

Proyecto 3

TAD Abb

Algoritmos y Estructuras de Datos II Laboratorio

7 de mayo de 2009

Implementar un TAD diccionario con un arbol binario de búsqueda *abb*.

Igual que en el proyecto anterior, el diccionario debe poder ser guardado y leído desde disco. Para ello se brindan ya programadas los TAD's cinta de lectura y escritura de pares (ver en la página de la materia).

El diseño de estas modificaciones será el siguiente:

- El diccionario debe implementarse ahora con un TAD *abb* conservando las funcionalidades del proyecto anterior.
- Hay que implementar el nuevo TAD *abb* como se vió en el teórico. Esto es, se puede definir una estructura como sigue:

Archivo **abb.h**:

```
typedef struct sAbb *Abb;

/* constructor */
Abb
abb_empty (void);

/* agrega una clave y un dato */
void
abb_add (Abb h, Key k, Data d);

/* devuelve si existe la clave en el abb */
Bool
abb_exists (Abb h, Key k);

/* busca un dato segun la clave
   debe llamarse solo si abb_exists (l, k)
   si no lo encuentra devuelve NULL */
Data
abb_search (Abb h, Key k);
```

```

/* borra la clave y el valor asociado */
void
abb_del (Abb h, Key k);

/* devuelve la cantidad de elementos */
int
abb_length (Abb h);

/* Lee un abb desde un archivo
Abb
abb_fromFile(char *nomfile);

/* Graba un abb a un archivo
void
abb_toFile(char *nomfile, Abb h);

/* Destructor.
   Destruye todo el abb y su contenido */
Abb
abb_destroy(Abb h);

```

Archivo **abb.c**:

```

#include "abb.h"

typedef struct Branch *Tree;
#define Leaf ((Tree) NULL);

struct sAbb {
    int cant;
    Tree t;
};

struct Branch {
    Key k; Data d;
    Tree l, r;
};

```

donde `cant` es la cantidad de tuplas ingresadas.

- Dejando todos los demás TAD's igual que en el proyecto anterior (inclusive la interfase con el usuario) testear esta nueva implementación.

Ejercicio ★ 1 ¿Que sucede si el diccionario se guarda en disco de forma ordenada?

*¿Como se puede solucionar esto?
Implementar la solución*

Ejercicio ★ 2 *Hacer que el árbol permanezca balanceado.*

Ejercicio ★ 3 *Hacer una interface para buscar un conjunto de palabras al azar (utilizar el diccionario que esta en la página de la materia). Probar las distintas performance del diccionario sobre `abb` con y sin balanceo y el proyecto anterior.*

Se puede usar el comando `time` (ver `man page`) o investigar funciones de librería en C (ver `info page`).

Además, de forma general, para hacer cada TAD se deberá tener en cuenta:

- Todos los TAD's deberán estar implementados con la técnica de punteros a registros vista en el teórico del laboratorio.
- Cada TAD se deberá escribir en un par de archivos separados, un `.h` ("headers") y un `.c`.
- Para implementar correctamente los TAD's, cada `.h` deberá exportar únicamente las funcionalidades del TAD que define, ocultando todos los aspectos que tienen que ver con su implementación.
- No definir las estructuras que implementan los TAD's en los archivos "headers" ya que allí solo va la parte pública del TAD y no los detalles de implementación.
- Los programas deben estar libres de "memory leaks".
- Recordar que todos los `.h` deben tener su cerradura `ifndef`.
- Utilizar `Makefile`.
- Utilizar todos los parámetros de `gcc` para detectar posibles errores a tiempo de compilación.
- Para obtener nota B el programa debe funcionar sin errores y debe cumplir con todos los puntos anteriores.
- Para obtener nota B⁺ el programa debe funcionar sin errores, debe cumplir con todos los puntos anteriores y se deben hacer todos los puntos estrella.