TP Árbol De Intervalos

Cassinerio Marcos - Cerruti Lautaro

Junio 7, 2020*

*Updated June 7, 2020

1 Introducción

El objetivo de este trabajo fue implementar un árbol AVL de Intervalos. Adicionalmente implementamos un intérprete para poder utilizar el árbol y sus funciones desde la entrada estándar.

2 Compilado y ejecucion

Para compilar el proyecto abrimos una terminal, y una vez ubicados en el directorio del proyecto, ejecutamos el comando `make`. Esto nos generará el ejecutable del intérprete.

El mismo lo corremos con:

```
./interprete
```

Este programa nos permitirá ingresar comandos para manipular un árbol. Los comando aceptados son:

- `i [a, b]`: inserta el intervalo [a, b] en el árbol. Ej: `i [5.4, 6.3]`.
- `e [a, b]`: elimina el intervalo [a, b] del árbol. Ej: `e [2.2, 1e4]`.
- `? [a, b]`: interseca el intervalo [a, b] con los intervalos del árbol. Ej: `? [-1, 0.5]`.
- `dfs`: imprime los intervalos del árbol con recorrido primero en profundidad.
- `bfs`: imprime los intervalos del árbol con recorrido primero a lo ancho.
- `salir`: destruye el árbol y termina el programa.

Los primeros 3 comandos tienen que respetar el formato `caracterOperacion [a, b]`.

3 Organizacion de los archivos

El programa se divide en 4 partes: Intervalo, Árbol de Intervalos, Cola (Queue) e intérprete
Por un lado tenemos la implementación y declaración de Queue en los archivos `queue.c` y `queue.h` respectivamente.
Por otro lado tenemos Intervalo hecho de la misma manera, en los archivos `intervalo.c` y `intervalo.h`.
Luego tenemos al Árbol de Intervalos que hace uso de las dependecias queue e intervalo. Su implementación y declaración se encuentra en los archivos `itree.c` y `itree.h`.
Finalmente tenemos en el archivo `interprete.c` el intéprete que es nuestra interfaz del progragama para manipular los Árboles de Intervalos.

4 Implementaciones y estructuras

4.1 Queue

La implementación de Queue esta basada en una Lista Doblemente Enlazada Circular, definida de la siguiente forma:

```
struct _QNode {
  void *dato;
  struct _QNode *ant;
  struct _QNode *sig;
};

typedef struct _QNode QNode;

typedef QNode *Queue;
```

En su cabecera declaramos sus únicas 3 funciones:

- `queue_crear`
- `queue_pop`
- `queue_push`
- `queue_destruir`

La función `queue_destruir` nunca es utilizada, pero para que la implentación de la Cola sea general se vio la necesidad de hacerla.
Todas las implementaciones se encuentran en `queue.c`.

4.2 Intervalo

La declaración de intervalo es la siguiente:

```
struct _Intervalo{
  double extremoIzq;
  double extremoDer;
};

typedef struct _Intervalo Intervalo;
```

En su cabecera declaramos las siguientes funciones:

- `intervalo_crear`
- `intervalo_destruir`
- `intervalo_extremo_izq`
- `intervalo_extremo_der`
- `intervalo_valido`
- `intervalo_interseca`
- `intervalo_imprimir`

Sus implentaciones se encuentran en `intervalo.c`.

4.3 ITree

El Árbol de Intervalos se encuentra definido de la siguiente manera:

```
struct _INodo {
  Intervalo *intervalo;
  double maxExtremoDer;
  int altura;
  struct _INodo *izq;
  struct _INodo *der;
};

typedef struct _INodo INodo;

typedef INodo *ITree;
```

En su archivo cabecera se encuentran declaradas las siguientes funciones:

- `itree_crear`
- `itree_destruir`
- `itree_altura`
- `itree_insertar`
- `itree_eliminar`
- `itree_intersecar`
- `itree_recorrer_dfs`
- `itree_recorrer_bfs`
- `itree_aplicar_a_intervalo`

Sus implementaciones se encuentran en el archivo `itree.c`, junto con las implementaciones de las funciones:

- `itree_nuevo_nodo`
- `itree_calcular_max_extremo_der`
- `itree_calcular_altura`
- `itree_actualizar_datos`
- `rotacion_izquierda`
- `rotacion_derecha`
- `itree_balance`
- `itree_balancear`
- `itree_obtener_menor`

4.4 Interprete

El interprete se encuentra el main del programa, este se encarga de leer la entrada estandar, validarla y ejecutar las funciones de ITree e intervalo según la entrada. En este archivo estan implementadas las siguientes funciones:

- `leer_cadena`
- `obtener_operacion`

5 Resultados Pruebas

5.1 Primer Caso de Prueba

Esta primer prueba la realizamos con 5000 personas.
Resultados:

5.1.1 Edad

```
Selection Sort: 0.147 segundos
Insertion Sort: 0.096 segundos
Merge Sort: 0.005 segundos
```

5.1.2 Largo Nombre

```
Selection Sort: 0.859 segundos
Insertion Sort: 0.329 segundos
Merge Sort: 0.007 segundos
```

5.2 Segundo Caso de Prueba

Esta segunda prueba la realizamos con 15000 personas.
Resultados:

5.2.1 Edad

```
Selection Sort: 1.522 segundos
Insertion Sort: 1.103 segundos
Merge Sort: 0.013 segundos
```

5.2.2 Largo Nombre

```
Selection Sort: 9.611 segundos
Insertion Sort: 3.647 segundos
Merge Sort: 0.021 segundos
```

5.3 Tercer Caso de Prueba

Esta última prueba la realizamos con 30000 personas.
Resultados:

5.3.1 Edad

```
Selection Sort: 10.479 segundos
Insertion Sort: 9.065 segundos
Merge Sort: 0.042 segundos
```

5.3.2 Largo Nombre

```
Selection Sort: 68.993 segundos
Insertion Sort: 33.761 segundos
Merge Sort: 0.051 segundos
```

Empezamos haciendo todo lo relacionado al Árbol de Intervalos, mientras trabajabamos en sus funciones, nos dimos cuenta que podiamos modularizar los intervalos como una estructura para facilitar el agregado, el borrado y la impresion de los mismos, y para mantener una mejor organización del código.
Luego de haber implementado las funciones necesarias de intervalos, proseguimos refactorizando lo ya hecho del árbol.
Realizamos estos cambios hasta llegar a la impresion del arbol con BFS. Esta la habiamos dejado para el final por la necesidad de implementar una Cola. La implementamos en su propio modulo

6 Bibliografia

https://en.wikipedia.org/wiki/Selection_sort
https://en.wikipedia.org/wiki/Insertion_sort
https://en.wikipedia.org/wiki/Merge_sort
https://www.geeksforgeeks.org/merge-sort-for-doubly-linked-list/
https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/