# Algoritmos y Estructuras de Datos I

## Entrega 2

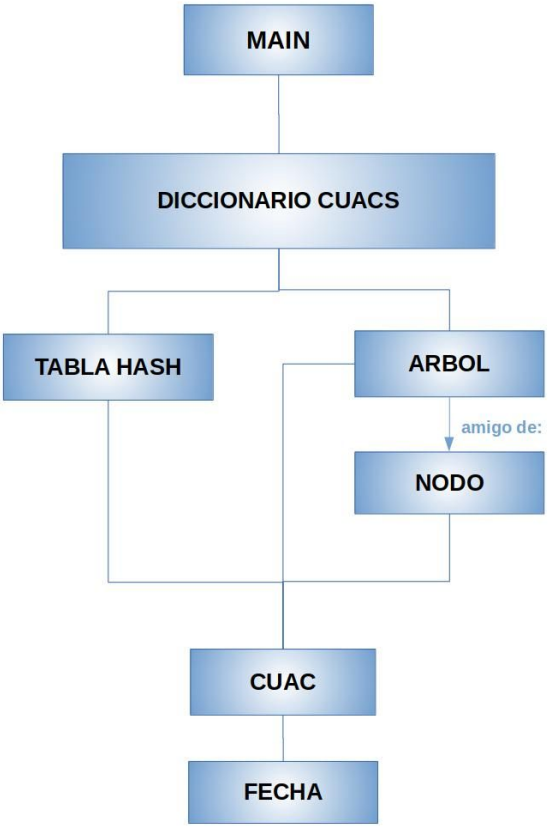
Actividad Temas 2 y 3. Cuacker.

Jose Antonio Lorencio Abril (joseantonio.lorencioa@um.es).
Pablo Miralles González (pablo.mirallesg@um.es).

## Análisis y diseño del problema.

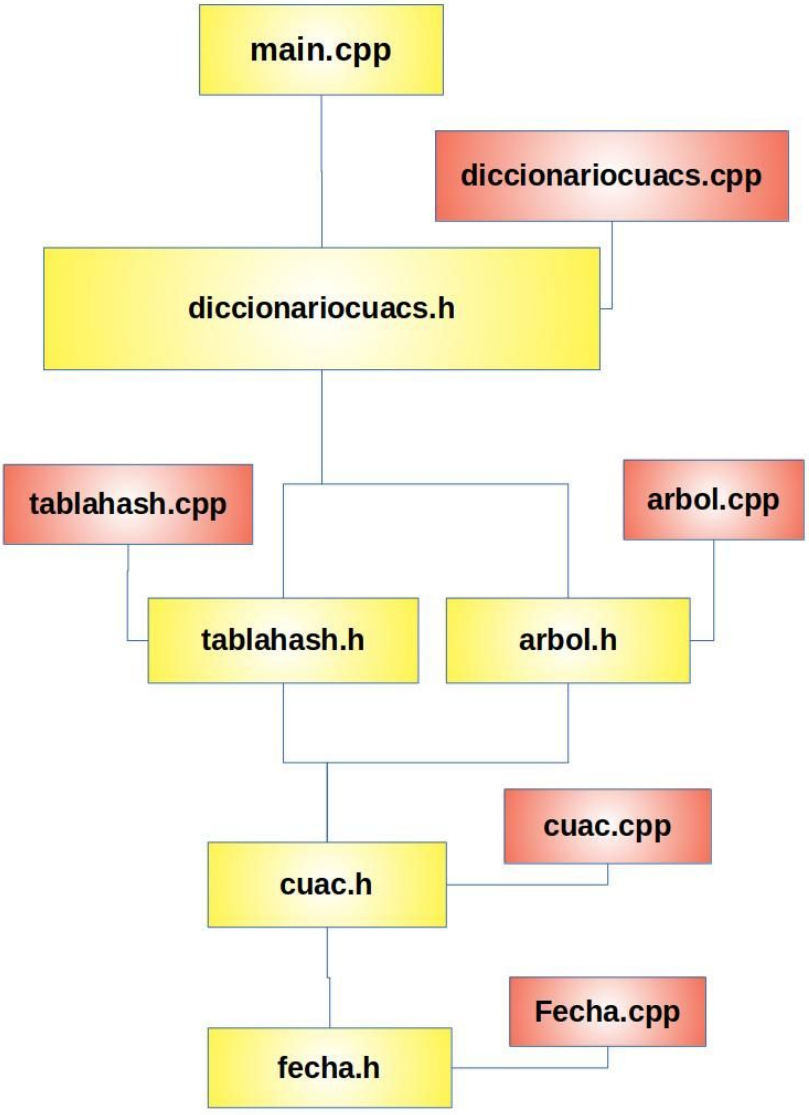
- ¿Qué clases se han definido y qué relación hay entre ellas? Incluir una representación gráfica de las clases.

