Tecnológico de Monterrey

Actividad Integradora 2 – Estructuras de Datos Lineales.

Yael García Morelos | A01352461 | Equipo 16.

Programación de estructuras de datos y algoritmos fundamentales. (Gpo 850.)

Profesor. Eduardo Arturo Rodríguez Tello.

07 de abril del 2025.

Desde mi perspectiva como profesional en desarrollo, me voy percatando como la manera en la que suelo racionar un problema e identifico una posible solución va madurando, pues si en mi primer año me hubieran puesto el mismo problema, lo más probable es que hubiera decidido implementar un array en vez de una *doubly linked list (DLL),* esto sobre todo porque me hubiera parecido más sencillo implementar un array ya que no debería de implementar métodos extras, constructores, destructores, entre otros.

Sin embargo, dentro de mí misma ignorancia, estaría descartando las ventajas que presenta la implementación de una *doubly linked list*, al manejar una gran cantidad de datos, la inserción se vuelve un aspecto favorable pues gracias a que se puede avanzar tanto hacia adelante como atrás permite una implementación más sencilla y segura en métodos de búsqueda y ordenamiento, además de eliminar o insertar algún nodo en la lista sin necesidad de recorrerla desde el principio.

Anteriormente se mencionaron las ventajas que hay en una *DLL*, sin embargo, ¿por qué no se utilizó una *linked list*? Si lo vemos desde la perspectiva de una persona que apenas se está familiarizando con las estructuras de datos, le podrían parecer muy similares e innecesario implementar una *DLL* por tener una mayor complejidad, sin embargo, es un pensamiento erróneo, pues implementar un *doubly linked list* permite recorrer la lista en ambas direcciones, favoreciendo búsquedas desde la cola o algún punto medio de la lista. En cambio, si se hubiera aplicado una *linked list*, cada vez que se realice una búsqueda se debería de empezar desde el inicio de la lista pues no se puede retroceder, para simular este efecto se tiene que reiniciar desde el principio.

En la gran mayoría de las funciones utilizadas en este algoritmo tienen una complejidad computacional de *O(n)* y *O(1)*, mientras que solo al ordenar y buscar se tiene una complejidad de *O(n log n)*. Esto favorece nuestro algoritmo ya que se prioriza un buen rendimiento, usar la mínima cantidad de recursos, memoria y tiempo, pues parte del desarrollo de un algoritmo es priorizar la experiencia del usuario y tener funciones y/o métodos con una complejidad de *O(n) y O(n log n)* favorece este aspecto.

Tras haber finalizado con este algoritmo me percaté que los algoritmos de ordenamiento no son complicados de aplicar, desde mi experiencia con este algoritmo, no se presentó ningún problema a la hora de realizar el ordenamiento por merge sort, sin embargo, considero que

lo más complicado de esta actividad fue la búsqueda, cabe resaltar que no es complicado implementar una búsqueda binaria, lo complicado fue adaptar *head* y *tail* para filtrar la información, en un principio ni siquiera lograba posicionar los nodos de manera correcta, sin embargo, tras analizar la lógica y replantear la solución, se logró superar este desafío.

Ya con experiencia en la implementación de *Estructura de Datos Lineales* como lo es un *DLL*, he llegado a la conclusión que la implementación de esta estructura fue la correcta a comparación de una *linked list,* pues la manera en la que te permite recorrer la lista genera una gran ventaja a la hora de manejar grandes cantidades de datos. Por último, considero de gran importancia analizar la problemática a profundidad antes de empezar con el algoritmo, además de realizar como mínimo un pseudocódigo sencillo para tener una idea de la estructura y jerarquía que tendrá nuestro algoritmo, pues realizarlo sin una previa planeación puede generar errores, problemas a la hora de compilar o tiene poca eficiencia.