Árbol Binario de Búsqueda

El trabajo que deben entregar de **forma grupal** es el tipo de dato abstracto Árbol Binario de Búsqueda (ABB). Se incluye en el sitio de descargas el archivo de main correspondiente al ejercicio. Se debe implementar:

```c
typedef struct abb abb_t;

typedef int (*abb_comparar_clave_t) (const char *, const char *);
typedef void (*abb_destruir_dato_t) (void *);

abb_t* abb_crear(abb_comparar_clave_t cmp, abb_destruir_dato_t destruir_dato);

bool abb_guardar(abb_t *arbol, const char *clave, void *dato);
void *abb_borrar(abb_t *arbol, const char *clave);

void *abb_obtener(const abb_t *arbol, const char *clave);
bool abb_pertenece(const abb_t *arbol, const char *clave);

size_t abb_cantidad(const abb_t *arbol);

void abb_destruir(abb_t *arbol);
```

La función de comparación (de tipo `abb_comparar_clave_t`), recibe dos cadenas y devuelve:

- Un entero menor que 0 si la primera cadena es menor que la segunda.
- Un entero mayor que 0 si la primera cadena es mayor que la segunda.
- 0 si ambas claves son iguales.

Qué implica que una cadena sea igual, mayor o menor que otra va a depender del usuario del TDA. Por ejemplo, `strcmp` cumple con esta especificación.

La función `destruir_dato` se recibe en el constructor, para usarla en `abb_destruir` y en `abb_insertar` en el caso de que tenga que reemplazar el dato de una clave ya existente.

Por otro lado deben implementar dos iteradores inorder.

Un iterador interno, que funciona usando la función de callback “visitar” que recibe la clave, el valor y un puntero extra, y devuelve true si se debe seguir iterando, false en caso contrario:

```c
void abb_in_order(abb_t *arbol, bool visitar(const char *, void *, void *), void *extra);
```

Y un iterador externo, con las siguientes definiciones y primitivas:

```c
typedef struct abb_iter abb_iter_t;

abb_iter_t *abb_iter_in_crear(const abb_t *arbol);
bool abb_iter_in_avanzar(abb_iter_t *iter);
const char *abb_iter_in_ver_actual(const abb_iter_t *iter);
bool abb_iter_in_al_final(const abb_iter_t *iter);
void abb_iter_in_destruir(abb_iter_t* iter);
```

Contamos con un script de pruebas que pueden ejecutar para verificar que la estructura que implementaron funciona correctamente. De todas formas, al igual que en entregas anteriores, deben realizar sus propias pruebas (pueden tomar las pruebas del hash como referencia, ya que el comportamiento de ambas estructuras es muy similar).

Como siempre, deben subir el código completo a la página de entregas de la materia y también entregarlo impreso con nombre y padrón de ambos integrantes, si su corrector así lo requiere.

No olviden revisar las preguntas frecuentes del árbol binario de búsqueda

---

Bibliografia recomendada

- Weiss, Mark Allen, “Data Structures and Algorithm Analysis”: *Chapter 4: Trees*, en particular desde *4.3. The Search Tree ADT- - Binary Search Trees*.
- Cormen, Thomas H. “Introduction to Algorithms”: *12. Binary Search Trees*.