Trabajo Práctico 2

[7541] Algoritmos y Programación II
Curso Buchwald
Primer cuatrimestre de 2020

Alumno:	Pancrazzi, Gustavo Roberto
Padrón:	104738
Mail:	gpancrazzi@fi.uba.ar
Ayudante:	González Lisella, Fiona
Mail:	fgonzalezlisella@gmail.com

Diseño de la estructura de datos

Para este trabajo practico cree un TDA “Turnos”, el mismo modela una lista de espera con elementos prioritarios y otros no prioritarios, por un lado, los elementos prioritarios se ordenan por orden de llegada y, por otro lado, los elementos no prioritarios se ordenan según lo defina el usuario del TDA.

La estructura interna de este TDA está definida mediante dos TDA ya implementados, el TDA Lista enlazada (para los elementos prioritarios) y el TDA Heap de máximos (para los elementos no prioritarios).

También agregue una primitiva en el ABB para obtener una lista de datos con los valores que se encuentren dentro de un rango de claves especificado.

Por otro lado, modele dos estructuras, una para los pacientes (contiene el nombre y año de inscripción) y otra para los doctores (contiene el nombre, la especialidad y la cantidad de pacientes que atendió).

Se reutilizan un paquete de funciones programadas para el TP 1, las librerías “strutil” e “input”, agregando a esta última dos funciones.

Se inicia el programa verificando la cantidad de parámetros recibidos e identificando los archivos. Posteriormente, leyendo línea a línea cada archivo, se crean tres estructuras de datos, que a medida que se ingresen comandos, se van a ir consultando para realizar las acciones y enviar los mensajes que correspondan.

Con el archivo de pacientes se crea un Hash con los nombres de paciente como claves y como dato la estructura paciente. Con el archivo de doctores se crea un Hash con las especialidades como clave y como dato el TDA turnos, por otro lado, también se crea un ABB con los nombres de los doctores como claves y como dato la estructura doctor.

Con estas estructuras creadas el programa queda a la espera de recibir instrucciones.

Comando Pedir turno:

Primero se realizan las validaciones correspondientes:

1- Que la cantidad de parámetros sea la correcta (se usa una función de “input”).

2- Que el paciente exista (Se busca en el Hash de pacientes).

3- Que la especialidad exista (se busca en el Hash de especialidades).

4- Que exista el grado de urgencia (se usa una función de “input”).

Estas operaciones tienen una complejidad temporal constante $O(1)$.

Si todo es válido, entonces se procesa el turno:

1- Se obtiene del Hash de pacientes el paciente correspondiente (se obtiene el paciente de buscarlo en el Hash).

2- Se crea una estructura paciente a guardar en el Hash de especialidades.

3- Se obtiene el TDA turnos de la especialidad (desde el Hash de especialidades).
4- Se encola un paciente en el turno de la especialidad con el tipo de prioridad solicitado (urgente o regular).
La complejidad temporal de este comando es:
- Si el turno es urgente:

$$Pedir\ turno = Validar\ comando + Procesar\ comando = O(1)$$

$Validar\ Comando = Validar\_cantidad\_cadenas + Hash_{pertenece} + Hash_{pertenece} + Validar\_operador$

$Procesar\ Comando = Hash_{obtener} + Turnos\_encolar$

$Validar\ comando = O(1) + O(1) + O(1) + O(1) = O(1)$

$Procesar\ comando = O(1) + O(1) = O(1)$

- Si el turno es regular:

$$Pedir\ turno = Validar\ comando + Procesar\ comando = O(\log(n))$$

Siendo “n” la cantidad de pacientes en espera de ser atendidos para esa especialidad.

$Validar\ Comando = Validar\_cantidad\_cadenas + Hash_{pertenece} + Hash_{pertenece} + Validar\_operador$

$Procesar\ Comando = Hash_{obtener} + Turnos\_encolar$

$Validar\ comando = O(1) + O(1) + O(1) + O(1) = O(1)$

$Procesar\ comando = O(1) + O(\log(n)) = O(\log(n))$

La diferencia de complejidad temporal se justifica siendo que para turnos urgentes se guarda el paciente en una lista, mientras que para los regulares se guarda al paciente en un Heap.

