Universidad Nacional
de Entre Ríos

Facultad de ingeniería

Tecnicatura en Procesamiento y Explotación de Datos

Algoritmos y Estructura de Datos

Docentes:

Dr. Javier Eduardo Diaz Zamboni
Bioing. Jordán F. Insfrán
Bioing. Diana Vertiz del Valle
Bruno M. Breggia

Actividad: Informe de análisis TP 2.

Alumnos: Starchevich Adriel , Escalante Axel

23 Octubre 2023

Introducción.

En el siguiente informe se detallarán las implementaciones de montículo binario de mínimos, Árbol AVL utilizando nodos y por último se verá implementado la estructura de datos de Grafo y el funcionamiento del algoritmo Dijsktra junto con montículos binarios levemente modificados para adaptados al problema.

Desarrollo

Implementación Sala de Emergencias

Para la correcta implementación de una sala de emergencias se utilizó un montículo binario de mínimos ya que este era el algoritmo adecuado para resolver el problema de la atención a los pacientes por nivel de riesgo. Para esto se creó un módulo con la clase de montículo binario, luego se modificó a la clase paciente agregándole métodos especiales __gt__ y __lt__ para que se puedan comprar a los objetos pacientes entre sí (utilizando el nivel de riesgo como comparación) ya que había un problema a la hora de utilizar el montículo. Después se implementa el TAD de sala de emergencias en donde se ingresan los pacientes, se atienden y muestra la cantidad de estos. Por último hicimos leves modificaciones en el main para utilizar el TAD de sala de emergencias.

En cuanto al orden de complejidad de las operaciones de inserción y eliminación en este montículo:

La inserción agrega un elemento al montículo y luego realiza el proceso de filtrado hacia arriba (infiltArriba) para mantener el orden del montículo. La inserción se realiza en tiempo constante (O(1)), ya que solo se agrega un elemento al final de la lista y luego se realiza el filtrado hacia arriba, que no toma más tiempo que log(n). Por lo tanto, se puede concluir que la complejidad de la inserción es O(log n).

La eliminación del mínimo elimina el elemento mínimo del montículo y luego realiza el proceso de filtrado hacia abajo (infiltAbajo) para mantener el orden del montículo. La eliminación del mínimo se realiza en tiempo constante (O(1)) al eliminar el elemento mínimo, que es el primer elemento en la lista. Luego, el proceso de filtrado hacia abajo se realiza en O(log n) ya que, en el peor caso, podría requerir que el elemento eliminado sea "descendido" hasta la hoja más profunda del montículo. Por lo tanto, la complejidad de la eliminación del mínimo es O(log n), donde "n" es el número de elementos en el montículo.

Implementación de Temperaturas DB

En este problema utilizamos un AVL con nodos en vez de una lista, implementamos un archivo nodoArbol para el árbol el cual contiene lo necesario para el correcto

funcionamiento del mismo, con respecto al AVL, agregamos el rotar derecha el cual tiene su modificación más importante en la actualización del factor de equilibrio ya que este cambia mucho con respecto al rotar izquierda. Otro método modificado (partiendo del método básico de ABB) fue el eliminar, a éste lo adaptamos para que actualice el equilibrio ya que a la hora de eliminar no lo hacia y no cumplia el requisito básico para ser un AVL, luego agregamos un método nuevo para actualizar equilibrio al eliminar, este se va a llamar recursivamente hasta que llegue a la raíz, también agregamos al metodo eliminar el re-equilibrar para el caso de que se desequilibre cuando se remueva el nodo.

Para el entorno utilizamos la librería datetime así creamos un objeto de tipo fecha y lo guardamos en el AVL, para sacar la máxima temperatura en un rango de fechas seguimos la misma lógica para obtener el máximo en una lista, en el método de la temperatura mínima es la misma estructura nada más que cambia la condición, ya que se trata del mínimo. Por último para devolver las temperaturas en un rango de fechas, iteramos el árbol y mientras la fecha se encuentre en ese rango, obtenemos las temperaturas.

