Proyecto 3 - Map con Árboles Binarios de Búsqueda

Algoritmos y Estructuras de Datos II - Laboratorio

Docentes: Leonardo Rodríguez, Gonzalo Peralta, Milagro Teruel, Diego Dubois, Jorge Rafael.

Descripción

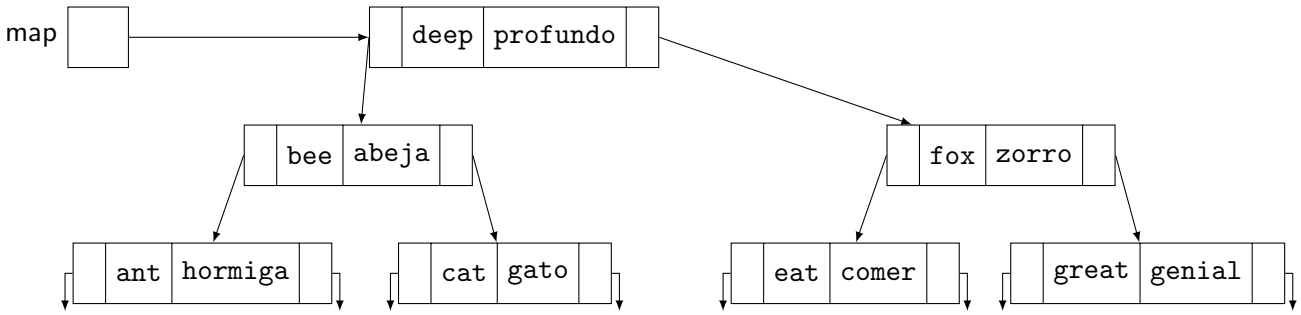
En este proyecto deben implementar el tipo `map` (el mismo del proyecto anterior) pero usando árboles binarios de búsqueda, en lugar de la lista enlazada. Se pide la implementación de árboles binarios de búsqueda usando punteros, tal como se vio en el teórico.

Respecto al proyecto anterior, sólo debe cambiar el archivo `map.c`, el resto debe quedar intacto, a menos que se encuentren errores. Tanto el diccionario como la interfaz de línea de comandos no necesitan tener ningún cambio.

Este proyecto no posee esqueleto. Los tests pueden reutilizarse para la nueva versión de `map`.

Implementación

El siguiente es un ejemplo de un mapeo representado con punteros a nodo:



Notar que ahora los nodos tienen cuatro elementos: nodo izquierdo, clave, valor y nodo derecho.

El mapa vacío se representará con un puntero `NULL`.

Invariante de representación: todos los nodos cumplen con la siguiente propiedad:

- Los nodos a la izquierda tienen una clave menor estricta (alfabéticamente).
- Los nodos a la derecha tienen una clave mayor estricta.

Según el dibujo, en el árbol con raíz en `deep` todos los nodos a la izquierda (`ant`,`bee`,`cat`) son menores que `deep` y todos los nodos a la derecha (`fox`,`eat`,`great`) son mayores que `deep`. Lo mismo vale para el árbol con raíz en `bee` y con raíz en `fox`.

Los nodos pueden tener 0, 1 o 2 nodos hijos. Si un nodo no tiene hijo izquierdo o derecho, el puntero correpondiente será `NULL`. En particular las "hojas" del árbol tienen ambos punteros a `NULL`.

Las funciones a implementar son las mismas que en el proyecto anterior, excepto que ahora se aprovechará la invariante de representación para hacerlas más eficientes. Los nuevos requisitos son:

- `map_put`, `map_get` y `map_remove` deben ser de complejidad logarítmica $\mathcal{O}(log\, n)$ en el caso promedio.
- `map_dump` debe imprimir los pares claves-valor ordenados alfabéticamente.
- Todas las funciones deben preservar el invariante de representación.

Todas las funciones pueden implementarse usando recursión, siguiendo el apunte del teórico.

Se agregan dos requisitos para aprobar:

- Seguir un estilo de código de manera coherente, cuidar la identación (de 4 espacios, sin tabs).
- Escribir un `Makefile` que permita compilar y ejecutar el proyecto, y además ejecutarlo con `valgrind`.

Punto Estrella

Implementar todas las funciones de manera imperativa, *sin* usar recursión. Podría ser necesario implementar un TAD Stack para lograr este objetivo.

Entrega

- Parcialito y Entrega 3: Martes 24 de Mayo.

Recordar

- Preguntar en clase o en la lista de mails las dudas. En caso de mandar un mail, no poner código.
- Comentar e indentar el código apropiadamente.
- Todo el código tiene que usar la librería estándar de C, y no se puede usar extensiones GNU de la misma.
- El programa resultante **no** debe tener *memory leaks* **ni** accesos (read o write) inválidos a la memoria.
- Recordar que la traducción de pseudocódigo al lenguaje C **no** es directa. En particular, tener en cuenta que lo que se llama `tuple` en el teórico, en C es un `struct`.