    1. Clase Fecha. Clase para guardar una fecha. La utilizamos para guardar la fecha en la que un Cuac se publicó.
    2. Clase Cuac. Consta de la fecha de publicación, un usuario y un texto, que es el mensaje del Cuac. La fecha, como ya hemos dicho, utiliza la clase Fecha.
    3. Clase Arbol. Más concretamente, es un árbol de Cuacs, que mantiene el orden por fecha de los Cuacs, utilizando la clase Cuac y la clase Fecha para ello. Utiliza para ello una clase auxiliar: la clase Nodo.
    4. Clase Nodo. Clase auxiliar para los nodos del árbol. Se apoya en la clase Cuac. La clase Arbol la hemos implementado como clase amiga de Nodo.
    5. Clase TablaHash. Almacena los Cuacs de forma que el acceso a través del usuario sea rápido. Utiliza la clase Cuac y la clase Fecha.
    6. Clase DiccionarioCuacs. Mantiene un Arbol y una TablaHash con Cuacs, utilizando ambas clases, y aquellas clases que estas utilizan, que son Fecha y Cuac.



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

    1. Módulo fecha. Contiene exclusivamente la clase Fecha. Consta, al igual que el resto, de un fichero cabecera fecha.h y uno de implementacion fecha.cpp.
    2. Módulo cuac. Contiene la clase Cuac. En el fichero de cabecera cuac.h se incluye fecha.h, haciendo uso así del módulo fecha.
    3. Módulo arbol. Contiene la clase Arbol, incluyendo el fichero cuac.h en su fichero de cabecera, pudiendo así usar tanto el módulo cuac como el módulo fecha. También contiene la clase Nodo, en la cual se apoya el Arbol.
    4. Módulo tablahash. Contiene la clase TablaHash, e incluye el fichero cuac.h para poder acceder al módulo cuac.
    5. Módulo diccionariocuacs. Contiene la clase DiccionarioCuacs, e incluye los ficheros arbol.h y tablahash.h, para acceder a sus respectivas clases.



- ¿Contiene el makefile todas las dependencias existentes?

Sí, ya que si un módulo A depende de otro, B, los cambios de B le afectan, luego si se producen cambios en el módulo B se debe volver a compilar el A también. De hecho, aparte de contener las dependencias directas también contiene las indirectas, es decir, si A depende de B y B depende de C, en el makefile la compilación de A depende tanto de cambios en B como de cambios en C.

- ¿Qué tipo de tabla de dispersión has usado y por qué? Justificar la decisión.

Hemos utilizado dispersión abierta, ya que el enunciado decía que íbamos a recibir una gran cantidad de Cuacs. Por supuesto, las operaciones en dispersión abierta siempre son más eficientes, y la implementación es relativamente sencilla. Por estos motivos vimos muy claro que había que utilizar dispersión abierta.

- ¿Qué función de dispersión se ha usado? ¿Se han probado varias?

