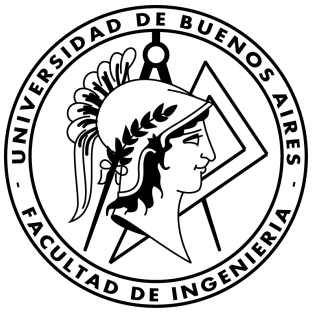


TP 2

Algoritmos y Programación II
Segundo cuatrimestre de 2022

Alumno:	Fernández, Facundo Nahuel
Número de padrón:	109097
Email:	ffernandez@fi.uba.ar

Índice

1. Introducción

El objetivo de este trabajo es modernizar la implementación del TP 1 utilizando algún TDA de los implementados a lo largo del cuatrimestre y crear un sistema de gestión de cajas que interactúe con el usuario.

Para compilar el programa (el archivo main.c), se debe acceder desde la terminal a la carpeta en donde se encuentra dicho archivo y escribir el comando ´make' para luego ingresar './main' seguido de los nombres de los archivos csv que se quieren cargar en el programa de gestión de cajas.

En este informe se explican los cambios realizados al TP 1 y la implementación de cada comando pedido en el trabajo.

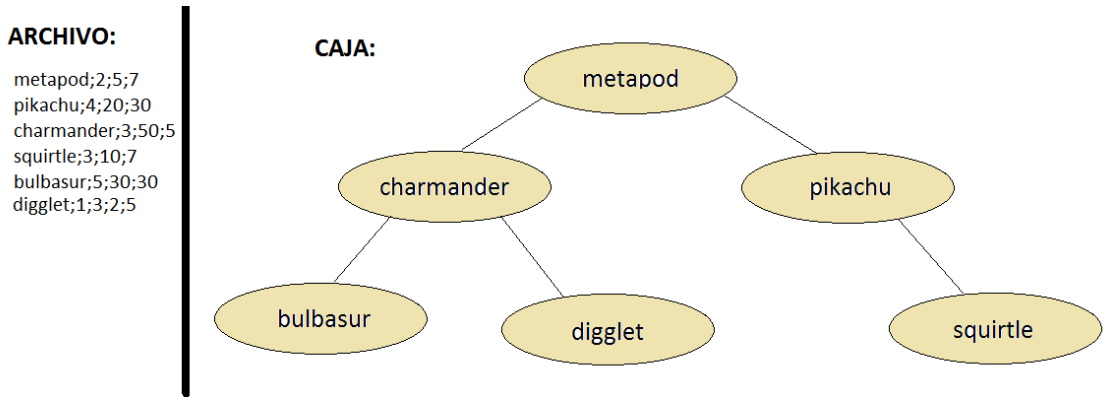
2. TP 1

Para la re-implementación del TP 1 decidí reemplazar el uso de vectores dinámicos por el uso del TDA ABB.

La implementación de pokemon.c no fue modificada ya que no era necesario.

La implementación de cajas.c fue la que modifiqué. El porqué del uso de tda abb tiene un motivo simple: la posibilidad de recorrer los elementos del abb en orden me facilitó la tarea.

La función caja_cargar_archivo de cajas.h carga el contenido de un archivo csv de pokemones en una caja, con los pokemones alfabéticamente ordenados. Cada caja es un abb de pokemones. Lo que se hace al crear una caja es simplemente leer el archivo csv e ir insertándolos. El comparador del abb es una función que compara los nombres de los pokemones, por lo que, al insertarlos, se guarda el primero como raíz y luego se insertarán en la rama izquierda si el nombre es menor alfabéticamente o en la rama derecha en caso contrario. Una caja tendrá una forma similar a la que se aprecia en la siguiente imagen (cada pokemon tendrá sus datos guardados, se muestran solo los nombres a modo ilustrativo):



Esto podría haberse hecho también con una lista (insertando cada pokemon en una posición) o con una tabla de hash (insertando cada pokemon con su nombre como clave), pero se hubiera dificultado la implementación de otras funciones. Por ejemplo, implementé caja_recorrer utilizando el iterador del abb, iterando con un recorrido inorden, lo que permite recorrer los pokemones de la caja en orden alfabético y aplicarles la función recibida (la complejidad en este caso es O(n), ya que se recorren todos los pokemones de la caja). Utilizar listas o tablas de hash no me permitía hacer esto, y tendría que habérmelas ingeniado para hacerlo de otro modo, cosa que no sentí que valga la pena teniendo disponible el abb.

2.1. Implementación de algunas funciones

- caja_guardar_archivo: se itera el abb de manera inorder, se pasa por parámetro la función guardar_pokemon y como contexto el archivo que anteriormente fue abierto en modo de escritura. Como se recorre inorden el abb, se va escribiendo en el archivo todos los pokemones en orden alfabético. Complejidad: O(n), se recorren todos los pokemones de la caja para escribir sus datos en el archivo.

