Reflexión Actividad 3.4

Daniel Alfredo Barreras Meraz - A01254805

Reflexionando sobre el problema de la gestión de pedidos de envío en la compañía naviera “International Seas, Ltd.”, comencé con el primer paso que fue reconocer el problema: un gran volumen de pedidos de envío que necesitan ser ordenados eficientemente. Este problema se agravó con la reducción de las operaciones portuarias debido al COVID-19.

Para resolver este problema, consideré e investigué sobre varios tipos de árboles, tales como los árboles binarios, los árboles AVL, los árboles rojo-negro, los árboles B, los árboles B+, los árboles de segmento y los árboles Trie.

Árboles Binarios: Los árboles binarios son una estructura de datos jerárquica que puede ser útil para almacenar datos que tienen una relación natural de “menor que” o “mayor que”. Sin embargo, en nuestro caso, los pedidos no se hacen de manera jerárquica, por lo que es mejor apuntar hacia otro tipo de árbol.

Árboles AVL: Los árboles AVL son un tipo especial de árbol binario que se autoequilibra para mantener las operaciones eficientes. Aunque son útiles en ciertos escenarios, no proporcionan la flexibilidad necesaria para nuestro problema.

Árboles B y B+: Los árboles B y B + son útiles para almacenar y recuperar grandes cantidades de datos. Aunque son eficientes para ciertos tipos de problemas, no son adecuados para nuestro problema debido a su complejidad y a la necesidad de un acceso aleatorio a los elementos.

Árboles de Segmento: Los árboles de segmento son útiles para responder a consultas de rango en un conjunto de puntos. Sin embargo, en nuestro caso, no necesitamos realizar consultas de rango, por lo que este tipo de árbol no es adecuado.

Árboles Trie: Los árboles Trie son útiles para almacenar palabras en un diccionario para búsquedas rápidas y eficientes. Aunque son útiles en ciertos escenarios, no proporcionan la flexibilidad necesaria para nuestro problema.

Árboles Heap: Los árboles Heap son un tipo especial de árbol binario que cumple con la propiedad de heap. Esto significa que el valor de cada nodo es mayor o igual (en un heap máximo) o menor o igual (en un heap mínimo) que los valores de sus nodos hijos. Esto hace que el nodo raíz sea el elemento máximo (heap máximo) o mínimo (heap mínimo). Los heaps son comúnmente utilizados para implementar colas de prioridad.

Los árboles heap, se utilizan comúnmente para implementar listas de prioridad, las cuales son una estructura de datos que permite almacenar elementos con una cierta prioridad y acceder al elemento de mayor prioridad de manera eficiente. Los heaps

son especialmente útiles en este contexto por varias razones. En primer lugar, permiten insertar nuevos elementos y eliminar el elemento de máxima prioridad en tiempo logarítmico, lo cual es muy eficiente. En segundo lugar, permiten acceder al elemento de máxima prioridad en tiempo constante.

En el contexto específico del problema que estamos considerando, los Heaps Mínimos resultaron ser particularmente útiles. Los heaps mínimos permiten acceder al elemento de menor prioridad (es decir, el pedido con la menor prioridad) de manera eficiente, lo cual es crucial para poder manejar rápidamente los pedidos más urgentes.

Después de haber encontrado que las estructuras de datos basadas en grafos/nodos, como los árboles Heap, son fundamentales para resolver problemas complejos en el mundo real, ya que permiten acceder al elemento de menor prioridad de manera eficiente, lo cual es crucial para poder manejar rápidamente los pedidos más urgentes. Esta eficiencia es vital en aplicaciones del mundo real donde el tiempo es un recurso valioso.

En mi experiencia, entender y ser capaz de implementar estas estructuras de datos es una habilidad valiosa para cualquier desarrollador de software o futuro ingeniero en tecnologías computacionales. No sólo me ha ayudado a resolver problemas más eficientemente, sino que también me hará más atractivo para los empleadores potenciales.

Para finalizar, me gustaría decir que aunque los heaps en general y los heaps mínimos en particular son solo uno de los muchos tipos de árboles disponibles, su capacidad para manejar eficientemente las prioridades los hace una opción muy adecuada para resolver este problema específico. Aunque otros tipos de árboles podrían haber sido útiles en otros contextos, las características específicas del problema hicieron que los heaps mínimos fueran la opción más adecuada. Como futuro ingeniero en tecnologías computacionales, estoy emocionado por las posibilidades que estas estructuras de datos avanzadas ofrecen para resolver problemas complejos.

Referencias de investigación:

GeeksforGeeks. (2023). Graph Data Structure And Algorithms. Recuperado de https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Sugandhi, A. (2023). Trees in Data Structure: Types, Properties and Applications. Knowledgehut.com. https://www.knowledgehut.com/blog/programming/types-of-trees-in-data-structure