Adjuntamos una tabla con distintos resultados:

| Función | Tiempo de ejecución |
|---|---|
| Suma posicional (multiplicando cada carácter por su posición) | 0,34 segundos de media |
| Método folding | 0.38 segundos de media |
| Método de extracción | 0.37 segundos de media |
| Suma posicional (multiplicando por una base) | 0.36 segundo de media |

Basándonos en estos resultados, hemos escogido la primera función. Estos resultados parecen poco significativos, pero algo de mejora sí hay. Cabe decir que estos tiempos de ejecución han llegado a variar bastante, por lo que utilizamos una gran cantidad de casos de prueba e hicimos la media para llegar a esos tiempos.

- Si se hace reestructuración de las tablas, explicar cómo y justificar la decisión.

Sí utilizamos reestructuración. Cuando el número de usuarios que tienen un cuac almacenado supera el doble del número de cubetas, llamamos a la función de reestructuración. Esta implementación basada en el número de usuarios en lugar del número de Cuacs se debe a que cada cubeta es una lista de punteros a pares de string (nombre de usuario) y lista de Cuacs. Esta lista contiene los Cuacs del respectivo usuario. Lo que insertamos en la nueva tabla es el par <usuario, lista>, lo cual es mucho más eficiente que mover cada cuac. Tenemos entonces que la ocupación depende del número de usuarios y no del número de elementos.

La razón para usar reestructuración es que no perjudica en nada la implementación, y mejora la eficiencia de las operaciones para un número grande de Cuacs. Además, teniendo en cuenta el enunciado de la práctica, se supone que debemos esperar varios millones de Cuacs, luego sin reestructuración necesitaríamos un número de de cubetas inicial enorme, desperdiciando mucha memoria.

- ¿Cómo se libera la tabla de dispersión?

La tabla está implementada como un array de listas de pares < usuario, lista <cuac> >.
Es por esta razón que la liberación de la memoria se realiza usando delete[] sobre el array, cuya memoria se reserva dinámicamente. Tras esta llamada, el delete[] llama a los destructores de cada uno de los elementos, en este caso de las listas de pares.
Una vez está liberada toda la memoria de sus elementos, se libera la memoria ocupada por el array.

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

Hemos implementado árboles AVL, ya que nos proporcionan una forma más rápida de acceso a los cuacs, ordenados por fechas, que un árbol binario de búsqueda no balanceado y, a su vez, una implementación considerablemente más sencilla que un árbol B.
Además, la eficiencia de un árbol B no es mucho mejor que la de un AVL, pues ambas búsquedas tienen un orden O(log n), si bien la base del logaritmo en los árboles B es mayor, esto no es una diferencia sustancialmente significativa para el problema que nos ocupa.

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

```
class Nodo{
    private:
        Nodo* derecho;
        Nodo* izquierdo;
        Cuac* elem;
        int altura;
    public:
        Nodo();
        ~Nodo();
    friend class Arbol;
};

class Arbol{
    private:
        Nodo* raiz;
    public:
        Arbol();
        ~Arbol();
        Nodo*& get_raiz(){
            return raiz;
        }
        void RSI(Nodo*& nodo);
        void RSD(Nodo*& nodo);
        void RDD(Nodo*& nodo);
        void RDI(Nodo*& nodo);
        int get_altura(Nodo* nodo);
        void insertar(Cuac *cuac, Nodo*& nodo);
        void last(int n,int &cont, Nodo* nodo);
        void date(int &cont,Nodo* nodo, Fecha &fecha1, Fecha &fecha2);
};

#endif
```

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

Hemos implementado las rotaciones vistas en clase, así como la operación de inserción de un nodo en un árbol, en la que efectuamos el análisis de casos en función de las alturas de los subárboles y la consecuente aplicación de las rotaciones necesarias para el balanceo.

- ¿ Cómo se liberan los árboles?

Para liberar un árbol, simplemente hacemos un delete de su nodo raíz, cuyo destructor liberará recursivamente la memoria de sus nodos hijos.

