ALGORITMOS Y ESTRUCTURA DE DATOS I

PRÁCTICAS TEMAS 2 Y 3

NOMBRES: ABELWANCHENG JIANGWAN — NIE: X7465921Y

RUBEN SANZ LÓPEZ — DNI: 49274367H

GRUPO: 1.1

PROFESOR: EDUARDO TIGERAS GIL

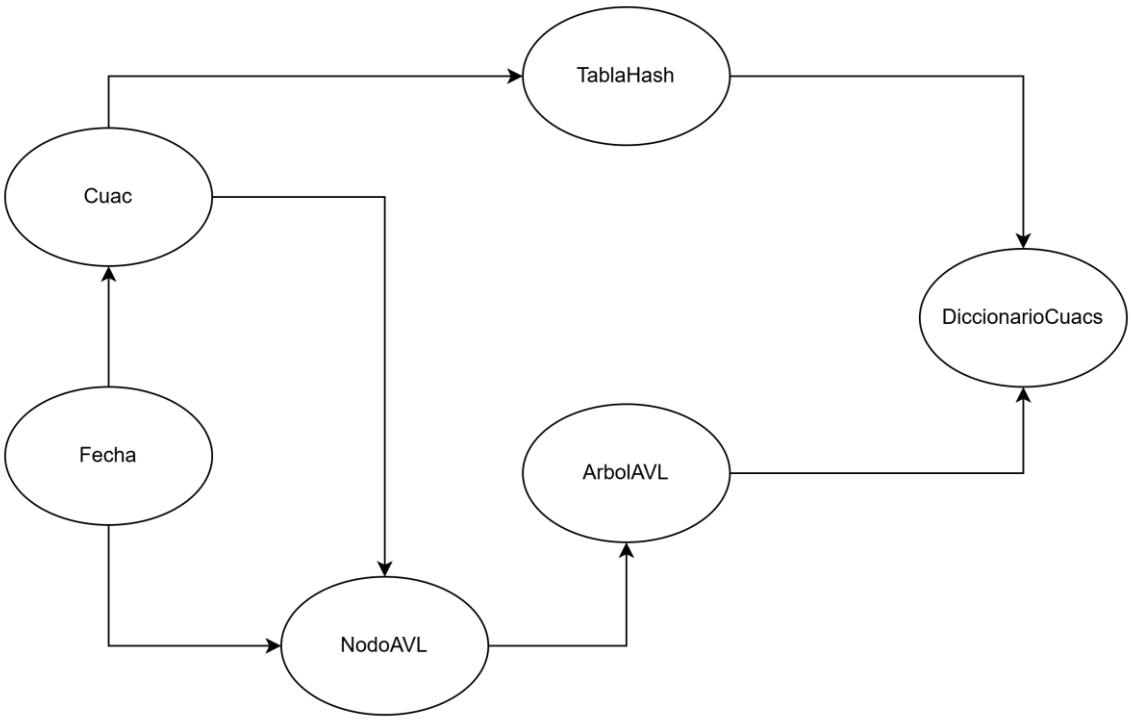
ÍNDICE

1. LISTADO DE PROBLEMAS RESUELTOS

Número del problema	Número de envio
001	87
002	706
003	1721
004	1725
005	3327
006	3344
200	4304
300	4366
301	5751
302	7103

2. ANÁLISIS Y DISEÑO DEL PROBLEMA

1.1 ¿Qué clases se han definido y qué relación existe entre ellas? Incluir una representación gráfica de las clases.



Fecha: Representa la fecha y la hora. Es utilizada por la clase Cuac

Cuac: Representa un mensaje, contiene la fecha, el usuario y el texto del mismo.

TablaHash: Contiene el tamaño, listas de cuacs y funcionalidad de la tabla de dispersión, así como la función de dispersión.

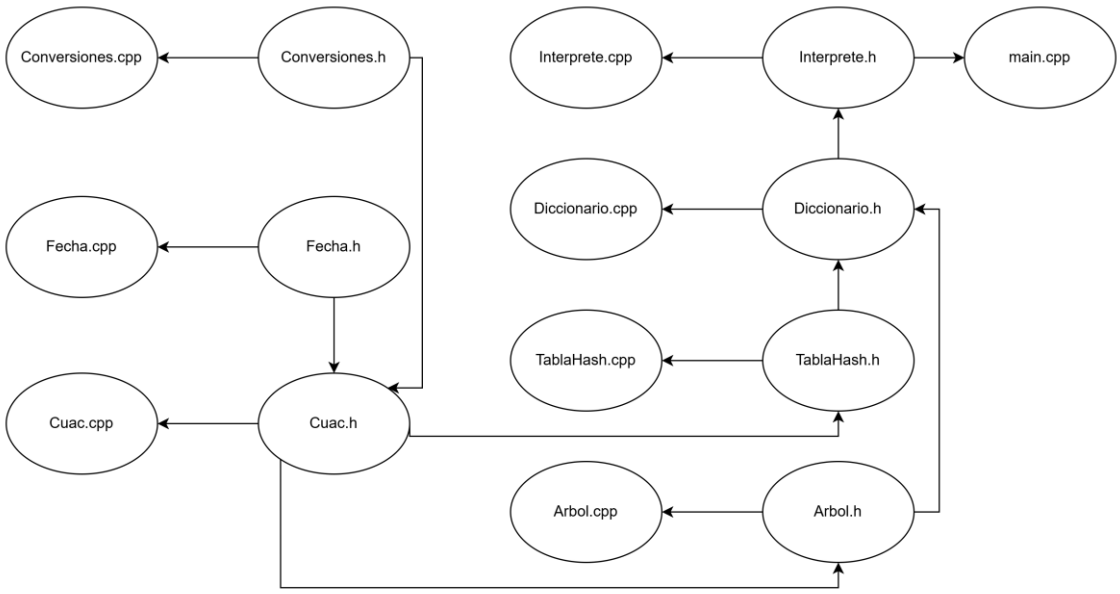
NodoAVL: Contiene una fecha y la lista de cuacs con dicha fecha, además de punteros a los hijos izquierdo y derecho y la altura para el balanceo

ArbolAVL: Gestiona la estructura del arbol, contiene un puntero a la raíz . Contiene un puntero a la raíz (NodoAVL*) y métodos para insertar, rotar y consultar. Tiene una relación de amistad con NodoAVL.

