AED I

BUSCADOR WEB

```cpp
if (autor1 == JOSÉ DANIEL MOYA MORENO)
    && (autor2 == ANDRÉS MARÍN PÉREZ) {
      grupo = 2.2;
      usuario_juez_online = B80;
      DNI_autor1 = 21065100K;
      DNI_autor2 = 23330945K;
 }
```

C++

UNIVERSIDAD DE MURCIA

ÍNDICE

1. Prueba de aceptación en Mooshak 3
2. Análisis y diseño del problema 4
3. Listado del código 14
4. Informe del desarrollo 29
5. Conclusiones y valoraciones personales 30

1- PRUEBA DE ACEPTACIÓN EN MOOSHAK

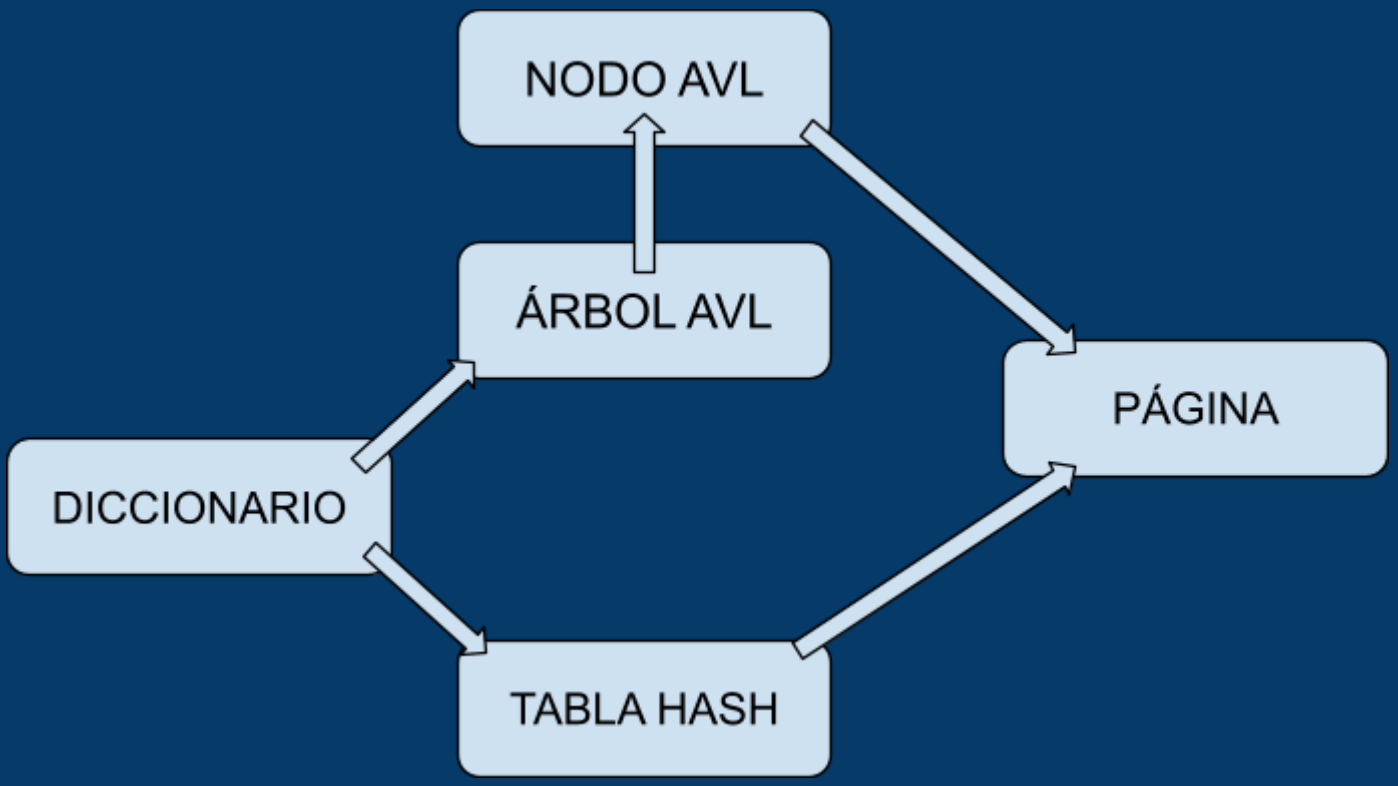
Todos los envíos han sido realizados desde la cuenta de mooshak : B80 (andres.m.p@um.es)

