Nombre: Josue Salvador Cano Martinez ***Matrícula:*** A00829022

Programación de estructuras de datos y algoritmos fundamentales

Módulo: Conceptos Básicos y Algoritmos fundamentales

Actividad 1.3: Conceptos básicos y algoritmos fundamentales

Investigación y reflexión individual

Fecha de entrega: 08 de septiembre de 2020

Objetivo

Realizar una investigación y reflexión en forma individual de la importancia y eficiencia del uso de los diferentes algoritmos de ordenamiento y búsqueda en una situación problema de esta naturaleza.

Parte 1: Investigación

Algoritmos de ordenamiento

- ✓ Bubble Sort

También conocido como ordenamiento burbuja, funciona de la siguiente manera: Se recorre el arreglo intercambiando los elementos adyacentes que estén desordenados. Se recorre el arreglo tantas veces hasta que ya no haya cambios. Prácticamente lo que hace es tomar el elemento mayor y lo va recorriendo de posición en posición hasta ponerlo en su lugar.

Procedimiento:

```
Paso 1: [Inicializa i al final de arreglo]          For i <- N downto 1 do
Paso 2: [Inicia desde la segunda posición]                  For j <- 2 to i do
paso 4: [Si a[j-1] es mayor que el que le sigue]              If a[j-1] < a[j] then
paso 5: [Los intercambia.                                       Swap(a, j-1, j)
paso 7: [Fin]                                       End.
```

Tiempo de ejecución del bubble sort:

1. Mejor caso: O(n)
2. Peor caso. n(n-1)/2
3. Promedio. $O(n^2)$

(Universidad de las Américas Puebla, s.f.)

✓ Merge Sort

En cada recursión se toma un array de elementos desordenados. Se divide en dos mitades, se aplica la recursión en cada una de estas y luego (dado que al finalizar estas recursiones se tienen las dos mitades ordenadas) se intercalan ambas para obtener el array ordenado.

Intercalar: Es la operación que le da el nombre a este algoritmo. La intercalación toma dos secuencias (arrays) de elementos y a partir de estas construye una tercera secuencia que contiene todos los elementos de estas en orden.

Características.

1. Es un algoritmo recursivo con un número de comparaciones mínimo. El tiempo de ejecución promedio es O (N log(N)).
2. Su desventaja es que trabaja sobre un array auxiliar lo cual tiene dos consecuencias: uso de memoria y trabajo extras consumido en las copias entre arreglos (aunque es un trabajo de tiempo lineal).
3. Es una aplicación clásica de la estrategia para resolución de algoritmos "divide y vencerás". Esta estrategia plantea el hecho de que un problema puede ser dividido en varios subproblemas y una vez resueltos estos se pueden unir las soluciones para formar la solución del problema general. La solución de los subproblemas más pequeños se realiza de la misma manera: es decir, se van resolviendo problemas cada vez más pequeños, hasta encontrarse con un caso base (problema no divisible).

(Gurin, 2004)

Algoritmos de búsqueda

- ✓ Secuencial Search

Consiste en recorrer y examinar cada uno de los elementos del array hasta encontrar el o los elementos buscados, o hasta que se han mirado todos los elementos del array. Este es el método de búsqueda más lento, pero si la información se encuentra completamente desordenada es el único que podrá ayudar a encontrar el dato buscado.

Este algoritmo se puede optimizar cuando el array está ordenado, en cuyo caso la condición de salida cambiaría; o cuando sólo interesa conocer la primera ocurrencia del elemento en el array.

Si al acabar el bucle, i vale N esto indica que no se encontró el elemento. El número medio de comparaciones que hay que hacer antes de encontrar el elemento buscado es: (N+1)/2.

Complejidad de la Búsqueda Lineal

(A) Mejor caso: el algoritmo termina tan pronto como encuentra el elemento buscado en el array. Puede ser que la primera posición examinada contenga el elemento buscado, en cuyo caso el algoritmo informará que tuvo éxito después de una sola comparación. Por tanto, la complejidad en este caso será O(1).

