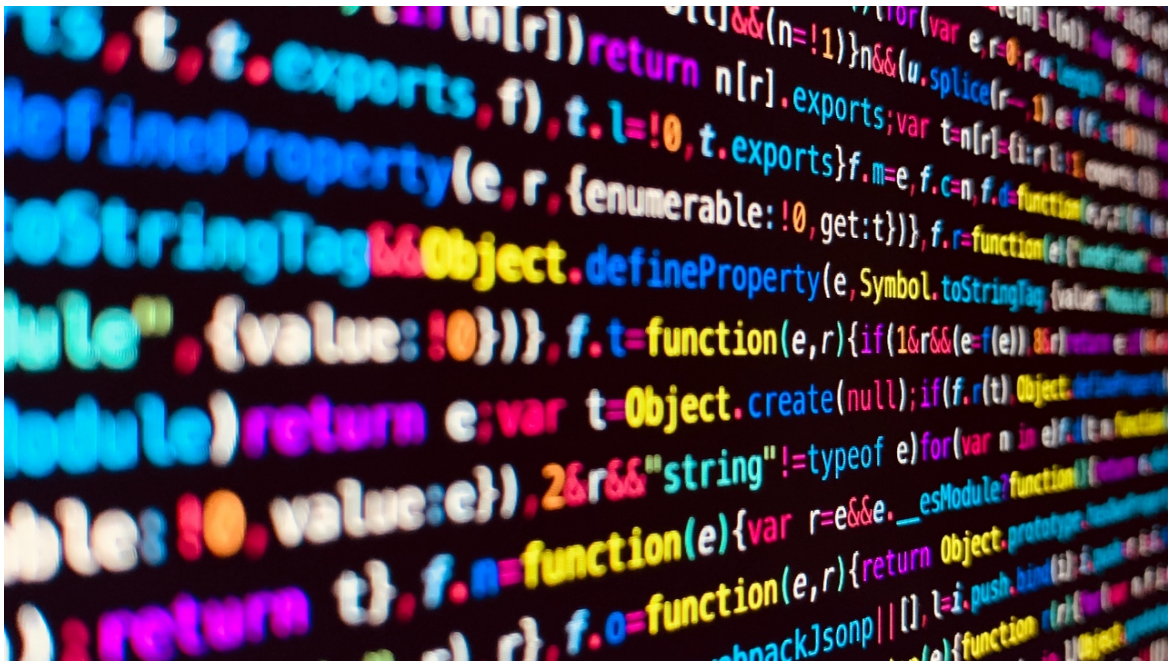
Act 2.3 - Actividad Integral estructura de datos lineales (Evidencia Competencia)

Programación de estructuras de datos y algoritmos fundamentales
Grupo 12



Marisol Rodríguez Mejía A01640086

15 de octubre de 2021

Reflexión

Las estructuras de datos son un proceso a través del cual podemos almacenar y organizar datos de la mejor manera en la cual los datos se puedan usar y manipular fácilmente en el futuro, así como realizar operaciones en ellos de la manera más efectiva. Una selección correcta de la estructura de datos puede mejorar la eficiencia del programa o algoritmo de computadora de una mejor manera. Permiten el almacenamiento seguro de información en una computadora. Por lo tanto, utilizar estructuras de datos en esta actividad, resulta algo prudente, considerando que lo que nosotros queremos hacer con los datos es almacenarlos, manipularlos y ordenarlos.

Las listas ligadas son una estructura de datos lineal muy utilizada que consta de un grupo de nodos en una secuencia. Cada nodo contiene sus propios datos y la dirección del siguiente nodo, por lo tanto, forma una estructura en forma de cadena.

Ventajas de las listas ligadas:

- Son de naturaleza dinámica que asigna la memoria cuando es necesario, tienen longitud variable.
- Las inserciones y eliminaciones son rápidas en cualquier punto de la lista y son fáciles de implementar.

Desventajas de las listas ligadas:

- La memoria se desperdicia ya que los punteros requieren memoria adicional para su almacenamiento.
- No se puede acceder a ningún elemento de forma aleatoria, se tiene que acceder a cada nodo de forma secuencial.

En esta actividad, se hace uso de una lista doblemente ligada. Ésta es una estructura de datos lineal similar a una lista enlazada individualmente, la diferencia es que cada nodo tiene un puntero adicional que almacena la dirección del nodo anterior correspondiente a cada nodo. Aparte de

tener las mismas ventajas que las listas ligadas, las listas doblemente vinculadas cuentan con otras ventajas:

- Se pueden recorrer tanto hacia adelante como hacia atrás.
- Podemos eliminar un nodo fácilmente ya que tenemos acceso a su nodo anterior, no necesita movimiento de elementos para su inserción y eliminación.
- Invertir la lista es fácil.
- Son buenas para ordenar, ya que puede intercambiar punteros en lugar de copiar datos.

