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 funcionamiento de los TDAs creados para este trabajo.

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 archivo de cabecera *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*, que internamente utiliza los TDA Paciente, TDA Cola de Pacientes y TDA Doctor, con los que se encarga de la mayor parte del funcionamiento del trabajo.

- Hicimos un TDA Paciente en los archivos *paciente.c y paciente.h*, que guarda la información de un paciente determinado al cargarlo en memoria.

- Elaboramos un TDA Cola de Pacientes en los archivos *colapac.c* y *colapac.h*, el cual permite encolar y desencolar los pacientes en el orden indicado según su urgencia.

- Creamos un TDA Doctor en los archivos *doctor.c* y *doctor.h*, que almacena los datos de cada doctor al cargarlos en memoria y la cantidad de pacientes atendidos por cada uno.

TDAs creados para el trabajo

TDA Paciente

Es simplemente una estructura que guarda el nombre del paciente y su año de antigüedad, con el cual permite comparar pacientes mediante una primitiva. Esta primitiva se utiliza en el TDA Cola de Pacientes para determinar el orden de los turnos no urgentes.

TDA Doctor

Se trata de otra estructura que guarda el nombre del doctor, su especialidad y la cantidad de pacientes atendidos. Con sus primitivas permite aumentar el contador de pacientes atendidos de un doctor y ver la cantidad actual.

TDA Clínica

El TDA Clínica es el principal TDA del trabajo en el cual se almacenan todos los doctores, pacientes y la cola de pacientes.

Las estructuras de este TDA son:

- Clínica: Esta es la estructura general, la cual contiene 3 diccionarios. Estos son, 2 tablas de hash y un árbol binario de búsqueda:
    - Pacientes: Almacena pacientes del TDA Pacientes. La clave de este diccionario es el nombre del paciente. Se decidió utilizar una tabla de hash porque necesitábamos que obtener un paciente y su información sea en tiempo *O(1)*. Esto es importante al momento de pedir los

turnos, que deben ser *O(1)* o *O(log n)*, siendo *n* la cantidad de pacientes encolados para esa prioridad. Como almacenamos todos los pacientes existentes juntos, cualquier otra complejidad al obtener un paciente implicaría que pedir un turno dependa 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 a los doctores del TDA Doctor, y la clave utilizada es el nombre del doctor. Elegimos utilizar un árbol binario 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)*, lo cual coincide con la primitiva de obtener del ABB. 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 mucho mayor. Con una lista o arreglo ordenados, sería lineal el informe y más lenta la carga de datos.
- Cola de Pacientes: Almacena una Cola de Pacientes, que se utiliza para encolar y desencolar pacientes al pedir y atender turnos.

- 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 del TDA Doctor.

TDA Cola de Pacientes

Es el encargado de administrar los turnos de la clínica. Sus estructuras son:

- Cola de Pacientes: Es la estructura general del TDA, la cual contiene un diccionario *Colas de Especialidad* que almacena las colas de pacientes de cada especialidad por separado. Al agregar una cola de una especialidad con la primitiva correspondiente, esta se agrega a este diccionario. La clave de este es la especialidad. Utilizamos un diccionario ya que necesitábamos que obtener cada cola individual sea en tiempo constante, porque la complejidad de los comandos no dependen de la cantidad de especialidades.

- Cola de especialidad: Es una cola de los turnos para una especialidad individual, almacenadas en el diccionario *Colas de Especialidad.* 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.
  - 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.