(B) Peor caso: sucede cuando se encuentra el valor en la última posición del array. Como se requieren n ejecuciones del bucle while, la cantidad de tiempo es proporcional a la longitud del array n, más un cierto tiempo para realizar las instrucciones del bucle while y para la llamada al método. Por lo tanto, la cantidad de tiempo es: O(n).

(C) Caso promedio: Suponiendo que cada elemento almacenado en el array es igualmente probable de ser buscado. La media puede calcularse tomando el tiempo total de encontrar todos los elementos y dividiéndolo por n:

Total = a (1 + 2 + ...+n) + bn = a (n(n+1) / 2) + bn, ‘a’ representa el costo constante asociado a la ejecución del ciclo y ‘b’ el costo

constante asociado a la evaluación de la condición. 1, 2, ..n, representan el costo de encontrar el elemento en la primera, segunda, ..., enésima posición dentro del arreglo. Media = (Total / n) = a((n+1) / 2) + b, que es: O(n).

Este es el algoritmo de más simple implementación, pero no el más efectivo. En el peor de los casos se recorre el array completo y el valor no se encuentra o se recorre el array completo si el valor buscado está en la última posición del array. La ventaja es su implementación sencilla y rápida, la desventaja, su ineficiencia.

(Diaz, 2006)

- ✓ Binary Search

Si los elementos sobre los que se realiza la búsqueda están ordenados, entonces se puede utilizar un algoritmo de búsqueda mucho más rápido que el secuencial, la búsqueda binaria. El algoritmo consiste en reducir paulatinamente el ámbito de búsqueda a la mitad de los elementos, basándose en comparar el elemento a buscar con el elemento que se encuentra en la mitad del intervalo y en base a esta comparación:

- Si el elemento buscado es menor que el elemento medio, entonces se sabe que el elemento está en la mitad inferior de la tabla.
- Si es mayor es porque el elemento está en la mitad superior.
- Si es igual se finaliza con éxito la búsqueda, ya que se ha encontrado el elemento.

Se puede aplicar tanto a datos en listas lineales (Vectores, Matrices, etc.) como en árboles binarios de búsqueda. Los prerrequisitos principales para la búsqueda binaria son:

- La lista debe estar ordenada en un orden especifico de acuerdo con el valor de la llave.
- Debe conocerse el número de registros.

La búsqueda binaria consiste en dividir el array por su elemento medio en dos subarrays más pequeños y comparar el elemento con el del centro. Si coinciden, la búsqueda se termina. Si el elemento es menor, debe estar (si está) en el primer subarray, y si es mayor está en el

segundo. Por ejemplo, para buscar el elemento 3 en el array {1,2,3,4,5,6,7,8,9} se realizarían los siguientes pasos:

Se toma el elemento central y se divide el array en dos:

{1,2,3,4}-5-{6,7,8,9}

Como el elemento buscado (3) es menor que el central (5), debe estar en el primer subarray:

{1,2,3,4}

Se vuelve a dividir el array en dos:

{1}-2-{3,4}

Como el elemento buscado es mayor que el central, debe estar en el segundo subarray:

{3,4}

Se vuelve a dividir en dos:

{}-3-{4}

Como el elemento buscado coincide con el central, ha sido encontrado. Si al final de la búsqueda aún no se encuentra, y el subarray a dividir está vacío {}, el elemento no se encuentra en el array.

(Diaz, 2006)

Parte 2: Reflexión

La problemática identificada de esta naturaleza consiste en dar solución a la necesidad de ordenamiento de datos para posteriormente efectuar consultas de búsqueda. En la vida diaria existen gran variedad de situaciones de distintas índoles donde la solución se encuentra en la implementación algorítmica como la utilizada en esta situación problema; prácticamente en todas las empresas se lleva a cabo el manejo y gestión de datos, podríamos imaginarnos una aerolínea o una cadena de hoteles, por mencionar un ejemplo, quienes tienen una base de datos de sus clientes con mayor fidelidad y que, por tanto, tienen un perfil dentro de la organización, imaginemos que uno de los clientes en cierta ocasión llega en busca de alquilar u servicio que ofrecen y para obtener un descuento del 20% se debe