PROBLEMA	NÚMERO DE ENVÍO
001	4292
002	480
003	1095
004	1336
200	2273
300	4279

2- *ANÁLISIS Y DISEÑO DEL PROBLEMA*

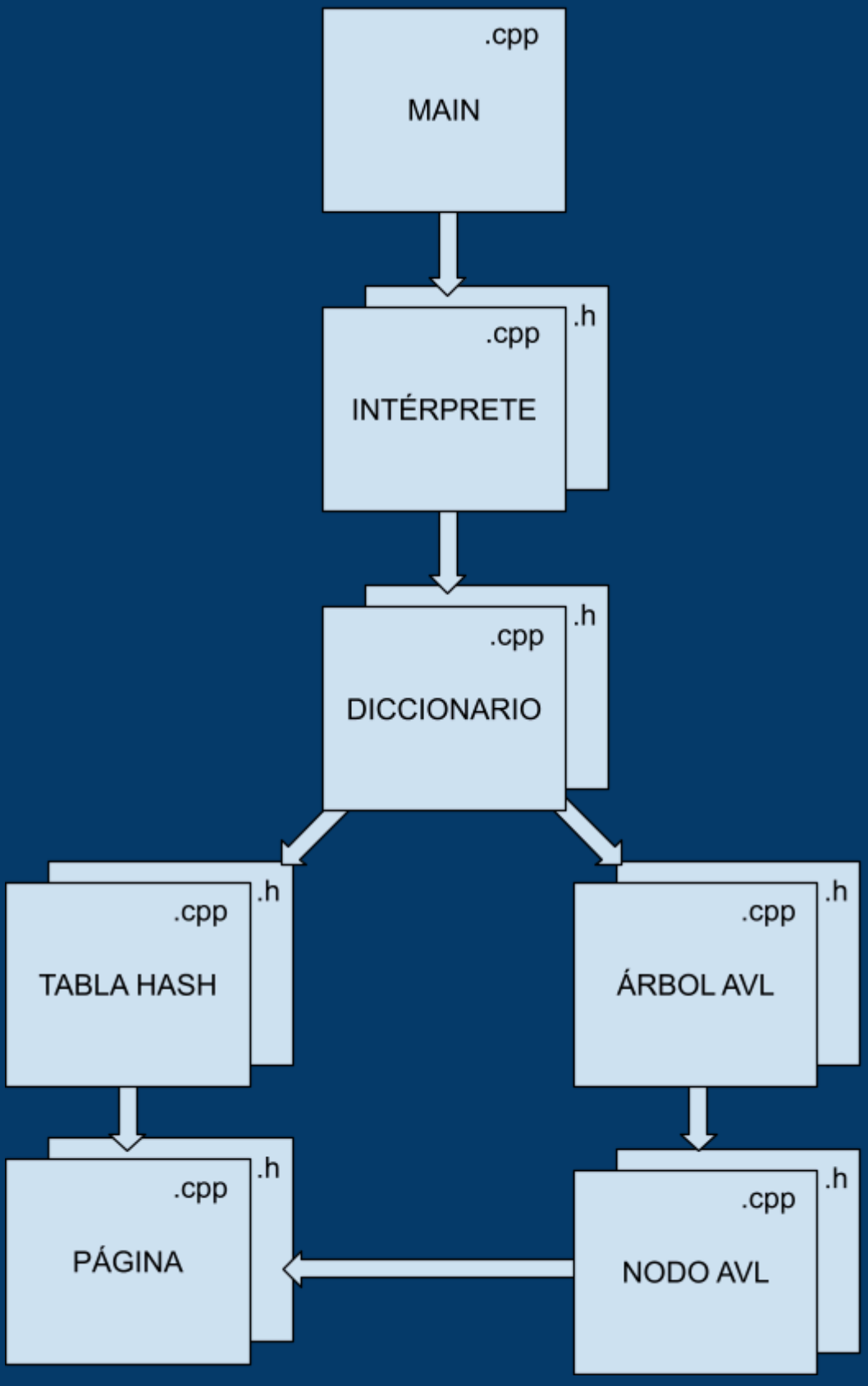
2.1) ¿Qué clases se han definido y qué relación existe entre ellas? Incluir una representación gráfica de las clases.

