FACULTAD DE INGENIERIA

Universidad de Buenos Aires

TP2 — Simulador de Hospital Pokemon

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre de 2021

Alumno:	Hurtado, Daniel
Número de padrón:	107404
Email:	dhurtado@fi.uba.ar

1. Introducción

Este trabajo práctico tiene como objetivo consolidar los conocimientos aprendidos en la implementación de los diversos TDA vistos en la cátedra y hacer uso de ellos como solución a nuestras necesidades de almacenamiento, procesamiento y lectura de diversa información en base a nuestras necesidades.

Es por lo antes mencionado, que este trabajo práctico consistirá en una continuación del TP1, realizado previo a adquirir los conocimientos e implementar los tda, y transformarlo para hacer un uso conveniente de los tda. Así mismo, este tp2 también plantea el trabajo o familiarización de un stack más completo con respecto a los trabajos pasados, formando parte del mismo una implementación visual (cli) para hacer uso de nuestra implementación total.

Siguiendo de igual manera el desarrollo enfocado en TDD (Test Development Drive, o en español, desarrollo guiado por pruebas), cuya finalidad es realizar la implementación conociendo de antemano cual sería su correcto y esperado funcionando, a la vez que dicha implementación

seguirá los criterios establecidos por la cátedra y el contrato facilitado (./tp2.pdf y ./src/simulador.h).

2. Teoría

Debido a que este tp2 final de cátedra tiene como objetivo el antes mencionado, todo lo implementado a continuación será referente a tda anteriormente implementados y/o conocimientos previamente adquiridos tanto en el tp0, tp1 así como también incluso conocimientos de algoritmos 1. Razón por la cual, procederemos a detallar la implementación en su lugar.

3. Detalles de implementación

Como ya se ha mencionado brevemente, la implementación de este tp consta de las siguientes etapas de interés:

1- Actualización Hospital: Retomaremos el hospital.c realizado anteriormente en el tp1 (y split de tp0) y transformaremos sus estructuras internas con el objetivo de abstraer las necesidades/formas de guardar/leer/procesar/eliminar la información contenida, esto abriendo hueco a la utilización de los mismos tdas realizados anteriormente, cuyas decisiones de elección se encuentran abiertas en base a los criterios elegidos, seguidamente mencionados:

- Hospital_t: Anteriormente, la estructura principal de hospital contenía un listado de entrenadores y pokemones en forma de vectores dinámicos. Posteriormente a la necesaria modificacion por la consiga, se han realizado los siguientes cambios:
    - Entrenadores: Ahora pasan a ser una lista haciendo uso del TDA Lista implementado anteriormente, la decisión de elección del mismo se basa en la necesidad de mantener el orden guardado en el que son leídos y poder realizar una lectura óptima de los mismos (sin limitantes de cola y pila). Así mismo, sumada a las razones de la elección, como conocemos de antemano el uso que le daremos, en donde no requerimos búsqueda de un único entrenador, nos resulta irrelevante también considerar opciones de búsqueda ventajosas como TDA Hash y TDA Abb. Así mismo, la estructura entrenador_t se ha actualizado para tener otra lista en su interior, la cual contendrá los pokemones pokemon_t, con el objetivo de facilitar la obtención de pokemones teniendo un determinado entrenador accesible, para evitar recorridos o información duplicada en otras estructuras.
    - Pokemones: Ahora pasan a ser un abb ordenado alfabéticamente en base a sus nombres, esto debido a que conocemos el uso y necesidades de recorrido de los mismos alfabéticamente, con lo cual si bien en una lista insertar sería O(1) a diferencia de O(log(n)) del Abb, al momento de realizar una iteración, es mucho el cómputo ahorrado al no tener que iterar toda la lista para obtener sus elementos y posteriormente volver a iterarla para ordenarla (o iterarla una vez con un ordenamiento agregando elementos, pero igualmente costoso).

- Guardado de pokemones: Si bien, tendremos a disposición el acceso desde dos lugares de nuestros pokemones, en el abb y la lista de cada entrenador, decisión tomada para obtener el mejor performance posible pensando en el uso que daremos, realmente los pokemones son guardados una única vez, por lo que (aunque no sea necesario) de editarse alguno, se actualizarán en ambos lugares al ser guardados punteros a un mismo pokemon.

