
# TP2

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre de 2022

Alumno: Chen, jiangjie

Número de padrón: 109747

Email: jchen@fi.uba.ar

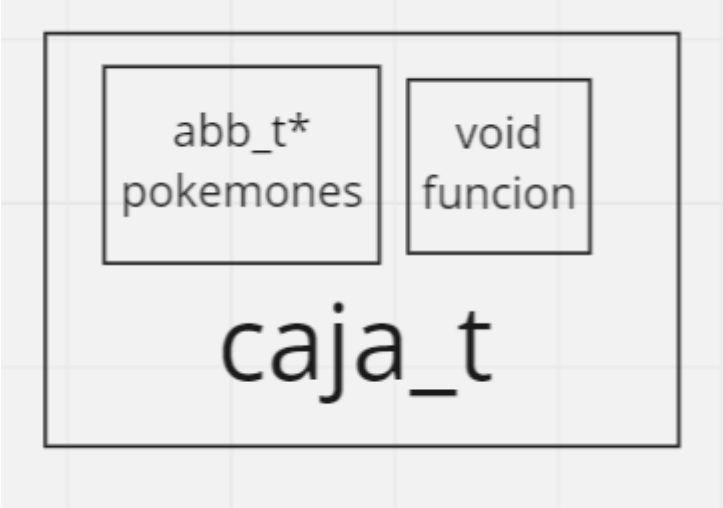
# ÍNDICE

**Explicación de los cambios implementados al TP1. Justificar los cambios (explicar por qué se decidió el TDA utilizado).** .....1

**Explicación de la implementación de cada comando pedido. Explicar cómo funcionan (y cómo utilizan el TDA Caja implementado en el TP1).**.....4

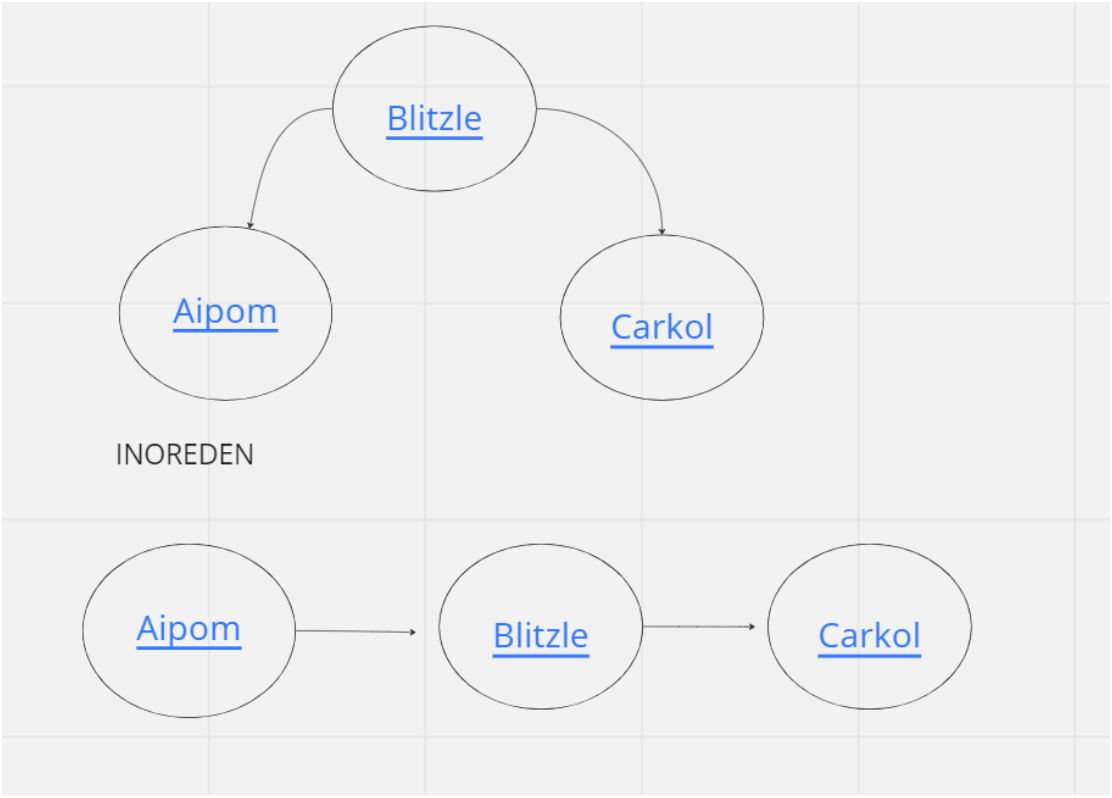
# Explicación de los cambios implementados al TP1. Justificar los cambios (explicar por qué se decidió el TDA utilizado).

