

PRÁCTICA TEMAS 2 Y 3
Algoritmos y Estructuras de Datos I
Curso: 2025/26

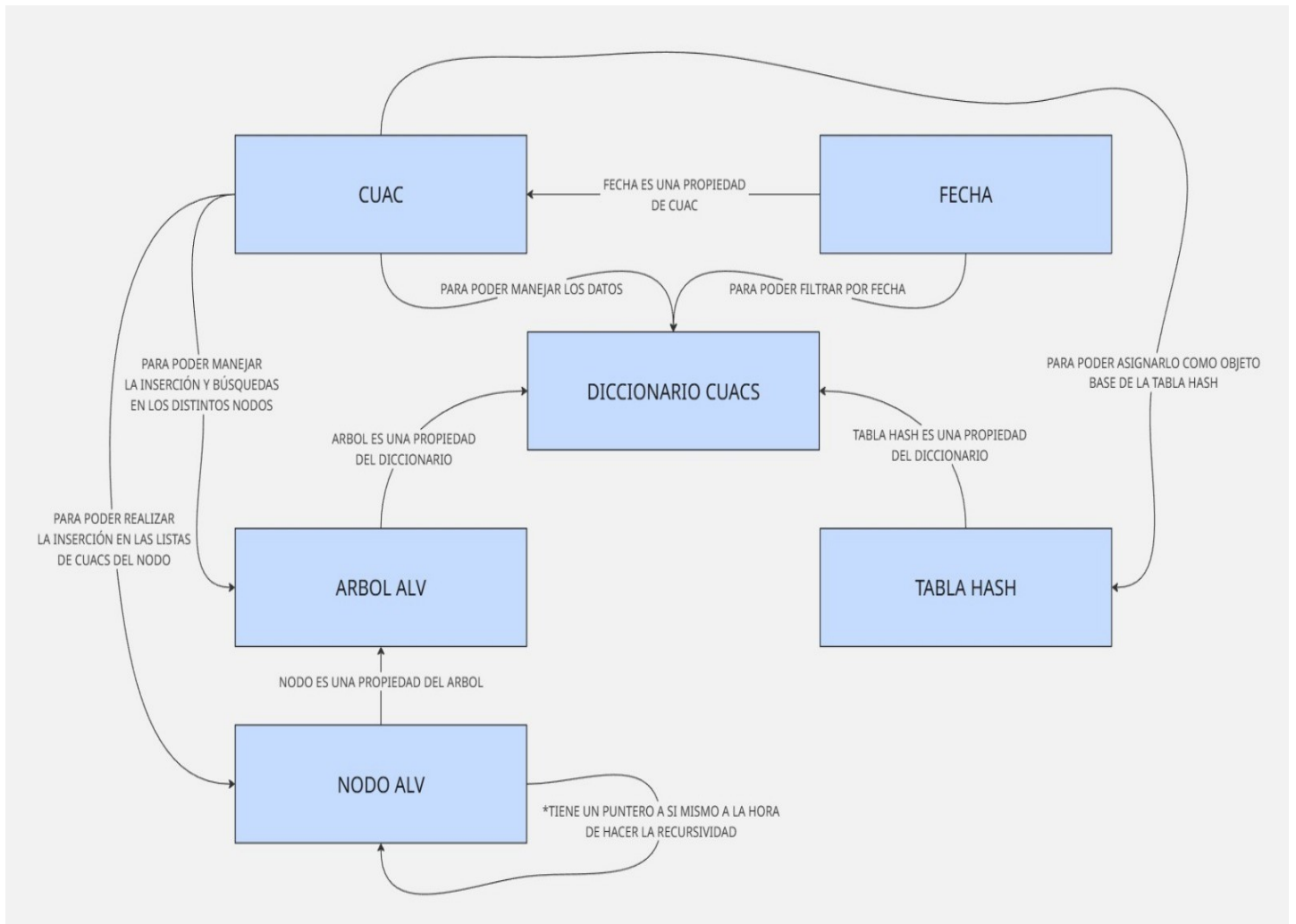
Autores:

Nombre completo	Subgrupo	DNI
Francisco Iñiguez Pérez	3.2	49308514X
Antonio Pujante Cuenca	3.2	49194300Z

1. Envíos a Mooshak:

Nº de problema	Nº de envío a Mooshak	Propietario de la cuenta del envío
001	320	Francisco Iñiguez Perez C50
002	562	Francisco Iñiguez Perez C50
003	1187	Francisco Iñiguez Perez C50
004	1212	Francisco Iñiguez Perez C50
005	1619	Francisco Iñiguez Perez C50
006	3410	Francisco Iñiguez Perez C50
200	3468	Francisco Iñiguez Perez C50
300	4244	Francisco Iñiguez Perez C50
301	6231	Francisco Iñiguez Perez C50
302	6233	Francisco Iñiguez Perez C50

