Algoritmos y Programación II

Curso Buchwald

Informe Trabajo Práctico N°2

Grupo: Gabriel Semorile (gsemorile@fi.uba.ar, padrón: 105681)

Felipe de Luca Andrea (fdeluca@fi.uba.ar, padrón: 105646)

Corrector: Martín Buchwald

En el presente informe se desarrollará sobre las diferentes decisiones de diseño que se llevaron a cabo al realizar el trabajo práctico. El informe está organizado en 2 partes: Distribución del programa, donde se detalla la disposición del código en los diferentes archivos; y el TDA Clínica, que se encarga de procesar la mayor parte del funcionamiento del programa.

Distribución del programa

Cuando empezamos a pensar sobre la distribución del código del programa, decidimos mantener en su mayoría las recomendaciones de la cátedra presentes en los archivos de descarga del trabajo. El contenido de los archivos de la descarga que modificamos y los que creamos es:

- En *zyxcba.c*, en el cual estaba presente el main del programa y se hace el procesamiento de los comandos ingresados por el usuario, mantuvimos prácticamente todo el código. Agregamos más que nada mensajes de error y llamadas a funciones para cargar los datos, moviendo un poco de código al main que se encontraba vacío. Desde aquí llamamos a las funciones de los 2 archivos siguientes.

- Creamos el archivo *funciones_tp2.c* y su header file *funciones_tp2.h*, que venía incluido en *zyxcba.c* pero no estaba creado. En este archivo decidimos hacer las funciones para los 3 comandos que podía ingresar el usuario, recibiendo los parámetros en un arreglo. De esta manera simplemente imprime los correspondientes mensajes de error y realiza las tareas utilizando el TDA Clínica, que maneja internamente todo lo necesario.

- Para el archivo *csv.c* y su cabecera *csv.h*, decidimos hacer alguna modificación mayor. Este venía con un constructor al cual le podíamos pasar una función que se ejecutaba con lo leído de los archivos, para poder cargar los datos a memoria.

  En primer lugar, cambiamos el código y su firma para que no devuelva una lista. Decidimos hacerlo de esta manera ya que no nos parecía necesario en ningún momento contar con una lista de los pacientes o doctores leídos cuando directamente podíamos usar la función para cargar los datos al TDA que luego decidimos crear con las primitivas correspondientes.

  En segundo lugar, hicimos que el constructor devuelva un booleano que nos diga si tuvo éxito la ejecución de la función dada (que también la cambiamos para que devuelva un booleano), ya que necesitábamos cortar la ejecución del programa en caso de error de lectura.

  Por último, le cambiamos el nombre a los archivos. En un principio, las funciones que le pasábamos al constructor se encontraban en *zyxcba.c*, pero nos pareció más lógico mantener el código de carga de datos en el mismo archivo, por lo que las movimos allí y le cambiamos el nombre a *lectura_archivos.c* y *lectura_archivos.h.* Luego creamos una función que se encargue de ejecutar el constructor para los pacientes y los doctores, la cual utilizamos en *zyxcba.c*.

- En el archivo *mensajes.h*, agregamos un par de mensajes para errores posibles que no se encontraban contemplados. Estos son: cantidad de parámetros inválidos para cada comando, insuficientes argumentos en un registro de algún archivo y error de memoria (para cuando falle un pedido de memoria dinámica). Como detalle menor incluímos la biblioteca <stdio.h> simplemente para evitar un error de compilación por la flag *pedantic (error: ISO C forbids an empty translation unit [-Werror=pedantic]).*

- Creamos una carpeta *dependencias* con todos los TDAs anteriores utilizados en el programa. Estos no se vieron modificados en el desarrollo de este programa, exceptuando el TDA ABB. En este caso tuvimos que agregarle un iterador interno para poder recorrer los elementos en *O(log n)* (siempre y cuando no sean muchos, sino se hace lineal). Esto lo utilizamos en el TDA Clínica para luego poder, utilizando una primitiva de este, imprimir el informe de doctores en la complejidad pedida. Este iterador interno es simplemente un iterador por rangos, es decir, que recorre el árbol in-order todos los elementos entre un mínimo y un máximo dados.