2- Simulador de Hospital: El simulador consta (además de su creación y eliminación) de una función principal para ejecutar un evento en específico, los cuales son descritos en el archivo eventos.txt y conforman el funcionamiento general y core de este tp. Internamente contamos con una estructura simulador_t conteniendo destacadamente la siguiente información:

- Cola de Entrenadores: Debido a que los entreadores dentro del simulador serán usados a modo de sala de espera, en donde nuestras acciones consistirán en pasar entrenador por entrenador a atender, una cola es suficiente para nuestras necesidades de llenar la cola en el orden que leemos los entrenadores contenidos dentro de hospital, y posteriormente sacar de uno según se atiendan.

- Heap de Pokemones: Una vez se antiende un entrenador, sus pokemones pasan a la equivalente sala de espera, siendo necesario en este caso pasar a atención a cada pokemon de menor a mayor nivel independientemente el orden en el que fueron cargados, por lo que el tda Heap (Cuya base fue implementada como ejercicios obligatorios de Chanutron) nos ayuda a realizar estas operaciones de manera óptima.

3- Interface del Simulador: Para hacer uso del simulador (y formando parte de la consigna), requerimos realizar una interface en la que hagamos uso de ella mediante línea de comandos (cli), con lo cual se ha implementado como programa principal (main.c) una sencilla interface que nos permita ejecutar los eventos disponibles dentro de nuestro simulador, y poder interactuar en tiempo real con él.

4. Pruebas

Como se mencionaba en la introducción, el enfoque de desarrollo utilizado en la implementación con respecto a las pruebas, fue TDD o Test Development Driven (Desarrollo Guiado por Pruebas), las cuales forman parte de la entrega y no son provistas por la cátedra. Las mismas se encontran en ./pruebas.c y pueden ser ejecutadas haciendo uso del comando `make pruebas` con el archivo makefile, o en su defecto con `gcc ./src/*.c ./src/*.h pruebas.c –o pruebas 2>&1` . Dichas pruebas abarcan en conjunto a cada uno de los aspectos o funcionalidades esperadas del tp2.

5. Importación, utilización y compilación del simulador e interface

Para utilizar el simulador, necesitamos importar todas las librerías contenidas dentro de ./src/*, de la siguiente manera:

Script de compilación de ejemplo previo a importación de simulador y dependencias: `gcc ./miprograma.c –o miprograma` Script de compilación de ejemplo posterior a importación de

simulador y dependencias: `gcc ./miprograma.c ./src/*.c ./src/*.h -o miprograma`

Con esta modificación a nuestro script de gcc, permitiremos la importación en la compilación de la compilación de las dependencias.

Seguidamente, para empezar a hacer uso de nuestro simulador, debemos contar con un hospital previamente cargado (y con información cargada) para ello hacemos uso de las siguientes funciones provistas por hospital.h:

1- `hospital_t* hospital_crear();` para crear un hospital nuevo.

2- `bool hospital_leer_archivo(hospital_t* hospital, const char* nombre_archivo);` para leer y cargar la información de un archivo de entrenadores en el hospital recibido por parámetro, siguiendo una estructura similar a csv mostrada en los ejemplos contenidos en ./ejemplos/*.

Una vez que contamos con nuestro hospital listo para ser usado, procedemos a trabajar con el simulador:

1- `simulador_t* simulador_crear(hospital_t* hospital);` para crear un simulador con el hospital previamente creado.

2- `ResultadoSimulacion simulador_simular_evento(simulador_t* simulador, EventoSimulacion evento, void* datos);` para simular un evento con el simulador y los datos requeridos para dicho evento (Los eventos, sus datos y posibles errores se encuentras descritos en ./eventos.txt).

3- `void simulador_destruir(simulador_t* simulador);` para destruir el simulador creado (y también destruye automáticamente el hospital recibido).

Así mismo, podemos preferir hacer uso de la implementación de interface que aprovecha las cualidades de simulador antes decritas, mediante cli. Para ello, debemos compilar la interface (main.c) mediante el siguiente comando: `gcc ./main.c ./src/*.c ./src/*.h -o main` o mediante el makefile con el comando `make main` .