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), las siguientes variables:

- `Intervalo *intv`, una estructura utilizada para almacenar los extremos del intervalo,
- `double 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), y
- `int altura`, que es la altura del nodo.

Si bien en un principio se prefirió no utilizar la estructura auxiliar `intv` y agregar dos doubles a la estructura para almacenar el intervalo, utilizar la estructura hace a la generalidad de la implementación, y modularizando el código adecuadamente se pueden evitar líneas que comprometan la legibilidad del código.

Por otro lado, 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 la cantidad de operaciones en estas funciones 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

Para manipular árboles de intervalos desde la entrada estándar, el intérprete 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, como se especificó en el enunciado del trabajo, es case-sensitive, y 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, se 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 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, decidimos que esta función realice el recorrido DFS siguiendo el criterio de ordenamiento in-order ya que éste permite recorrer los elementos del árbol de forma ordenada (de menor a mayor) siguiendo el orden lexicográfico.

5 Dificultades encontradas

La mayor dificultad con la que nos encontramos podría decirse que fue decidir qué función utilizar para leer de la entrada estándar. La primera opción que consideramos fue `scanf`, pero terminamos decidiendo por escanear el input de la entrada estándar con `fgets` y parsearlo con `sscanf`, de acuerdo al formato establecido de los comandos. Es de gran importancia en la validación del input del programa el hecho de que `sscanf` devuelve un entero con la cantidad de variables a las que les asignó un valor según el formato dado.

Por otro lado, con respecto a la impresión de los intervalos, consideramos las diferentes opciones y el formato que elegimos es `%g`. Este formato imprime la opción más corta entre la notación decimal y científica de un número. La ventaja que posee por sobre `%lf` es que, teniendo en cuenta 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.

6 Bibliografía

- *The C Programming Language*, Kernighan & Ritchie, Sección 7.4 - Formatted Input.
- `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`
- `https://www.lemoda.net/c/double-decimal-places/`