- ¿Se usan variables globales en el programa final?

Sí, dic, una variable de la clase diccionariocuacs. Esta variable es el diccionario de almacenamiento de los cuacs.

## Listado del código.

- **fecha.h**

```
#ifndef _FECHA
#define _FECHA

class Fecha
{
 private:
   int day, month, year, hour, minutes, seconds;

 public:
   Fecha();
   bool leer();
   void escribir();
   bool es_menor(Fecha &otra);
   bool es_igual(Fecha &otra);
};

#endif
```

- **fecha.cpp**

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

Fecha::Fecha()
{
	day = month = year = hour = minutes = seconds = 0;
}
bool Fecha::leer()
{
	char aux;
	cin >> day >> aux >> month >> aux >> year >> hour >> aux >> minutes >>
aux >> seconds;
	return true;
}
void Fecha::escribir()
{
	cout << day << "/" << month << "/" << year << " ";
	if (hour < 10)
```

```
        cout << "0";
        cout << hour << ":";
        if (minutes < 10)
        cout << "0";
        cout << minutes << ":";
        if (seconds < 10)
        cout << "0";
        cout << seconds;
}
bool Fecha::es_igual(Fecha &otra)
{
        return ((otra.day == day) && (month == otra.month) && (otra.year == year)
&& (otra.minutes == minutes) && (otra.seconds == seconds) && (otra.hour ==
hour));
}
bool Fecha::es_menor(Fecha &otra)
{
        return (year < otra.year) ? true : (year > otra.year) ? false : (month <
otra.month) ? true : (month > otra.month) ? false : (day < otra.day) ? true :
(day > otra.day) ? false : (hour < otra.hour) ? true : (hour > otra.hour) ?
false : (minutes < otra.minutes) ? true : (minutes > otra.minutes) ? false :
(seconds < otra.seconds) ? true : false;
}
```

- **cuac.h**

```
#ifndef _CUAC
#define _CUAC

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

class Cuac
{
  private:
        Fecha fecha;
        string usuario;
        string texto;

  public:
        bool leer_mcuac();
        bool leer_pcuac();
        void escribir();
        bool es_anterior(Cuac &otro);
        bool es_de_usuario(string nombre);
        Fecha get_fecha(){return fecha;}
        string get_nombre(){return usuario;}
};

#endif
```

- **cuac.cpp**

```
#include "cuac.h"
using namespace std;

const int MAX_TEXTOS = 30;
string arr[MAX_TEXTOS] = {"Afirmativo.", "Negativo.", "Estoy de viaje en el
extranjero.", "Muchas gracias a todos mis seguidores por vuestro apoyo.",
"Enhorabuena, campeones!", "Ver las novedades en mi pagina web.", "Estad atentos
a la gran exclusiva del siglo.", "La inteligencia me persigue pero yo soy mas
rapido.", "Si no puedes convencerlos, confundelos.", "La politica es el arte de
crear problemas.", "Donde estan las llaves, matarile, rile, rile...", "Si no te
gustan mis principios, puedo cambiarlos por otros.", "Un dia lei que fumar era
malo y deje de fumar.", "Yo si se lo que es trabajar duro, de verdad, porque lo
he visto por ahi.", "Hay que trabajar ocho horas y dormir ocho horas, pero no
las mismas.", "Mi vida no es tan glamurosa como mi pagina web aparenta.", "Todo
tiempo pasado fue anterior.", "El azucar no engorda... engorda el que se la
toma.", "Solo los genios somos modestos.", "Nadie sabe escribir tambien como
yo.", "Si le molesta el mas alla, pongase mas aca.", "Me gustaria ser valiente.
Mi dentista asegura que no lo soy.", "Si el dinero pudiera hablar, me diria
adios.", "Hoy me ha pasado una cosa tan increible que es mentira.", "Si no
tienes nada que hacer, por favor no lo hagas en clase.", "Que nadie se
vanaglorie de su justa y digna raza, que pudo ser un melon y salio una
calabaza.", "Me despido hasta la proxima. Buen viaje!", "Cualquiera se puede
equivocar, inclusivo yo.", "Estoy en Egipto. Nunca habia visto las piramides tan
solas.", "El que quiera saber mas, que se vaya a Salamanca."};

