Universidad Nacional de Rosario

Facultad de ciencias exactas, ingenieria y agrimensura

Estructuras de datos y algoritmos I

Trabajo práctico final - Mesa Agosto

Autores:
Trincheri Lucio

Agosto 2020

EDyAI - Informe TP Final

Trincheri Lucio, LCC UNR FCEIA 18/08/2020

1. Aproximación al problema

Lo primero que fue realizado fue la lectura completa del trabajo a desarrollar, pensando en que partes este se iba a dividir esto y como organizar estas partes en distintos archivos con sus funcionalidades correspondientes. Aunque dos de las tres partes en las que dividí el trabajo ya sabía más o menos como las iba a encarar, la última parte del desarrollo de la interfaz me tomó por sorpresa debido a la complejidad y longitud de las misma.

2. Elección de estructuras y organización de datos

2.1. Conjuntos

A la hora de elegir la estructura de datos que vamos a utilizar para la representación de conjuntos, fue necesario tener en cuanto la posibilidad de admitir tanto conjuntos dados por extensión como por comprensión. Como primer paso, ya que se trata de conjuntos, decidí utilizar como base a la estructura de árboles de intervalo ya desarrollada durante el TP Nº2, y ampliar sobre ella el funcionamiento necesario para los conjuntos. Esto decisión surgió de la capacidad de representar tanto conjuntos por comprensión en forma de intervalos en sí, como a su vez números individuales mediantes intervalos con el valor de inicio y valor final iguales.

Algunas modificaciones que fueron surgiendo durante la implementación de las funciones de conjuntos fue la necesidad de crear una manera de aplicar una operación entre un intervalo y el árbol destino. De esta idea surgió el crear una variación de las funciones visitantes, como a su vez una variación de la función recorrer-dfs, la cual ya no se usaría para imprimir el árbol. Otra situación que fue aparente fue la necesidad de crear nuevos árboles copia, situación que no era una necesidad en la entrega anterior.

2.2. Pensamiento de operaciones de conjuntos

2.2.1. Unión

El desarrollo de la función unión fue la base de las demás funciones. Acá surgió la idea de duplicar uno de los árboles, sobre el cual se harían los cambios necesarios. En el caso de unión, desarrolle una función auxiliar que dado un intervalo a unir, se buscaba la intersección del mismo con el árbol, se expandía el nuevo intervalo unión y se repetía el proceso. Esto se realizaba con todo los intervalos del árbol a unir, hasta llegar al resultado de un árbol de intervalos disjuntos que tenia los elementos de ambos árboles operando en su interior.

2.2.2. Intersección

La idea de la intersección fue parecida a la de la unión, en cual se duplicaba un árbol (el más chico en este caso ya que intersecar es mas caro a grande es el árbol contra el cual se realiza) y se iba agregando de manera disjunta las intersecciones entre el árbol y los intervalos a un nuevo árbol intersección.

2.2.3. Unión

La idea de la resta usa como base a las funciones de la intersección. Aprovechándose de estas, la resta se desarrolla mediante la intersección del árbol a restar y la copia del árbol al cual se le restará. Si se encuentra una intersección, ella será removida del árbol duplicado, dejando así dos arboles disjuntos entre sí, siendo uno de ellos la resta final.

2.2.4. Complemento

El desarrollo del complemento fue el mas sencillo, ya que este se baso en la propiedad $A = U - A$, lo que fue fácil de crear ya que la función resta ya estaba definida.

2.3. Hash

En cuento a la parte de almacenar los arboles en alias generados por el usuario, fue necesario encontrar una forma de almacenar información mediante un índice alfanumérico. Debido a esto, fue aparente la idea de utilizar una tabla hash para poder ubicar y almacenar fácilmente a los árboles según su alias.

Sobre la manera de almacenar los árboles y su alias correspondiente, me decidí a utilizar nuestra implementación de listas doblemente enlazadas, simplificándola a una simplemente enlazada debido a la falta de necesidad del puntero para regresar a nodos anteriores.

Además de esto, en cuanto la generación del índice para ubicar la información, me base en una implementación de djb2. Esta decisión fue debido a la buena dispersión en casi cualquier largo y composición de palabras. Por otra parte, también fueron desarrolladas las funciones básicas de hash, como devolver el árbol dado un alias, almacenar un árbol y otras necesarias.