- caja_combinar: Se reserva memoria para lo que será la caja combinada y se procede a insertar los pokemones de cada caja en esta con el iterador del abb. La implementación es simple, la función que se le pasa al iterador recibe como primer parámetro un pokemon (sobre el que se está iterando) y lo inserta en la caja combinada (que se recibe como segundo parámetro). El recorrido que se le pasa al iterador es preorden, ya que si se hacía un recorrido inorden la caja combinada iba a degenerarse en lista (se iban a ir insertando los pokemones en orden alfabético y sólo se guardarían en la rama derecha del abb).

  Sobre la complejidad: se recorren todos los pokemones de la primera caja y luego los pokemones de la segunda caja, lo que son dos acciones de complejidad O(n), por lo que sería O(n) la complejidad de la función. Pero, el insertar en un abb tiene complejidad O(log(n)), por lo que la complejidad de cada recorrido es O(n.log(n)), ya que se recorren n pokemones y luego se insertan en otro abb (la caja combinada). En el caso que la caja combinada degenere a lista por algún motivo, la complejidad de insertar sería O(n) y, por ende, la de la función caja_combinar sería $O(n^2)$

- caja_obtener_pokemon: Como ya no tengo un vector dinámico, no puedo acceder directamente a la posición n de la caja y devolver el pokemon en ella. Por este motivo, se recorre el abb inorden y, con la ayuda de una función y una estructura auxiliar, se logra lo buscado. La estructura tiene tres campos: un puntero a pokemon, un int que indica la posición del pokemon que se busca (hasta) y otro int que indica la posición del pokemon sobre el que se está iterando (recorridos). La función recibe el pokemon sobre el que se itera y la estructura auxiliar. Si se llego al pokemon que se quiere (hasta = recorridos), se guarda en la estructura a este y se devuelve false para no seguir iterando, caso contrario se aumenta recorridos en uno y se devuelve true para seguir iterando. Complejidad de la función: O(n).

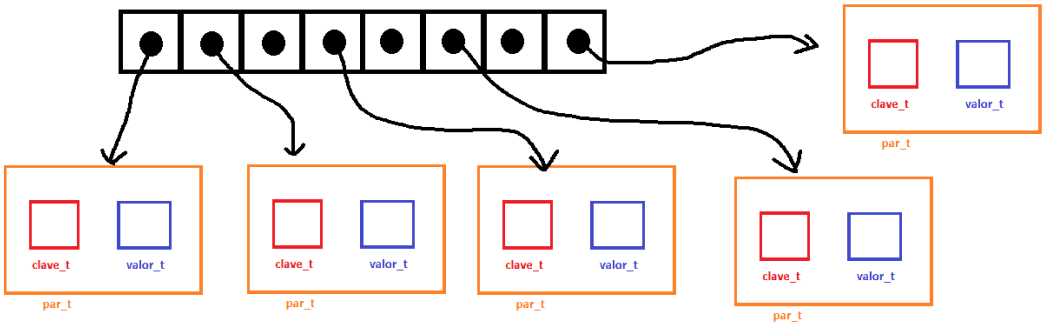
3. Sistema de gestión: TDA Gestor

Decidí crear un TDA gestor para la implementación del sistema de gestión de cajas, para así poder implementar el menú de una forma más prolija y luego poder realizar las puebas correspondiente de los comandos pedidos.

Abro un paréntesis para hacer el siguiente comentario: por razones médicas (un derrame en el ojo izquierdo que causó que tenga que estar frente a la pantalla el menor tiempo posible, espero que el corrector sepa entender) no pude ponerle mucho empeño al formato del menú con el usuario y no pude implementar algunas cosas que pensé que serían útiles. Por ejemplo: Para el menú, simplemente cree un vector estático de comandos (no vi problema en hacer esto ya que sólo hay seis) y con una función que pide al usuario una letra hice que se ejecutaran los comandos del menú. Me hubiera gustado que el menú forme parte del gestor, así podía hacer pruebas de que los comandos ingresados por el usuario se lean bien y se ejecute lo que corresponde, pero en un principio lo implementé de este modo y no llegué a modificarlo. Otro problema que me surgió fue el de no poder evitar hacer algunos pritf dentro de las funciones de gestor.c. Para el programa de main.c no hay problema, pero para las pruebas causa molestia ya que se imprimen por pantalla cosas además de las pruebas efectuadas. Quizás esto podría haber sido evitado de algún modo, pero me quedó en el tintero.

Ahora bien, sobre la implementación (se hablará por último sobre buscar_cajas y el hash de índices de pokemones utilizado para esto): el gestor es básicamente un hash de cajas, que tienen como clave el nombre del archivo que corresponde a cada caja. Además, tiene un booleano

que indica si la gestión (el programa) finalizó o no. El tener un hash de cajas permite acceder directamente a las cajas (complejidad O(1) en la búsqueda si no hubo colisiones al insertar), lo que no sería posible con una lista o un abb.



En la imagen puede verse una tabla de hash abierta. El campo en rojo representaría el nombre de la caja y el azul la variable de tipo caja_t, que es la caja que se guarda junto con su nombre.

