Trabajo Practico N°2

Alumnos:

- Paz Blanco, Pilar – 105600
- Rivero Trujillo, Tobías Luciano – 106302

Ayudante:

- Agustín López Núñez

En el siguiente informe explicaremos nuestra resolución para el Trabajo Practico número dos y presentaremos las estructuras utilizadas y los motivos detrás de su implementación.

Para el trabajo distinguimos tres grupos fundamentales: especialidades, pacientes y doctores, para lo cual, creamos sus respectivos TDAs para poder manejar más fácilmente sus datos.

Para las especialidades, debido a los requerimientos de complejidad, creamos un *struct* que se separaba en dos Hashes: uno para la espera de especialidades urgentes y otro para las mismas pero regulares, esto nos permitió poder chequear rápidamente si ya se encontraban en el sistema, poder guardarlas y poder acceder a ellas en O(1). En el Hash de especialidades que contendría a los pacientes que vayan de urgencia , creamos una Cola para cada especialidad debido a su propiedad de *first in first out* y que gracias a que la estructura cuenta con una referencia al primer elemento encolado, permite trabajar en O(1) a la hora de pedir un turno de urgencia. Para el Hash de especialidades que contendria a los pacientes regulares, empleamos un Heap de mínimos en base a su antigüedad, en donde un paciente con mas antigüedad, tiene un año de inscripción mas pequeño, el cual es el dato que se recibe por archivos, con dicha estructura se respeta complejidad temporal requerida (Heap_encolar() funciona en O(log n) siendo n la cantidad de pacientes encolados).

En tanto a los pacientes, decidimos también almacenarlos en un Hash con sus nombres como clave y sus antigüedades como valor, logrando tener primitivas, como antigüedad y guardar paciente, que funcionen en O(1), puesto que son utilizadas de manera frecuente para el funcionamiento del hospital.

Finalmente, en tanto a los doctores, decidimos crear un TDA donde estos estuviesen almacenados en un ABB, utilizando a la función *strcmp* como función de comparación entre claves, que en este caso correspondían al nombre de los doctores. Para el valor, creamos un *struct* con el nombre del doctor, su especialidad y la cantidad de pacientes atendidos para acceso rápido a los datos. De esta manera, para pedir un turno, utilizando la primitiva para obtener la especialidad, se buscaba al doctor en el ABB en O(log d) (donde d es la cantidad de doctores). Luego, con ese dato, podíamos acceder al Hash de especialidades con los pacientes urgentes y atender al siguiente paciente en O(1) como fue explicado previamente. En caso de no haber más pacientes en la Cola de urgencia, se accede a los pacientes regulares, resultando en una complejidad de O(log d) + O(log n).

El principal motivo de haber elegido un ABB para los doctores en vez de un Hash, como con las especialidades y los pacientes, es que para el informe debíamos tener los nombres ordenados de manera alfabética. Al ordenar los doctores con *strcmp* si realizábamos un recorrido *in order* (Realizado recursivamente debido a los requerimientos de complejidad) obtendríamos la lista de los doctores ordenada alfabéticamente, por lo que solo restó crear una condición que acotase la lista con los nombres dados.

Otra de las limitaciones para el informe de doctores, es que, al tener que saber cuántos doctores había en sistema con las características deseadas, tuvimos que en un primer lugar almacenar todos los doctores que cumpliesen con el rango y recién ahí podíamos mostrar en pantalla sus características, para lo cual, creamos una función visitar (puesto que el usuario no tiene manejo ni conocimiento sobre doctor_t , es decir la estructura con los datos del doctor) que guarda en una lista los datos principales de los doctores en el rango. Para esta función, si bien al utilizar diccionarios sacrificamos más memoria que utilizando listas de lista u otros medios, preferimos priorizar un fácil y claro acceso por parte del usuario a los datos almacenados del doctor, puesto que con acceder a “nombre”, “especialidad” y “atendidos” en el diccionario, recibe un puntero a estos datos. El motivo también por el que no guardamos las claves y utilizamos las primitivas ya existentes para obtener los datos, es que nos manejaríamos con iteraciones en una lista de d elementos (d siendo la cantidad de doctores ya acotada) más log D (donde D es la cantidad de doctores totales).