Con respecto al orden de complejidad:

Método	Complejidad Big-O	Explicación
‘__init__’	O(1)	Inicializa un objeto de la clase, que crea una instancia de un árbol AVL vacío. Esto es una operación de tiempo constante.
‘guardar_temperatura’	O(log n)	Agrega una temperatura a un árbol AVL y realiza una inserción, que tiene una complejidad de O(log n) en promedio. La conversión de la fecha a objeto de fecha también tiene una complejidad constante.
‘devolver_temperatura’	O(log n)	Obtiene una temperatura del árbol AVL utilizando la fecha como clave de búsqueda. La operación de búsqueda en un árbol AVL tiene una complejidad de O(log n) en promedio.
‘max_temp_rango’	O(k*log n)	Itera a través del árbol AVL para encontrar la temperatura máxima en un rango de fechas (desde

		fecha1 hasta fecha2). En el peor caso, se iterará a través de "k" elementos ("k" es la cantidad de elementos en el rango, y puede ser igual a "n"). La búsqueda de cada temperatura en el árbol AVL tiene una complejidad de O(log n), y se realiza un máximo de "k" veces.
‘min_temp_rango’	O(k*log n)	El orden de complejidad es el mismo a max_temp_rango, ya que solamente cambia la condición de búsqueda.
‘temp_extremos_rango’	O(k*log n)	Este método llama a min_temp_rango y max_temp_rango, que tiene una complejidad total de O(2k * log n) pero la constante multiplicativa (2 en este caso) no afecta la complejidad, por lo que la complejidad total es O(k * log n).
‘borrar_temperaturas’	O(log n)	Elimina una temperatura del árbol AVL utilizando la fecha como clave de búsqueda. La operación de eliminación en un árbol AVL tiene una complejidad de O(log n) en promedio.
‘devolver_temperaturas’	O(k*log n)	Itera a través del árbol AVL para mostrar las temperaturas dentro de un rango de fechas. La complejidad es similar a la de max_temp_rango y min_temp_rango.
‘cantidad_muestras’	O(1)	Devuelve la cantidad de muestras almacenadas en el árbol AVL, por lo que se obtiene en tiempo constante.

Implementación del Servicio de Transporte

En la última actividad propuesta por la cátedra, abordamos un ejercicio en el que una empresa de CasaBella deseaba determinar cuál era el peso máximo que podía transportarse desde la ciudad de Buenos Aires a un destino específico, así como conocer el precio mínimo para realizar dicho transporte. Para abordar este desafío, decidimos emplear una estructura de grafo, en la que los vértices representan las ciudades y las aristas representan las rutas que conectaban dichas ciudades.

En este proceso, creamos un grafo para representar los caminos con sus respectivos pesos y precios. Para identificar el cuello de botella (es decir, el peso máximo de los mínimos) entre Buenos Aires y un destino determinado, utilizamos el algoritmo de Dijkstra con algunas adaptaciones. Primero, implementamos una cola de prioridad que opera sobre un montículo de máximos de tuplas. Inicialmente, asignamos una distancia infinita al vértice de inicio, mientras que los demás vértices se mantenían en un estado de infinito. Al final, este algoritmo retorna una lista de grafos los cuales contienen la secuencia de los diferentes caminos (si es que existen esos caminos). Esta estrategia se empleó con el propósito de identificar los máximos entre los mínimos posibles.

En cuanto a la búsqueda del precio mínimo para el transporte desde la Ciudad de Buenos Aires hacia un destino, aplicamos el algoritmo de Dijkstra (sobre cada grafo de la lista de grafos que retorna el anterior algoritmo) con una cola de prioridad basada en un montículo de mínimos de tuplas. En este caso, asignamos un valor de inicio igual a 0 y consideramos el resto de los vértices como infinito. Esta configuración se aplicó para garantizar que se cumpliera la fase de relajación en la búsqueda del precio mínimo. Es importante mencionar que luego de aplicar este algoritmo a cada grafo de la lista (si es que hay más de un camino), se guarda el camino junto con su respectivo precio, después se busca el mínimo de la lista y nos quedamos con el camino y su precio.

Esta implementación nos permitió proporcionar la respuesta que necesitábamos, tanto en términos de capacidad de carga como de costo mínimo, para su servicio de transporte entre Buenos Aires y sus diversos destinos.

Conclusión:

En este informe, se exploraron implementaciones de estructuras de datos y algoritmos en el contexto de aplicaciones específicas. En primer lugar, se utilizó un montículo binario de mínimos para diseñar una Sala de Emergencias que prioriza la atención de pacientes según su nivel de riesgo.

En segundo lugar, se empleó un Árbol AVL con nodos en lugar de una lista para el seguimiento de temperaturas en una base de datos. Esta implementación se hizo con funciones modificadas de rotación, eliminación y reequilibrio, asegurando que el árbol siempre cumpla con las propiedades de un AVL. Además, se permitió el acceso a máximas y mínimas temperaturas en un rango de fechas.

Por último la implementación del Servicio de Transporte, demostró la utilidad de las estructuras de grafo y el algoritmo Dijkstra en la resolución del problema. La utilización de colas de prioridad basadas en montículos de máximos y mínimos, junto con sus modificaciones, permitió identificar el cuello de botella en términos de peso máximo y determinar el precio mínimo para el transporte de manera eficaz.