Reflexión Actividad 2.3 (Evidencia)

Santiago Reyes Moran - A01639030 - 13/10/2021

Eficiencia de las estructuras de datos lineales

La ventaja de usar estructuras de datos lineales en lugar de estructuras de datos no lineales es que son mucho más simples de trabajar para los propósitos de esta actividad. Ordenar una lista doblemente ligada e implementar algoritmos de ordenamiento y búsqueda eficientes en ella es relativamente fácil en comparación a las estructuras de datos no lineales. A pesar de que las estructuras de datos lineales la memoria no se utiliza de forma óptima, siguen siendo estructuras sumamente potentes al implementar algoritmos como el quick sort y la búsqueda binaria.

En esta actividad se logró implementar de forma eficiente la lista doblemente ligada, existen otros tipos de estructuras de datos lineales que permiten el uso de estos algoritmos:

- El arreglo o vector son posiblemente las estructuras de datos más utilizadas debido a su simplicidad y a la disponibilidad de recursos en línea que existen para ellos. Estas estructuras permiten el acceso directo a sus elementos al utilizar corchetes, lo cual también permite que los algoritmos de búsqueda y de ordenamiento sean bastante eficientes para estas estructuras.
- El stack tiene una implementación bastante sencilla, en donde los elementos que se añaden al stack se van apilando uno tras otro, pero únicamente el último elemento puede ser eliminado (utilizando .pop), es decir que se van eliminando en orden opuesto al que fueron insertados (primero el último) . Dentro de la actividad desarrollada, un stack fue implementado para el algoritmo de ordenamiento iterativo quicksort; esto permitiría guardar información de forma ordenada para acceder a ella posteriormente.
- El queue funciona de manera opuesta al stack; esto se debe a que el único elemento que puede ser eliminado es el primero que se encuentra en la lista, es decir, se van eliminando en el mismo orden en el que se insertaron al queue
- La lista ligada consiste en relacionar (o ligar) un atributo con el siguiente (next), por lo cual únicamente se puede llegar a un nodo si todos los nodos previos a él ya fueron recorridos.
- Finalmente está la lista doblemente ligada, la cual fue implementada en esta actividad; esta funciona de forma similar a la lista ligada, pero tiene un atributo adicional (previous), el cual permite acceder al nodo previo a cualquier nodo de la

lista. Poder acceder a los nodos de esta forma permite que se pueda recorrer la lista de forma lineal y en reversa.

Doubly Linked List en lugar de Singly Linked List

La característica principal que diferencia a una doubly linked list y una singly linked list es el hecho de que la doubly linked list tenga los atributos de "previous" en sus nodos, es decir que una doubly linked list tiene el privilegio de poder acceder al nodo previo desde cualquier otro nodo en la lista. Tener acceso al elemento previo de cualquier nodo facilita mucho el proceso de comparación entre los valores de la lista, con lo cual se puede hacer el ordenamiento de una forma más conveniente.

Uso de Quick Sort

Para realizar esta actividad utilicé el algoritmo de quicksort para ordenar la lista doblemente ligada de objetos conforme su mes, día y hora. Este es uno de los algoritmos de ordenamiento más eficientes que hemos aprendido, junto con el mergesort, los cuales son los dos algoritmos de ordenamientos más recomendables para utilizar en esta actividad. Decidí utilizar quick sort en lugar del mergesort debido a que se puede implementar de forma iterativa usando stacks, lo cual hace que el algoritmo corra de forma más rápida; esto, al considerar que la complejidad temporal promedio de ambos algoritmos es de $O(nlog(n))$, hace que el quicksort sea el algoritmo más recomendable para ordenar la lista doblemente ligada. La complejidad del algoritmo en el peor de los casos (cuando la lista doblemente ligada ya está ordenada) es de $O(n^2)$; la complejidad temporal de este algoritmo es un poco menor que la complejidad temporal de una lista ligada simple si es que la lista ligada simple no tiene una propiedad tail. Poder acceder a tail nos permite mandar llamar el algoritmo de ordenamiento de inicio a fin de manera lineal, es decir, no existe necesidad de atravesar toda la lista doblemente ligada para encontrar el último elemento (tail).

Tabla: Complejidad de los algoritmos de ordenamiento.

Algoritmo	Mejor	Promedio	Peor	Estable	Espacio
Swap sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Sí	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	No	$O(1)$
Buble sort	$O(n)$	$O(n^2)$	$O(n^2)$	Sí	$O(1)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	Sí	$O(1)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Sí	$O(n)$
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	Sí	$O(\log n)$

Uso de Búsqueda Binaria

El uso del algoritmo de búsqueda binaria para este ejercicio es algo bastante obvio, sobre todo cuando comparamos su complejidad temporal con la búsqueda secuencial, en donde se tendría que recorrer el arreglo completo para completar la búsqueda $O(log(n))$. Al tener grandes cantidades de datos (Como en este ejercicio en donde se tienen que crear 16807 objetos) es sumamente importante utilizar algoritmos eficientes, como lo viene siendo la búsqueda binaria $O(log(n))$.

De los algoritmos de búsqueda que cubrimos en la clase, el de búsqueda binaria es el más eficiente, por lo cual decidí implementarlo en mi programa.

Conclusión

La complejidad temporal total de esta actividad es de $O(nlog(n))$, lo cual se debe a la implementación del método quick sort, cuya complejidad temporal también es $O(nlog(n))$. Es importante tomar en consideración que ese algoritmo de ordenamiento fue implementado de forma iterativa y no de forma recursiva dentro del programa, lo cual hace que el algoritmo corra de forma más rápida y por ello es más eficiente.

Considero que implementar la lista doblemente ligada junto con sus algoritmos apropiados fue bastante útil en esta actividad debido a que teníamos que recrear el resultado obtenido de la actividad integradora pasada (en donde teníamos que lograr el mismo objetivo pero utilizando vectores), ya que estamos más familiarizados con los datos que estaríamos manejando y eso nos permitió concentrarnos en la implementación de la lista doblemente ligada junto con el algoritmo de ordenamiento iterativo y la búsqueda binaria.