DIccionarioCuacs: Encapsula todo el sistema, contiene una TablaHash y un ArbolAVL y procesa las instrucciones del interprete con estos

1.2 ¿Qué módulos existen, qué contienen y cuál es la relación de uso entre ellos? Poner una representación gráfica de los ficheros y sus dependencias (includes).

(Conversiones): Conversiones de texto a número y de número a texto.
(Fecha): Define la clase Fecha.
(Cuac): Define la clase Cuac. Depende de Fecha y Conversiones.
(Interprete): Contiene los procesos de las funciones disponibles. Depende del Diccionario.
(Diccionario): Dirige cada proceso a la tabla hash o al árbol según cuál sea necesario. Depende por tanto de TablaHash y de Arbol
(TablaHash): Define la clase TablaHash. Depende Cuac.
(Arbol): Definimos las clases ArbolAVL y NodoAVL. Depende de Cuac.



1.3 ¿Contiene el makefile todas las dependencias existentes?

Si, ya que contiene todas las dependencias necesarias para compilar el proyecto correctamente.

1.4 ¿Qué tipo de tablas de dispersión se ha usado y por qué? Justificar la decisión.

Se ha usado una tabla de dispersión abierta en vez de cerrada ya que el tiempo de operación es proporcional al tamaño de las listas (crece linealmente), mientras que en la dispersión cerrada solo puede haber tantos elementos como cubetas y la probabilidad de colisión crece conforme más elementos haya, reduciendo la eficiencia.

En cuanto al tamaño, hemos decidido crear una tabla de tamaño fijo en vez de tamaño variable debido a que es más sencillo de implementar, evita la complejidad de la reestructuración dinámica de la tabla.

1.5 ¿Qué función de dispersión se ha usado? ¿Se han probado varias? Explicar y comparar los resultados de las distintas funciones probadas. Si se hace reestructuración de las tablas, explicar cómo y justificar la decisión.

Sí, hemos probado en concreto 3 funciones de dispersión diferentes. También para realizar esta prueba hemos redireccionado la salida a /dev/null ya que en nuestro caso los ordenadores que tenemos nos demoraban mucho tiempo en realizar las operaciones de entrada y salida del mooshak, agilizando por mucho este resultado y obteniendo con mejor precisión las funciones de dispersión realizadas.

Suma posicional djb2:

(0.795+0.825+0.753+0.768+0.887)/5= 0.8056 seg.

Método de suma con exponente:

(0.904+0.801+0.859+0.848+0.746)/5= 0.8316 seg.

Método de suma simple:

(1.152+ 1.264+ 1.109+ 0.950 + 0.776)/5= 1.0502 seg.

Por último queremos recalcar que la diferencia entre una función de dispersión y otra en nuestro caso puede ser insignificante debido a la potencia de los procesadores actuales.

1.6 ¿Cómo se libera la tabla de dispersión?

Dado que hemos utilizado una tabla de dispersión de tamaño fijo esta no requiere destructor.

1.7 ¿Qué tipo de árboles se han implementado y por qué?

Hemos implementado los árboles AVL ya que son ideales para la ordenación de mensajes cronológicamente.

Por otra parte no hemos usado los árboles B ya que este lenguaje son más eficientes para una base de datos y tampoco hemos implementado los árboles Tries ya que nos daría mucha complejidad a la hora de procesar las fechas desde nuestro punto de vista.

1.8 ¿Cómo es la definición del tipo árbol y del tipo nodo?

NodoAVL: contiene los punteros hijos (izq, der) y el factor de equilibrio(altura).

ArbolRaiz: contiene un puntero a la raíz del nodo

1.9 Si se han implementado árboles AVL, ¿cómo se hace el balanceo?

Mediante la aplicación el uso de las operaciones conocidas como rotaciones en ABB, cambiando algunos punteros y obtener un árbol que siga siendo un ABB. Tenemos 4 diferentes tipos de rotaciones en un ABB: RSD, RSI, RDD, RDI.

1.10 ¿Cómo se liberan los árboles?

Al finalizar el main se llama automáticamente al destructor del arbolAVL, que llama a su vez a los destructores de los nodosAVL derecha e izquierda recursivamente.

1.11 ¿Se usan variables globales en el programa final?

Para procesar los comandos del interprete se ha necesitado un diccionarioCuacs global.