verificar que haya tenido al menos 5 visitas concurrentes en el último año (dato que se encuentra en su perfil), para esto, el prestador del servicio requiere confirmar que efectivamente el cliente puede ser acreedor a dicho descuento y realiza una búsqueda dentro del sistema, sin embargo, no se trata del único cliente que tienen las compañías, sino que aparte de él existe otro millón de usuarios con un perfil, por tanto, buscarlo de forma manual resulta imposible, se necesita de un proceso rápido y efectivo que en cuestión de segundos permita tener acceso a la información del perfil buscado; es aquí donde entran en juego las arquitecturas computacionales, la implementación codificada de algoritmos que permitan ordenar los registros que se tienen para posteriormente dar la posibilidad de efectuar búsquedas. Un caso similar, y tal vez más sencillo de entender, resulta ser cuando se efectúa una búsqueda de un perfil en Facebook, que claro, muy seguramente su infraestructura computacional y los algoritmos implementados son más sofisticados, sin embargo, el desafío de búsqueda tiene una misma naturaleza compartida con el caso de la situación problema planteada. En los ejemplos mencionados anteriormente sólo se está realizando la búsqueda de un registro a la vez (correspondiente a un perfil buscado), sin embargo, para aterrizarlo de la manera más similar a lo que consiste la problemática de dicha actividad integradora se podría pensar en los procesos gubernamentales que muy constantemente se efectúan; por ejemplo, recientemente dio inicio el periodo de clases para educación básica en México y, al menos en Jalisco, existe un programa gubernamental que otorga de uniformes y de útiles escolares a los estudiantes de preescolar, primaria y secundaria, ¿qué debe de hacer el gobierno del estado para saber exactamente la demanda a satisfacer?, ¿Cómo darse cuenta de quiénes serán los beneficiados de su programa social? Para ello es necesario realizar una búsqueda entre dos parámetros (los cuales podrían ser las edades de la población); el gobierno tiene una base con los datos de toda la población que habita el estado (debido al registro que se hace cuando nacemos), entonces, ordenando los registros en forma ascendente por edad y realizando una consulta (o búsqueda) tomando como datos una edad inicial y una edad final (que crean el rango de edades de los estudiantes de educación básica) podrá obtener como resultado los registros correspondientes a las personas que serán apoyadas con el programa, teniendo así un criterio más seguro acorde a la demanda a satisfacer. Se han mencionado situaciones reales donde la aplicación de los conceptos y metodologías abordadas en la solución a la problemática planteada en esta actividad integradora resulta bastante útil, pero ahora, es momento de entender de una forma más técnica el funcionamiento y las implicaciones ingenieriles que hay detrás de la solución desarrollada.

La manera de implementar soluciones ingenieriles computacionales a situaciones con la naturaleza de los casos anteriormente planteado es haciendo uso de algoritmos de ordenamiento y de búsqueda. Los algoritmos de ordenamiento, como su nombre lo indica, son procedimientos lógicos-matemáticos

