ALGORITMOS Y PROGRAMACIÓN II

TP N° 2

Integrantes:

Pinto, Nicolás
Ybarra, Diego

Corrector:

Collinet, Jorge

Para la implementacion del este TP decidimos crear dos TDAs que nos ayudarian en el planteo del problema y en la generalizacion del codigo, estos son los TDA paciente y doctor, con sus respectivas primitivas.
El TDA doctor posee internamente un struct doctor que tiene en sus campos el nombre del doctor, su especialidad y la cantidad de atendidos. Estos datos se completan cuando se usa la primitiva para crear un doctor, y para acceder a los mismos se proveen las primitivas para obtener cada uno de ellos y también se puede modificar la cantidad de atendidos mediante la primitiva doctor_sumar atendidos. Finalmente el doctor se puede destruir con doctor_destruir.

Para el TDA paciente tenemos el struct paciente que tiene como campos el numero de inscripcion y el nombre y tiene sus correspondientes primitivas que funcionan como las del TDA doctor.

Implementacion:

Al inicio del programa, para guardar los doctores decidimos implementar un ABB en donde cada clave es el nombre del doctor y el dato es el TDA doctor correspondiente. De esta forma los doctores quedan ordenados por nombre y se puede hacer facilmente un recorrido inorder para obtenerlos alfabeticamente.
El ABB fue modificado para este tp, le agregamos las primitivas correspondientes para que también pueda iterar de manera inorder sobre los elementos en un cierto rango especificado(dando un inicio o un fin de rango) y se pueda obtener la cantidad de elementos que fueron iterados en ese rango.

Para guardar los pacientes decidimos usar un Hash para poder acceder a ellos de forma rápida mediante la clave, que en nuestro caso sería el nombre del paciente. De esta forma podemos acceder rápidamente a cada uno de ellos para poder obtener asi su año de inscripción.

Finalmente también guardamos en un Hash, a medida que iteramos sobre la lista de doctores inicial, las distintas especialidades de los doctores. Usamos entonces como clave el nombre de la especialidad y como dato creamos una lista en donde el primer elemento es un Heap vacío y el segundo elemento es una Cola vacía. El Heap es de minimos y tiene como objetivo guardar todos los pacientes que piden turno que no tengan carácter de urgencia. Se ordenan mediante el año de inscripción de manera que a la hora de atender tengan prioridad los que tienen el menor año de

incripción. En la Cola se guardan los pacientes que tengan que ser atendidos de urgencia, por lo que a la hora de atender un paciente, primero se obtiene la especialidad del doctor que va a entender, luego se busca en el Hash de especialidades y se obtiene el paciente de urgencia que haya ingresado primero para esa especialidad y, en caso de que no hubiera ninguno, el paciente con menor año de inscripcion que esté esperando.

Complejidad:

Para el informe de doctores, decidimos guardar a estos ultimos en un abb, ya que esto nos permitiria cumplir con la complejidad pedida, O(log d), en un caso regular y nos facilitaria el muestreo. Lo que hicimos fue crear dos iteradores, uno para saber la cantidad final de doctores a mostrar y luego lo borramos y mostramos con el segundo iterador los doctores pedidos, haciendo asi 2O(log d) = O(log d)

Para obtener siguiente paciente utilizamos dos tdas, una cola y un heap, mas el abb de los doctores. Esto nos permite asi tener complejidad O(log d) en casos de pacientes urgentes ya que la complejidad de desencolar es O(1) y O(log d + log n) en casos regulares ya que debemos sumar la complejidad de heap desencolar

Para pedir turno sucede algo similar, al ser cola_encolar O(1) y heap_encolar O(log n), se cumple la complejidad pedida.