- Creamos un TDA Clínica en el archivo *clinica.c* y su cabecera *clinica.h*, el cual se desarrollará a continuación.

TDA Clínica

Para cumplir con lo pedido y organizar el programa, decidimos crear un TDA Clínica que se encarga de administrar prácticamente todo exceptuando la carga de datos y mensajes de error.

Las estructuras internas utilizadas en este TDA son:

- Clínica: Esta es la estructura general del TDA, la cual contiene 3 diccionarios. Estos son, 2 tablas de hash y un árbol binario de búsqueda:
    - Antigüedades: Este hash almacena el año de ingreso a la clínica de los pacientes. La clave de este diccionario es el nombre del paciente. Se decidió utilizar una tabla de hash porque necesitábamos que obtener la antigüedad de un paciente sea en tiempo *O(1)*. La antigüedad se utiliza al pedir los turnos de prioridad regular, que debe ser *O(log n)*, siendo *n* la cantidad de pacientes encolados para esa prioridad. Como almacenamos las antigüedades de todos los pacientes juntos, cualquier otra complejidad dependendería de la cantidad total de pacientes y no de los encolados. Por ejemplo, en un ABB, la complejidad de obtener una antigüedad sería *O(log k),* donde k es todos los pacientes existentes, lo cual ya excede lo pedido.
    - Doctores: Este ABB almacena estructuras con los datos de los doctores, y la clave utilizada es el nombre del doctor. Elegimos este TDA ya que nos permite obtener los datos de los doctores en los tiempos necesarios para los comandos de atender siguiente e informe. En el caso del comando atender siguiente, la información de los doctores debe ser obtenida en *O(log d)*. Para el informe, se pide que este esté ordenado alfabéticamente y que si son pocos doctores el orden sea *O(log d)*. Esto último no lo podíamos lograr con una tabla de hash, ya que no almacenaría los doctores alfabéticamente y tendríamos que ordenarlos en cada informe, lo cual tendría una complejidad mayor. Con una lista o arreglo, sería lineal.
    - Colas: Son las colas de los turnos de los pacientes de cada especialidad, almacenadas en una tabla de hash. La clave es la especialidad. Necesitábamos que sea en tiempo constante ya que la complejidad de los comandos no dependen de la cantidad de especialidades, por lo que decidimos usar este TDA para lograrlo.

● Paciente: Contiene el nombre y año de ingreso a la clínica del paciente. El nombre del paciente es lo que se encola y desencola al pedir y atender un turno. Se crea esta estructura cuando el turno no es urgente de manera que quede el año de antigüedad disponible como parámetro para priorizar a los pacientes más antiguos en la cola de prioridad utilizada.

● Colas de especialidad: Son las diferentes colas de los turnos de cada especialidad almacenadas en el diccionario *Colas* de la estructura *Clínica*. Estas contienen:

- Cola de urgentes: Esta es una cola de los nombres de los pacientes que piden turno con prioridad urgente. Es simplemente el TDA Cola, ya que nos permite desencolar en el orden de llegada en tiempo *O(1)*, como fue pedido.
- Cola de regulares: Esta es una cola de los nombres de los pacientes que piden turno con prioridad regular. En este caso es el TDA Heap, ya que nos permite encolar y desencolar los pacientes en función de su antigüedad en la complejidad *O(log n)* pedida. Aquí es donde se utiliza la estructura *Paciente*.
- Cantidad de pacientes en espera: Es nada más un contador de los pacientes encolados en ambas prioridades. Lo decidimos implementar aquí ya que el TDA Cola no cuenta con un *cola_cantidad()* como si lo tiene el TDA Heap, y así ahorrarnos modificar el primero.

● Datos de doctor: Es lo que se almacena en el diccionario *Doctor* en la estructura *Clínica.* Contiene la especialidad del doctor, necesaria para saber en donde desencolar un paciente al atender un turno, y la cantidad de pacientes atendidos que incrementa al hacerlo.

● Extras para visitar doctores: Es una estructura auxiliar que utilizamos para poder pasar dos extras al iterador interno del TDA ABB en la primitiva *clinica_visitar_doc()*, abstrayendo al usuario de la estructura de *Datos de doctor*.