que se aplican en la codificación de programas que ayudan a facilitar ciertos procesos de ordenamiento de datos. Si bien es cierto que los datos resultan ser un elemento sumamente importante y valioso para las tomas de decisiones y para la gestión de diversas situaciones, también es necesario recalcar la importancia de su ordenamiento, datos que no están ordenados resultan ser datos sin valor alguno, cuestión que en el Big Data y en la Ciencia de Datos se recalca muy precisamente. Según la Harvard Business School, la Ciencia de Datos resulta ser la profesión más sexi del siglo XXI, y no cabe duda de ello al percatarnos de que cada vez el mundo se somete a una era mayormente digitalizada, donde los datos pasan a ser igualmente digitalizados y donde el control de ellos resulta ser muy útil para un sinfín de situaciones. Si bien es cierto que para cuestiones relacionadas con Ciencias de Datos y Big Data el ordenamiento resulta ser muy sofisticado y completo, es importante señalar que el ordenamiento efectuado en dicha situación problema (donde los datos se ordenan en forma ascendente/descendente según su cronología) forma parte de los diferentes procesos que podrían llegar a verse implícitos dentro de esta analítica de datos sofisticada. Tener los datos ordenados brinda la posibilidad de efectuar búsquedas de una forma más rápida y sencilla, que es aquí donde entra en juego el otro algoritmo del que se hace uso en el proceso de solución: algoritmo de búsqueda. Se trata de dos algoritmos que, para solución a problemáticas de esta naturaleza, resultan ir de la mano, la implementación previa de uno de ellos (el de ordenamiento) es necesario para la ejecución del otro de ellos (el de búsqueda). Una vez que los datos se encuentran ordenados es posible implementar un algoritmo de búsqueda que posibilite una consulta específica tomando en cuenta ciertos parámetros de entrada; haciendo una símil se podría imaginar a una biblioteca, donde los libros se encuentran ordenados según diversas categorías (autor, género, fecha, edición, etc.), cuestión que facilita encontrar de forma rápida cierto libro; algo similar pasa con las situaciones que mantienen la naturaleza de la situación planteada: ordenar los datos de manera ascendente según su eventualidad cronológica (por fecha) facilita las consultas que pretenden realizarse. Para el caso de ordenamiento se llegaron tener en cuenta dos algoritmos: Merge Sort y Bubble Sort, y para el caso de búsqueda se consideraron de la misma manera dos algoritmos: Secuencial Search y Binary Search. En un principio la decisión de elegir entre uno otro se había tomado al tener en consideración únicamente su complejidad computacional, que, como se mencionó en la investigación y como fue demostrado matemáticamente en la documentación del programa, entre los algoritmos de ordenamiento, el Merge Sort resulta tener una menor complejidad: (nLog(n)) en comparación con el método Bubble Sort, que resulta tener una complejidad cuadrática: O(n^2), a partir de ello se logra plantear que entre mayor sea la cantidad de datos el tiempo de respuesta del algoritmo aumenta en mayor proporción para el Bubble Sort en comparación del requerido por el Merge Sort. Fue a partir de este planteamiento que en la codificación se optó por utilizar el Merge Sort como método de ordenamiento para los datos del

fichero que se tenía, donde se logró confirmar su utilidad en cuestión de ahorro de tiempos para una entrada masiva de datos (1 millón de datos específicamente) en comparación del tiempo de respuesta que tenía el Bubble Sort. Para el caso de las búsquedas estaba planteado seguir la misma idea en la toma de decisión acorde a qué algoritmo utilizar, guiándonos por la complejidad computacional, el Binary search resulta ser más eficiente en cuanto a tiempo comparado con el Secuencial Search, donde el primero resulta tener una complejidad O(log(n)) , mientras que el segundo tiene una complejidad computacional: O(n); para un caso donde los datos resulten ser pocos puede resultar que el Secuencial Search no presente mucha variedad con respecto al Binary Search, sin embargo, este tipo de implementaciones computacionales se realiza con datos masivos (porque eso es lo que al final se busca con el apoyo de programas computacionales), resultado que si se tuvieran un millón de datos el valor de 'n' resultaría ser del mismo valor: 'un millón'. A partir de este planteamiento, en un inicio, se había definido utilizar el Binary Search y se desarrolló un programa que hasta cierto punto era eficiente, sin embargo, podría serlo mucho más, y la solución estaba en implementar ese algoritmo que se había descartado con tanta certeza previamente, una mejor solución se encontraba en el uso del Secuencial Search. Antes de explicar esto que parece ser una controversia misma donde en un inicio se argumenta y se explica como el Secuencial Search resulta ser menos eficiente que el Binary Search, al final resultó implementarse y se obtuvo un programa mucho más eficiente. Para poder explicar la situación anterior es necesario decir que la solución planteada se basó en la implementación de objetos, la razón por la cual se utilizaron consistió en el hecho de que cada registro (o cada renglón del fichero) correspondía a información que no sólo contenía la fecha del acontecimiento (que es el parámetro utilizado para ordenar en forma ascendente los datos), sino que también se incluían otros parámetros tales como la dirección IP, la hora y la razón de falla, se contaban con datos dentro de cada registro que no resultaban útiles para el proceso de ordenamiento, sólo se requería abstraer la fecha y para ello fue que se realizó la implementación de POO, donde cada registro corresponde a la creación de un objeto y de esta manera resulta más sencillo separar la fecha (lo que resulta importante y necesario) de los demás datos para efectuar así un ordenamiento ascendente de dichos registros. Volviendo a la explicación del cómo la búsqueda secuencial resultó ser mejor que la binaria existe la necesidad de recalcar un error en la interpretación de las instrucciones: en un inicio en el equipo de trabajo habíamos interpretado que el programa iba a generar un fichero nuevo con los registros correspondientes al rango encontrado entre las fechas ingresadas como parámetros de entrada, es decir, cada vez que se corriera el programa se generaría un fichero de extensión '.txt' con las salidas correspondientes, siendo una interpretación errónea que cuando logró aclararse fue posible determinar que la búsqueda secuencial resultaba ser mejor que la binaria. Lo que realmente se solicitaba era un único archivo con los 16,807 registros ordenados, sin

