



UNR - FCEIA

Estructuras de Datos y Algoritmos I

# Trabajo Práctico Final - Conjuntos Numericos

*Poma Lucas*

# Trabajo Práctico Final
# Conjuntos Numericos

Poma Lucas 18/08/2020

## Introducción

El objetivo de este trabajo fue desarrollar un programa que permitiese ingresar conjuntos y realizar acciones y operaciones con ellos.
Para esto fueron utilizadas distintas estructuras, como tablas hash para almacenar los conjuntos, listas simplemente enlazadas para resolver las colisiones en la tabla y listas doblemente enlazadas para almacenar los elementos del conjunto.

## Compilado y ejecución del programa

Contamos con un archivo makefile, el cual permite compilar rápidamente el programa.
Para ello se utiliza el comando **make** en el terminal estando situado en el directorio principal del proyecto, generando así el ejecutable **interprete**.
Una vez terminada la ejecución se puede utilizar el comando **make clean** para eliminar los archivos compilados.

## Intérprete

### Mensajes de error

- **No se encontró ningún conjunto con ese nombre** Se muestra si ocurre un error buscando el conjunto asociado al alias en las operaciones de un conjunto.
- **No se encontró alguno de los conjuntos** Se muestra si ocurre un error buscando los conjuntos asociados a los alias en las operaciones de dos conjuntos.
- **Sintaxis incorrecta** Se muestra si el comando ingresado no es reconocido por el intérprete.
- **Sintaxis del conjunto incorrecta** Se muestra si se cometió un error al definir un conjunto, ya sea por extensión o compresión.
- **Operación incorrecta** Se muestra si se utiliza un carácter no reconocido como operación entre conjuntos.

### Formato alias

Se permiten los alias con caracteres especiales y diferenciando mayúsculas de minúsculas, hasta 1000 caracteres y sin espacios de por medio.

### Comandos

- **alias = {k0,k1,...,kn}** (se pueden colocar espacios entre comas), este comando creara ese conjunto de enteros bajo ese alias.
- **alias = {x : k0 <= x <= k1}** (se puede utilizar cualquier letra minúscula, mientras coincidan, y también se permite el uso de "<" para denotar el intervalo sin esos elementos), este comando creará ese intervalo bajo ese alias.
- **alias1 = alias2 | alias3** Realiza la unión entre esos dos conjuntos y guarda el resultado bajo el nombre de alias1.
- **alias1 = alias2 alias3** Realiza la intersección entre esos dos conjuntos y guarda el resultado bajo el nombre de alias1.
- **alias1 = alias2 - alias3** Realiza la resta entre esos dos conjuntos y guarda el resultado bajo el nombre de alias1.
- **alias1 = alias2** Realiza el complemento del conjunto y guarda el resultado bajo el nombre de alias1.
- **imprimir alias** Imprime en pantalla los elementos del conjunto bajo el nombre de alias en orden de menor a mayor.
- **salir** Cierra el programa liberando la memoria solicitada.

## Estructuras

- **Intervalo** Esta estructura representa los elementos del conjunto, ya que los números enteros son almacenados de esta manera.
- **DList** Esta fue la estructura seleccionada para representar los conjuntos, la cual almacena el alias asociado, y punteros al inicio y final del conjunto.
- **Tabla Hash** Esta estructura permite almacenar los conjuntos y acceder a ellos. Consiste en un array con punteros a SList permitiendo añadir conjuntos de manera eficiente en caso de colisiones.
- **SList** Esta estructura se utiliza para resolver las colisiones en la tabla hash, cada elemento de la tabla tiene un puntero a SList, donde sus elementos almacenan como dato un puntero a DList, es decir un conjunto.

```
1  typedef struct _DNodo {
2    void *dato;
3    struct _DNodo *ant;
4    struct _DNodo *sig;
5  } DNodo;
6
7  typedef struct {
8    DNodo *primero;
9    DNodo *ultimo;
10   char *nombre;
11 } DList;
12
13 typedef struct _Intervalo {
14   int inicio;
15   int final;
16 } Intervalo;
17
18 typedef struct _SNodo {
19   void *dato;
20   struct _SNodo *sig;
21 } SNodo;
```

## Dificultades encontradas

### Interprete

Una de las principales dificultades fue el encontrar un balance entre flexibilidad y funcionalidad a la hora de ingresar comandos en el interprete. Llegue a una forma de leer la entrada que permite al usuario cometer errores minimos a la hora del ingreso por teclado sin que estos afecten al funcionamiento del programa.

### Operaciones entre conjuntos

Otra dificultad encontrada fue a la hora de realizar operaciones entre conjuntos, sobretodo a la hora de la union, ya que era la operacion donde mas factores tenian que tenerse en cuenta. Fue resuelto analizando meticulosamente todos los posibles casos de uniones entre intervalos, y extendiendolos correctamente cuando fuese necesario.

### Tabla hash

Otro problema fue al pensar en como resolver las colisiones dentro de la tabla hash. Llegue a la conclusion de que utilizar listas simplementes enlazadas, iba a ser la forma mas adecuada de resolver el problema.

## Fuentes consultadas

https://www.tutorialspoint.com/data_structures_algorithms/hash_table_program_in_c.htm#:~:text=Hash%20Table%20is%20a%20data,its%20own%20unique%20index%20value.
https://www.geeksforgeeks.org/data-structures/linked-list/singly-linked-list/
https://cp-algorithms.com/string/string-hashing.html
http://www.cplusplus.com/reference/cstdlib/strtol/
https://www.tutorialspoint.com/c_standard_library/limits_h.htm