Primero, sobre la estructura de la caja_t, tiene dos campos, un es abb_t  y otro es una función que recibe parámetro pokemon_t*.




Elijo abb_t sea un campo de caja_t porque, en caja.h dice que necesita ORDENADOS ALFABETICAMENTE, entonces me conviene mucho más crear un árbol de pokemones, solo tengo crear una función de abb para comparar el orden alfabético de pokemones, y con recorrido inorden me da un orden alfabético.

Si fuese una lista o hash se va a complicar mucho más, porque necesita alguna función rebuscada para ordenarlo, en mi punto de vista.




En este ejemplo, inserto tres pokemones en abb, con la fuincion que comparar el orden, se queda como un árbol, después recórrelo inorden ya me queda un orden alfabético.

El otro campo es una función porque me sirve en la función caja_recorrer(caja_t *caja, void (*funcion)(pokemon_t *)), como tengo un abb de pokemones, para recorrerlo, tengo que usar abb_con_cada_elemento, problema está que los parámetros de esa función recibe una función con parámetro pokemon_t*, entonces, puedo guardar esa función en la estructura de caja_t, después lo aplico.

En la función caja_cargar_archivo, creo una función carga_pokemoens_en_caja, en esa función ya no tengo que agrandar un vector dinámico, sino ya puedo reutiliza abb_insertar cuando hay un pokemon corresponde.

En caja_guardar_archivo, reutilizo abb_con_cada_elemento, y lo recorro con inorden así, en el archivo queda un orden alfabético.

En caja_combinar simplemente reutilizo abb_con_cada_elemento con inorden, y aplico inserta_pokemones para insertar los dos abbs a un abb conjunto, como lo que había dicho, por inorden no tengo preocuparme por orden alfabético.

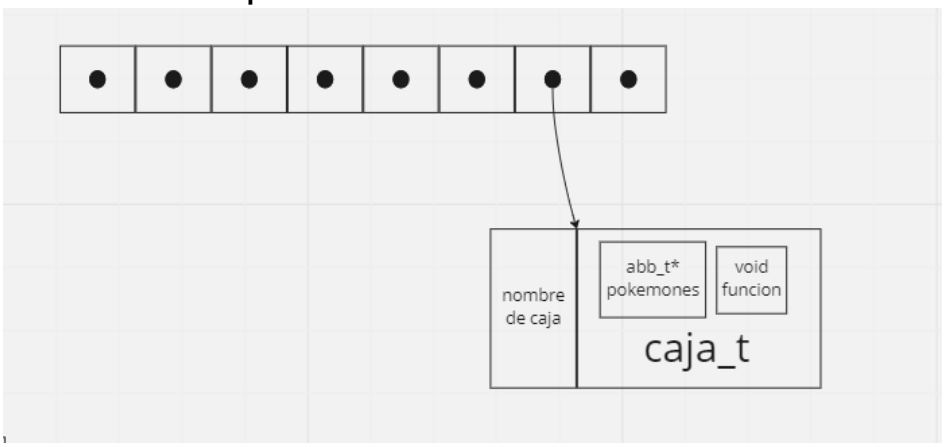
En caja_obtener_pokemon, creo una estructura de pokemon_bucado, contiene un pokemon_t*  pokemon y int n_esimo. en en abb_con_cada_elemento, el parametro auxiliar, mando esa estructura. Cada vez recorre un pokemon, voy a restando n_esimo, cuando n_esimo == 0, encontra el pokemon, y lo guarda en el pokemon de la estructura, así después lo puedo acceder.

Por ultimo en caja_destruir, reutilizo abb_destruir_todo, como esa función solo recibe un destructor con parámetro void*, entonces creo una función de ese tipo, ahí dentro meto pokemon_destruir.

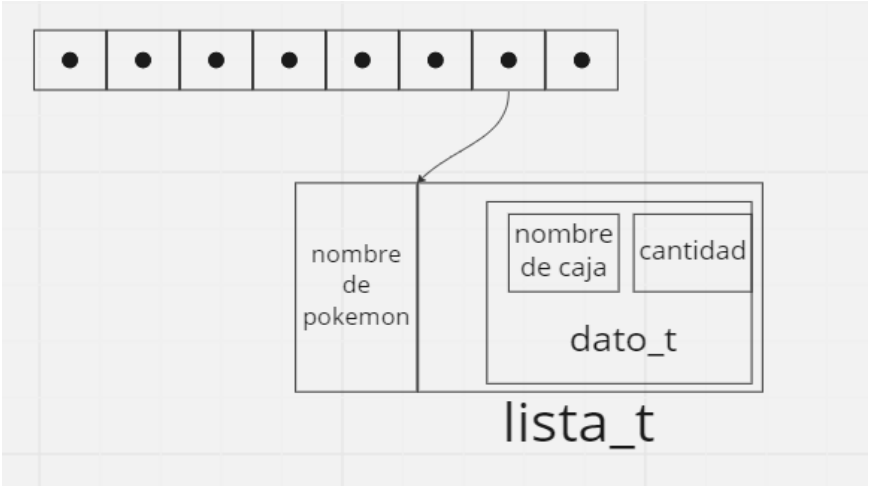
## Explicación de la implementación de cada comando pedido. Explicar cómo funcionan (y cómo utilizan el TDA Caja implementado en el TP1).

