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 Elección de estructura de datos

La estructura de datos utilizada a lo largo del programa es Árboles de Intervalos donde los nodos siguen un orden lexicográfico, e implementados como árboles binarios balanceados. La estructura de árbol de intervalos 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 y por lo tanto son el dato de la estructura, y `max` que representa el máximo extremo derecho entre los intervalos contenidos en ese subárbol. También incluimos una variable del tipo `int` que es la altura del nodo.

Si bien hace a la generalidad de la estructura del árbol y de la implementación de la estructura el uso de una estructura auxiliar para almacenar el dato, en este caso un intervalo, no lo creimos necesario ya que no nos pareció que agregase una generalidad significante y entorpecia la legibilidad del código. 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.

2 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 siguientes comando:

`./interprete`.

3 Decisiones en el comportamiento

3.1 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, es decir, distingue minúsculas de mayúsculas, 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 nos pareció significativa. Imprime el mensaje `Intervalo no valido` 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 valido` si se ingresa cualquier comando que rompa con las especificaciones antes dadas.

3.2 Recorrido DFS

La función que realiza el recorrido primero en profundida del árbol de intervalos fue hecha de forma recursiva, y no iterativa. Esto fue porque la estructura de árbol con la que trabajamos es recursiva y al trabajar con árboles balanceados la altura de los mismos se ve limitada al logaritmo de la cantidad de nodos del árbol, y por lo tanto la recursión en este contexto no representaba dificultades mayores en cuanto al uso de memoria.

Por otro lado, realiza el recorrido DFS siguiendo el orden in-order, pues siendo que el árbol sigue un orden lexicográfico, este orden permite recorrerlo, y consecuentemente imprimirlo, de forma ordenada de manor a mayor.

3.3 Impresión de los intervalos

El formato elegido para la impresión de los intervalos en la salida estádar es `%g` porque ...

4 Dificultades encontradas

Cómo leer por la entrada estándar; porque usamos fgets.

5 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`