3. LISTADO DEL CÓDIGO

Comandos:

make
time ./a.out <301a.in> salida
time ./a.out <301a.in> /dev/null
diff -y 301a.out salida

Código:

Main.cpp

```
#include "Interprete.h"
#include <iostream>

using namespace std;

int main() {
   string comando;
   while (cin >> comando && comando!="exit")
      Interprete(comando);
}
```

Interprete.h

```
#ifndef INTERPRETE_H
#define INTERPRETE_H

#include "Diccionario.h"

#include <string>
using namespace std;
```

```
void procesar_pcuac();
void procesar_mcuac();
void procesar_last();
void procesar_follow();
void procesar_date();
void procesar_tag();
void Interprete(string comando);

#endif
```

Interprete.cpp

```
#include "Interprete.h" // interprete

#include <iostream>
using namespace std;


// Hemos tenido que rehacer el ejercicio 5 ya que a la hora de compilarlo con el make nos daba error
// por lo que aquí está hecho de otra forma para poder compilarlo y ejecutar el programa.

// declaramos el diccionario global para poder usarlo en las funciones
DiccionarioCuacs diccionario;

// definimos las funciones de procesar cada comando
void procesar_pcuac(){
    Cuac nuevo; // creamos un cuac nuevo temporal
    nuevo.leer_pcuac(); // leemos el pcuac con todos sus parametors
    diccionario.insertar(nuevo); // insertamos el cuac en la lista ordenada del diciocnario
    cout << diccionario.numElem() << " cuac" << endl; // imprimimos el cuac
}

void procesar_mcuac(){
    Cuac nuevo;
    nuevo.leer_mcuac();
    diccionario.insertar(nuevo);
    cout << diccionario.numElem() << " cuac" << endl;
}

void procesar_last() {
    int n;
    cin >> n;
    diccionario.last(n);
```

```
}

void procesar_follow(){
    string usuario;
    cin >> usuario;
    diccionario.follow(usuario);
}

void procesar_date(){
    Fecha f1, f2;
    f1.leer();
    f2.leer();
    diccionario.date(f1, f2);
}

void procesar_tag(){
    string tag;
    cin >> tag;
    cout << "tag " << tag << endl;
    cout << "Total: 0 cuac" << endl;
}

void Interprete(string comando){
    if (comando == "pcuac") procesar_pcuac();
    else if (comando == "mcuac") procesar_mcuac();
    else if (comando == "last") procesar_last();
    else if (comando == "follow") procesar_follow();
    else if (comando == "date") procesar_date();
    else if (comando == "tag") procesar_tag();
}
```

Diccionario.h

```
#ifndef DICCIONARIO_CUACS_H
#define DICCIONARIO_CUACS_H

#include "TablaHash.h"
#include "Arbol.h"

// DEFINICION DEL DICCIONARIO DE CUACS AVL CON LA TABLA HASH DEL EJERCI-
CIO 200
class DiccionarioCuacs {
   private:
        TablaHash tabla;
        ArbolAVL arbol;
   public:
        DiccionarioCuacs() {}
        void insertar (Cuac nuevo);
        void follow (string nombre);
        int numElem ();
```

```
        void last (int N);
        void date (Fecha f1, Fecha f2);
};

#endif
```

Diccionario.cpp

```
#include <iostream>
#include "Diccionario.h"

using namespace std;

void DiccionarioCuacs::insertar(Cuac nuevo) {
    Cuac *ref = tabla.insertar(nuevo);
     arbol.insertar(ref);
}

void DiccionarioCuacs::follow(string nombre) {
    tabla.consultar(nombre);
}

int DiccionarioCuacs::numElem() {
    return tabla.numElem();
}

void DiccionarioCuacs::last(int N) {
    arbol.last(N);
}
void DiccionarioCuacs::date(Fecha f1, Fecha f2) {
    arbol.date(f1, f2);
}
```

Tablahash.h

```
#ifndef TABLAHASH_H
#define TABLAHASH_H

#include "Cuac.h" // Incluimos la clase Cuac
#include <list>
#include <string>
#include <vector>

using namespace std;

// DEFINICION DE LA TABLA HASH CON LISTAS
class TablaHash {
    private:
        static const int M = 5381; // Tamaño de la tabla (número primo)
        list<Cuac> tabla[M]; // Tabla de listas de cuacs
```

```
        int nElem; // Numero de elementos en la tabla
        int funcionHash(string clave);

    public:
        TablaHash();
        Cuac* insertar(Cuac nuevo); //REVISAR
        void consultar(string nombre);
        int numElem() { return nElem; }
};

#endif
```

Tablahash.cpp

```
#include "TablaHash.h"
#include <iostream>

using namespace std;

TablaHash::TablaHash() {
    nElem = 0;
}

//Función de dispersion1 djb2
int TablaHash::funcionHash(string clave) {
   unsigned int h = 5381; // Valor inicial y hacemos que no sea un numero
negativo
     for (int i = 0; i < clave.length(); i++) { // Recorremos cada ca-
rácter
         h = (h * 33) + clave[i]; // Actualizamos el hash con el carácter
actual
     }
     return h % M; // Retornamos el índice dentro del tamaño de la tabla
}

/*

// método de suma posicional con exponente
int TablaHash::funcionHash(string clave) {
   unsigned int h=0;
   int E=33;
     for (int i = 0; i < clave.length(); i++) {
         h = h*E + clave[i];
     }
     return h % M;
}

// metodo de suma simple
int TablaHash::funcionHash(string clave) {
```

```
    unsigned int h=0;
    for (int i = 0; i < clave.length(); i++) {
        h+=clave[i];
    }
    return h % M;
}

*/

// insertar un cuac en la tabla hash y devolver un puntero al cuac inser-
tado
Cuac* TablaHash::insertar(Cuac nuevo) {
    string usuario = nuevo.getUsuario(); // Obtener el nombre de usuario
del cuac como indica en el enunciado
    int indice = funcionHash(usuario);  // Calcular el índice usando la
función hash
    list<Cuac>::iterator itLista = tabla[indice].begin();
    // Buscamos la posicion correcta cronologicamente
    while (itLista != tabla[indice].end() && nuevo.es_anterior(*itLista))
{
        itLista++;
    }
    // Insertamos y obtenemos el iterador al nuevo elemento
    list<Cuac>::iterator itNuevo = tabla[indice].insert(itLista, nuevo);
    nElem++;
    return &(*itNuevo);
}

// consultar los cuacs de un usuario
void TablaHash::consultar(string nombre) {
    cout << "follow " << nombre << endl;
    int indice = funcionHash(nombre); // Calcular el índice usando la
función hash
    int count = 0;
    list<Cuac>::iterator it;
    for (it = tabla[indice].begin(); it != tabla[indice].end(); it++) {
        if (it->getUsuario() == nombre) {
            count++;
            cout << count << ". ";
            it->escribir(); // Imprimimos el cuac
        }
    }
    cout << "Total: " << count << " cuac" << endl;
}
```

