# TP2 - Algoritmos y Programación II - 2C 2020

**Alumnos:** Sebastian Aznárez y Máximo Santoro.

**Corrector:** Ezequiel Genender.

---

## Introducción

Para poder implementar todo lo pedido en la consigna, tuvimos que usar varias estructuras de datos que fuimos viendo a lo largo de toda la cursada para poder resolver los problemas presentados en el enunciado, de una manera eficiente y cumpliendo con las restricciones de complejidad.

Decidimos separar el informe en pequeñas partes, en el orden de ejecución del programa, para lograr explicar el procedimiento de una manera más ordenada.

### pre - Lista de espera (“lista_esp_t”)

Para loguear todos los pacientes usamos un nuevo TDA "lista de espera" (“lista_esp_t”). Este TDA consiste en una Cola para almacenar los pacientes urgentes, y un Heap de mínimos para almacenar los pacientes regulares. En el primer caso, es trivial el uso de la Cola, ya que los turnos urgentes se acomodan en orden de llegada. Por otro lado, para los pacientes regulares usamos un Heap de mínimos. Se eligió esta estructura de datos ya que la prioridad para la asignación del turno será dada a partir de la antigüedad que tengan como clientes del hospital, por lo que se irá siempre desencolado el que tenga el numero de año más chico.

## Carga de CSV

La carga de los dos CSVs se hace de la siguiente forma:

***Carga de doctores:***

Para almacenar la información de cada doctor, usamos una estructura “doctor_t”, que almacena su nombre en una cadena, su especialidad en otra y un size_t con su cantidad de pacientes atendidos.

Aquí decidimos usar 2 estructuras distintas para 2 usos distintos. Primero, la estructura que decidimos usar para almacenar los doctores, fue un ABB. Esto debido a que es una forma muy

eficaz de guardarlos en orden alfabético, cosa que después nos va a ser muy útil para cuando nos pidan el informe (en el comando número 3) y poder buscar rápido, o sea, O(d) en caso de ser todos los doctores y O(log d) en caso de ser un rango más acotado de doctores. Guardamos cada doctor con su nombre como clave, y su respectiva estructura “doctor_t” como dato.

Luego usamos un Hash (“especialidades_t”) para guardar cada especialidad con su nombre como clave y una lista de espera (“lista_esp_t”) como dato para cada una.

Todo esto para que el pedido de turno sea en O(1) para los urgentes, y O(log n) en caso de ser pacientes regulares.

Todos los doctores (ABB, hash (“especialidades_t”) y un size_t contando la cantidad de doctores leidos), son almacenados un estructura “doctores_t”.

Luego de la creación de estas estructuras, se leen todos los doctores. Primero recibiendo una lista desde “csv_crear_estructura”, y de la lista, ingresamos los doctores a su ABB y cada especialidad con su respectiva lista de espera al hash.

***Carga de pacientes:***

Para almacenar la información de cada paciente, usamos una estructura “paciente_t”, que almacena su nombre en una cadena y un size_t con su año de entrada a la clínica.

Se leen todos los pacientes, primero recibiendo una lista desde “csv_crear_estructura”. De la lista, los ingresamos en un hash, usando su nombre como clave, y la estructura “paciente_t” como dato.

Usamos un Hash para que cuando pidamos un turno podamos acceder y buscar a un paciente en O(1).

## Pedir Turno

Para pedir turno, se procesan los comandos que entran por consola, siendo nombre de paciente, especialidad y urgencia. Luego se corrobora que todos los comandos sean válidos y posteriormente se procede a buscar la lista de espera de la especialidad requerida en el hash de especialidades (“especialidades_t”).

Una vez ya con referencia a la lista de espera necesaria, dependiendo de la urgencia del turno, se encola en la cola o en el heap de mínimos (urgentes o regulares respectivamente).

Así logramos pedir turno para pacientes urgentes en O(1) y para regulares en O(log p), siendo “p” los pacientes en espera de esa especialidad

## Atender Siguiente

Para atender siguiente, se procesan los comandos que entran por consola, siendo el único comando, el nombre del doctor.

Una vez corroborado que exista ese doctor (usamos internamente “abb_obtener”), se obtiene referencia al mismo, se adquiere la especialidad de ese doctor, y luego con esa especialidad se busca en el hash (“especialidades_t”) la lista de espera respectiva para esa especialidad.

Luego desencolamos al siguiente paciente a ser atendido, discriminando por urgencia, orden de llegada y antigüedad en la clínica.

Una vez atendido, sumamos uno al contador interno del doctor que haya atendido y terminará de atender en ese instante, listo para volver a trabajar.

## Informe Doctores

Acá usaremos el ABB de doctores que creamos al principio para poder hacer un recorrido ordenado alfabéticamente de una manera eficaz. Y cumpliendo con los requerimientos de complejidad declarados en el enunciado. O sea en O(d), siendo “d” todos los doctores (el peor de los casos) o O(log d) en el caso de que se pida un rango acotado de doctores.

Para esta parte explotamos al máximo las conveniencias de un ABB, usando un iterador interno in-order.

La función comienza chequeando el inicio y el fin (si es que son determinados por el usuario a la hora de ejecutar), y luego se empieza a iterar comparando los nombres de los doctores en el abb contra el “inicio” y “fin”. Ahí usamos un contador para saber cuántos doctores entran en el subgrupo pedido por el usuario, y posteriormente se imprime el número, seguido de cada doctor, avanzando uno por uno con el iterador interno in-order, e imprimiendo los datos pedidos en el enunciado.