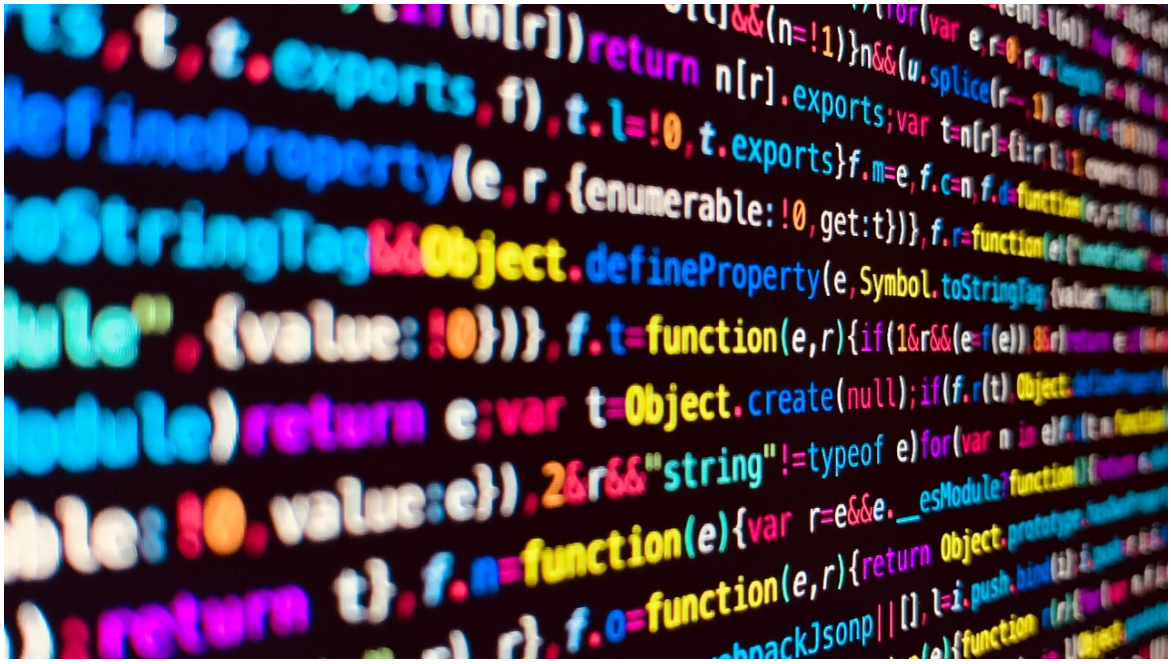
Act 1.3 - Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales (Evidencia Competencia)

Programación de estructuras de datos y algoritmos fundamentales
Grupo 12



Marisol Rodríguez Mejía A01640086

24 de septiembre de 2021

Reflexión

Buscar algo en una plataforma es algo que la gran mayoría de los humanos hemos hecho, desde meterse a Instagram para encontrar el usuario de un amigo hasta buscar vuelos en la página de una aerolínea. Los algoritmos de búsqueda tienen miles de aplicaciones, por ejemplo, nos han permitido llegar a tener buscadores tan eficientes como Google, en donde se trabaja con conjuntos de datos enormes.

Asimismo, si no fuera por los algoritmos de ordenamiento, las búsquedas serían mucho más complejas, son un concepto esencial en programación. Éstos organizan los datos para obtener una base de datos más sencilla de manejar, muchos de los cómputos que tiene que realizar un programa son más eficientes si los datos sobre los que operan están ordenados.

En una situación problema como la de esta actividad, implementar estos algoritmos se vuelve algo absolutamente necesario, y el resultado que se obtiene al usarlos en mi programa es el deseado, gracias a ellos me fue posible buscar ambas fechas y mostrar la base de datos en un formato mucho más organizado.

Los algoritmos que yo elegí fueron el de merge sort para el de ordenamiento y binary search para realizar la búsqueda.

Al elegir mi algoritmo de ordenamiento, reduje mis opciones a dos: quick sort o merge sort. No consideré a los otros algoritmos que vimos en clase debido a que su complejidad era mayor. Por esa misma razón preferí el ordenamiento merge al quick sort, tanto en su mejor caso como en su caso promedio ambos tienen una complejidad de O(nlogn), sin embargo, el peor caso de quick sort tiene una complejidad de $O(n^2)$ y el peor caso del ordenamiento merge tiene una complejidad igual a las demás de O(nlogn), por esa razón mi elección final fue merge sort.