Arbol.h

```
#ifndef ARBOL_H
#define ARBOL_H
#include <list>
#include <algorithm> // Para max
#include "Cuac.h" // Cuac

using namespace std;

class NodoAVL {
    friend class ArbolAVL; // El árbol puede acceder a los privados del
nodo
    private:
        Fecha fecha;           // Clave del árbol
        list<Cuac*> lista;     // Valor: Lista de punteros a los cuacs
con esa fecha
        NodoAVL *izq;          // Hijo izquierdo
        NodoAVL *der;          // Hijo derecho
        int altura;            // para el balanceo

    public:
        NodoAVL(Fecha f, Cuac* c);
        ~NodoAVL(); // Destructor para liberar memoria recursivamente
        void insertarOrdenado(Cuac* c);
};

// Clase Arbol AVL (Implementación de Last y Date)
class ArbolAVL {
    private:
        NodoAVL *raiz;

        // métodos internos recursivos
        int altura(NodoAVL* nodo); // Altura segura
        void insertar(NodoAVL* &nodo, Cuac* ref);  // Inserción recursiva

        // Rotaciones para balanceo
        void RSI(NodoAVL* &nodo); // Rotación Simple Izquierda II
        void RSD(NodoAVL* &nodo); // Rotación Simple Derecha DD
        void RDI(NodoAVL* &nodo); // Rotación Doble Izquierda ID
        void RDD(NodoAVL* &nodo); // Rotación Doble Derecha DI
        // Recorridos para operaciones
        void recorridoLast(NodoAVL* nodo, int &n, int &total); // Reco-
rrido inverso para last
        void recorridoDate(NodoAVL* nodo, Fecha f1, Fecha f2, int
&count); // Recorrido para date
    public:
        ArbolAVL();
        ~ArbolAVL();
```

```
        void insertar(Cuac* ref);
        void last(int N);
        void date(Fecha f1, Fecha f2);
};


#endif
```

