Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Programación de estructuras de datos y algoritmos fundamentales – TC1031

Actividad 1.3: Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales (Evidencia Competencia)

Reflexión personal

Grupo 14

Adrián Alejandro Salgado Martínez – A00828843

12 de septiembre de 2020

Algoritmos fundamentales - Reflexión

Los algoritmos de búsqueda y ordenamiento, tales como los que fueron utilizados en el desarrollo de dicha situación problema, son de suma importancia ya que se presentan como una de las bases fundamentales del área de la computación. A través de estos, es posible realizar la manipulación de datos para lo cual es tan útil el uso de la informática, puesto a que nos permite estructurar y desglosar grandes cantidades de información que de hacerse de manera manual sería no solo impráctico, sino también inviable.

Asimismo, un aspecto crucial que diferencia a los algoritmos que existen es su eficiencia, pues a pesar de los múltiples avances que han habido en la computación la realidad es que existen limitaciones de tiempo y de potencia, combinado con el hecho que en ocasiones como en esta situación problema la cantidad de datos con los que se trabaja es inmensa, lo que significa que elegir o diseñar e implementar un buen algoritmo de búsqueda o de ordenamiento trae un gran impacto en la administración de los recursos limitados a nuestra disposición.

Es por esto mismo que para esta actividad se decidió utilizar los siguientes algoritmos de búsqueda y ordenamiento disponibles en la librería estándar de C++:

- `std::sort()` - Este algoritmo de ordenamiento, disponible en el encabezado `<algorithm>`, organiza los elementos en orden ascendente. Su especificación indica que tiene una complejidad promedio de O(n log n). La forma en la que está implementado es por medio del algoritmo Introsort, el cual es una implementación híbrida que combina el uso de algoritmos increíblemente eficientes como lo son Quicksort, Heapsort e Insertionsort, aprovechando sus fortalezas y compensando sus debilidades, obteniendo así una alta eficiencia promedio en comparación a si se utilizara solo uno de ellos.

- `std::lower_bound()` y `std::upper_bound()` - Estos algoritmos de búsqueda, también disponibles en el encabezado `<algorithm>`, encuentran el apuntador a un elemento que es el límite inferior o el anterior al superior comparado a un valor dado en un vector ordenado. Su especificación indica que tienen una complejidad de O(log n). La manera en la que están implementados, como es evidente con su complejidad, es a través de búsqueda binaria modificada.

La ventaja que tuvo usar estos algoritmos internos, en lugar de implementar uno por nuestra cuenta, es que están diseñados para ser increíblemente eficientes y seguros. Sin embargo, una desventaja importante es que existe mucha limitación respecto a cómo podemos modificarlos para necesidades específicas, aunque no haya sido tan impactante gracias a la sobrecarga de operadores.