TP FINAL EDyAL

Bolzan Francisco Mesa Julio 2020

Estructuras utilizadas

El programa consta de tres estructuras, las dos principales siendo una modificación de los arboles de intervalos y una tabla hash de dichos arboles; y luego un stack como estructura auxiliar.

Los arboles de intervalos son la estructura que representa y almacena a cada conjunto individual, decidí usar esta estructura ya que desde un principio la iba a necesitar para los conjuntos por compresión, y con algunas modificaciones pude almacenar ambos tipos de datos en los mismos.
Algunas modificaciones presentes son:

- Nueva funcion de inserción que "recorta" cada intervalo a medida que lo inserta en el arbol
- Adicion de funciones de operacion entre arboles como unir, intersectar, etc.
- Algunas correcciones a funciones ya existentes de acuerdo a las devoluciones del TP anterior (principalmente, cambiar el calculo de la altura para que no recorra todo el arbol en cada llamada).

La siguiente estructura es una implementación de tabla hash mediante direccionamiento abierto con doble hashing y un índice de carga para redimensionar la tabla en caso necesario. El tamaño de la tabla es siempre primo y se calcula mediante una función auxiliar.

Finalmente el stack está para poder realizar el recorrido de los arboles.

Formatos aceptados para alias

Los alias aceptados son de carácter alfabético y largo máximo de 100 caracteres. El largo máximo de toda operación son 254 caracteres, además de esto, las variables para la representación de un conjunto por compresión son también de carácter alfabético y largo máximo de 1 caracter.

Mensajes de error

1. "Límite de caracteres excedido": cuando el largo del comando supera el límite de caracteres permitidos.

2. "Comando inválido": cuando el comando ingresado no corresponde con el formato esperado para ningún comando real.

3. "Comando inválido para inserción de conjunto": cuando se reconoce que se quiso insertar un conjunto pero el formato del comando no es el adecuado.

4. "Caracter invalido en el conjunto" y "Formato de conjunto por extension invalido": cuando se reconoce que se quiso realizar una inserción de un conjunto por extensión, pero éste no cumple con el formato esperado.

5. "Formato inválido para operacion entre conjuntos": cuando se reconoce cierta estructura de una operación entre conjuntos pero esta no es válida.

6. "Formato inválido para complemento": cuando se reconoce que se quiso realizar un complemento pero este no cumple el formato esperado

7. "Formato inválido para operacion": cuando se reconoce que se quiso realizar una operacion de algun tipo pero esta no cumple con el formato esperado.

8. "Esa clave no corresponde a ningun elemento", "Algun conjunto pedido no existe" y "El conjunto pedido no existe": cuando las operaciones de impresión, operación entre conjuntos y complemento se procesaron de manera correcta pero uno o mas de los conjuntos necesarios para la operación no existen.

Los formatos esperados mencionados previamente son:
Salir: salir
Imprimir conjunto: imprimir alias
Op. entre conjuntos: alias = alias X alias (sea X: |,&,-)
Comp. de conjunto: alias = ~alias
Insercion por extension: alias = {n,...,m} (sean n hasta m números enteros dentro del rango [INT_MIN:INT_MAX])
Insercion por compresion: alias = {c : n <= c <= m} (sea c un caracter alfabético, y n m números dentro del rango [INT_MIN:INT_MAX])

Uso del programa

Siguiendo las recomendaciones del apartado de formatos aceptados y de errores, sólo queda ejecutar el comando **make** en la terminal y luego ingresar los conjuntos u operaciones deseados. Si se desea limpiar la carpeta, ejecutar **make clean**.

Dificultades y resoluciones

La primer dificultad encontrada fue la decisión de qué estructuras utilizar. La primer estructura que decidí usar fue la tabla hash, ya que esta me permitiría almacenar el alias de

una manera útil para el funcionamiento del programa al poder buscar conjuntos en la misma con costo constante mediante su alias. Decidí postergar la decisión de qué tipo de tabla y direccionamiento utilizar hasta luego de decidir en la/las otras estructuras principales necesarias. Eventualmente opté por hacer la tabla hash con direccionamiento abierto mediante doble hashing, decidí esto ya que me permite busquedas rápidas sin tener una cantidad excesiva de memoria sin utilizar.
La primer función de hash esta inspirada en preguntas y algunos análisis leídos por internet[1][2][3], mi objetivo con esto era por sobre todo evitar la función mas común de hash para strings (la suma de los valores de cada caracter) ya que esta evidentemente generaría un número considerable de colisiones en la tabla, lo cual aumentaría el tiempo de busqueda. La segunda función de hash es la dada en la práctica de tablas hash[4][5], para esta requería que el tamaño de la tabla fuese siempre primo, por este motivo cree una función que calcula el siguiente primo desde un entero dado, la cual implementa múltiples mejoras para que el tiempo de cálculo del mismo no sea tan costoso[6].