Debido a todo lo anterior, considero que está justificado el uso de una lista doblemente ligada para esta actividad. Se requería realizar un almacenamiento y ordenamiento de más de 16,000 líneas de datos y la libertad que nos brindan este tipo de estructuras de datos es que podemos fijar el tamaño dinámicamente, sin necesidad de usar memoria extra; asimismo, el destructor de DoubleLinkedList se asegura de liberar todo el espacio utilizado en la lista con los objetos de tipo Registro. Y, debido a que estamos usando listas doblemente ligadas, el algoritmo de quick sort fue posible de implementar intercambiando lo punteros a sus nodos, logrando así un ordenamiento bastante eficiente. En general, fue mejor utilizar listas doblemente ligadas sobre las sencillas debido a que las operaciones que debíamos de implementar requerían de varias inserciones y recorridos, sobretodo al ordenar sus elementos. De igual manera, utilizar listas vinculadas nos permitió implementar un quick sort iterativo con stacks.
Los stacks o pilas son una estructura de datos simple que permite la adición y eliminación de elementos de una manera específica. Siempre que hay una adición de algún elemento, se envía al punto más alto de las pilas. Por lo tanto, el primer elemento que se agregue será el último en salir. De esta manera, fue posible implementar el quick sort iterativo, utilizando stacks para simular la pila de llamadas recursivas.

Durante el algoritmo de ordenamiento, en la primera iteración de la función “partition” (la cual aprovecha bastante el hecho de estar trabajando con una lista doblemente ligada), se divide en dos particiones. Eso toma O(n). En la siguiente pasada se tiene dos particiones, cada una de las cuales es de tamaño n/2, por lo tanto, se necesita O(n/2) para dividir cada uno de ellos. El tiempo

total para la segunda pasada es O(n/2 + n/2), lo cual es igual a O(n). Cada iteración tiene más particiones, pero las particiones son cada vez más pequeñas. En total, realiza log(n) iteraciones, cada una de los cuales requiere O(n) de tiempo total.

De esta manera, al estarse dividiendo los datos en mitades, el quick sort tanto en su mejor caso como en su caso promedio tiene una complejidad de O(nlogn), no obstante, el peor caso de quick sort tiene una complejidad de $O(n^2)$.

La ventaja de utilizar un algoritmo iterativo contra uno recursivo es que la recursividad llama repetidamente el mecanismo, por lo que se puede causar una sobrecarga de las llamadas a métodos. Esto puede resultar caro tanto en tiempo de procesador como en espacio de memoria, mientras que la iteración no lo es. En general, las funciones iterativas son más rápidas que las recursivas.

Por otro lado, las inserciones (los métodos de addFirst y addLast) en mi código de listas doblemente ligadas tienen una complejidad temporal constante (O(1)). Lo anterior se debe a que sólo se instancia un nuevo objeto tipo apuntador a Node el cual se va asignando a la cabeza o a la cola de la lista, ya sea si se manda a llamar un addFirst o un addLast; no es necesario realizar un recorrido de la estructura.

En cuanto a la eliminación, su complejidad en listas doblemente ligadas es de O(n) debido a que tenemos que buscar el elemento para poder borrarlo. Es decir, se debe de hacer un recorrido de la lista, la cual tiene n elementos.

Por último, para llevar a cabo la búsqueda de información, implementé el algoritmo búsqueda binaria. Esta búsqueda puede ser considerada eficiente, comparada a los demás algoritmos de búsqueda, 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). Esto se debe a que

la búsqueda binaria reduce el número de elementos a buscar al filtrar casi la mitad del número de elementos en cada iteración.

El pseudocódigo del algoritmo es el siguiente:

$\boldsymbol{busquedaBinaria(A, n, k)}$
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 implementar este algoritmo en la lista doblemente ligada, el arreglo se remplaza por la lista y para comparar las fechas accede al nodo necesario, obteniendo la fecha registrada en éste.

Para concluir, cabe recalcar que las estructuras de datos son un concepto extensamente útil en programación, nos pueden facilitar la implementación de muchas ideas y soluciones. Nunca se acabarán los usos para las estructuras de datos, nos conducen a comprender la naturaleza de los problemas a un nivel más profundo y, por lo tanto, a una mejor comprensión del mundo.

Referencias:

Doubly Linked List | Set 1 (Introduction and Insertion) - GeeksforGeeks. (2014, March 12). Retrieved October 15, 2021, from GeeksforGeeks website: https://www.geeksforgeeks.org/doubly-linked-list/

Qué son las estructuras de datos y por qué son útiles. (2019, October 18). Retrieved October 15, 2021, from OpenWebinars.net website: https://openwebinars.net/blog/que-son-las-estructuras-de-datos-y-por-que-son-tan-utiles/

Listas enlazadas vs vectores: una gran decisión a nivel de programación. (2020). Retrieved October 15, 2021, from Delfino.cr website: https://delfino.cr/2020/10/listas-enlazadas-vs-vectores-una-gran-decision-a-nivel-de-programacion