Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructura de Datos y Algoritmos

Informe Tarea 0

20 de mayo de 2024
INFORME DE EJEMPLO TIPO
12345678

Motivación

Heaps y los Árboles de Búsqueda son estructuras de datos fundamentales para algoritmos, donde se tiene que tener una cola de prioridad, como A-Estrella, o índices de búsqueda, como en sistemas de bases de datos. En esta tarea, se utilizan estas estructuras, empleando técnicas algorítmicas para mantener colas de prioridad para simular un mercado en el caso de la parte 1, y diferentes estrategias con árboles de búsqueda binaria en la parte 2.

Informe

Parte 1: Heaps

Para simular el libro de órdenes de compra y venta, max y min-heap respectivamente, como cola de prioridad. Se asume que la cantidad máxima que puede tener la cola es la cantidad de eventos, empleando un arreglo para guardar la cola.

- El evento estatus solo ve el tope de ambos heaps, lo que es $O(1)$. Notar que si fuera un árbol de búsqueda, encontrar estar órdenes sería $O(\log(n))$.
- Eventos SEEL y BUY añaden una orden de venta y compra respectivamente. Sin perdida de generalidad, si se ejecuta una orden de venta SELL, y se ve que la orden de compra mayor es mayor ($O(1)$ ver mayor en max-heap), se guarda la orden de venta al min-heap ($O(\log(n))$ inserción). En el caso de que la orden de compra mayor sea menor que la orden de venta, se ejecuta las órdenes, eliminando la orden de compra del max-heap ($O(\log(n))$. Ambas operaciones, por lo tanto, son $O(\log(n))$.

Parte 2: Árboles de búsqueda

Para esta parte, lo que se hace es construir diferentes árboles de búsqueda con diferentes índices. Los nodos resultantes apuntan a un arreglo que almacena toda la información correspondiente. A medida que se van guardando los nodos en el arreglo, se van insertando en los diferentes árboles empleando el balanceo de los árboles AVL.

- Búsqueda por ID: Se hace un simple árbol de búsqueda binario por ID, obteniendo el índice en los registros para imprimir lo buscado, en complejidad $O(\log(n))$
- Búsqueda por año: Se hace un árbol binario, pero cada nodo, almacena una lista ligada con todos los elementos del mismo año. El orden de esta lista es la de inserción, por lo que encontrar el año obtiene la lista ordenada de resultados. La cantidad está en el nodo para no tener que recorrer dos veces la lista para realizar lo pedido. Esto resulta en complejidad $O(\log(n) + k)$, con k resultados.
- Búsqueda por año y región: Se construye a partir del árbol anterior, pero con árboles anidados. Cada nodo de un árbol de año tiene además un árbol de región, y cada nodo de región, tiene la lista de elementos en el mismo año y región. Cada árbol tiene operación de búsqueda $O(\log(n))$, quedando en $O(\log^2(n) + k)$ por la anidación del árbol. Al igual que la parte anterior, se obtiene el orden esperado.

- Búsqueda por rango X: Se utiliza la misma estructura que por año, solo cambia el algoritmo. El algoritmo, basado en Range Tree, busca el nodo más alto que esté dentro del rango, y con eso, se busca el nodo a la izquierda y a la derecha del rango, incluyendo en los resultados que hay entre estos. Realizando esta iteración recursiva en profundidad en orden, se obtienen los resultados en orden. Recorrer el árbol de esta manera e imprimir los resultados tiene complejidad $O(\log(n) + k)$
- Búsqueda con círculo: Se emplea las estrategias de los 2 árboles y algoritmos anteriores. Se reutiliza el árbol X, pero cada nodo tiene un subárbol y. Se itera primero por $x = X \pm R$, y en cada subárbol con valor x, en $y = Y \pm \sqrt{R^2 - x^2}$. No es necesario tener la lista ligada en este caso, porque no existe elemento con mismo (x, y). Siguiendo la misma iteración de la búsqueda anterior, se logra $O(\log^2(n) + k)$.

La complejidad de la búsqueda con llaves de múltiples atributos (año-región y círculo), se podría optimizar a $O(\log(n) + k)$ utilizando versiones optimizadas de Range-Trees, pero se escapa a lo realizado en la tarea con $O(\log(n))$ de operación de búsqueda en cada árbol.

Conclusión

En esta tarea, se logró revolucionar problemas que involucran heaps y árboles de búsqueda, empleando técnicas algorítmicas para adaptar estas estructuras a las características del problema pedido. En la parte 1, se empleó heaps para realizar de forma eficiente, con $O(\log(n))$, las ejecuciones de órdenes, y $O(1)$ para ver el estado. En la parte 2, se utilizó diferentes de árboles búsqueda con ciertas modificaciones, como permitir múltiples elementos con la misma llave y múltiples atributos en la llave. Además, se utilizó iteración por rangos con búsqueda por profundidad, cuando se busca un rango de llaves, obteniendo $O(\log^2(n)+k)$ como peor complejidad considerando búsqueda $O(\log(n))$ e iteración $O(k)$ de lo buscado.

Referencias

- J. Antoni Sellars. Lecture 8: Range trees, Computational Geometry, Universitat de Girona. https://ima.udg.edu/~sellares/ComGeo/RangeTrees4Ms.pdf