Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Estructuras de Datos y Algoritmos I

Trabajo Práctico 2

Integrantes:

Cipullo, Inés

Palumbo, Matías

Universidad Nacional de Rosario

2020

A continuación, se detallan las elecciones en general y particularidades que nos resultaron relevantes en la resolución del Trabajo Práctico 2.

1 Preliminares

El desarrollo del Trabajo Práctico fue realizado en su totalidad en el sistema operativo Mac OS 10.15.2. Se testeó tanto en Mac OS como en Ubuntu 19.10. Además, se solucionaron problemas de manejo de memoria corriendo el programa con Valgrind en la misma versión de Ubuntu.

2 Elección de estructura de datos

La estructura de datos utilizada a lo largo del programa representa Árboles de Intervalos, donde los nodos siguen un orden lexicográfico y son implementados como árboles binarios con altura balanceada. La estructura incluye, además de los elementos usuales de la estructura de árbol binario (los punteros `left` y `right`, al subárbol izquierdo y derecho respectivamente), tres variables del tipo de dato `double`:

- `izq` y `der`, que representan los extremos del intervalo almacenado en el nodo, y
- `max`, que representa el máximo extremo derecho entre los intervalos contenidos en los subárboles del nodo (a lo largo del trabajo se referirá a él como el "máximo" de un nodo).

También se incluye una variable de tipo `int` que es la altura del nodo.

Si bien hace a la generalidad de la implementación de la estructura el uso de una estructura auxiliar para almacenar el dato, en este caso un intervalo, preferimos utilizar los dos doubles por separado por cuestiones de legibilidad del código y ya que no nos pareció que agregase una generalidad significativa en lo que respecta a este trabajo en particular. Asimismo, la decisión de agregar la variable `altura` a la estructura se basó principalmente en buscar evitar recorrer el árbol completo cada vez que se necesite la altura o el factor de balance de un nodo. Se aprovechó para reducir operaciones en el cálculo de altura y máximo de un nodo el hecho de

que las funciones `itree_insertar` y `itree_eliminar` se llaman recursivamente hasta llegar a la posición del nodo en cuestión, y luego los datos se actualizan desde abajo hacia arriba (desde las hojas hacia la raíz).

3 Compilación de los archivos

La estructuración de los archivos y sus dependencias se encuentra detallada en el archivo *makefile*. Para compilar el programa se utiliza el comando `make interprete` o `make`. A su vez, luego de la compilación, el ejecutable del intérprete se corre mediante el siguiente comando:

`./interprete`.

4 Decisiones en el comportamiento

4.1 Comandos del intérprete

El intérprete para manipular árboles de intervalos desde la entrada estándar soporta los siguientes comandos:

- `i [a, b]`
- `e [a, b]`
- `? [a, b]`
- `dfs`
- `bfs`
- `salir`

donde `[a, b]` es un intervalo bien definido ($a \leq b$). Además, vale aclarar que es case-sensitive, y que en los tres primeros casos los comandos llevan un espacio entre el primer caracter y el corchete que inicia el intervalo, y entre la coma y el segundo número.

En cuanto a los mensajes de error, hace una distinción que consideramos significativa: se imprime el mensaje "`Intervalo no válido.`" en caso de que la estructura del comando ingresado sea correcta pero el intervalo ingresado no este correctamente definido (es decir, $a > b$), e imprime el mensaje "`Comando no válido.`" si se ingresa cualquier comando que no cumpla con las especificaciones antes dadas.

4.2 Recorrido DFS

La función que realiza el recorrido primero en profundidad del árbol de intervalos es de naturaleza recursiva, lo cual no presenta dificultades en cuanto a memoria ya que la altura del árbol, al ser balanceado, se ve limitada al logaritmo de la cantidad de nodos del árbol.

Por otro lado, realiza el recorrido DFS siguiendo el criterio de ordenamiento in-order, pues éste permite recorrer los elementos del árbol de forma ordenada (de menor a mayor) siguiendo el orden lexicográfico.

4.3 Impresión de los intervalos

El formato elegido para la impresión de los intervalos en la salida estádar es `%g`. Este formato imprime la opción más corta entre la notación decimal y científica de un número. La ventaja que posee esta opción por sobre `%lf` es que, considerando números no muy grandes y con menos de cuatro decimales aproximadamente (un rango de números que podemos considerar como los más usados), `%g` los imprime en la gran mayoría de los casos en notación decimal, sin ceros a la izquierda del último decimal (conocidos como "trailing zeros"). `%lf` imprime los números con un número fijo de decimales luego de la coma, por lo que se prefirió utilizar `%g` para evitar trailing zeros en algunos casos.

5 Dificultades encontradas

Cómo leer por la entrada estándar; por qué usamos fgets.

6 Bibliografía

- `https://www.geeksforgeeks.org/avl-tree-set-1-insertion/`
- `https://www.geeksforgeeks.org/avl-tree-set-2-deletion/`
- `https://en.wikipedia.org/wiki/Interval_tree`
- *The C Programming Language*, Kernighan & Ritchie, Sección 7.4 - `scanf`.