string convertir(int n)
{
    return arr[n - 1];
}

int convertirinv(string str)
{
    int i = 0;
    while (str.compare(arr[i]) && i < MAX_TEXTOS)
    {
        i++;
    }
    return i + 1;
}

bool Cuac::leer_mcuac()
{
    cin >> usuario;
    fecha.leer();
    cin.ignore();
    getline(cin, texto);
    return true;
}
bool Cuac::leer_pcuac()
```

```
{
    cin >> usuario;
    fecha.leer();
    int n;
    cin >> n;
    texto = convertir(n);
    return true;
}
void Cuac::escribir()
{
    cout << usuario << " ";
    fecha.escribir();
    cout << endl;
    cout << "    " << texto << endl;
}
bool Cuac::es_anterior(Cuac &otro)
{
    if (otro.fecha.es_menor(fecha))
        return true;
    else if (fecha.es_igual(otro.fecha))
    {
        if (!texto.compare(otro.texto))
        {
            return usuario.compare(otro.usuario) < 0;
        }
        return texto.compare(otro.texto) < 0;
    }
    else
        return false;
}
bool Cuac::es_de_usuario(string nombre){
    return nombre==usuario;
}
```

- **arbol.h**

```
#ifndef _ARBOL
#define _ARBOL
#include "cuac.h"

class Nodo{
       private:
       Nodo* derecho;
       Nodo* izquierdo;
       Cuac* elem;
       int altura;
       public:
       Nodo();
       ~Nodo();
       friend class Arbol;
};
```

```
class Arbol{
	private:
	Nodo* raiz;
	public:
	Arbol();
	~Arbol();
	Nodo*& get_raiz(){
		return raiz;
	}
	void RSI(Nodo*& nodo);
	void RSD(Nodo*& nodo);
	void RDD(Nodo*& nodo);
	void RDI(Nodo*& nodo);
	int get_altura(Nodo* nodo);
	void insertar(Cuac *cuac, Nodo*& nodo);
	void last(int n,int &cont, Nodo* nodo);
	void date(int &cont,Nodo* nodo, Fecha &fecha1, Fecha &fecha2);
};

