Spoletini Bruno - LCC

Trabajo Práctico Nº 3

28 de julio del 2020 - UNR - FCEIA

Introducción

Este trabajo consiste de un intérprete, mediante el cual es posible ingresar conjuntos por extensión, comprensión, y realizar operaciones sobre los mismos, tales como Unión, Intersección, Resta y complemento.

Para esto, utilizamos tablas hash, para almacenar los conjuntos, listas doblemente enlazadas, para almacenar los elementos de cada conjunto, y árboles AVL, para resolver colisiones en la tabla hash.

Compilación y ejecución del programa

El programa cuenta con un archivo makefile, que se encarga de compilar los archivos correspondientes al funcionamiento del mismo. Para ejecutarlo, basta con utilizar el comando **make** en la terminal, estando situado en el directorio principal. Esto generará un ejecutable llamado “Interprete”.

Dentro de ./estructuras/hash.h, en la línea 8, se define el tamaño de la tabla. Esto puede ser ajustado de forma acorde a la cantidad de conjuntos que se desea ingresar para aumentar la eficiencia del programa.

Intérprete

El intérprete dispone de ocho comandos que nos permiten crear conjuntos, realizar operaciones sobre estos, mostrar dichos conjuntos, y por último salir del mismo.

1. **alias = {k0,k1,...,kn}**: Este comando crea el conjunto de enteros {k0, k1, . . . , kn} y lo asocia con alias.
2. **alias = {x: k0 <= x <= k1}**: Este comando crea el conjunto {x : k0 ≤ x ≤ k1} y lo asocia con alias. El valor 'x' puede ser cualquier carácter, y debe ser el mismo en ámbos términos.
3. **alias1 = alias2 | alias3**: Este comando computa la unión de los conjuntos asociados con alias2 y alias3, y asocia el conjunto resultante con alias1.
4. **alias1 = alias2 & alias3**: Este comando computa la intersección de los conjuntos asociados con alias2 y alias3, y asocia el conjunto resultante con alias1.
5. **alias1 = alias2 - alias3**: Este comando computa la resta de los conjuntos asociados con alias2 y alias3, y asocia el conjunto resultante con alias1.
6. **alias1 = ~alias2** : Este comando computa el complemento del conjunto asociado con alias2, y asocia el conjunto resultante con alias1.
7. **imprimir alias**: Este comando imprime los elementos del conjunto asociado con alias, respetando el orden de los números enteros.
8. **salir**: Este comando cierra el intérprete, liberando adecuadamente los recursos solicitados.

Estructuras de datos utilizadas

Para este trabajo, se utilizaron árboles AVL, una modificación de listas doblemente enlazadas no circulares, tablas hash, y una estructura llamada ElemConj que contiene dos números enteros. A continuación detallaremos el uso de cada una de estas:

1. ElemConj

Esta estructura fue utilizada representar cada intervalo de los conjuntos ingresados. Todos los números ingresados son interpretados como intervalos, por lo que cada elemento de los conjuntos serán guardados como inicio:extremo del intervalo.

2. DList

   Esta estructura fue utilizada para contener a los conjuntos ingresados, los cuales son interpretados como una DList, que contiene el alias del conjunto, un puntero al inicio del conjunto, y un puntero al final de este.

3. Tabla hash

   La tabla hash se utiliza para almacenar los conjuntos ingresados, y poder acceder a ellos en un tiempo acotado. Esta tabla consiste en un array de punteros a CTree, de forma tal que en el caso de que haya colisiones, el nuevo conjunto pueda añadirse al árbol AVL, y no se pierda eficacia a la hora de ingresar y buscar elementos

4. CTree

   Esta estructura se utiliza para resolver las colisiones en la tabla hash. Cada elemento del array de la tabla contiene NULL, o un puntero a CTree, el cual a su vez, contiene un puntero a DList, donde se encuentra el conjunto.

Formatos aceptados para el alias

El alias permite un máximo de hasta 999 caracteres, permite el uso los caracteres alfanuméricos, y hace diferenciación entre mayúsculas y minúsculas.

Mensajes de error

El programa cuenta con una función que imprime por pantalla los mensajes de error. Existen dos tipos de mensaje:

- **Ingrese un comando válido**: esto se produce al ingresar un comando que el intérprete no reconozca.
- **No se encontró el alias buscado**: este mensaje aparece al intentar realizar una operación con un elemento que no fue ingresado previamente.

Dificultades encontradas

- **Resolución de colisiones en la tabla hash**
    - Uno de los principales problemas encontrados fue el cómo resolver las colisiones en la tabla, ya que desde un principio, se desconoce la cantidad de conjuntos que será ingresado.
    - Recurrí a técnicas conocidas como linear probing y double hashing, hasta que llegué a la que considero la más eficiente y balanceada en términos de memoria y velocidad de almacenamiento y búsqueda.
    - Para resolver este problema, utilicé árboles AVL, los cuales se ordenan según el orden lexicográfico del alias del conjunto.
- **Complejidad del intérprete**
    - Una parte importante del trabajo consistió en considerar los casos posibles en los que el programa pueda fallar al ingresar los comandos de forma incorrecta. Una solución que consideré posible, fue utilizar expresiones regulares, que permiten un control estricto del input, pero desistí de la idea por no ser librería estándar en windows.
    - Esto se resolvió mediante la modularización del código, de forma tal que cada comando ingresado es analizado de forma meticulosa.

Bibliografía

https://cp-algorithms.com/string/string-hashing.html

https://www.smartick.es/blog/matematicas/numeros/numeros-primos-criba-eratostenes/.

http://www.hci.uniovi.es/Products/DSTool/hash/hash-queSon.html

https://en.wikipedia.org/wiki/Linear_probing

https://www.cs.yale.edu/homes/aspnes/pinewiki/C(2f)HashTables.html?highlight=%28CategoryAlgorithmNotes%29

https://www.geeksforgeeks.org/hashing-set-3-open-addressing/

https://www.gnu.org/software/libc/manual/html_node/Regular-Expressions.html

https://www.educative.io/edpresso/how-to-write-regular-expressions-in-c

https://stackoverrun.com/es/q/193144

https://regexr.com/