Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Programación de estructuras de datos y algoritmos fundamentales – TC1031

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

Reflexión personal

Grupo 14

Adrián Alejandro Salgado Martínez – A00828843

10 de octubre de 2020

Estructuras de datos lineales - Reflexión

Las listas doblemente encadenadas, tal como la que fue utilizada para el desarrollo de esta actividad, representan una de las estructuras de datos más importantes para el almacenamiento de información computacional. Lo que las diferencia de una estructura de datos más simple como es un arreglo es que estas están compuestas de distintos “nodos”, donde además de el tipo de dato almacenado también se encuentran apuntadores a las direcciones de memoria del siguiente nodo, del nodo anterior (solo en las encadenadas dobles), o nullptr si es que se trata del primer o el último nodo. De esta manera se logra conseguir que agregar un elemento a la lista se pueda llevar a cabo de manera constante cuando se tiene acceso a los nodos relevantes, como es el caso del inicio o del final de la lista, y la única limitación para la extensión máxima de esta es la cantidad de memoria total disponible en la computadora.

Sin embargo, una de las más grandes limitaciones de este tipo de estructuras de datos es que obtener acceso a un elemento en un índice determinado no se puede llevar a cabo de manera trivial, como es el caso con un vector o un arreglo, puesto a que se tiene que visitar cada uno de los nodos necesarios para saber cual es el siguiente, ya que la organización en la memoria de estos no es contigua y no se puede llevar a cabo de manera aleatoria, esto significa que para listas considerablemente largas leer y escribir en un índice medio se vuelve realmente costoso, especialmente si no se utilizan algoritmos eficientemente diseñados con estas limitaciones. Es por esto mismo que para esta actividad se decidió implementar una clase iterador además de la clase de la lista, consiguiendo así utilizar los siguientes algoritmos de búsqueda y ordenamiento:

- `DLinkedList<T>::sort()` - El algoritmo que se implementó para ordenar la lista está prácticamente compuesto en su totalidad por el algoritmo utilizado para la lista de la implementación de GCC de STL. Este, como lo dicta la especificación de C++ para cualquier sort de una lista, tiene una complejidad de O(n log n), lo cual es bastante inusual para un contenedor de dicho tipo. Se utiliza en lugar de `std::sort()` ya que este último requiere de iteradores de acceso aleatorio para mantener esta eficiencia, en contraste con los iteradores bidireccionales implementados.

- `std::lower_bound()` - Este algoritmo de búsqueda, disponible en el encabezado `<algorithm>`, encuentra el iterador correspondiente al primer elemento que no es menor a un valor dado en un contenedor ordenado. Su especificación indica que normalmente tiene una complejidad de O(log n), no obstante, al tratarse de iteradores bidireccionales y no aleatorios significa que este tendrá realmente complejidad de O(n).

Se concluye entonces que para esta situación problema el uso de una lista doblemente encadenada es bastante útil por la gran cantidad de datos con los que se trabajan, que permiten inserción y eliminación constante, sin embargo no se piensa que es la mejor que pudo haber utilizada puesto a las limitantes al ordenarla y realizar búsquedas.