Arbol.cpp

```
#include "Arbol.h"
#include <iostream>

using namespace std;
// IMPLEMETACION DE NODO AVL

// constructor
NodoAVL::NodoAVL(Fecha f, Cuac* c) {
    fecha = f;
    lista.push_back(c); // Insertamos el puntero en la lista
    izq = NULL;
    der = NULL;
    altura = 0;
}
// destructor
NodoAVL::~NodoAVL() {
   delete izq; // hijo izquierdo
   delete der; // hijo derecho
}

// tenemos que insertar el cuac en la lista de forma ordenada por texto y
usuario
void NodoAVL::insertarOrdenado(Cuac* c) {
    list<Cuac*>::iterator it = lista.begin();
    while (it != lista.end()) { // Recorremos la lista
        if (c->getTexto() < (*it)->getTexto()) { // Comparamos por texto
            lista.insert(it, c); // Insertamos antes de it
            return;
        }
        else if (c->getTexto() == (*it)->getTexto()) { // Si los textos
son iguales
            if (c->getUsuario() < (*it)->getUsuario()) {//En caso de que
los textos sean iguales comparamos por usuario
                lista.insert(it, c);  // Insertamos antes de it
                return;
            }
        }
        it++;
    }
```

```
    lista.push_back(c); // Si es el mayor, lo añadimos al final
}
// IMPLEMENTACION DE ARBOL AVL
// constructor
ArbolAVL::ArbolAVL() {
    raiz = NULL;
}
// Destructor
ArbolAVL::~ArbolAVL() {
    delete raiz;
}

// devolvemos la altura de un nodo
int ArbolAVL::altura(NodoAVL* nodo) {
    if (nodo == NULL) return -1;
    return nodo->altura;
}
// ROTACIONES PARA BALANCEO

// Rotación Simple Izquierda (Caso II)
void ArbolAVL::RSI(NodoAVL* &A) {
    NodoAVL* B = A->izq;
    A->izq = B->der; // subárbol derecho de B pasa a ser hijo izquierdo
de A
    B->der = A; // A pasa a ser hijo derecho de B
    // Actualizar alturas
    A->altura = 1 + max(altura(A->izq), altura(A->der));
    B->altura = 1 + max(altura(B->izq), A->altura);
    A = B; // A ahora apunta a la nueva raíz del subárbol
}

// Rotación Simple Derecha (Caso DD)
void ArbolAVL::RSD(NodoAVL* &A) {
    NodoAVL* B = A->der;
    A->der = B->izq;
    B->izq = A;
    // Actualizar alturas
    A->altura = 1 + max(altura(A->izq), altura(A->der));
    B->altura = 1 + max(altura(B->der), A->altura);
    A = B;
}
// Rotación Doble Izquierda (Caso ID)
void ArbolAVL::RDI(NodoAVL* &A) {
    RSD(A->izq); // Convertimos en caso II
    RSI(A);      // Resolvemos caso II
}

// Rotación Doble Derecha (Caso DI)
void ArbolAVL::RDD(NodoAVL* &A) {
```

```
    RSI(A->der); // Convertimos en caso DD
    RSD(A);      // Resolvemos caso DD
}

void ArbolAVL::insertar(Cuac* ref) {
    insertar(raiz, ref); // Llamada recursiva pasando por la raíz
}

// 3.3.3 OPERACION DE INSERCION EN AVL
// inserta un cuac en el árbol AVL
void ArbolAVL::insertar(NodoAVL* &A, Cuac* x) {
    Fecha f = x->getFecha(); // Obtenemos la fecha del cuac
    if (A == NULL) {
        A = new NodoAVL(f, x); // Creamos un nuevo nodo si A es NULL
    }
    // Si la fecha es anterior insertamos por la izquierda
    else if (f.esMenor(A->fecha)) {
        insertar(A->izq, x);
        if (altura(A->izq) - altura(A->der) > 1) {
            if (x->getFecha().esMenor(A->izq->fecha))
                RSI(A);
            else
                RDI(A);
        }
        else {
            A->altura = 1 + max(altura(A->izq), altura(A->der));
        }
    }
    // Si la fecha es posterior insertamos por la derecha
    else if (A->fecha.esMenor(f)){
        insertar(A->der, x);
        if (altura(A->der) - altura(A->izq) > 1) {
            if (!x->getFecha().esMenor(A->der->fecha))
                RSD(A);
            else
                RDD(A);
        }
        else {
            A->altura = 1 + max(altura(A->izq), altura(A->der));
        }
    }
    else {
        A->insertarOrdenado(x); // Misma fecha, insertamos en la lista
    }
}

// tenemos que buscar los cuacs mas nuevos con
void ArbolAVL::last(int N) {
    int count = 0;
```

```
    cout << "last " << N << endl;
    recorridoLast(raiz, N, count); // llamamos a la función recorridoLast
    cout << "Total: " << count << " cuac" << endl;
}

// comprobamos el funcionamiento del recorrido last
void ArbolAVL::recorridoLast(NodoAVL* nodo, int &n, int &count) { // pa-
rámetros nodo, n cuacs a imprimir, count cuacs impresos
    if (nodo == NULL || count >= n) return; // Caso base
    // Primero recorremos el subarbol derecho (mas nuevos)
    recorridoLast(nodo->der, n, count);
    if (count < n) { // Si no hemos impreso suficientes cuacs
        list<Cuac*>::iterator it;
        // recorremos la lista de cuacs
        for (it = nodo->lista.begin(); it != nodo->lista.end(); ++it) {
            if (count < n) { // Si no hemos impreso suficientes cuacs
                count++;
                cout << count << ". ";
                (*it)->escribir(); // Imprimimos el cuac
            }
        }
    }
    // Finalmente recorremos el subárbol izquierdo (mas viejos)
    recorridoLast(nodo->izq, n, count);
}


// creamos la función date y llamamos al recorrido date
void ArbolAVL::date(Fecha f1, Fecha f2) {
    int count = 0;
    cout << "date ";
    f1.escribir();
    cout << " ";
    f2.escribir();
    cout << endl;
    recorridoDate(raiz, f1, f2, count);
    cout << "Total: " << count << " cuac" << endl;
}
// hacemos el recorrido de date
//f1 anterior f2 posterior
void ArbolAVL::recorridoDate(NodoAVL* nodo, Fecha f1, Fecha f2, int
&count) {
    if (nodo == NULL) return; // caso base
    // Si el nodo es menor que f2, exploramos la rama derecha
    if (nodo->fecha.esMenor(f2)) {
        recorridoDate(nodo->der, f1, f2, count);
    }
    // si el nodo esta entre f1 y f2, lo procesamos
```

```
    if (!nodo->fecha.esMenor(f1) && !f2.esMenor(nodo->fecha)) { //
nodo >= f1 && nodo <= f2
        list<Cuac*>::iterator it;
        // Recorremos la lista de cuacs en el nodo
        for (it = nodo->lista.begin(); it != nodo->lista.end(); ++it) {
            count++;
            cout << count << ". ";
            (*it)->escribir();
        }
    }
    // Si el nodo es mayor que f1, exploramos la rama izquierda
    if (f1.esMenor(nodo->fecha)) {
        recorridoDate(nodo->izq, f1, f2, count);
    }
}
```

Cuac.h

```
#ifndef CUAC_H
#define CUAC_H
#include <string>
#include "Fecha.h" // Fecha
#include "Conversiones.h"

using namespace std;
//EJERCICIO 4
class Cuac {
  private:
    Fecha fecha;
    string usuario;
    string texto;
  public:
    string getUsuario() {return usuario;} // añadimos método get para po-
der usarlos en el ejercicio 6
    Fecha getFecha() {return fecha;} // añadimos método get para poder
usarlos en el ejercicio 6
    string getTexto() {return texto;} // casos en los que los textos
    bool leer_mcuac();
    bool leer_pcuac();
    void escribir();
    bool es_anterior(Cuac &otro);
};

#endif
```

Cuac.cpp

```
#include "Cuac.h"
#include <iostream>
using namespace std;
// Función que lee un mcuac
bool Cuac::leer_mcuac(){
```

```
    cin>>usuario;
    if (!fecha.leer()) return false;
    cin.ignore(); // ignora el salto de línea que va después de la fecha
    getline (cin, texto); // lee todo el texto hasta presionar enter
    return true;
}
// Función que lee un pcuac
bool Cuac::leer_pcuac(){
    cin>>usuario;
    if(!fecha.leer()) return false;
    int n;
    cin>>n;
    texto=convertir_num_text(n);
    return true;
}
// Función que escribe un cuac
void Cuac::escribir() {
    cout<<usuario<<' ';
    fecha.escribir();
    cout<<"\n   "; //nos ha llevado tiempo ya que en vez de 3 espacios
teníamos "\n\t" donde por culpa de eso nos daba error;
    cout<<texto<<endl;
}
// modificacion para realizar el ejercicio 6 debido a si las fechas,
texto, usuario son iguales
bool Cuac::es_anterior(Cuac &otro){
    // comparamos de más antiguo a más reciente
    if (!fecha.esIgual(otro.fecha)) {
        return fecha.esMenor(otro.fecha);
    }
    // en el caso de tengamos la misma fecha entonces comparamos por
texto
    if (texto != otro.texto) {
        return texto > otro.texto;
    }
    // en el caso de que también el texto sea igual entonces por usuario
    return usuario > otro.usuario;
}
```