Antes de explicar los comandos, tengo que explicar algunas funciones y estructuras importantes.

En en este tp2, tengo dos hashes, uno que se llama cajas_de_pokemones, que su clave es el nombre de la caja, y valor es un árbol de pokemones.

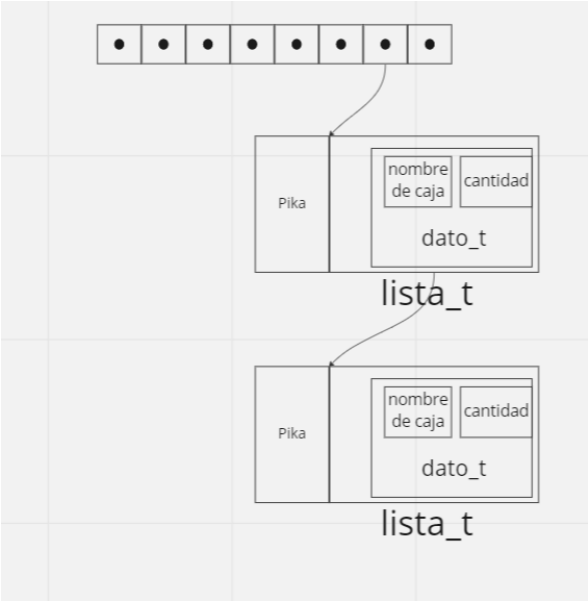



Otro hash es lista_pokemones, que se clave es el nombre de un pokemon, y su valor es una lista de dato_t:




En una estructura de dato_t, nombre va a guardar el nombre de la caja corresponde, y la cantidad de la dicha caja.

En la función inicia_menu, se va a cargar cajas_de_pokemones con argc y argv. Después lista_pokemones se va a cargar con la función carga_lista_pok, básicamente se accede cada pokemon de la caja, y inserto el nombre de pokemon como clave de hash, además, guarda datos con variables correspondes. Cuando tengo variables preparadas, inserto datos en lista de datos, despues inserto esa lista en el valor de hash. Así tengo un hash que su clave es nombre de pokemon, y su valor es el nombre de la caja y la cantidad de dicha caja. En esa función tiene que cuidar con el hash, porque si inserto el mismo nombre de pokemon, se va a actualizar el valor, entonces, cuando hay un mismo nombre, tengo que obtener la lista de ese nombre primero, y inserto nuevos datos de caja en esa lista , después la inserto en hash.



**(I)Mostrar inventario**: Reutilizo hash_con_cada_clave, y mando un hash de cajas_de_pokemones y una función que muestra el nombre de la caja y la cantidad de la caja.

**(C) Cargar otra caja**: Para pedir nombre al usuario, uso scanf y devuelve a un int para no se queja el compilador (así para todos los casos de pedir nombre de un pokemon o un archivo). Con el nombre que ingresa, voy a chequear si ya existe ese nombre de caja, si ya existió no se puede cargar, si no hay que cargar esa nueva caja, y inserto a la cajas_de_pokemones. Mientras, actualiza otro hash lista_pokemones, eso me sirve para el comando bucar_caja.

**(M) Combinar cajas**: Primero pedirse a usuario ingresa 2 nombres de cajas y un nombre de arichivo, después un par de chequeos para verificar si son nombres válidos. Reutilizo las funciones de abb para combinar esas dos cajas valias en una nueva caja y se guardar en el archivo que pide el usuario. Cuando hay nueva caja, hay que actualiza los dos hashes.

**(D) Mostrar caja**: Con el nombre que ingreso el usuario, chequeo si existe o no, si existe, uso caja_recorre con mostrar_datos_pokemoens, muestra los datos de esos pokemones.

**(B) Buscar caja**: Como ya tengo un hash lista_pokemones, solo tengo que usar un iterador interno para mostrar los datos correspondes.

**(Q) Salir**: Es comando se termina el programa, y liberar las memorias bien. Tengo usar hash_destruir_todo con distintos destructores. Lista_pokemones es un hash que contiene valor de una lista de datos, entonces adentro de destructor de lista tengo que aplicar lista_destruir_todo con otre destructor para datos. Y cajas_de_pokemones, necesita un destructor de caja.

Comentario: para que se puede ejecutar main.c en makefile atrás de ./main agrego un archivo.csv