Informe del TP2, Algoritmos y programación II
Curso -04 Méndez

Alumno: Víctor Zacarías, Padrón: 107080

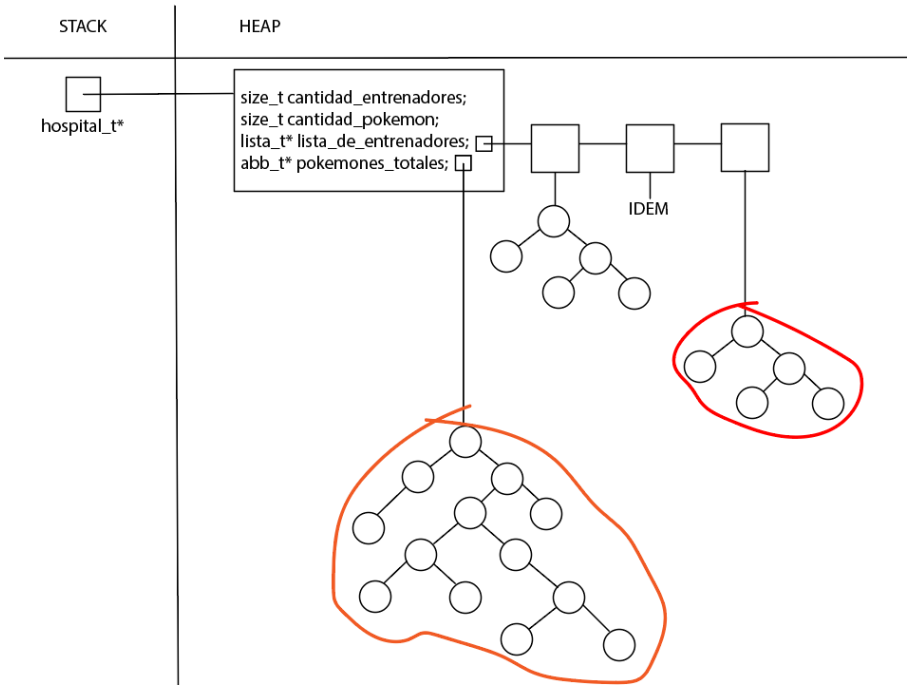
Mail: vzacarias@fi.uba.ar**, DNI: 95.979.449**

-Una breve introducción: Para poder implementar el TP2, se realizaron algunos cambios sobre el TP1. Estos cambios serán explicados brevemente a través de diagramas y algunas explicaciones en texto.

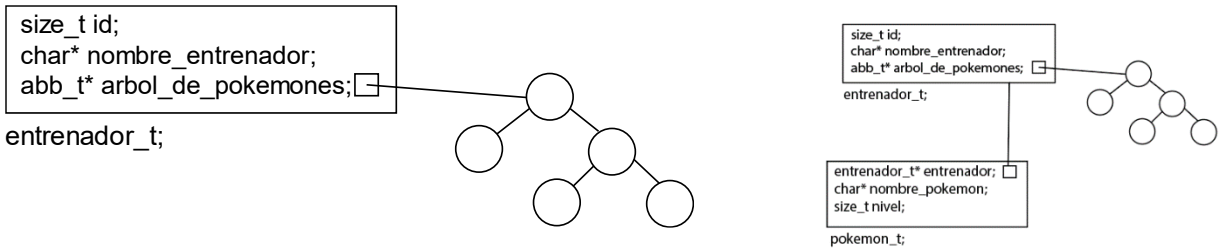
El informe se encontrará dividido en 3 secciones:

1. Cambios en el TP1.
2. implementación del simulador.
3. implementación del main.

Cambios en el TP1: Para poder implementar el simulador haciendo uso de las funcionalidades del tp1 se tuvieron que realizar los siguientes cambios en la estructura del hospital.



- En lugar de usar un vector dinámico para guardar a todos los pokemones se hace uso de una lista, en donde en cada nodo se guarda a un entrenador con un puntero a un árbol con sus pokemones (VER: círculo rojo.
- Se agrega como adicional un puntero a un árbol con todos los pokemones del hospital (VER: circulo naranja).



- A la izquierda se detalla como es el tipo de dato entrenador_t, se puede apreciar también que cada entrenador tiene un puntero a un árbol con sus pokemones (Los entrenadores son guardados en la lista de entrenadores.
- A la derecha se detalla como es el tipo de dato pokemon_t (representa a los pokemones que son guardados en el hospital). Cada pokemon tiene un puntero a su entrenador.