2. Clases definidas y dependencias (no incluir el código solo nombres de clases y dependencias)



3. Módulos implementados, archivos de cada módulo (no incluir el código, solo nombres de módulos y sus dependencias)

- **Cuac:**
 - Archivos: *cuac.cpp* y *cuac.h*
 - Dependencias: *fecha.h*
- **DiccionarioCuacs:**
 - Archivos: *diccionarioCuacs.cpp* y *diccionarioCuacs.h*
 - Dependencias: *cuac.h*, *fecha.h*, *tablaHash.h* y *arbolALV.h*
- **TablaHash:**
 - Archivos: *tablaHash.cpp* y *tablaHash.h*
 - Dependencias: *cuac.h*
- **ArbolAVL:**
 - Archivos: *arbolALV.cpp* y *arbolALV.h*
 - Dependencias: *cuac.h* y *nodoALV.h*
- **NodoAVL:**
 - Archivos: *nodoALV.cpp* y *nodoALV.h*
 - Dependencia: *cuac.h*
- **Fecha:**
 - Archivos: *fecha.cpp* y *fecha.h*
 - Dependencias: No tiene dependencias
- **Main:**
 - Archivos: *main.cpp*
 - Dependencias: *cuac.h*, *diccionarioCuacs.h* y *fecha.h*

4. Tabla de dispersión utilizada. Explicación y justificación de las decisiones tomadas.

Durante esta práctica hemos utilizado de forma definitiva una tabla de dispersión abierta de tamaño fijo $M=101$ aunque realizamos distintas pruebas con tablas de dispersión abiertas de tamaño dinámico.

Esta tabla es un array, en el que en cada posición encontramos una lista, y cada lista almacena pares. Cada par almacena por un lado la clave, que es el nombre de usuario, y por otro lado almacena el valor, que son una lista de cuacs pertenecientes al usuario.

Elegimos esta tabla en vez de otras, ya que fue la que mejor entendimos en las clases de teoría y por lo tanto, es la que mas nos gustaba. Así mismo para el almacenamiento de datos ilimitados teníamos claro que una tabla de dispersión cerrada no iba a ser lo suficientemente eficiente como para sernos útil a no ser que dispusiésemos de un espacio de almacenamiento de memoria muy limitada.

La función de dispersión que hemos usado es la siguiente:

$$h = (h * 31 + \text{carácter}) \% M$$

Hemos considerado correcta esta función, ya que, reparte bastante bien, es rápida y al usar $M=101$, como es primo minimiza el numero de colisiones, ademas usamos de base el 31.

La idea es ir acumulado cada carácter del nombre, empiezas con $h=0$ y para cada carácter haces la función de dispersión, y el valor final es la posición del usuario en la tabla.

Y el tamaño fijo hemos considerado que $M=101$, ya que es primo, es un tamaño suficiente para el numero de usuarios y reduce las colisiones.

También pensamos en que utilizar una tabla de dispersión cerrada no seria mala idea, ya que funciona como un array, pero el tema de la redispersión y el crecimiento de la tabla, nos pareció que podía ser mas lioso y difícil, por lo que la descartamos.

```
antonio@Tonako:/mnt/c/Users/anton/OneDrive/Documentos/UMU/2º/AED/Cuacker$ time ./a.out < 200a.in > salida
real    0m8,698s
user    0m0,785s
sys     0m0,578s
```

Tiempo de la tabla de dispersión abierta con tamaños "fijos".

5. Árbol utilizado. Explicación y justificación de las decisiones tomadas.

Hemos utilizado un *árbol AVL*, este es un árbol binario de búsqueda que se mantiene siempre balanceado, para que la altura sea **$O(\log n)$** , lo cual es necesario para las operaciones de inserción y consulta.

Elegimos este árbol porque se balancea automáticamente, lo cual como hemos dicho antes garantiza inserción y consulta, el *AVL* permite recorrer los elementos desde el mas reciente hasta el mas antiguo y no es necesario recorrer todo el árbol lo cual lo hace mas eficiente, es decir, podemos descartar subárboles, ya sea de la parte izquierda o derecha.

En definitiva, pensamos que este seria el mejor árbol dado que los datos están mejor estructurados si queríamos realizar búsquedas por fechas (*last y date*).

Otros tipos de arboles como el *TRIE*, no eran una mala idea, pero parecían mas complejos y ademas los entendíamos menos a nivel teórico.

El modulo de arbol lo forman dos clases: *NodoALV* y *ArbolALV*.

La clase *NodoALV* representa un nodo del arbol, guarda la fecha como clave y guarda una lista de punteros a cuacs con esa fecha, donde los punteros son izquierda o derecha y implementa las rotaciones *RSI*, *RSD*, *RDI* y *RDD*.

