.:: Correcciones TDA ABB ::.

.:: De Nobili, Lautaro 107394 ::.

• Informe:

¿Qué es un Árbol/Árbol Binario/Árbol Binario de Búsqueda?

“Es un tipo de dato aleatorio…”

¿A que te referis con “tipo de dato aleatorio”?

En general no queda muy clara la definición de Árbol, por ejemplo, no se menciona es una estructura naturalmente recursiva y el concepto de sub-árbol. Algo importante es separar lo que es el concepto de Árbol como TDA y la implementación utilizando nodos.

Con respecto a la diferencia entre Árbol/Árbol Binario/ABB también estaría bueno agregar que se engloban unos a otros, un ABB es un tipo de Árbol Binario y un Árbol Binario es un Árbol.

Detalles de implementación:

Estaría bueno que en vez de explicar en forma de texto lo que hacen las funciones lo hagas como un puntualización, es decir:

1. Creó el nodo a insertar
2. Busco la posición donde debo insertarlo
3. etc…

Diagramas:

En general muy claros, pero estaría bueno que también hayan diagramas que muestran los recorridos.

• Código:

- Muy bien modularizado, las funciones son muy claras.
- `destruir_nodo_con_dos_hijos(nodo_abb_t* nodo_inicio)` esta bien, pero se le podría hacer una pequeña optimización, en vez de buscar el nodo `izquierdisimo_del_predecesor` podrías sencillamente asignar al nodo padre del predecesor los hijos izquierdos del predecesor.Un ejemplo sería:

```
                      11
                     /
                    9
                  /   \
                2       10
              /   \
            1       6
                      \
                       8
                      /
                     7
                    /
                  6.5
Borro (9)
                      11
                     /
                    8
                  /   \
                2       10
              /   \
            1       6
                      \
                       7
                      /
                     6.5
```

• Pruebas:

- Utilizar int’s en el heap no es necesario, podes tener un puntero a un int que vive en el stack. De esa manera no tenes que preocuparte por crearlos y después destruirlos, más allá de cuando quieras probar el destructor.

- En estos casos no es necesario castear a `void*` ya que C lo hace por su cuenta, recorda que `void*` es el puntero comodín

  ```
  abb_recorrer(árbol, INORDEN, (void**)vector_enteros, 20);
  ```

- Deberías chequear posición por posición los array resultantes de recorrer el árbol.

```
ABB in-order: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ✓ e pudieron imprimir 15 obje
tos (INORDER)
ABB pre-order: 8 4 2 1 3 6 5 7 12 10 9 11 14 13 15 ✓Se pudieron imprimir 15 obj
etos (PREORDEN)
ABB post-order: 1 3 2 5 7 6 4 9 11 10 13 15 14 12 8 ✓Se pudieron imprimir 15 ob
jetos (POSTORDER)
```

Imaginate que si el resultado de ABB pre-order es 8 4 2 1 3 5 6 7 12 10 9 11 14 13 15. Es muy difícil darse cuenta que cambio y que ahora no devuelve el resultado correcto.

Nota 1.5/2