Comando Atender siguiente paciente:

Primero se realizan las validaciones correspondientes:
1- Que la cantidad de parámetros sea la correcta (se usa una función de “input”).
2- Que el doctor exista (Se busca en el ABB de doctores).
La búsqueda en el ABB tiene una complejidad temporal $O(\log(d))$. Siendo “d” la cantidad de doctores en el ABB.
Si todo es válido, entonces se atiende el siguiente paciente:
1- Se obtiene el doctor del ABB de doctores.
2- Se obtiene el turno del Hash de especialidades.
3- Si hay pacientes en la especialidad, entonces se desencola el paciente más prioritario y se lo atiende (sumando un paciente atendido en el doctor).
La complejidad temporal de este comando es:
- Si el paciente tiene un turno urgente:

$$Atender\ siguiente\ paciente = Validar\ comando + Procesar\ comando = O(\log(d))$$

$Validar\ Comando = Validar\_cantidad\_cadenas + ABB_{pertenece}$

$Procesar\ Comando = ABB_{obtener} + Hash_{obtener} + Turnos\_desencolar$

$Validar\ comando = O(1) + O(\log(d)) = O(\log(d))$

$Procesar\ comando = O(\log(d)) + O(1) + O(1) = O(\log(d))$

- Si el paciente tiene un turno regular:

$Atender\ siguiente\ paciente = Validar\ comando + Procesar\ comando = O(\log(d) + \log(n))$

$Validar\ Comando = Validar\_cantidad\_cadenas + ABB_{pertenece}$

$Procesar\ Comando = ABB_{obtener} + Hash_{obtener} + Turnos\_desencolar$

$Validar\ comando = O(1) + O(\log(d)) = O(\log(d))$

$Procesar\ comando = O(\log(d)) + O(1) + O(\log(n)) = O(\log(d) + \log(n))$

Nuevamente, para explicar la diferencia de complejidad temporal, aplica el mismo criterio que en el comando anterior.

Comando Informe doctores:

Primero se realiza la validación correspondiente:

1- Que la cantidad de parámetros sea la correcta (se usa una función de "input").

Si esto es válido, entonces se procede a armar el informe:

1- Se obtiene la lista con el rango de doctores ordenado alfabéticamente (se usa la primitiva del ABB armada para este TP).

2- Se itera la lista mostrando los datos de cada doctor.

La complejidad temporal de este comando es:

- Si la cantidad de doctores no es demasiado grande:

$Informe\ doctores = Validar\ comando + Procesar\ comando = O(\log(d))$

$Validar\ Comando = Validar\_cantidad\_cadenas$

$Procesar\ Comando = ABB_{obtener\ rango} + lista_{iterar}$

$Validar\ comando = O(1) = O(1)$

$Procesar\ comando = O(\log(d)) + O(\log(d)) = O(\log(d))$

- Si la cantidad de doctores es muy similar a la totalidad que hay en el ABB:

$Informe\ doctores = Validar\ comando + Procesar\ comando = O(d)$

$Validar\ Comando = Validar\_cantidad\_cadenas$

$Procesar\ Comando = ABB_{obtener\ rango} + lista_{iterar}$

$Validar\ comando = O(1) = O(1)$

$Procesar\ comando = O(d) + O(d) = O(d)$

Oportunidades de mejoras o sugerencias:

Por ejemplo, se podría implementar un Hash and Displace ya que es posible obtener la cantidad de claves a guardar y se sabe que no se van a agregar nuevas claves, tanto como para el Hash de pacientes como para el de especialidades. Obviamente

no resulta algo sencillo de programar, pero las condiciones de la situación a modelar permiten efectuar esta mejora.
Por otro lado, para el comando “atender siguiente paciente”, podría también usarse un Hash con los nombres de los doctores como claves y como dato la estructura doctor (ambos datos serian iguales tanto para el ABB como para el Hash). Si bien se usa mas espacio en memoria debido a la construcción de este Hash (además del ABB que se mantiene ya que es necesario para armar el informe de doctores), para este comando en particular, se ganaría mucho en la complejidad temporal, ya que las búsquedas serian en tiempo constante y no logarítmicas.