Informe
Trabajo Práctico 2

Grupo: G17

Alumno: Santander Valentín, Padrón: 105637

Alumno: Higa Axel Tomas, Padrón: 105848

Corrector: Jorge Collinet

Materia: Algoritmos y programación II

Introducción

El trabajo práctico que se va a presentar es una simulación de un hospital de nombre Zyxcba que va a tener en cuenta tres comandos principales: PEDIR TURNO, ATENDER e INFORME.

Mediante el uso de los diferentes TDAs ya realizados con anterioridad y la creación de otros se intentará resolver el trabajo cumpliendo con las condiciones dadas para cada comando. El funcionamiento del mismo será por entrada y salida estándar de la consola.

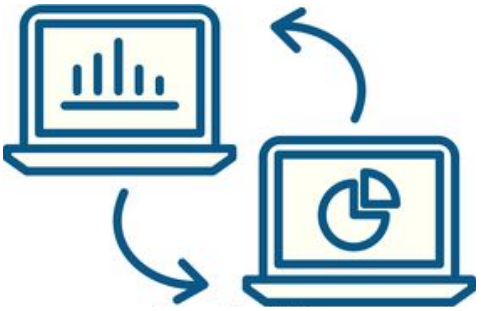
Antes de comenzar con la introducción de los comandos, se introducirán previamente los datos de los doctores y los pacientes que están registrados en el hospital, los pacientes contarán con nombre y año de inscripción, y en caso de los doctores contarán con nombre y la especialidad en la que trabajan.

Para los comandos se deberán cumplir las órdenes de complejidad dadas. En caso de PEDIR TURNO Y ATENDER habrá dos órdenes de complejidad dependiendo de la urgencia (URGENTE O REGULAR), en caso de PEDIR TURNO si el paciente es de caso urgente deberá ser en O(1), si es regular podrá ser en O(n) siendo (n) la cantidad de pacientes en la especialidad. En caso de ATENDER si fuera urgente deberá ser O(log(d)) siendo (d) la cantidad de doctores en el sistema, y si fuera regular podrá ser O(log(d) + log(n)) siendo (d) la cantidad de doctores en el sistema y (n) la cantidad de pacientes en espera en la especialidad. Por último el comando INFORME deberá de imprimir la cantidad de doctores en el sistema en un rango dado, y su orden debe ser en promedio O(log(d)) siendo (d) la cantidad de doctores que hay en el sistema, aunque en el peor caso puede llegar a ser O(d).

Tipos de dato abstracto auxiliares

Para poder modelar el funcionamiento de la clínica Zyxcba, fue necesario utilizar algunas estructuras auxiliares. Algunas de estas estructuras son tipos de dato abstracto genéricos y conocidos los cuales ya fueron entregados anteriormente (Por ejemplo el Árbol Binario de Búsqueda o el Diccionario).

Sin embargo, también fue necesario crear estructuras y tipos de dato específicos del problema. Éstos tienen propiedades o particularidades que no son lo suficientemente genéricas como para ser considerados en otro tipo de proyectos. Las estructuras auxiliares específicas son las siguientes:



- ➔ Cola_turnos

  Funciona como un TDA Cola implementado de forma enlazada pero tiene la particularidad de tomar registro de la cantidad de elementos que tiene. En una Cola_turnos se guardan los turnos urgentes de una determinada especialidad. Uno de los requisitos del comando PEDIR_TURNO es guardar el mismo y avisar al Usuario cuántos pacientes más hay en espera. La Cola_turnos junto con su contador se encargan de cumplir esa función (Pero solo en el caso de pedir un turno urgente).

- ➔ Doctor

  El TDA doctor consiste en una estructura que contiene todos los datos importantes de los doctores que trabajan en la clínica Zyxcba. Cada doctor tiene un nombre, una especialidad y una cierta cantidad de atendidos desde que arrancó el sistema la cual inicia en 0.