necesidad de creación de un fichero cada vez que se hiciese la ejecución del programa. Planteado lo anterior fue posible determinar la repetición de ciertas fechas, existían varios registros que se habían dado en una misma fecha, de esta manera podrían agruparse asignándoles un índice, por ejemplo, la primera fecha que aparecía era 'Jun 1' en el índice '0', la primera vez que aparecía 'Jun 2' era en la posición '111', la primera vez que aparecía 'Jun 3' era en la posición '247' y así sucesivamente se creó otro archivo con las 150 fechas únicas (que se repetían para formar las 16,807 totales) con su respectiva asignación de índice; una vez que esto se había implementado, cuando el usuario ingresaba los datos de entrada (fechas de inicio y fin) se llevaba a cabo una lectura del archivo de índices para retornar el índice donde se encontraba la fecha que había sido el dato de entrada, a partir de ello se comenzaba a desplegar (con un ciclo while) los registros posteriores hasta donde la condición secundaria (fecha final) se cumpliera. De esta manera se estaba dejando de implementar la búsqueda binaria para encontrar los índices del rango de datos que debía imprimirse y se sustituyó por la implementación de la búsqueda lineal, lo cual, si bien es cierto que para la realización de una primera consulta resulta ser menos eficiente, cuando se trata de muchas consultas consecutivas (algo más apegado a la realidad cuando son miles de usuarios y no sólo uno) los tiempos son mucho menores, resultando ser un algoritmo mejor optimizado. Con esto se logró apreciar cómo el definir qué algoritmo se debe implementar no depende únicamente de su complejidad, sino en sí mismo de la situación que pretende modelarse y solucionarse.

Referencias:

Diaz, N. (2006). *Universidad del Cauca*. Recuperado el 08 de 09 de 2020, de http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap02.htm

Gurin, S. (30 de 11 de 2004). Recuperado el 08 de 09 de 2020, de http://es.tldp.org/Tutoriales/doc-programacion-algoritmos-ordenacion/alg_orden.pdf

Universidad de las Américas Puebla. (s.f.). *Interactive and Cooperative Technologies Lab*. Recuperado el 08 de 09 de 2020, de http://ict.udlap.mx/people/ingrid/Clases/IS211/Ordenar.html

Estructuras de Datos y Algoritmos, Mark Allen Weiss, Addison-Wesley Iberoamericana, 1995, ISBN 0-201-62571-7.

Fundamentos de Algoritmia, G. Brassard y P. Bratley, Prentice Hall, 1998, 84-89660-00-X.

Estructuras de Datos y Algoritmos, Aho, Hopcrotf y Ullman, Addison Wesley Iberoamericana, 1988, 0-201-64024-4.

Técnicas de Diseño de Algoritmos, Rosa Guerequeta and Antonio Vallecillo , Servicio de Publicaciones de la Universidad de Málaga. 1998, 84-7496-666-3, Segunda Edición: Mayo 2000.

Análisis de Algoritmos, Sebastián Gurin (Cancerbero), 2004.