Fecha.h

```
#ifndef FECHA_H
#define FECHA_H

using namespace std;

class Fecha {
    private:
        int dia, mes, ano;
        int hora, minuto, segundo;
```

```
    public:
        Fecha();
        bool leer();
        void escribir();
        bool esMenor(Fecha &fecha);
        bool esIgual(Fecha &otra);
};

#endif
```

Fecha.cpp

```
#include "Fecha.h"
#include <iostream>
using namespace std;

// inicializamos el constructor a 0;
Fecha::Fecha()
{
    dia=0; mes=0; ano=0; hora=0; minuto=0; segundo=0;
}
// comprobamos si la fecha esta en el formato correcto
bool Fecha::leer() {
   char barra1, barra2, puntos1, puntos2;
   // leemos la fecha con el formato indicado
   if (!(cin >> dia >> barra1 >> mes >> barra2 >> ano >> hora >> pun-
tos1 >> minuto >> puntos2 >> segundo)) {
    return false;
   } else
    return barra1 =='/' && barra2=='/' && puntos1==':' && puntos2==':';
}

// escribimos la fecha
void Fecha::escribir(){
    cout << dia << '/' << mes << '/' << ano << ' ';
   if (hora<10) { // nos ha demorado mucho tiempo en encontrar este fallo
        cout << '0';
    } cout << hora << ':';
    if (minuto<10) {
        cout << '0';
    } cout << minuto << ':';
    if (segundo<10) {
        cout << '0';
    } cout << segundo;
}

// Comprobamos si la fecha es menor
// primero comprobamos que no sean iguales y observamos que este es menor
bool Fecha::esMenor(Fecha &f) {
```

```
    if (ano != f.ano) return ano < f.ano; // f1<f2? si es que si entonces
true si no entonces pasamos al mes
    if (mes != f.mes) return mes < f.mes;
    if (dia != f.dia) return dia < f.dia;
    if (hora != f.hora) return hora < f.hora;
    if (minuto != f.minuto) return minuto < f.minuto;
    if (segundo != f.segundo) return segundo < f.segundo;
    else
        return false;
}

// Comprobamos si las fechas son iguales
bool Fecha::esIgual(Fecha &f){
    if (segundo==f.segundo && minuto==f.minuto && hora==f.hora &&
dia==f.dia && mes==f.mes && ano==f.ano){
        return true;
    } else
        return false;
}
```

Conversiones.h

```
#ifndef CONVERSIONES_H
#define CONVERSIONES_H

#include <string>

using namespace std;

string convertir_num_text(int n);
void convertir_text_num(string& s, int& n);

#endif
```

Conversiones.cpp

```
#include "Conversiones.h"

//EJERCICIO 001
// Función que convierte un numero en su texto correspondiente
string convertir_num_text (int n) {
 switch(n) {
        case 1: return "Afirmativo.";
        case 2: return "Negativo.";
        case 3: return "Estoy de viaje en el extranjero.";
        case 4: return "Muchas gracias a todos mis seguidores por vuestro
apoyo.";
        case 5: return "Enhorabuena, campeones!";
        case 6: return "Ver las novedades en mi pagina web.";
        case 7: return "Estad atentos a la gran exclusiva del siglo.";
        case 8: return "La inteligencia me persigue pero yo soy mas ra-
pido.";
```

```
        case 9: return "Si no puedes convencerlos, confundelos.";
        case 10: return "La politica es el arte de crear problemas.";
        case 11: return "Donde estan las llaves, matarile, rile,
rile...";
        case 12: return "Si no te gustan mis principios, puedo cambiarlos
por otros.";
        case 13: return "Un dia lei que fumar era malo y deje de fumar.";
        case 14: return "Yo si se lo que es trabajar duro, de verdad,
porque lo he visto por ahi.";
        case 15: return "Hay que trabajar ocho horas y dormir ocho horas,
pero no las mismas.";
        case 16: return "Mi vida no es tan glamurosa como mi pagina web
aparenta.";
        case 17: return "Todo tiempo pasado fue anterior.";
        case 18: return "El azucar no engorda... engorda el que se la
toma.";
        case 19: return "Solo los genios somos modestos.";
        case 20: return "Nadie sabe escribir tambien como yo.";
        case 21: return "Si le molesta el mas alla, pongase mas aca.";
        case 22: return "Me gustaria ser valiente. Mi dentista asegura
que no lo soy.";
        case 23: return "Si el dinero pudiera hablar, me diria adios.";
        case 24: return "Hoy me ha pasado una cosa tan increible que es
mentira.";
        case 25: return "Si no tienes nada que hacer, por favor no lo ha-
gas en clase.";
        case 26: return "Que nadie se vanaglorie de su justa y digna
raza, que pudo ser un melon y salio una calabaza.";
        case 27: return "Me despido hasta la proxima. Buen viaje!";
        case 28: return "Cualquiera se puede equivocar, inclusivo yo.";
        case 29: return "Estoy en Egipto. Nunca habia visto las piramides
tan solas.";
        case 30: return "El que quiera saber mas, que se vaya a Sala-
manca.";
        default: return "Error";
    }
}

// EJERCIO 002
// Función que convierte un texto en su número correspondiente
void convertir_text_num( string& s, int& n) {
    if (s == "Afirmativo.")n= 1;
    else if (s == "Negativo.")n= 2;
    else if (s == "Estoy de viaje en el extranjero.")n= 3;
    else if (s == "Muchas gracias a todos mis seguidores por vuestro
apoyo.")n= 4;
    else if (s == "Enhorabuena, campeones!")n= 5;
    else if (s == "Ver las novedades en mi pagina web.")n= 6;
    else if (s == "Estad atentos a la gran exclusiva del siglo.")n= 7;
```

```
    else if (s == "La inteligencia me persigue pero yo soy mas ra-
pido.")n= 8;
    else if (s == "Si no puedes convencerlos, confundelos.")n= 9;
    else if (s == "La politica es el arte de crear problemas.") n= 10;
    else if (s == "Donde estan las llaves, matarile, rile, rile...") n=
11;
    else if (s == "Si no te gustan mis principios, puedo cambiarlos por
otros.") n= 12;
    else if (s == "Un dia lei que fumar era malo y deje de fumar.") n=
13;
    else if (s == "Yo si se lo que es trabajar duro, de verdad, porque lo
he visto por ahi.") n= 14;
    else if (s == "Hay que trabajar ocho horas y dormir ocho horas, pero
no las mismas.") n= 15;
    else if (s == "Mi vida no es tan glamurosa como mi pagina web apa-
renta.") n= 16;
    else if (s == "Todo tiempo pasado fue anterior.") n= 17;
    else if (s == "El azucar no engorda... engorda el que se la toma.")
n= 18;
    else if (s == "Solo los genios somos modestos.") n= 19;
    else if (s == "Nadie sabe escribir tambien como yo.") n= 20;
    else if (s == "Si le molesta el mas alla, pongase mas aca.") n= 21;
    else if (s == "Me gustaria ser valiente. Mi dentista asegura que no
lo soy.") n= 22;
    else if (s == "Si el dinero pudiera hablar, me diria adios.") n= 23;
    else if (s == "Hoy me ha pasado una cosa tan increible que es men-
tira.") n= 24;
    else if (s == "Si no tienes nada que hacer, por favor no lo hagas en
clase.") n= 25;
    else if (s == "Que nadie se vanaglorie de su justa y digna raza, que
pudo ser un melon y salio una calabaza.") n= 26;
    else if (s == "Me despido hasta la proxima. Buen viaje!") n= 27;
    else if (s == "Cualquiera se puede equivocar, inclusivo yo.") n= 28;
    else if (s == "Estoy en Egipto. Nunca habia visto las piramides tan
solas.") n= 29;
    else if (s == "El que quiera saber mas, que se vaya a Salamanca.") n=
30;
    else s= "ERROR. Cadena no encontrada: [" + s + "]";
}
```