#endif
```

- **arbol.cpp**

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

Nodo::Nodo()
{
	derecho = NULL;
	izquierdo = NULL;
}
Nodo::~Nodo()
{
	delete izquierdo;
	delete derecho;
}

void Arbol::RSI(Nodo *&nodo)
{
	Nodo *aux = nodo->izquierdo;
	nodo->izquierdo = nodo->izquierdo->derecho;
	aux->derecho = nodo;
	nodo = aux;
	nodo->derecho->altura = 1 + max(get_altura(nodo->derecho->izquierdo),
get_altura(nodo->derecho->izquierdo));
	nodo->altura = 1 + max(get_altura(nodo->derecho),
get_altura(nodo->izquierdo));
}
void Arbol::RSD(Nodo *&nodo)
```

```
{
	Nodo *aux = nodo->derecho;
	nodo->derecho = nodo->derecho->izquierdo;
	aux->izquierdo = nodo;
	nodo = aux;
	nodo->izquierdo->altura = 1 + max(get_altura(nodo->izquierdo->izquierdo),
get_altura(nodo->izquierdo->izquierdo));
	nodo->altura = 1 + max(get_altura(nodo->derecho),
get_altura(nodo->izquierdo));
}
void Arbol::RDD(Nodo *&nodo)
{
	RSI(nodo->derecho);
	RSD(nodo);
}
void Arbol::RDI(Nodo *&nodo)
{
	RSD(nodo->izquierdo);
	RSI(nodo);
}
int Arbol::get_altura(Nodo *nodo)
{
	if (nodo == NULL)
	return -1;
	return nodo->altura;
}

Arbol::Arbol()
{
	raiz = NULL;
}
Arbol::~Arbol()
{
	delete raiz;
}

void Arbol::insertar(Cuac *cuac, Nodo *&nodo)
{
	if (nodo == NULL)
	{
	nodo = new Nodo();
	nodo->elem = cuac;
	nodo->izquierdo = nodo->derecho = NULL;
	nodo->altura = 0;
	}
	else if (!cuac->es_anterior(*nodo->elem))
	{
	insertar(cuac, nodo->izquierdo);
	if (get_altura(nodo->izquierdo) - get_altura(nodo->derecho) > 1)
	{
```

```
                if (!cuac->es_anterior(*(nodo->izquierdo->elem)))
                RSI(nodo);
                else
                RDI(nodo);
        }
        else
        {
                if (get_altura(nodo->izquierdo) > get_altura(nodo->derecho))
                nodo->altura = nodo->izquierdo->altura + 1;
                else
                nodo->altura = nodo->derecho->altura + 1;
        }
        }
        else
        {
        insertar(cuac, nodo->derecho);
        if (get_altura(nodo->derecho) - get_altura(nodo->izquierdo) > 1)
        {
                if (!cuac->es_anterior(*(nodo->derecho->elem)))
                RDD(nodo);
                else
                RSD(nodo);
        }
        else
        {
                if (get_altura(nodo->derecho) > get_altura(nodo->izquierdo))
                nodo->altura = nodo->derecho->altura + 1;
                else
                nodo->altura = nodo->izquierdo->altura + 1;
        }
        }
}

void Arbol::last(int n, int &cont, Nodo *nodo)
{
        if (nodo != NULL)
        {
        if (nodo->derecho != NULL)
        {
                last(n, cont, nodo->derecho);
        }
        if (cont < n)
        {
                cout << ++cont << ". ";
                nodo->elem->escribir();
        }
        if (cont < n)
        {
                last(n, cont, nodo->izquierdo);
        }
```

```
	}
}
void Arbol::date(int &cont, Nodo *nodo, Fecha &fecha1, Fecha &fecha2)
{
	if (nodo != NULL)
	{

	if (nodo->elem->get_fecha().es_menor(fecha1))
	{
		date(cont, nodo->derecho, fecha1, fecha2);
	}
	else if (!nodo->elem->get_fecha().es_menor(fecha2) &&
!nodo->elem->get_fecha().es_igual(fecha2))
	{
		date(cont, nodo->izquierdo, fecha1, fecha2);
	}
	else
	{
		{
		date(cont, nodo->derecho, fecha1, fecha2);
		cout << ++cont << ". ";
		nodo->elem->escribir();
		date(cont, nodo->izquierdo, fecha1, fecha2);
		}
	}
	}
}
```

- **tablahash.h**

```
#ifndef _TABLA_HASH
#define _TABLA_HASH
#include "cuac.h"
#include <list>
#include <utility>

class TablaHash{
	private:
	int M;
	int nelem;
	int nusuarios;
	list<pair<string, list<Cuac> >* > *tabla;
	int hash_function(string nombre);
	public:
	TablaHash();
	~TablaHash();
	Cuac* insertar(Cuac nuevo);
	void consultar(string nombre);
	int numelem(void){return nelem;};
	void reestructurar(void);
};
```

```
#endif
```

- **tablahash.cpp**

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

TablaHash::TablaHash()
{
	M = 25000;
	nelem = 0;
	nusuarios = 0;
	tabla = new list<pair<string, list<Cuac> >* >[M]();
}
TablaHash::~TablaHash()
{
	delete[] tabla;
	tabla = NULL;
}
int TablaHash::hash_function(string nombre)
{
	int k = 0;
	for (int i = 0; i < nombre.length(); i++)
	{
	k += ((i + 1) * nombre[i]);
	}
	if (k < 0)
	k *= (-1);
	return k % M;
}