Listing 1: Implementación de la estructura Nodo.

```
1  typedef struct _HashLista {
2      AVLTree conjunto;
3      char *alias;
4      struct _HashLista *siguiente;
5  } HashLista;
6
7  // Puntero tabla hash
8  typedef struct {
9      // Lista enlazada
10     HashLista *lista;
11     unsigned int largo;
12 } Hash;
```

3. Desarrollo de la interfaz y manejo de alias

Sin duda, la parte más difícil del trabajo fue la del desarrollo de una interfaz funcional y relativamente fácil de leer y comprender. Para ello, la nueva interfaz se basa en la desarrollada en el TP2, sobre la cual se expandió en gran cantidad el checkeo y el reconocimiento de caracteres, para así poder descubrir comandos inválidos y errores en la sintaxis. Para interactuar con la misma, se creo una función que valida la entrada retornando un identificador para que luego un gran switch decida que acción realizar. El mismo cuenta con las funciones pedidas y con códigos de error para los distintos inconvenientes que se pueden encontrar. Por su parte, las funciones son bastantes permisivas en cuanto a errores, pudiendo permitir espacios de más, intervalos formados de forma erronea pero sin fallas graves (ej. a = {1,,2} es permitido) etc.

3.1. Formato soportado y operaciones

A continuación, se detalla la forma de los comandos aceptados por la interfaz

- Conjunto por extensión: A = {1,2,3,4,5,6}
- Conjunto por comprensión: A = {x: 4 <= x <= 10}
- Unión: A = B | C
- Intersección: A = B & C
- Resta: A = B - C
- Complemento: A = ∼B
- Imprimir: imprimir A
- Salir: salir

Cabe destacar, que los alias soportan una amplia gamas de caracteres tanto alfanuméricos como símbolos. Lo que si quedan reservadas para el buen funcionamiento de la interfaz son los comandos (salir, imprimir, etc).

4. Compilación del programa

Para compilar el ejecutable de manera simple, desarrollamos un archivo MakeFile. Llamando a make, se crearan todos los archivos object necesarios y el ejecutable nombrado "main". Para utilizar el programa debemos realizar lo siguiente:

```
./main
```

Una vez ejecutado el main, el usuario podrá interactuar con el programa mediante la consola. Finalmente, invocando make clean, se borraran los archivos object y el ejecutable del directorio.

5. Progreso y dificultades durante el desarrollo

Uno de los problemas más grandes que me surgió fue al momento de realizar el parseo de los comandos ingresados mediante la interfaz. Este problema nació debido a no saber que tan riguroso debía ser el parser en cuanto a la información ingresada por el usuario, así como que tan mal escrito esta podría llegar a estar (espacios de más, llaves mal colocados, etc). El código no era demasiado largo, sino que engorroso y difícil de leer. Finalmente, decidí reutilizar la implementación desarrollada durante el trabajo práctico 2, donde separo el comando en sus partes significativas, las cuales al ser analizadas nos otorgan la información necesaria para saber que función ejecutar.

Antes de dar como terminado el trabajo, puse énfasis en revisar y testear el uso dinámico de la memoria, para que no presente problemas ni perdidas de la misma. Las funciones de liberación de memoria y cuando se utilizan fue un punto importante a la hora de realizar el trabajo.

Para una mejor organización de los archivos, coloque el trabajo ya realizado (avltrees y sus estructuras necesarias en otra subcarpeta) en un subcarpeta, ya que estas no eran el énfasis del trabajo (excepto avltree, donde se encuentran los métodos de conjuntos). En el directorio raíz finalmente se encuentra el código con el cual interactuará el usuario (main). En otra subcarpeta, almacene los nuevos archivos de hash para que queden organizados. Finalice el trabajo modificando el makefile correspondiente al proyecto y limpie el directorio para poder ser entregado.

6. Bibliografía

Para realizar nuestra implementación, nos guiamos del pseudocódigo y las explicaciones detalladas de las siguientes páginas:

- http://www.cse.yorku.ca/~oz/hash.html
- Documentación de la cátedra.

Al igual que el tp anterios, volví a utilizar la siguiente herramienta para guiarme durante el desarrollo de los nuevos metodos de avltree:

- https://www.cs.usfca.edu/~galles/visualization/AVLtree.html