info[“UFOS] una sola vez. Dentro del ciclo se comprueba si tanto la latitud como la longitud del avistamiento están en el intervalo que se indicó como parámetro, en caso de estarlo, se agrega una nueva pareja a info_avistamiento. La nueva pareja tendrá como llave un arreglo con la latitud, la longitud y el tiempo, mientras que el valor será la información del avistamiento.

Teniendo en cuenta que la lista solo se recorre una sola vez y que no existe ninguna otra operación con complejidad variable, es correcto afirmar que la complejidad temporal del requisito es de O(N), en donde N es el número de elementos dentro del rango. Por otro lado, debido a que la única estructura de datos que se utiliza para la función es info_avistamiento, eso quiere decir que la cantidad de memoria extra usada es aproximadamente 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:

Latitud mínima: 31.33

Latitud máxima: 37.0

Longitud mínima: -109.05

Longitud máxima: -103.0

Tiempo de ejecución aproximado: 1,3s

Prueba 2:

Latitud mínima: 20.10

Latitud máxima: 47.12

Longitud mínima: -120.04

Longitud máxima: 10.9

Tiempo de ejecución aproximado: 2,1s

Prueba 3:

Latitud mínima: 10.15

Latitud máxima: 90.98

Longitud mínima: -150.01

Longitud máxima: 9.03

Tiempo de ejecución aproximado: 2,7s

Debido a que en las diferentes pruebas realizadas los tiempos que se reportaron son pequeños, eso significa que la función es bastante eficiente en el apartado de complejidad temporal. Con respecto a la memoria, como solo se agregan al map ordenado los elementos del intervalo, en lugar de todos los de la lista, la memoria adicional que se uso es reducida si se compara con el total de elementos. Por último, el crecimiento en los tiempos de respuesta de la solución concuerda con un crecimiento de orden lineal, razón por la cual es muy posible que el análisis de la complejidad temporal hecho sea acertado.