En el programa final hemos definido un total de 5 clases que se relacionan entre sí tal y como podemos ver en la representación:



- La clase **Diccionario** hace uso de las clases **ArbolAVL** y **TablaHash** para operaciones como la insercion de páginas, palabras, etc. en dichas estructuras de datos.
- La clase **ArbolAVL** hace uso de **NodoAVL**, ya que su atributo principal es un puntero a un nodo raíz. Para acceder a los atributos privados de NodoAVL, ambas clases son amigas.
- La clase **TablaHash** hace uso de la clase **Página**, puesto que se define como un array cuyas posiciones contienen listas enlazadas de Páginas.
- La clase **NodoAVL** hace uso de la clase **Página** puesto que cada NodoAVL tiene como atributo una lista de punteros a Páginas.

2.2) ¿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).



2.3) ¿Cómo se hace la normalización del texto?

El intérprete tiene definida una función normalizar() que recibe como parámetro la cadena a normalizar (string). Recorre todos los caracteres de dicha cadena y realiza los cambios pertinentes en función del caso en el que se encuentre:

Letra	Valor		Letra	Valor
á	A1		Á	81
é	A9		É	89
í	AD		Í	8D
ó	B3		Ó	93
ú	BA		Ú	9A
ü	BC		Ü	9C
ñ	B1		Ñ	91

1) Si es letra mayúscula la pasa a minúscula
2) Si es un carácter no ASCII, tendrá un primer byte que en hexadecimal es 0xC3, y un segundo byte (cadena[++i]) que depende de la letra

```
string normalizar(string cadena){
      string salida = "";
      for(unsigned int i = 0; i<cadena.length(); i++){
          if (cadena[i] >= 'A' && cadena[i] <= 'Z'){
            salida.push_back(tolower(cadena[i]));
          }else if(cadena[i] == char(0xC3)){ //si es caracter no ASCII
            switch(cadena[++i]){
                case char(0xA1): case char(0x81): salida.push_back('a'); break;
                case char(0xA9): case char(0x89): salida.push_back('e'); break;
                case char(0xAD): case char(0x8D): salida.push_back('i'); break;
                case char(0xB3): case char(0x93): salida.push_back('o'); break;
                case char(0xBA): case char(0x9A): case char(0xBC): case char(0x9C):
                       salida.push_back('u');
                       break;

                case char(0x91): salida.push_back((char) 0XC3);
                    salida.push_back((char) 0XB1);
                    break;

                default: salida.push_back((char) 0XC3);
                   salida.push_back(cadena[i]);
            }
          }else salida.push_back(cadena[i]);
      }
      return salida;
```

2.4) ¿Cómo es el Makefile? ¿Contiene todas las dependencias existentes?

Así ha quedado el Makefile tras incluir todas las dependencias existentes entre los módulos que hemos utilizado:

Makefile:

```
a.out: main.o Interprete.o Pagina.o Diccionario.o TablaHash.o NodoAVL.o ArbolAVL.o
  g++ main.o Interprete.o Pagina.o Diccionario.o TablaHash.o NodoAVL.o ArbolAVL.o

Interprete.o: Interprete.cpp Interprete.h Diccionario.h
  g++ -c Interprete.cpp

Diccionario.o: Diccionario.cpp Diccionario.h TablaHash.h ArbolAVL.h
  g++ -c Diccionario.cpp

TablaHash.o: TablaHash.cpp TablaHash.h Pagina.h
  g++ -c TablaHash.cpp

Pagina.o: Pagina.cpp Pagina.h
  g++ -c Pagina.cpp

NodoAVL.o: NodoAVL.cpp NodoAVL.h Pagina.h
  g++ -c NodoAVL.cpp

ArbolAVL.o: ArbolAVL.cpp ArbolAVL.h NodoAVL.h
  g++ -c ArbolAVL.cpp

main.o: main.cpp Interprete.h
  g++ -c main.cpp


clean:
  rm -f *.o a.out

.PHONY: clean
```

En la parte final incluimos ‘clean:’ para borrar los ficheros de código objeto (contienen el código compilado resultante de los archivos fuente de C++) para prevenir interferencias con los ficheros antiguos que hubiese en el directorio.