Cuac* TablaHash::insertar(Cuac nuevo)
{
	if(nusuarios >= 2*M) reestructurar();
	string usuario = nuevo.get_nombre();
	int k = hash_function(usuario);
	list<pair<string, list<Cuac> >* >::iterator it = tabla[k].begin();
	while ((it != tabla[k].end()) && ((*it)->first.compare(usuario) < 0))
	{
	++it;
	}
	if ((it == tabla[k].end()) || ((*it)->first.compare(usuario) > 0))
	{
	tabla[k].insert(it, new pair<string, list<Cuac> >(string(usuario),
list<Cuac>()));
	it--;
	nusuarios++;
	}
	list<Cuac>::iterator it2 = (*it)->second.begin();
	while (it2 != (*it)->second.end() && it2->es_anterior(nuevo))
	{
```

```
	++it2;
	}
	(*it)->second.insert(it2, nuevo);
	it2--;
	cout << ++nelem << " cuac" << endl;
	return &*it2;
}
void TablaHash::consultar(string usuario)
{
	int k = hash_function(usuario);
	int cont = 0;
	list<pair<string, list<Cuac> >* >::iterator it = tabla[k].begin();
	while ((it != tabla[k].end()) && ((*it)->first.compare(usuario) < 0))
	{
	++it;
	}
	if ((it != tabla[k].end()) && ((*it)->first.compare(usuario) == 0))
	{
	for (list<Cuac>::iterator it2 = (*it)->second.begin(); it2 !=
(*it)->second.end(); ++it2)
	{
		cout << ++cont << ". ";
		it2->Cuac::escribir();
	}
	}
	cout << "Total: " << cont << " cuac" << endl;
}

void TablaHash::reestructurar(void){

	list<pair<string, list<Cuac> >* > *tablaAux = new list<pair<string,
list<Cuac> >* >[2*M]();
	for(int i=0; i<M; i++){
	for(list<pair<string, list<Cuac> >* >::iterator it = tabla[i].begin(); it
!= tabla[i].end(); it++){
		string aux = (*it)->first;
		int k = hash_function(aux);
		list<pair<string, list<Cuac> >* >::iterator it2 =
tablaAux[k].begin();
		while ((it2 != tablaAux[k].end()) && ((*it2)->first.compare(aux) <
0))
		{
		++it2;
		}
		tablaAux[k].insert(it2, *it);
	}
	}
	M*=2;
	delete[] tabla;
	tabla = tablaAux;
```

```
}
```

- **diccionariocuacs.h**

```
#ifndef _DICCIONARIO_CUACS
#define _DICCIONARIO_CUACS
#include "tablahash.h"
#include "arbol.h"
using namespace std;

class DiccionarioCuacs{
	private:
	TablaHash tabla;
	Arbol arbol;
	public:
	void insertar(Cuac nuevo);
	void follow(string nombre);
	void last(int n);
	void date(Fecha fecha1, Fecha fecha2);
	int numElem(){
		return tabla.numelem();
	}
};

#endif
```

- **diccionariocuacs.cpp**

```
#include "diccionariocuacs.h"
using namespace std;

void DiccionarioCuacs::insertar(Cuac nuevo)
{
	Cuac *ref = tabla.TablaHash::insertar(nuevo);
	arbol.Arbol::insertar(ref, arbol.get_raiz());
}
void DiccionarioCuacs::follow(string nombre)
{
	tabla.TablaHash::consultar(nombre);
}
void DiccionarioCuacs::last(int n)
{
	int cont = 0;
	arbol.Arbol::last(n, cont, arbol.get_raiz());
	cout << "Total: " << cont << " cuac" << endl;
}
void DiccionarioCuacs::date(Fecha fecha1, Fecha fecha2)
{
	int cont = 0;
	arbol.Arbol::date(cont,arbol.get_raiz(), fecha1, fecha2);
	cout << "Total: " << cont << " cuac" << endl;
}
```

- **main.cpp**