Su pseudocódigo incluye dos funciones, la de merge y la de ordenaMerge:

$\boldsymbol{merge(A, l, r, m)}$
Input: Un arreglo A, índices de inicio y fin de A: l y r
Output: El arreglo A ordenado
$n1 \leftarrow m - l + 1;$

$n2 \leftarrow m - l + 1;$
for $i \leftarrow 0;\ i < n1;\ i++$ do $L[i] \leftarrow A[l + i];$
for $j \leftarrow 0;\ j < n2;\ j++$ do $R[i] \leftarrow A[m + j + 1];$
$i \leftarrow 0, \quad j \leftarrow 0, \quad k \leftarrow l$
while $i < n1$ and $j < n2$ do
    if $L[i] \leq R[j]$ then
        $A[k] \leftarrow L[i];$
        $i \leftarrow i + 1;$
    else
        $A[k] \leftarrow R[i];$
        $j \leftarrow j + 1;$
    end
    $k \leftarrow k + 1;$
while $i < n1$ **do** $A[k] \leftarrow L[i],\ i \leftarrow i + 1,\ k \leftarrow k + 1;$
while $j < n2$ **do** $A[k] \leftarrow R[j],\ j \leftarrow j + 1,\ k \leftarrow k + 1;$

$\boldsymbol{ordenaMerge(A, l, r)}$
Input: Un arreglo A, índices de inicio y fin de A: l y r
Output: El arreglo A ordenado
if $l < r$ **then**
    $m \leftarrow \left\lfloor l + \frac{r}{2} \right\rfloor;$
    $ordenaMerge(A, l, m);$
    $ordenaMerge(A, m + 1, r);$
    $merge(A, l, r, m);$
end

Como mencioné anteriormente, este algoritmo de ordenamiento tiene una complejidad de orden linearítmico, lo anterior considero que se debe a que, al realizar el ordenamiento, siempre divide la matriz en dos mitades y toma un tiempo lineal para fusionar esas mitades.

Por otra parte, fue prudente utilizar el algoritmo búsqueda binaria. Opino lo anterior debido a que en clase aprendí que esta búsqueda puede ser considerada eficiente, comparada a las demás, tiene una complejidad menor, sobretodo en su caso promedio y en su peor caso. La búsqueda binaria comparte el mejor caso de complejidad O(1) con los otros tipos de algoritmos que vimos en clase (como lo son la búsqueda secuencial y la secuencial ordenada), pero considerando que el

archivo de bitácora no se encontraba ordenado, la complejidad que se debe de considerar es la de caso promedio de O($\log_2 n$) y la de peor caso, O(logn).

La búsqueda binaria tiene las anteriores complejidades es porque divide a la mitad el conjunto de entrada en cada iteración. Por lo tanto, solo necesitamos saber en cuántos pasos obtenemos 1 a medida que dividimos la lista entre 2. Para una lista de longitud n,

$$[n]/2 + [(n\ /2)]/2 + [((n\ /2)/2)]/2 + \dots$$

Es decir,

$$n\ /2^1 + n\ /2^2 + n\ /2^3 + \dots + n\ /2^x$$

Por lo tanto,

$$n\ /2^x\ = 1$$
$$n\ = 2^x$$
$$\log_2 n = \log_2(2^x)$$
$$\log_2 n = x$$

$$T_{promedio} = O(\log_2 n)$$

El pseudocódigo del algoritmo es el siguiente:

***busquedaBinaria*(A, n, k)**
Input: Un arreglo A ordenado de tamaño n, una llave de búsqueda k
Output: El índice del primer elemento en A igual a k o -1 si no se encuentra

```
l ← 0
r ← n − 1
while l ≤ r do
        m ← l + (r − l)/2;
        if k == A[m] then
                return m;
        else if k < A[m] then
                r ← m − 1;
        else
                l ← m − 1;
        end
end
return -1;
```

Para concluir, cabe recalcar que la complejidad fue mi principal factor decisivo porque el archivo bitácora con el cual tenía que trabajar, es un archivo con miles de registros, por ello consideré que lo mejor sería elegir a los algoritmos más eficientes, a diferencia de tener un archivo de muy pocos datos, en donde la complejidad de los algoritmos no llega a generar incrementos de tiempo tan grandes.

Sabemos que las complejidades de los algoritmos se expresan en función de n debido a que dependen del tamaño de la base utilizada. Por lo tanto, yo esperaría que al tener un caso más grande que el de la bitácora, el tiempo de ejecución de mi programa incremente. No obstante, ese incremento no será exageradamente grande gracias a la complejidad de los algoritmos que seleccioné.

Referencias:

Tema 8: Algoritmos de ordenación y búsqueda. (n.d.). Recuperado el 24 de septiembre de 2021 de http://biolab.uspceu.com/aotero/recursos/docencia/TEMA%208.pdf