Lo siguiente fue decidir qué estructura representaría a cada conjunto siendo la tabla hash la que los almacena. Para esto una primer idea fue crear una estructura que represente a un conjunto y posea punteros a dos estructuras distintas, una para los elementos por extensión y otra para los elementos por compresión. Esta idea fue descartada rápidamente por múltiples motivos, principalmente la dificultad de las operaciones entre estructuras distintas y la complejidad de realizar dos implementaciones distintas pero que cumplan el mismo proposito. Lo que quedó de esa idea fue la utilización de la estructura itree (arbol de intervalos) para la representación de los conjuntos por compresión. Con esta idea en mente quedaba ver como "mezclar" los elementos por compresión y extensión en una sola estructura, lo cual implicaba modificar itree para que tome tanto intervalos como enteros o transformar los enteros en intervalos. Luego de una consulta y debido a la dificultad que proponia la primer solución, decidí optar por representar los enteros como intervalos, lo cual si bien implica almacenar un elemento innecesario, me da la posibilidad de recorrer el arbol de manera sencilla y rápida.

Con esto, las estructuras principales quedaron decididas y pude comenzar con el código. Uno de los primeros problemas en este punto surgió en la implementación de itree para representar conjuntos. Debido a la propiedad de los conjuntos que hace que no repitan elementos, requerí deshacerme de las repeticiones de intervalos; para esto modifiqué la función **itree_insertar** para que a medida que el intervalo "baja" por el árbol se vaya "recortando" para representar elementos que no pertenezcan al mismo. Esta nueva función probó ser el pilar principal de mi estrategia para realizar las operaciones en el futuro, ya que me permitió definir todas las operaciones siguiendo esta modalidad de "comparar y recortar", dejando a **itree_insertar** el trabajo de evitar repeticiones.
Siguiendo esta idea, la diferencia entre las operaciones está en qué se hace en caso de haber intersecciones entre intervalos y qué árboles se utilizan para comparar e insertar.
En la inserción, tanto las comparaciones como la inserción del intervalo se realizan en el mismo árbol. Las partes del intervalo que se intersectan con el árbol se ignoran y la inserción se retoma con las otras partes del mismo.

La union copia el árbol de mayor altura en un árbol resultado y luego inserta en el mismo todos los elementos del otro árbol, dejando a la función **itree_insertar** el trabajo de evitar colisiones.
La intersección toma dos árboles, y realiza la comparación de los elementos de uno sobre el otro "simulando" el camino que seguiría la inserción del elemento en dicho árbol; de haber intersecciones entre los intervalos estas son insertadas en un tercer árbol resultado y las partes que no se intersecten siguen su camino bajando el árbol, de llegar un intervalo hasta una hoja queda seguro que dicho intervalo no esta presente en ambos árboles.
De manera opuesta, el complemento compara el intervalo [INT_MIN:INT_MAX], contra todo el árbol para luego insertar en un árbol resultado solo aquellas partes del intervalo original que hayan llegado hasta el final.
Similarmente, la resta sigue la misma lógica que el complemento sólo que ese proceso se repite para todos los elementos de uno de los árboles sobre el otro arbol entero, a modo de quedar solo con aquellos intervalos provenientes de uno pero no presentes en el otro.

Hasta este punto, todos los resultados de las operaciones e inserciones existen como árboles nuevos, los cuales todavía no están en la tabla, para esto hay una tercer parte del programa, que sirve de vínculo entre la tabla y los árboles. El archivo **conjuntos** contiene las funciones que se encargan de insertar, buscar y eliminar elementos de la tabla; el propósito de estas funciones es aliviar la carga del intérprete, y servir como "implementación" definitiva de los conjuntos como tales. Como se dijo antes, luego de una operación el resultado es un nuevo árbol que debe ser insertado en la tabla, para esto, se elimina el elemento en su alias, a modo de sobreescribir el árbol de ser necesario y luego se inserta el nuevo árbol con su alias correspondiente.

Otro problema fue la creacion del interprete. Mi intención era obtener un interprete que distinguiese entre distintos "grupos" de errores. Para esto utilicé **sscanf** a modo de parsear la entrada de manera progresiva y utilizando ciertos caracteres o formatos como indicadores de qué comando se quiso ejecutar. Sólo para la inserción por extensión fue necesario utilizar **strtok** al ser esta mas apta para una lectura "iterativa" de la entrada. En este punto la mayoría de problemas eran referidos a estas y otras funciones especificas como **atoi**; algunos de estos inconvenientes fueron por ejemplo el hecho de que **sscanf** no reemplaza sino que hace un append del terminador y **atoi** que para ciertos errores no devuelve 0 correctamente (por ejemplo al escribir 1-2, **atoi** devuelve 1, por esto cree mi propia función que la reemplaza).

Bibliografía utilizada

(1)Funciones de hash para strings 1 StackOverflow
(2)Funciones de hash para strings 2 Computinglife
(3)Funciones de hash para strings 3
(4)Double hashing 1 Geeksforgeeks

(5)Double hashing 2
(6)Busqueda de primos Wikipedia
Data de scanf y sscanf
Funciones para operación de strings
Guía profunda de sscanf