```
#include <iostream>
#include <string.h>
#include <list>
#include "diccionariocuacs.h"
using namespace std;

DiccionarioCuacs dic;

Cuac actual;

void procesar_mcuac()
{
	actual.leer_mcuac();
	dic.DiccionarioCuacs::insertar(actual);
}
void procesar_pcuac()
{
	actual.leer_pcuac();
	dic.DiccionarioCuacs::insertar(actual);
}
void procesar_follow()
{
	string usuario;
	cin >> usuario;
	cout << "follow " << usuario << endl;
	dic.DiccionarioCuacs::follow(usuario);
}

void procesar_last()
{
	int N;
	cin >> N;
	cout << "last " << N << endl;
	dic.DiccionarioCuacs::last(N);
}

void procesar_date()
{
	Fecha finicial, ffinal;
	finicial.leer();
	ffinal.leer();
	cout << "date ";
	finicial.escribir();
	cout << " ";
	ffinal.escribir();
	cout << endl;
	dic.DiccionarioCuacs::date(finicial, ffinal);
}

void procesar_tag()
```

```
{
	string etiqueta;
	cin >> etiqueta;
	cout << "tag " << etiqueta << endl;
	cout << "1. ";
	actual.escribir();
	cout << "Total: 1 cuac" << endl;
}

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

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

## Informe de desarrollo

El trabajo ha sido llevado a cabo por los dos dos de forma conjunta, trabajando en clase, en casa de alguno de los dos o comunicándonos vía online.

| **Proyecto:** Cuacker | | | | **Fecha inicio:** 17/10/2018 | |
|---|---|---|---|---|---|
| **Programadores:** Lorencio Abril, Jose A<br>Miralles González, Pablo | | | | **Fecha fin:** 5/12/2018 | |
| **Dia/mes** | **Análisis** | **Diseño** | **Implementación** | **Validación** | **Total** |
| 17/10 | 5 | 10 | 15 | 8 | 38 |
| 18/10 | 5 | 5 | 25 | 10 | 45 |
| 24/10 | 10 | 15 | 35 | 20 | 80 |
| 26/10 | 12 | 10 | 20 | 5 | 47 |
| 31/10 | 10 | 20 | 25 | 10 | 65 |
| 1/11 | 5 | 12 | 20 | 5 | 42 |
| 5/11 | 15 | 10 | 30 | 15 | 70 |
| 7/11 | | | 15 | 5 | 20 |

| Día/mes | Análisis | Diseño | Implementación | Validación | Total |
|---|---|---|---|---|---|
| 9/11 | | 20 | 35 | 5 | 60 |
| 23/11 | 30 | 45 | 70 | 20 | 165 |
| 5/12 | | 15 | 40 | 10 | 65 |
| **Total (minutos)** | 92 | 162 | 330 | 113 | 697 |

## Conclusiones y valoraciones personales.

Este ha sido el proyecto que más hemos disfrutado hasta el momento. Hemos asentado robustamente los conceptos tratados en las clases de teoría y hemos vivido de primera mano el desarrollo de un programa de cierta complejidad.
Es la primera vez que realmente hemos tenido que discernir entre las distintas implementaciones posibles, intentando minimizar uso de memoria y de tiempo.
Es también la primera vez que no nos dan la comida masticada, sino cuchillo y tenedor, lo cual, aunque dificulte el trabajo, consideramos que resulta beneficioso para nuestro desarrollo como ingenieros informáticos.

En resumen, podemos asegurar que este proyecto marca un antes y un después en nuestra visión del grado en particular y de la informática en general. Ahora valoramos la verdadera importancia en la forma de almacenar y manejar la información.