Y la clase *ArbolALV* gestiona el árbol completo, mantiene un puntero raíz, inserta nuevos *CUACS* en el árbol creando nuevos nodos si la fecha no existe o insertando en la lista del nodo si coincide la fecha, actualiza alturas y ejecuta las rotaciones necesarias, implementa las operaciones de cuacker, etc.

Separar las clases así tiene ventajas ya que *NodoALV* es simple y contiene la estructura mínima, *ArbolALV* mantiene toda la lógica de inserciones, rotaciones y recorridos, permite modificar el árbol sin afectar al código y hace que el propio código sea mas fácil de mantener.

6. Variables globales utilizadas (en su caso)

En el *main* final unicamente usamos tres variables globales al inicio, que son:

- `DiccionarioCuacs dic;`
- `Cuac cuac;`
- `list<Cuac> lista;`

La variable *dic*, es la estructura principal del programa y encapsula la *Tabla Hash* para búsquedas por usuario y el *Árbol AVL* para búsquedas por fecha. La hemos declarado así porque evitar pasar referencias constantes del diccionario y porque el programa tiene un único diccionario.

La variable *lista* se utiliza para almacenar el resultado de *last*, *follow* y *date*, y la hacemos global porque permite que la función *imprimirListadoCuacs* pueda acceder al resultado sin necesidad de pasar parámetros.

Y la variable *cuac* no se utilizaba en las primeras versiones en las funciones de inserción de *cuacs* (*pcuac* y *mcuac*) pero nos hemos dado cuenta mientras realizábamos la documentación de que ya que cada operación crea su propio *cuac* local y ya no se usa.

7. Otras decisiones de diseño relevantes.

Otra decisión de diseño relevante es, donde definir el objeto **DiccionarioCuacs**.

Nosotros hemos decidido definirlo como una variable global, ya que simplifica el interprete de comandos y evita tener que pasar el diccionario por referencia a cada función.

También hemos tomado la decisión de agrupar en carpetas los ficheros de cada código para así tener una mejor visión y gestión de los archivos con un repositorio de trabajo mucho mas claro y eficiente. Esto ha implicado unos cambios necesarios en el fichero **Makefile** y, a la hora de realizar el tar, una modificación del comando.

Así mismo, en el fichero **main** hemos creado una función que, dada una lista que recibe por parámetro, la recorra escribiendo en el formato adecuado los **cuacs** y el número total al finalizar el recorrido. Esto lo hemos realizado para evitar repetir código en cada una de las funciones.

8. Resumen de tiempo dedicado a cada apartado.

- **Practica 1:** La primera práctica que recoge los ejercicios **001** y **002**, no nos llevo mucho tiempo, ya que era una simple conversión de numero a texto y de texto a numero. Por lo que tardamos alrededor de 45 minutos cada ejercicio.
- **Practica 2:** Para los ejercicios **003** y **004**, tardamos un poco mas, ya que empezamos a ver cosas nuevas como las clases y métodos, por lo tanto nos tomo alrededor de 2 horas cada ejercicio, aunque el 004 nos tomo mas tiempo ya que no funcionaba como queríamos.
- **Practica 3:** Esta práctica recoge los ejercicios **005** y **006**, ya nos costo varias horas, ya que comenzamos con los comandos del interprete (*follow*, *date*, *pcuac*, *mcuac*, *last*), pero sobre todo lo que mas nos costo fue implementar el diccionario con listas, ya que no habíamos visto nunca el tipo list y nos daba algún que otro error.
- **Practica 4:** Para esta practica, que recoge el ejercicio **200**, nos llevo alrededor de 6 horas, ya que empezaba a ser todo mas complejo y probamos tablas abierta con tamaño fijo y tablas cerradas, pero enseguida seguimos usando tabla abierta dado que los tiempos de ejecución de las otras tablas generaban un error de *Time Limit Exceded* ya que tardaban demasiado.
- **Practica 5:** En esta ultima práctica, que recoge los ejercicios **300**, **301** y **302**, fue la mas dura de todas y tardamos varios días en poder acabarlas, sobre todo el 301, ya que cuando hicimos el *diff*, no nos daba la salida que esperábamos en muchas ocasiones. Hubo una que incluso, sin ninguna diferencia, nos daba respuesta errónea y tuvimos que revisarlo con el profesor. El error estaba en el cuarto caso de prueba. La mayor complejidad aquí, aparte de elegir el mejor árbol, fue ordenar correctamente los elementos del listado con las mismas fechas a la hora de realizar el *date* y el *last*.

En resumen la practica de los temas 2 y 3, nos ha llevado bastante tiempo, ya que aunque los primeros ejercicios fueran mas normales y sencillos, los últimos, eran bastante teóricos y necesitabas comprender bastante bien la teoría para poder hacerlos. Ademas, necesitamos realizar bastantes intentos para que el juez online nos aceptara los ejercicios. En general, hemos aprendido bastante a implementar listas, tablas de dispersión y arboles a nivel practico y hemos mejorado bastante nuestra lógica de programación.