3.1. Implementación de funciones y comandos

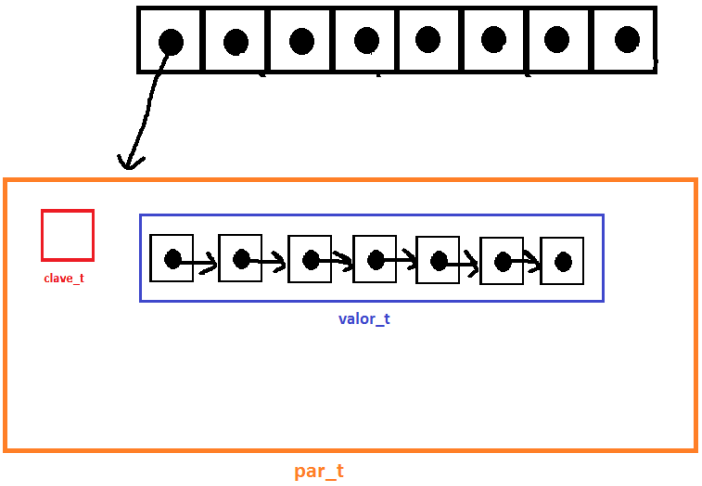
- gestor_inicializar: Es una función que recibirá como parámetros a argc y argv, como se pide en el enunciado del trabajo (en realidad le envió como parámetros argv+1 y argc, así evito enviar el comando ./main y envió directamente los nombres de los archivos que se quieren cargar en el gestor). Lo que hace esta función es encargarse de reservar la memoria correspondiente para el gestor y luego, con los nombres de archivos ingresados, se crean las cajas (si no se pueden crear o hay un nombre repetido no se hace nada) y se guardan en el hash del gestor, con sus nombres como claves. Esta operación es algo compleja (sin considerar los índices de cada pokemon), se inserta cierta cantidad de cajas, y crear las cajas tiene su complejidad ya que se debe insertar en cada una (cada una es una abb) todos los pokemones que correspondan.

- gestor_mostrar_inventario: Se muestran los nombres de las cajas cargadas en el gestor utilizando el iterador de hash. Complejidad es O(n) ya que se recorre todo el hash (todas las cajas).

- gestor_cargar_caja: Para esta función opté por recibir por parámetro una función que se encarga de leer lo que ingresa el usuario (también puede ser una función que directamente devuelva un string sin necesidad de pedir nada al usuario, lo que me sirvió para realizar las pruebas). Se guarda en un string lo que devuelve esta función y, en caso de que no haya una caja cargada con lo que contenga el string como nombre, se crea una nueva caja con este nombre y se carga en el gestor. La complejidad es O(n.log(n)), ya que se deben insertar (O(log(n)) todos los pokemones (O(n)) del archivo en un abb (en el caso de que degenere en lista será O((n^2)).

- gestor_combinar_cajas: Al igual que con la anterior, se recibe una función que, en el caso del programa de main.c, se encarga de pedir al usuario que ingrese tres nombres de archivo y devuelve un string con todos ellos juntos, separados por salto de linea para poder diferenciarlos y separarlos. Con hash_obtener se obtienen las dos cajas que se quieren combinar y con caja_combinar se las combina y se carga esta nueva caja en el gestor (se usa caja_guardar_archivo para generar el archivo que se pide).

- gestor_buscar_caja: Teniendo un gesto que solo tenga un hash con las cajas, hubiera sido imposible implementar esto. Es por este motivo que tuve que añadir al gestor otra tabla de hash de índices de pokemones: las claves de este hash serán los nombres de los pokemones presentes en el gestor y los valores serán listas que tendrán los nombres de las cajas en las que se encuentran dichos pokemones. Cada vez que se cargue una caja en el gestor, se deben actualizar los índices de los pokemones que ya se encontraban en él o se deben crear nuevos índices si un nuevo pokemon aparece en el gestor. Se puede ver en la figura como es un hash como clave tendra el nombre de un pokemon (en rojo) y como valor tendrá una lista de nodos enlazados (cada nodo tendrá el nombre de una caja)



Lo que se hace para buscar caja es simplemente pedir al usuario que ingrese el nombre del pokemon y, como tenemos un hash, se accede directo al índice de cajas asociado a este pokemon (con hash_obtener) y se utiliza el iterador de lista para mostrar estas cajas.

Esta operación tiene complejidad O(n), lo costoso en esto es que cada vez que se carga una caja se deben actualizar los índices, por lo que se debe insertar en cada lista un nombre de caja, resrvando memoria extra para cada lista y para cada nombre de caja, e insertando cada nombre en cada lista de pokemon (inserto en la primera posición de la lista para que sea O(1) la complejidad de la inserción, al igual que la complejidad de la inserción en el hash en el caso de tener que insertar una nueva lista para un nuevo pokemon, por lo que la complejidad de actualizar el índice al cargar una caja es O(n), ya que se deben actualizar los índices para todos los pokemones de la caja que se carga).