Jorge Loredo Meléndez A01284185

Reflexión Act 2.3

Las listas encadenadas al igual que los arreglos, vectores y árboles binarios son estructuras de datos que permiten interactuar con la información de una manera específica. La mayor diferencia entre esta estructura y otras estructuras dinámicas es la manera de accesar y mutar los datos que comprenden. Mientras que los arreglos requieren almacenar la información en espacio de memoria contiguos, las listas encadenadas solamente requieren dos espacios contiguos, el primero que contenga el valor del tipo de dato y el siguiente que contenga un apuntador al espacio de memoria siguiente de la lista. A este pequeño segmento de la lista se le llama Nodo.

El tiempo de ejecución y complejidad de diversos algoritmos enfocados en Listas Encadenas son los siguientes:

- AddFirst() : Añade al principio de la lista, complejidad O(1)
- Add / AddLast() : Añade en la posición de la lista indicada, complejidad O(n)
- DeleteFirst() : Borra el nodo al principio de la lista. Complejidad O(1)
- Delete / DeleteLast() : Borra el nodo en la posición indicada, complejidad O(n)
- Set(): Cambia el valor de un nodo O(n)
- Change(): Intercambia dos nodos cualesquiera O(n)
- Get(): Regresa el valor de cierta posición insertada O(n)

La mayor ventaja que presentan las listas encadenadas sobre los vectores y arreglos es la mutación de posiciones. Tiene una mayor eficiencia al momento de insertar y remover datos. Al tener una lista doblemente encadenada si se busca remover o añadir a la primera o última posición, se tendrá una complejidad O(1), de igual manera en cualquier otra posición, no se dejará rastro del elemento removido. Por otro lado, los arreglos no pueden destruir los elementos insertados, solamente se podrán igualar a 0 a menos que todos los valores siguientes cambien de posición. Sin embargo, el ordenamiento de una lista encadenada tiene una complejidad $O(n^2)$, ya que no se puede accesar a valores anteriores al actual. Otra desventaja es que tanto el Set como el Get tendrán una complejidad O(n), debido a que a diferencia de los arreglos y su posición definida, se tiene que recorrer la lista hasta encontrar la posición solicitada.

Para el problema reto una estructura de datos de este tipo resulta favorable ya que las salidas de los buques pueden cambiar constantemente de posición. Por ejemplo si un buque se retrasa, simplemente se apunta el siguiente buque disponible y este anterior se reinserta en otra posición. De igual manera como las salidas pueden utilizar un modelo FIFO o LIFO tener disponibles los dos extremos de la estructura permite un acceso más rápido. Sin embargo, una desventaja muy pronunciada es que si se requiere monitorear cierto buque en específico y no el siguiente en salir, se necesitaría recorrer toda la lista.