Trabajo Práctico N°2

Estructura de Datos y Algoritmos I

Barbageleta Blas
Castro Rojas Tomás

Junio 10, 2020

1 Introducción

El objetivo de este trabajo es la implementación de arboles de intervalos cerrados de numeros reales con la motivacion de realizar operaciones como insertar y eliminar intervalos de manera dinamica, y consultar la interseccion de dos intervalos de forma eficiente. El tipo de arbol implementado es Arbol AVL.

2 Organización y funcionamiento

El Trabajo Práctico se organiza en dos carpetas. En la carpeta `itree` se encuentra la implementacion de arbol de intervalos juntos con la implementacion de la estructuras Intervalo y la implementacion de Cola para el recorrido utilizando BFS.
Dentro de la carpeta `interprete` se encuentran el makefile con las reglas de compilación, el archivo interprete.c que es el ejecuta el interprete y dos archivos con funciones que validan la entrada para el interprete.
Para compilar el Trabajo Práctico ejecutar el comando:
`make interprete`

3 Dificultades

La estructura implementada en este trabajo fue la siguiente:

```
typedef struct _INodo {
Intervalo* intervalo;
double maxExtremoDer;
struct _INodo* der;
struct _INodo* izq;
}INodo;

typedef INodo* ITree;
```

Se intento expandir esta estructura agregando un campo `altura` donde guardaba la altura del arbol para no estar calculandola cada vez que queriamos determinar si el arbol estaba balanceado o no. Tratar de determinar como se actualizaba la altura luego de agregar un nodo era bastante engorroso pero el verdadero problema era que cuando se realizaba una rotación, cualquiera de ellas, no encontramos forma de generalizar el comportamiento de cambio de altura. Entonces terminamos desistiendo de esta idea.
De igual modo se penso en un campo `factorBalance` pero surgian los mismos problemas que con el campo altura.
Ademas, la funcion `itree_insertar` esta implementada de tal modo que si se intenta ingresar un intervalo que ya se encuentra en el arbol, no realiza ninguna operacion y comienza a ir para atras terminando las distintas recursiones. Con el campo `altura` o `factorBalance` hubieramos tenido que tener una manera de decir si se agregó o no un nodo.

Luego, tuvimos bastantes dificultades para implentar las funciones `itree_intersecar` e `itree_eliminar`.

En el caso de `itree_eliminar` el problema surgio cuando el nodo a eliminar tenia dos hijos. Segun nuestro criterio, no queriamos crear un nuevo intervalo con los mismos datos del intervalo que reemplaza al nodo a eliminar. Pero esto traia problemas como, por ejemplo, era muy engorroso tener que actualizar el hijo izquierdo del padre del nodo sucesor, ademas que con esta solucion no verificaba si al eliminar el nodo sucesor el arbol quedaba balanceado o no. Terminamos copiando el intervalo sucesor y llamando recursivamente a `itree_eliminar` sobre el arbol derecho.

En el caso de `itree_intersecar` nos rompimos la cabeza pensando en todos los casos posibles de interseccion entre dos intervalos, pensando en ejemplos y contraejemplos que resulto en una primera implementacion con 7 `if`. Luego pudimos reducir a 3 pero era dificil decir si siempre funcionaba por mas que al testearlo devolviera resultados correctos.
Leyendo con mayor detenimiento el ejemplo en la consigna del TP logramos escribir el pseudocodigo implicito en el ejemplo. Sin embargo, seguiamos sin entender por qué siempre funciona. Buscando en internet encontramos una pagina de GeeksforGeeks donde implementaban arboles de intervalos y lo más importante explicaban el algoritmo para la funcion `itree_intersecar`.

En cuanto al interprete, la funcion `strtod` nos facilito mucho el trabajo.

References

[1] Arbol AVL Wikipedia: `https://es.wikipedia.org/wiki/%C3%81rbol_AVL`

[2] Deletion in AVL Tree: `https://www.youtube.com/watch?v=LXdi_4kSd1o`

[3] `https://www.geeksforgeeks.org/interval-tree/`

[4] Funcion strtod() `https://www.tutorialspoint.com/c_standard_library/c_function_strtod.htm`