- ➔ Paciente

  _____El TDA paciente es una estructura que contiene los datos de cada uno de los pacientes registrados en la clínica desde su inauguración. Cada paciente tiene un nombre y un año en el que fue inscripto. Los pacientes inscritos con mayor anterioridad tienen prioridad al ser atendidos.

- ➔ Datos
  _____Consiste en una estructura donde se guardarán todos los datos obtenidos de los archivos CSV. Contiene un Diccionario de especialidades, un Diccionario de pacientes y un Árbol Binario de Búsqueda de doctores. Utilizando las subestructuras, el TDA datos se encarga de proporcionar la información necesaria a las distintas funciones que ejecutan los comandos recibidos.

- ➔ Turno
  Consiste en un TDA donde se guardan los turnos solicitados por entrada estándar. Contiene un Diccionario para turnos urgentes y otro para turnos regulares que permitirán encolar y desencolar a los pacientes según la especialidad y el carácter de su turno.

Diseño

La inicialización del programa comienza por la introducción de dos archivos CSV correspondientes a los datos de los doctores y los pacientes. Estos archivos se procesarán mediante el CSV proporcionado por la cátedra en la que se creará una lista para cada uno. En caso de los pacientes la lista guardará estructuras del tipo paciente_t que contarán con el nombre y el año de inscripción y en caso de los doctores se guardarán estructuras del tipo doctor_t que contarán con el nombre, la especialidad, y la cantidad de
pacientes que atendió.

Luego de que los datos de los archivos csv se hayan procesado en las listas correspondientes, se procederá a organizar dicha información con el TDA dato que creará una estructura en la que guardará un hash de pacientes, que guardara las estructuras de los pacientes, un hash de especialidades que se utilizará para ver las especialidades que haya en el hospital y un ABBI de doctores que guardara las estructuras de los pacientes.

Cuando los datos se hayan organizado, se creará una estructura del tipo turno_t que será la encargada de llevar la información de los pacientes que pedirán los turnos en el hospital, y cuenta con dos hash para la urgencia que haya, ya sea urgente o regular.

Una vez creadas estas dos estructuras se dará comienzo al procesamiento de la de comandos que se dará por entrada estándar de la consola.

Los comandos se procesarán mediante el zyxcba_lib.c que verifica los comandos introducidos si posee algún error.

Una vez que los comandos y parámetros sean correctos se procesa la información.

En caso de pedir turno,se usa el TDA turno y se encolan a los pacientes dependiendo de su urgencia (urgente o regular), en caso de que sea urgente, el paciente será encolado en una estructura de cola_turno dentro de un hash teniendo a la especialidad correspondiente como clave, en caso de regular se encola en un heap dentro de un diccionario y se imprimirá si se encolo el paciente, y cuántos pacientes hay en total en la espera en la especialidad pedida.

Para el comando atender, también se usa el TDA turno como el comando pedir turno, a diferencia que se desencolan y se imprimirá cual paciente se atendió y cuantos quedan en espera de la especialidad atendida.

Para el comando informe se usará una función auxiliar, que utilizara el recorrido abb in order por rangos, dicha función consta de dos recorridos, uno para contar los doctores en el sistema de dicho rango y otro para imprimir la información del doctor (nombre, especialidad y cantidad pacientes atendidos).

Manejo de errores

Errores en los casos de inicialización se abortará con código 1:

1. No se recibieron exactamente dos argumentos por la línea de comandos: Si no se introducen los dos archivos csv de los doctores y pacientes se cortara la ejecución del programa.
2. Alguno de los archivos CSV no existe, o no pudo ser leído: si alguna de las listas que devuelve el lector csv proporcionado por la cátedra devuelve NULL se cortara la ejecución del programa.
3. Algún año no es un valor numérico: En este caso se implementó un centinela iniciado en NULL en el caso de que atoi no pueda cambiar el string a un int en el procesamiento el csv de pacientes mediante el constructor o parseo, devuelve 0 y el centinela tendrá el valor del string que no pudo cambiar y cortara la ejecución del programa imprimiendo el valor del string y su mensaje correspondiente.

