TP Final Interprete de Conjuntos

Cerruti Lautaro

Agosto 15, 2020*

*Updated August 18, 2020

1 Introducción

El objetivo de este trabajo práctico es la implementación de un intérprete para operar con conjuntos numéricos. Entre las operaciones que se deberán soportar se encuentran: unión, intersección, resta y complemento.

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 operar con conjuntos. Dados 3 conjuntos con nombres `alias1`, `alias2` y `alias3`, los comandos aceptados son:

- `alias1 = alias2 | alias3`: une los conjuntos alias2 y alias3 y almacena el resultado en el conjunto alias1.
- `alias1 = alias2 & alias3`: interseca los conjuntos alias2 y alias3 y almacena el resultado en el conjunto alias1.
- `alias1 = alias2 - alias3`: resta al conjunto alias2 el conjunto alias3 y almacena el resultado en el conjunto alias1.
- `alias1 = ~alias2`: calcula el complemento de alias2 y lo almacena en el conjunto alias1.
- `alias1 = {1, 2, 3, 4, 5}`: almacena en alias1 el conjunto definido por extensión.
- `alias1 = {x: 1 <= x <= 5}`: almacena en alias1 el conjunto definido por comprensión.
- `salir`: destruye la tabla con los conjuntos y termina el programa.

3 Organizacion de los archivos

El programa se divide en 5 partes: Simple Linked List, Glist que es una Circular Duble linked List, THash, Conjunto e intérprete
Por un lado tenemos la implementación y declaración de SLList en los archivos `sllist.c` y `sllist.h` respectivamente.
Por otro lado tenemos GList hecho de la misma manera, en los archivos `glist.c` y `glist.h`.
Luego tenemos THash que hace uso de SLList. Su implementación y declaración se encuentra en los archivos `thash.c` y `thash.h`.
Luego tenemos Conjunto que hace uso de Glist. Su implementación y declaración se encuentra en los archivos `conjunto.c` y `conjunto.h`.
Finalmente tenemos en el archivo `interprete.c` el intéprete que es nuestra interfaz del progragama para manipular los Conjuntos.

4 Implementaciones y estructuras

4.1 GList

La implementación de GList es una Lista Doblemente Enlazada Circular, definida de la siguiente forma:

```
typedef struct _GNode {
  void *data;
  struct _GNode *prev;
  struct _GNode *next;
} GNode;

typedef GNode *GList;
```

En su cabecera declaramos sus funciones:

- `glist_create`
- `glist_destroy`
- `glist_pop`
- `glist_concat`
- `glist_insert_last_position`
- `glist_merge`
- `glist_split`
- `glist_merge_sort`

Todas las implementaciones se encuentran en `glist.c`.

4.2 SLList

La declaración de SLList es la siguiente:

```
struct _SLNode {
void *data;
struct _SLNode *next;
};

typedef struct _SLNode SLNode;

typedef SLNode *SLList;
```

En su cabecera declaramos las siguientes funciones:

- `sllist_create`
- `sllist_destroy`
- `sllist_insert_with_replace`
- `sllist_find`

Sus implentaciones se encuentran en `sllist.c`.

4.3 THash

La Tabla Hash THash es nada mas que un array de SLList, y se encuentra definida de la siguiente manera:

```
typedef SLList* THash;
```

En su archivo cabecera se encuentran declaradas las siguientes funciones:

- `tabla_hash_crear`
- `tabla_agregar_elemento`
- `tabla_buscar_elemento`
- `tabla_destruir`

Sus implementaciones se encuentran en el archivo `thash.c`.

4.4 Conjunto

Los Conjuntos se encuentran definidos de la siguiente manera:

```
typedef struct {
  char *nombre;
  GList conjunto;
} Conjunto;
```

En su archivo cabecera se encuentran declaradas las siguientes funciones:

- `conjunto_compara_nombre`
- `conjunto_hash`
- `conjunto_create_empty`
- `conjunto_create_extension`
- `conjunto_destroy_conjunto`
- `conjunto_agregar_elemento`
- `conjunto_normalize`
- `conjunto_imprimir`
- `conjunto_union`
- `conjunto_interseccion`
- `conjunto_complemento`
- `conjunto_resta`
- `comparar_conjunto_string`

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

- `conjunto_elemento_valores_interseca`
- `conjunto_comparar_elementos_by_extremo_izquierdo`
- `conjunto_destroy_elemento`
- `conjunto_extension_elemento`
- `extremo_mas_uno`
- `extremo_menos_uno`

4.5 Interprete

El interprete se encuentra el main del programa, este se encarga de leer la entrada estandar, validarla, buscar los conjuntos, realizar las operaciones y almacenar los resultados en la tabla. En este archivo estan implementadas las siguientes funciones:

`check_alphanumeric`
`hasheo_string`
`leer_cadena`
`imprimir`
`inserta_conjunto_comprension`
`inserta_conjunto_extension`
`realizar_operacion`

5 Formatos de Alias Aceptados y Mensajes Devueltos

Los formatos de alias aceptados son nombres alfanumericos, o sea, letras y numeros mezclados. No se aceptan caracteres que no sean del código ASCII.

Los mensajes de error devueltos por el sistema son:

- `Entrada Incorrecta`: cuando se escribió algo que no fue reconocido como comando valido (simbolos no reconocidos, alias con caracteres no aceptados, etc.).
- `Conjunto inexistente`: cuando todo el formato es valido, pero alguno de los conjuntos a operar no existe.

6 Desarrollo y complicaciones

La principal complicación de este trabajo fue pensar con que estructura sería mas eficiente, en un principio habia pensado en un AVL de intervalos, pero esta era bastante difícil de implementar, y luego de pensarla varios dias me di cuenta que como tenia que crear arboles nuevo por las operaciones y no modificiar los existentes, el proceso de incersión de los nodos se hacia muy lento. Entonces pense en implementar los conjuntos como una lista de intervalos ordenados que no se colapsen unos con otros.
Una decisión que considero importante que tome fue la implementación de la resta, ya que no pensaba hacerla utilizando las demas operaciones, pero me di cuenta que la mejora era muy pequeña y que no aprovechaba nada de la teoria de conjuntos. Por lo que decidí finalmente implementar una resta como la intersección con el complemento.
Luego de pensar como implementaría todo con esta estructura, decidí quedarme con las listas. La mayor dificultad que tuve fue como hacer la definición de un conjunto por extensión, pero decidí aprovechar lo ya realizado en trabajos anteriores y optar por insertar todos los elementos, ordenarlos con Merge Sort y luego unificar los elementos si era posible.
Y definitivamente la parte que me tomo mas tiempo fue el intérprete, porque no quería hacerlo muy estricto con la entrada, pero tampoco quería que rompiera o que permitiera cosas que no debía. Luego de muchas horas de desarrollo llegue al estado final al cual no le encontré fallas.
Realice todos los testeos que se me ocurrieron y también los brindados por la catedra y el funcionamiento del programa es correcto. También utilice Valgrind para fijarme los leaks de memoria, y los que había fueron arreglados.

7 Bibliografía

https://es.wikipedia.org/wiki/Tabla_hash
https://cp-algorithms.com/string/string-hashing.html
http://www.cplusplus.com/reference/cstdlib/strtod/
https://stackoverflow.com/questions/12824134/undefined-reference-to-pow-in-c-despite-including-m