Se modifico el tp1 de forma que se pudieran guardar a cada entrenador y sus pokemones en listas, donde cada entrenador tiene un árbol con sus pokemones. Esto último permite tener una estructura ordenada, en donde puedo acceder a los pokemones de un entrenador (los entrenadores están separados en una lista) con solo llegar a ese entrenador en la lista, el usar un árbol binario de búsqueda para guardar los pokemones permite mejorar el tiempo de búsqueda de estos; O(log(n)), a diferencia de búsqueda lineal O(n).

***Cambios significativos en el TP1:**

Se hicieron cambios en las funciones:

1-Actualización de hospital: Lee un vector de strings con un entrenador y sus pokemones, y los carga al hospital en donde corresponda. Para esto recorre iterativamente el vector de strings usando estructuras auxiliares para cargar la información. Usa las funciones auxiliares:

1.1-Cargar árbol de pokemones: En cada iteración carga la información de un pokémon a un ABB (estos son los pokemones de un entrenador).

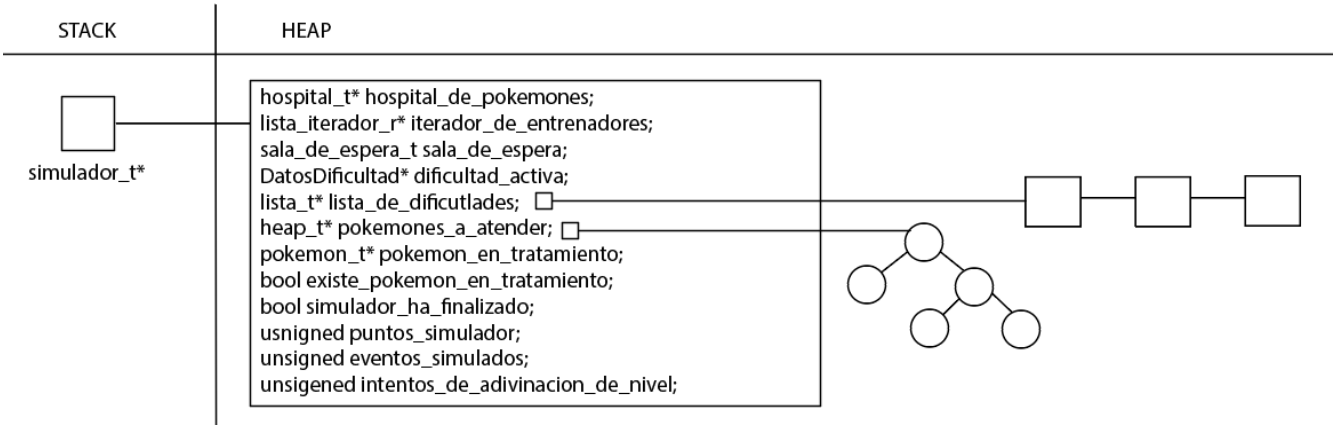
1.2Cargar árbol total de pokemones: En cada iteración carga un pokémon al ABB con el total de pokemones del hospital.

1.3-Agregar entrenador al hospital: Agrega un entrenador a la lista de entrenadores del hospital anexándole el árbol con los pokemones del entrenador.

2-Funcion hospital_a_cada_pokemon: Se encarga de recorrer alfabéticamente todos los pokemones guardados en el hospital mientras les aplica una función, esto pasa mientras queden pokemones para recorrer o mientras la función devuelva true. Para esto, se cargan los pokemones contenidos en el árbol con todos los pokemones en un vector, luego se recorre iterativamente el vector, aplicándole la función a cada pokémon contenido en él.

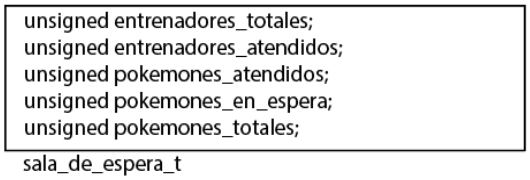
Implementación del simulador:

En la implementación del simulador se creó una estructura del tipo simulador_t en la que se guardan los datos necesarios para poder hacer que este funcione correctamente. Esta estructura tiene la forma:



***Importante:**

- El simulador hace uso de una estructura del tipo hospital_t en donde se guarda los datos necesarios para el funcionamiento del simulador.
- Iterador_de_entrenadores es un iterador de la lista de entrenadores del hospital, permite recorrerlos y atender a los entrenadores en el orden en que están en la lista.
- dificultad_activa es una estructura del tipo DatosDificultad en donde se guarda la información de la dificultad del simulador que se encuentra activa (como determina los puntajes, los mensajes que se muestran por pantalla, etc.)
- lista_de_dificultades es una lista con todas las dificultades disponibles a usar en el simulador.
- pokemones_a_atender es un heap con los pokemones que se encuentran en atención, cada vez que se atiende a un entrenador sus pokemones se guardan en un heap minimal, estos se ordenan por nivel en un el heap, lo que permite atender a los pokemones en orden ascendente de nivel.
- sala_de_espera es un registro que contiene la información sobre lo que pasa cuando se atiende a un entrenador y pokémon. Su utilidad es la de permitir crear estadísticas del estado del simulador a partir de la información contenida en este registro.
- pokemon_en_tratamiento guarda la información del pokémon que se encuentra en tratamiento.



Estructura de la sala de espera.

***Funciones importante implementadas en el simulador:** Las funciones que se describen a continuación son funciones auxiliares usadas por las funciones públicas del simulador, o funciones auxiliares usadas por funciones auxiliares del simulador.

1-Funcion actualizar_estadisticas: Se encarga de llenar un registro que representa a las estadísticas del simulador con la información contenida en el campo sala_de_espera del simulador.

2-Funcion atender_proximo_entrenador: Se encarga de atender al próximo entrenador usando el iterador de entrenadores, esto es, toma al entrenador actual en el iterador, luego carga todos los pokemones de su árbol de pokemones en un vector, el contenido de este vector es insertado en el heap de pokemones_en_tratamiento, una vez finalizado esto se avanza el iterador hacia el siguiente iterador para un futuro uso de esta función. A su vez esta función agrega a un pokémon en tratamiento si es que no lo hay.

3-Funcion obtener_info_pokemon_en_tratamiento: Llena un registro con la información del pokémon que se encuentra en tratamiento (su nombre, y de su entrenador), en caso de no haber pokémon en tratamiento, llena los campos del registro con NULL.

4-Funcion analizar_nivel_adivinado: Se encarga de recibir un intento de adivinación y a partir de la información del pokémon en tratamiento y de la dificultad, determina si este intento fue erróneo o correcto, agrega información adicional al registro. Si el intento resulta ser correcto, se actualiza el pokémon en tratamiento con la raíz del heap (si esta existe).

5-Funcion agregar_dificultad_al_simulador: Se encarga de insertar una dificultad recibida por parámetro en la lista de dificultades del simulador.

6-Funcion seleccionar dificultad: Permite actualizar el campo dificultad_activa con la dificultad deseada, esto a partir de un id, la función hace uso de lista_elemento_en_posicion (función del TDA lista) para obtener la dificultad correspondiente al id y seleccionarla como dificultad_activa.

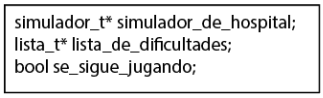
7-Funcion obtener_info_dificultad: A partir de un registro con el id de una dificultad de la que se desea obtener información (este registro es un dato el usuario), busca la dificultad en la lista usando lista_elemento_en_posicion y actualiza el registro con la información de la dificultad.

8-Funcion cargar_dificultades: Se encarga de cargar las dificultades que por defecto vienen en el simulador (fácil, normal y difícil). A partir del nombre de la dificultad que se desea agregar y la lista de dificultades, crea registros con la información de la dificultad y las carga en la lista.

9-Funcion destruir dificultades: Usa un iterador para recorrer cada dificultad de la lista de dificultades y liberar la memoria usada por estas.

10-Funcion es_dificultad_repetida: Recibe una dificultad, y se encarga de recorrer la lista de dificultades verificando si la dificultad recibida ya se encuentra en la lista.

implementación del main: La implementación del main no es más que una integración de las utilidades del simulador en conjunto a entrada de datos por pantalla a través mensajes mostrados por pantalla al usuario. Esto último permite a través de una interfaz mostrada por pantalla hacer uso de cada una de las funcionalidades del simulador.



Juego

El main hace uso de esta estructura para poder hacer un uso correcto del simulador, a continuación, se hará una breve descripción de cada campo del registro.

- Simulador hospital es un puntero a un registro del tipo simulador_t, que permite usar las funcionalidades del simulador.
- lista_de_dificultades es un puntero a la lista de dificultades del hospital del simulador, su utilidad es la de mostrar por pantalla las dificultades de la lista con sus IDs cuando el usuario desee cambiar una dificultad
- se_sigue_jugando permite determinar si el juego se debe seguir jugando o no. Si el false, el programa da por terminado el simulador y deja de ejecutarse.