Informe Tp2

Grupo: G14

Integrantes, Padron:
Nicolas Amigo, 105832
Franco Nicolini, 105632

Ayudante: Fiona González Lisella

Estructuras de datos utilizadas: Para poder simplificar el trabajo y poder abstraernos a la hora de hacer el programa principal decidimos crear los TDA especialidad, doctor y paciente.
Creamos estos TDA con el motivo de no tener que implementar funciones más complejas como el pedir turno o el atender paciente dentro del programa principal zyxcba.c. Adems de las estructuras creadas para el programa también usamos TDA adicionales dentro del programa principal
Primero utilizaremos un hash para guardar todas la especialidades donde la clave del hash es el nombre de la especialidad y su valor es el TDA especialidad. Decidimos usar un hash porque nos importaba que acceder a una especialidad sea en el menor tiempo posible. Por mismas razones usamos un hash para los pacientes donde también la clave es el nombre del paciente y el dato es su TDA paciente. Finalmente para los doctores también buscamos que acceder a uno de estos sea de menor complejidad temporal posible pero que estos estén guardados en una estructura donde se respeta un orden alfabético de los doctores para el comando informe. Por estas dos condiciones razonamos que la mejor estructura a utilizar en este caso sería un abb (arbol de busqueda binaria)

1. Especialidad Tda

El objetivo de este TDA es simplificar muchísimo el código de nuestro programa principal. Para implementar lo pedido usamos dos TDA previamente vistos en clase. Usamos dos "colas de prioridad" , una para los pacientes regulares y otra para los pacientes de urgencia.

- Regulares: Está implementada con un heap ¿Porque? Porque el sistema se basa en el año de inscripción para determinar en qué orden se debe atender a los pacientes. Creemos que es la mejor estructura para conseguir la mejor complejidad temporal.
- Urgentes: Está implementada con una cola. ¿Por qué? Porque estos se atienden por orden de llegada sin importar su año de inscripción. Este TDA nos permite lograr atender pacientes en complejidad O(log(d)) (Explicado ms abajo).

1.1. Órdenes pedidos

Pedir turno: En el caso urgente, logramos el O(1) ya que este comando depende de tres cosas:

- Verificar que exista la especialidad. (Es acceder a un hash O(1)).

- Verificar que existe el paciente. (Es acceder a un hash O(1)).
- Encolar al paciente en la cola de urgencia (O(1)).

En el caso regular deberá ser O(log(n)). Esto se debe a que encolar al paciente en la cola de regulares es O(log(n)).

1.2. Atender paciente

En el caso regular:

- Verificar que exista el doctor. (Es acceder a un abb O(log(d))).
- Verificar que existe el paciente. (Es acceder a un hash O(1)).
- Desencolar al paciente en la cola de regulares(O(log(n))).

En el caso urgente:

- Verificar que exista el doctor. (Es acceder a un abb O(log(d))).
- Verificar que existe el paciente. (Es acceder a un hash O(1)).
- Desencolar al paciente en la cola de urgencia (O(1)).

Entonces la complejidad total es la suma de todas las complejidades individuales:

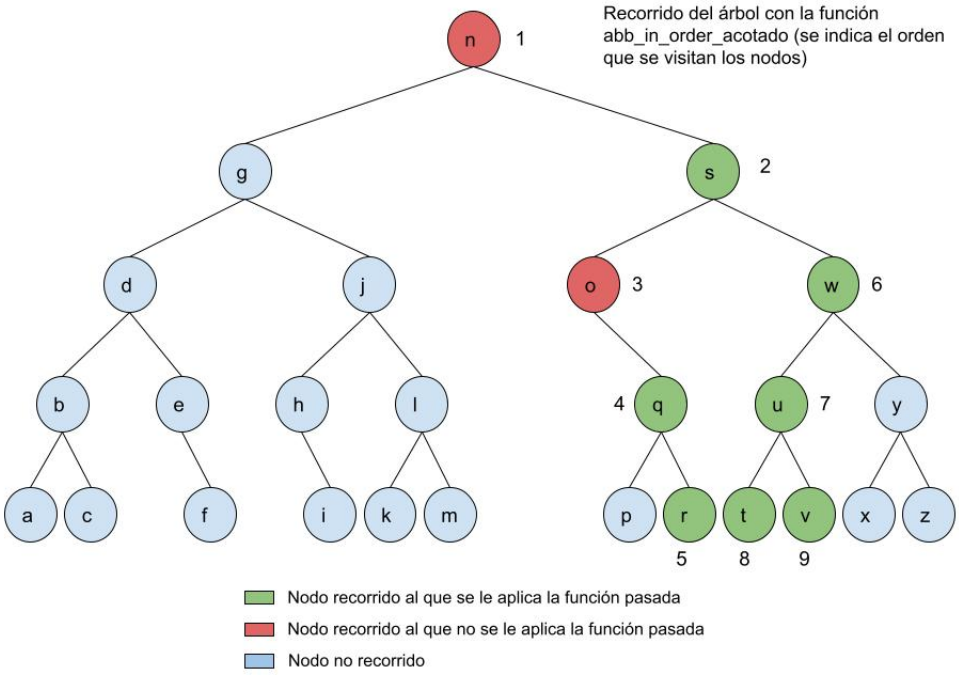
$$T(n) = O(log(d)) + O(1) + O(log(n)) = O(log(d) + log(n)) \quad (1)$$

En el caso urgente:

$$T(n) = O(log(d)) + O(1) + O(1) = O(log(d)) \quad (2)$$

1.3. Informe

Como ya mencionamos los doctores están guardados en un abb. Por lo tanto, si hiciéramos un recorrido in-order, la complejidad seria O(d) para todo caso que nos pidieran hacer el comando informe, lo cual no cumple con las pautas de trabajo.
Debido a esto creamos una nueva función que haga un recorrido similar a un in-order pero únicamente en la franja de valores que le pasamos.
Debajo hay un ejemplo de esta funcion en un arbol acotado ente 'q' y 'w'



Viendo el ejemplo podemos ver que la complejidad del algoritmo (para rangos chicos) no dependera de la cantidad total de nodos sino con la altura del arbol, que sera en promedio log(d).
De esta forma cumplimos la consigna, logrando una complejidad de O(log(d)) en el caso en que se piden pocos doctores.

Tambien en el caso que se nos pida un informe de todos los doctores la funcion utilizada actuara exactamente como un recorrido in-order, el cual tiene una complejidad temporal lineal.