MAKEFILE

```
a.out: main.o Interprete.o Diccionario.o TablaHash.o Arbol.o Cuac.o Fe-
cha.o Conversiones.o
    g++ main.o Interprete.o Diccionario.o TablaHash.o Arbol.o Cuac.o Fe-
cha.o Conversiones.o -o a.out
# MAIN
main.o: main.cpp Interprete.h
    g++ -c main.cpp
# INTÉRPRETE
```

```
Interprete.o: Interprete.cpp Interprete.h Diccionario.h
    g++ -c Interprete.cpp
# DICCIONARIO
Diccionario.o: Diccionario.cpp Diccionario.h TablaHash.h Arbol.h
    g++ -c Diccionario.cpp
#TABLA HASH
TablaHash.o: TablaHash.cpp TablaHash.h Cuac.h
    g++ -c TablaHash.cpp
# ÁRBOLES AVL
Arbol.o: Arbol.cpp Arbol.h Cuac.h
    g++ -c Arbol.cpp
# CUACS
Cuac.o: Cuac.cpp Cuac.h Fecha.h Conversiones.h
    g++ -c Cuac.cpp
# FECHAS
Fecha.o: Fecha.cpp Fecha.h
    g++ -c Fecha.cpp
# CONVERSIONES
Conversiones.o: Conversiones.cpp Conversiones.h
    g++ -c Conversiones.cpp
```

4. INFORME DEL DESARROLLO

001 De número a texto

Hemos creado una función con un switch donde convierte un numero a texto.

002 De texto a número

Hemos creado una función donde implementamos un if else para convertir los textos a número.

003 Leyendo la fecha y la hora

Hemos creado funciones para leer y escribir fechas con el formato adecuado, por otra parte. Nos hemos encontrado dificultades a la hora de leer la fecha dentro de su formato, escribir la fecha correctamente y para comparar si una fecha era anterior a otra.

004 El tipo de datos cuac

Contiene los atributos de un mensaje cuac y es capaz de leer mcuacs y pcuacs, escribir mensajes y determinar si un cuac es anterior a otro. La principal complicación ha sido a la hora de obtener una respuesta aceptada debido al espaciado de la solución.

005 Intérprete de comandos

Contiene el intérprete y se encarga de procesar cada orden.
Lo hemos modificado tras el 006, ahora utiliza un diccionario de cuacs, que hemos decidido definir como una variable global por simplicidad.

006 Diccionario de cuacs con listas

Contiene el diccionario de cuacs, que ejecuta los distintos procesos del intérprete. El principal problema de este ejercicio ha sido a la hora de imprimir las fechas ya que no considerábamos el orden de esta. Al hallar dificultades a la hora de acceder a los atributos del cuac y siguiendo lo que hemos dado en la asignatura de programación orientada a objetos hemos decidido implementar getters.

200 Tablas de dispersión de cuacs

Hemos implementado la tabla hash con su función de dispersión principal, las funciones de prueba y la capacidad de insertar cuacs y consultar los mensajes de un determinado usuario. Se ha optado por implementar una tabla de dispersión abierta de tamaño fijo.