Mensajes de error:

Los mensajes de error se producen cuando se introduce un dato erróneo, o el comando se introdujeron con errores.

Los errores de comandos se imprimen por las funciones dadas por la cátedra en el main del programa.

Los errores de datos y parámetros lo manejan el zyxcba_lib.c mediantes las funciones que procesan los comandos.

Los errores pueden ser, cantidad de parámetros erróneos, que el paciente o doctor no esté registrado en el hospital o que el hospital no trabaje con alguna especialidad.

Complejidades

El programa que ejecuta los comandos de la clínica debe cumplir con las instrucciones de cada uno. Pero además se deben ejecutar en un determinado orden de complejidad específico para que el funcionamiento sea lo más eficiente posible.

En primer lugar se solicita que el comando Pedir Turno tenga complejidad O(1) en el caso de pacientes urgentes y O(log(n)) en el caso de pacientes regulares, siendo n la cantidad de pacientes encolados en la especialidad del sistema.

En el caso de pedir un turno urgente,primero se accede al hash de turnos urgentes del TDA turno. En la especialidad que se quiere pedir el turno, se encolará al paciente en una cola_turnos. Acceder al hash es O(1) y encolar en una cola_turnos (Implementado como cola enlazada) también es O(1). Por lo tanto se cumple el orden pedido por la consigna.

Ahora bien, en el caso de un turno regular, se accede al hash de turnos regulares. En la especialidad que se quiere pedir el turno, se encolará al paciente un heap o Cola de Prioridad. Acceder al Diccionario de regulares es O(1). Pero desencolar en una cola de prioridad es de orden logarítmico. En ese heap solo se encolarán los pacientes que hayan pedido turno en esa especialidad determinada. Entonces el orden de encolar a un paciente en ese heap es O(log(n)) siendo n la cantidad de pacientes regulares que pertenecen a esa especialidad. Ergo, también se cumple el orden solicitado.

En cuanto al comando atender se tiene en cuenta los tiempos del pedir turno ya que utiliza las mismas estructuras y se suma la búsqueda del doctor en el ABB que es la estructura en la que se encuentra alojada la información de los mismos. En el caso de atender un paciente urgente, la búsqueda del nombre del doctor en un árbol es O(log(d)) con (d) cantidad de doctores en el sistema sumado al O(1) que sería el acceder a un hash y desencolar de cola_turnos que utiliza una cola enlazada, entonces, la ecuacion seria O(log(d)) + O(1) quedando O(log(d)), en el caso de ser regular, al utilizar un hash y un heap, el desencolar quedaría O(log(n)) con (n) cantidad de pacientes encolados en aquella especialidad debido al heap, por lo

tanto la ecuación de búsqueda de doctor y acceder al paciente quedaría O(log(d)) + O(log(n)) quedando así O(log(d) + log(n)).

Finalmente, se pide que el comando Pedir Informe tenga una complejidad de O(log(d)) en el caso promedio y O(d) en el peor caso, siendo d la cantidad de doctores que hay en la clínica. Para poder imprimir el informe se recorre alfabéticamente el ABB de doctores en un determinado rango el cual es dado como parámetro al momento de escribir el comando. Se indica un doctor inicial y uno final. En caso de no especificar el inicio, se itera desde el primer doctor. En caso de no especificar el fin, se itera hasta el último doctor. Cada vez que se visita un nodo del ABB se imprime una línea lo cual cuesta O(1). Por lo tanto lo único significativo para el orden es la cantidad de nodos visitados en total.

En caso de que se quiera recorrer una parte específica del árbol, el orden termina siendo O(log(d)) siendo d la cantidad total de doctores, pues el iterador por rangos del ABB está programado para saltear los nodos menores al inicio y mayores al fin.

Pero si el rango solicitado es similar a la cantidad total de doctores, entonces no se saltean los suficientes doctores como para que el orden disminuya significativamente. Entonces el orden final queda como O(d) siendo (d) la cantidad total de doctores. Se puede pensar que el comportamiento del ABB simula al de una lista ordenada.