TP N° 2
Clínica Zyxcba Inc

Algoritmos y Programación 2
Curso Buchwald
2° Cuatrimestre 2020

Alumnos:

Aceval Daniel Adrian (104082)
Olmos Gonzalo Agustin (105410)

Corrector: Accini Tomas

Diseño del programa

Para empezar a diseñar el programa nos pusimos a analizar principalmente las complejidades que cada comando deberia tener. Siendo que el **pedir turno** deberia ser ***O(1)*** en el caso de ser urgente y ***O(log n)*** (n cantidad de pacientes en la especialidad) para los pacientes regulares ,pensamos que en la especialidad deberia tener un heap para los casos regulares y un cola para los urgentes dentro del TDA hash, donde la clave fuera la especialidad y el dato un struct con las dos colas. Entonce acceder a la especialidad seria ***O(1)*** y encolar un urgente seria otra vez ***O(1)*** , la misma complejidad para acceder a la especialidad de los pacientes regulares, pero encolar ***O(log n)***.

De igual manera para el comando **informe** de los doctores en el sistema, como la complejidad tiene que ser ***O(log d)*** (d cantidad de doctores en el sistema), para un caso promedio, y ***O(d)*** en el peor caso. Directamente pensamos en un ABB ya que tenemos que recorrerlo en orden alfabetico y no un hash porque la principal ventaja en este caso es que el ABB nos provee de obtener los elementos en orden si quisieramos. Agregando unas primitivas nuevas al ABB obtenemos la complejidad en ***O(log d)***.

Para el caso del comando **atender al sigiente** debemos buscar al doctor en el abb que eso cuesta ***O(log d)*** y "conectar" de alguna manera la especialidad del doctor con sus lista de pacientes y es ahi donde hacemos el uso de hash que contiene las especialidades en ***O(1)*** entonces desencolar para un paciente urgente es simplemente desencolar de la cola que es ***O(1)*** cumpliendo el ***O(log d)*** de complejidad. Para atender un paciente regular es el mismo procedimiento, menos al desencolar en un heap ***O(log n)***, donde la complejidad total es ***O(log d + log n)***.

Primitivas nuevas ABB

Iterador interno por rangos

Para el iterador interno por rangos, es igual al iterador interno normal, excepto que recibe dos paramentros un min y un max donde estos son los rangos donde mi iterador "visita" con la funcion visitar. Para este trabajo era necesario crear este iterador para cumplir con la complejidad pedida. La complejidad para este iterador es ***O(log n) + k*** (siendo n la cantidad de elementos en el arbol y k los elementos de mi rango). Si **k** es igual a **n** la complejidad se vuelve ***O(n)***, en cambio si **k** no es un numero de elementos tan alto se podria despreciar y dejar la complejidad en ***O(log n)***.

Devolver menor clave

Esta primitiva solo me devuelve la menor clave dentro del arbol,para el TP lo usamos para reemplazar el parametro min del iterador interno por rangos en caso de que el usuario no especifique nada por parametro para este.

Devolver mayor clave

Esta primitiva solo me devuelve la mayor clave dentro del arbol,para el TP lo usamos para reemplazar el parametro max del iterador interno por rangos en caso de que el usuario no especifique nada por parametro para este.