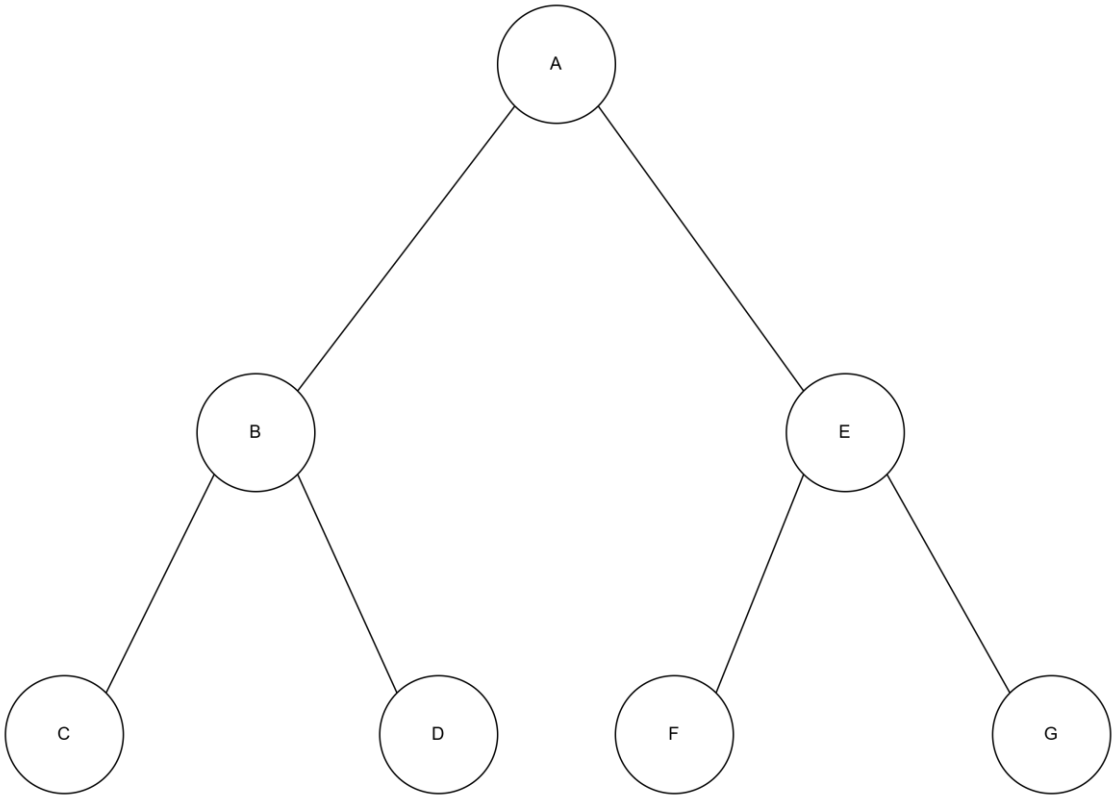
300 Nodoavl y arbolavl para last

Hemos decidido resolver este problema aplicando los árboles AVL, por lo que contiene las clases nodoAVL, y árbolAVL. Para las rotaciones de este último nos hemos ayudado del tema 3 de teoría. Hemos utilizado la ia para descubrir la librería que nos permite utilizar el max.

Por otra parte nos ha demorado mucho tiempo la realización de este ejercicio debido a que, mientras que nuestras comprobaciones nos daban una salida acertada, mooshak rechazaba la solución como incorrecta, por lo que al revisar el código nos dimos cuenta de que no habíamos implementado la inserción de texto y del usuario de forma ordenada como indicaba el enunciado del ejercicio 006.

301 Nodoavl y arbolavl para date

Hemos implementado la función date, donde recorremos el árbol comenzando por el nodo hijo derecha, continuando con el padre y siguiendo con el nodo hijo izquierda, comprobando en los tres casos si se encuentran entre las fechas solicitadas.



Suponiendo que f1 sea una fecha anterior a C y f2 una fecha posterior a G, la forma de recorrer el árbol sería en el orden: G->E->F->A->D->B->C.

302 El motor cuacker

Para mayor simplicidad del proyecto hemos decidido colocar el diccionario de cuacs y la tabla hash en módulos separados y por ello hemos tenido que modificar el makefile para que el proyecto compilara. Por otra parte también para su mayor legibilidad del código hemos decidido cambiar los nombres de dichos ficheros ya que hasta ahora era muy poco intuitivo. Todos los comandos funcionaban correctamente de forma conjunta, por lo que ha sido posible entregar el 302 sin dificultades.

5. CONCLUSIONES Y VALORACIONES PERSONALES

En este proyecto nos hemos visto en la necesidad de implementar diversas técnicas y herramientas como las tablas de dispersión o los árboles AVL, para lo que nos hemos ayudado en gran medida de las explicaciones y código mostrados en teoría, a aprender a aplicar la modularidad a un proyecto, uniendo sus diversas partes y a tomar decisiones cruciales en el diseño de un programa funcional: que tipo de tabla utilizar, que función de dispersión aplicar, que árbol utilizar, entre otras.

Proyecto: Tema 2 y 3 de AED				Fecha de inicio: 8/10/2025	
Programadores: Abelwanchen JiangWan Rubén Sanz López				Fecha de fin: 4/11/2025	
Día/Mes	Análisis	Diseño	Implement.	Validación	Total
8/10 (001)	10	10	15	0	35
15/10 (002)	5	5	10	50	70
27/10 (003)	20	20	30	240	310
27/10 (004)	10	10	20	60	100
28/10 (005)	5	5	10	10	30
11/11 (006)	20	40	140	100	300
24/11 (200)	35	40	350	10	435
26/11 (300)	30	40	210	100	380
1/12 (301)	20	30	40	120	210
1/12 (302)	15	20	120	0	155
TOTAL(min)	170	220	945	690	2025 min
Medias %	8.4%	10.86%	46.67%	34.07%	100%