Trabajo Práctico N°2
Algoritmos y Estructuras de Datos

Villaverde, Federico
Pauligk, Maximo

Objetivos

- Aplicar conceptos teóricos sobre estructuras jerárquicas, grafos y sus algoritmos asociados.

Resolución del Ejercicio 1:

En la situación inicial nos encontramos con la sala de espera de un centro de salud en la cual el orden en el que se atendían los pacientes estaba regido por el orden de llega de éstos (estructura lineal de cola).

Era necesario desarrollar un sistema de triaje que permita ir atendiendo los pacientes según un orden de prioridad de su estado (1- crítico, 2-moderado, 3-bajo),

Con ésta situación, optamos por resolver el problema apoyándonos sobre la estructura de Montículo Binario, en éste caso montículo mínimo. Esta estructura de datos en particular tiene la peculiaridad de diagramar como un árbol binario, pero internamente su estructura es una lista. Cómo es un montículo mínimo, en su raíz está el menor elemento, ósea que al quitar un elemento (atender un paciente) se al de menor valor en la lista. A continuación, analizamos el comportamiento y el orden de complejidad de las inserciones y eliminaciones en ésta estructura.

El montículo binario tiene la particularidad que se puede creer que agregar y eliminar un elemento tiene orden O(1). Pero en realidad, no sólo tenemos que mantener la estructura del montículo sino que también el orden:

Insertar: (Ingresa un paciente) Al insertar un elemento, se agrega al final de la lista (en las hojas del árbol binario) y entra en juego el método *infiltrarArriba.* Explicado de forma sencilla, lo que hace el método es ir comparando éste elemento con su padre de forma recursiva y reemplazarlo en caso de que éste sea mayor. Así hasta que se cumpla el orden o llegue a la raíz. El peor caso ocurre cuando el elemento tiene que ascender hasta la raíz y por lo tanto recorrer todos los niveles, si dividimos la posición del último elemento (i) por 2 dor forma recursiva, entonces: —----------> O(log n)

Eliminar: (Atender un paciente) Cuando se atiende un paciente, se elimina el elemento de la raíz e internamente el montículo binario lo reemplaza con el último elemento de la lista para conservar su estructura. Ahora, para conservar su orden, se utiliza el método *infiltrarAbajo*. Este método compara el elemento en cuestión con el menor de sus hijos y en caso de ser mayor, se intercambian. Así recursivamente hasta que cumpla el orden del montículo o llegue al final. cabe destacar que se accede al menor de los hijos de un elemento a través de *hijoMin* y la comparación entre pacientes está definida en su clase a través de ***__lt__*** (primero los compara por el riesgo y luego lo hace por el orden llegada).

En el peor caso, el método recorre todos los niveles del árbol hasta llegar al final, es decir tiene un orden de complejidad de —------> O(logn)

Resolución del Ejercicio 2:

Para resolver el segundo ejercicio, implementamos como lo sugiere la consigna un Árbol Binario de Búsqueda AVL. Primero, desarrollamos el árbol y luego lo utilizamos para la implementación de la clase Temperaturas, la cual es la interfaz con la que trabaja el usuario.

La clase Temperaturas cuenta con los métodos solicitados en la consigna así como también un método “fecha_dato”el cual toma como parámetro un fecha ingresada en la forma “dd/mm/aaaa” y lo convierte en un objeto datetime que nos permite realizar comparaciones entre ellas.

A continuación se presenta una tabla con los análisis de complejidad de los métodos de la clase Temperaturas:

Método	Complejidad	Justificación
`guardar_temperatura`	O(log n)	Inserta en un AVL y convierte fecha (O(1) + O(log n)).
`devolver_temperatura`	O(log n)	Búsqueda de una clave en AVL.
`max_temp_rango` / `min_temp_rango`	O(k + log n)	Búsqueda en rango de k nodos dentro del AVL.
`temp_extremos_rango`	O(k + log n)	Ejecuta dos búsquedas de rango.
`borrar_temperatura`	O(log n)	Eliminación con rebalanceo en AVL.
`devolver_temperaturas`	O(k + log n)	Recorre k nodos dentro del rango del AVL.
`cantidad_muestras`	O(1)	Devuelve una variable interna del árbol.
`fecha_dato`	O(1)	Convierte string a `datetime`.

Recordemos que en los árboles AVL, las operaciones fundamentales como inserción, eliminación y búsqueda tienen una complejidad de O(log n) debido a que el árbol se mantiene balanceado automáticamente después de cada modificación, lo cual garantiza una altura logarítmica.

Resolución del Ejercicio 3:

En el tercer y último ejercicio se nos plantea un desafío en donde se debe complacer una red de información en donde una aldea solo puede enviar datos a su aldea vecina, y a su vez se utiliza como ponderación la distancia a la que se encuentra dicha aldea en referencia a su vecina. El objetivo es minimizar la distancia recorrida por las palomas, es decir, encontrar la forma mas eficiente de que el mensaje llegue a cada una de las aldeas sin recurrir a ciclos entre las mismas.

Partiendo de esto, la solución al problema es realizar un grafo con el algoritmo Prim, un método eficiente para encontrar un árbol de expansión de costo mínimo (AEM) en un grafo ponderado no dirigido. Ayudándonos de esto se encontró una solución viable al problema de las aldeas que previamente se nos presentó.

Se utiliza el método Prim dado que es un grafo ponderado y no direccional en donde el objetivo es encontrar la forma mas eficiente de transferir la información a los distintos vértices del mismo, dicho de otra forma, la forma mas eficiente de interconectar todas las aldeas de modo que la distancia total recorrida por las palomas sea la mínima.

Como resultado final se llegó al objetivo propuesto hallando el camino mínimo o de ponderación mínima y, adicionalmente, se ordenaron las aldeas en orden alfabético, se muestra el paso de la noticia a través de las distintas aldeas y se realiza la suma de las distancias recorridas.