TECNOLÓGICO DE MONTERREY®

TC1031 – PROGRAMACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS FUNDAMENTALES
GRUPO 5

Actividad 2.3
Actividad Integral
Estructura de Datos Lineales

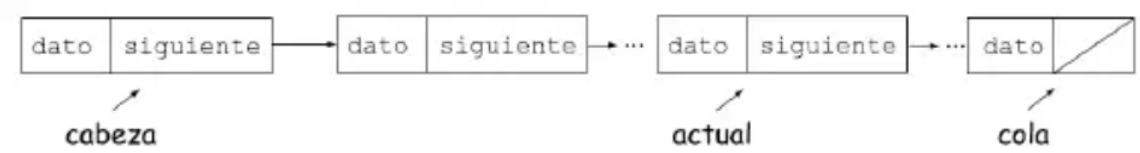
Brenda Elena Saucedo González – A00829855

14 de octubre de 2021

Investigación

Una lista enlazada o encadenada es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos.

La lista encadenada simple consiste en una secuencia de nodos (elementos), los cuales están compuestos por 2 partes, la primera parte contiene la información, y es, por consiguiente, un valor de un tipo genérico, y la segunda parte es una referencia que apunta al siguiente elemento de la lista.



El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.

Las listas encadenadas permiten inserciones y eliminación de nodos en cualquier punto de la lista, eliminando el espacio en memoria que ocupó ese mismo nodo en caso de removerlo. También elimina el problema de los traslados de los nodos. Otra de las ventajas es que no se requiere saber previamente el número de elementos a ser almacenados, puesto que son de tamaño dinámico, lo que permite agregar o eliminar elementos a petición del usuario, además de que implica la optimización de la memoria.

Cuando se tiene un conjunto de elementos del que se quiere remover algunos elementos dependiendo de alguna condición, removerlos de una lista enlazada consiste simplemente en destruir el elemento preciso y actualizar los enlaces de sus nodos “siguiente” adecuadamente. Removerlos de un array dinámico implicaría recorrer todos los elementos siguientes en cada eliminación, mucho menos eficiente.

Aunque posee muchas ventajas el usar este tipo de estructura de datos, también presenta algunas desventajas, como no conseguir de manera directa acceder a los elementos de la lista en tiempo constante, es decir, para acceder a un elemento de un arreglo, solo necesitamos darle la indicación del índice en el que se encuentra el elemento y nos lo retorna inmediatamente, por otro lado, para obtener un elemento de una lista encadenada, debemos de recorrer cada elemento de la lista hasta llegar al índice solicitado.

Otra de las desventajas es que utiliza un mayor número de operaciones para mantener la integridad de los datos, puesto que, para acceder a los demás elementos de la lista, necesita recorrerse así misma, lo que lo hace menos eficiente.

Complejidad Computacional para las Operaciones Solicitadas

Para la solución de la situación problema planteada, hicimos uso de las clases “Node” y “List”, para poder implementar las listas encadenadas en la solución. En la clase “List” tuvimos que hacer uso de diversos métodos para poder agregar nuevos elementos en la lista, ya sea en el orden como se van ingresando los valores o por medio de un orden ascendente (solo aplica para listas encadenadas de tipo numérico), así como acceder a la información de la lista.

Métodos implementados en la clase List:

El método *create*, que cuenta con un solo parámetro de entrada, solo ejecuta las instrucciones una vez, por lo que su nivel de complejidad en el peor de los casos es de O(1), ya que no depende de los elementos ni del tamaño de la lista para ejecutar dichas instrucciones.

El método *create*, que cuenta con dos parámetros de entrada, es de un nivel de complejidad de O(n) en el peor de los casos, ya que hace uso de un ciclo iterativo para recorrer la lista “n” veces hasta llegar a la posición en la que se desea insertar el nuevo elemento, donde “n” depende del tamaño de la lista.

El método *indexOrder* es de un nivel de complejidad de $O(n^2)$ en el peor de los casos, ya que primeramente hace uso de un ciclo iterativo para contar cuantas veces el valor ingresado es menor a cada uno de los elementos de la lista, por lo que va a recorrer la lista “n” veces. Después de dar por finalizada esta iteración, sale del ciclo y envía como parámetros a la función *create*: el valor a insertar y la posición en donde se desea insertar (acumulador), donde este mismo método es de complejidad O(n), razón por la cual el nivel de complejidad del método *indexOrder* es de $O(n^2)$, ya que hace uso de dos ciclos iterativos.

El método *get* es de un nivel de complejidad de O(n) en el peor de los casos, ya que hace uso de un ciclo iterativo para recorrer la lista “n” veces hasta llegar a la posición de la que se desea saber el contenido de ella.

El método *clear* es de un nivel de complejidad de O(n) en el peor de los casos, ya que también hace uso de un ciclo iterativo para eliminar cada elemento de la lista, por lo que recorre a esta misma “n” veces hasta vaciarla por completo.

Finalmente, el método *length* es de un nivel de complejidad de O(1) en el peor de los casos, ya que sólo ejecuta una instrucción que no depende del tamaño de la lista.

Reflexión

Como se había investigado y expresado anteriormente, las listas encadenadas son más prácticas que los vectores y los arreglos en algunos aspectos, ya que estas estructuras hacen el uso de los punteros de asignación dinámica, por lo que nos facilita el trabajo cuando queremos eliminar un nodo (elemento de la lista) y liberar el espacio que ocupaba en memoria, así como agregar un nodo en una cierta posición de la lista sin modificar la estructura ordenada de los nodos que le siguen, es decir, existe un traslado de los nodos de una manera mucho más sencilla. En cambio, para realizar este mismo procedimiento en los arreglos, tendríamos que modificar y recorrer los elementos uno a uno para que siguiera teniendo la misma consistencia. Sin embargo, las listas encadenadas cuentan con la desventaja de que no podemos acceder directamente a una posición de la lista, por lo que para hacer esto posible se tienen que realizar muchas más operaciones para lograrlo.

En situaciones problemas de esta misma naturaleza, las listas encadenadas facilitan el orden y acomodo de los nodos según la estructura que se quiera seguir, así como eliminar la información que ya no nos sea útil. También facilita la manipulación de los elementos guardados o almacenados en este tipo de estructuras de datos, haciendo más eficiente el algoritmo.

Teniendo en cuenta tanto sus ventajas como desventajas, la implementación de las listas encadenadas es mucho mas eficiente en este tipo de situaciones que requieren de un orden estructurado como lo son las fechas y horas (tiempo).

Referencias

Colaboradores de Wikipedia. (s.f.). *Lista enlazada*. Octubre 10, 2021, de Wikipedia. Sitio web: https://es.wikipedia.org/wiki/Lista_enlazada

Rivera, L. (s.f.). *Listas Encadenadas*. Octubre 10, 2021, de SCRIBD. Sitio web: https://es.scribd.com/document/241293518/LISTAS-ENCADENADAS

rogikato. (2009). *Listas Encadenadas*. Octubre 11, 2021, de Programacion C++ y Java. Sitio web: https://codigodeprogramacion.blogspot.com/2009/12/listas-encadenadas.html

Bascón, E. (2019). *¿Cuál es la utilidad de las listas enlazadas?* Octubre 11, 2021, de Quora. Sitio web: https://es.quora.com/Cu%C3%A1l-es-la-utilidad-de-las-listas-enlazadas