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

Hemos optado por utilizar tablas de dispersión con dispersión abierta por su manejo más eficiente de las colisiones. Cuando varias URLs generan el mismo valor de dispersión, esta alternativa utiliza listas enlazadas (cubetas). Esto supone una ventaja frente a la necesidad de tener una función de redispersión en dispersión abierta. De esta forma hemos podido almacenar y acceder a varias páginas en la misma posición sin afectar mucho al rendimiento. Esta solución mejora la eficiencia general del sistema al evitar la saturación de la tabla y mantener un buen tiempo de acceso.

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

```
unsigned int TablaHash::hash(string url){
  unsigned int valorHash = 0;
  int suma =17;
  for (unsigned int i = 0; i < url.length(); i++) {
     valorHash = (valorHash * suma ^ (unsigned char) url[i])% capacidad;
  }
  return valorHash % capacidad;
}
```

La función en cuestión calcula un valor a partir de los caracteres de la URL, combinándolos mediante multiplicaciones y operaciones XOR, para distribuir las URLs de manera uniforme en la tabla. Hemos optado por la operación XOR y no a la suma ya que analizamos que mezcla de manera más homogénea los bits y reduce las colisiones.

Hemos escogido esta función de dispersión ya que es la que mejores resultados nos ha dado de todas las que hemos probado. Además, hemos realizado diversas pruebas variando el valor de la variable ‘suma’ con diferentes números primos (31, 5…). También probamos alterando la capacidad de la tabla y 100001 fue con la que mejores resultados obtuvimos.

COMPARATIVA DE LAS DISTINTAS FUNCIONES

Para garantizar la fiabilidad y estabilidad de los resultados, repetimos 1000 veces estables la órden 'time ./a.out < 200a.in > salida'. Para automatizar estas pruebas programamos nuestro propio guión bash:

SCRIPT BASH -> REPETIR 'TIME' 100 VECES

```
#!/bin/bash

contador=0
cien=100
media=0
while test $contador != $cien
do
     (time ./a.out < 200a.in > salida) 2> tiempo.txt && (head -n -2
tiempo.txt | tail -n +2 > tiempo_tmp.txt && mv tiempo_tmp.txt
tiempo.txt) && grep -oP '\d+,\d+' tiempo.txt >> nuevo.txt
      let contador=$contador+1
done
```

FUNCIÓN 1 — 25.341/100 = 0,25341 seg

```
unsigned int TablaHash::hash(string url){
   unsigned int valorHash = 0;
   int suma = 5;
   for (int i = 0; i < url.length(); i++) {
     valorHash = (valorHash * suma + (unsigned char) url[i])% capacidad;
   }
   return valorHash % capacidad ;
}
     //CAPACIDAD INICIAL = 100001
```

FUNCIÓN 2 — 23.208/100 = 0,23208 seg

```
unsigned int TablaHash::hash(string url){
  unsigned int valorHash = 0;
  int suma = 5;
  for (int i = 0; i < url.length(); i = i+2) {
     valorHash = (valorHash * suma + (unsigned char) url[i])% capacidad;
  }
  return valorHash % capacidad;
}
     //CAPACIDAD INICIAL = 50003
```

FUNCIÓN 3 — 19,692/100 = 0,19692 seg (MEJOR OPCIÓN)

```
unsigned int TablaHash::hash(string url){
   unsigned int valorHash = 0;
   int suma = 17;
   for (int i = 0; i < url.length(); i++) {
     valorHash = (valorHash * suma ^ (unsigned char) url[i])% capacidad;
   }
   return valorHash % capacidad;
}
     //CAPACIDAD INICIAL = 100001
```

FUNCIÓN 4 — 24.769/100 = 0,24769 seg

```
unsigned int TablaHash::hash(string url){
   unsigned int valorHash = 0;
   int suma = 17;
   for (int i = 0; i < url.length(); i++) {
     valorHash = (valorHash * suma ^ (unsigned char) url[i])% capacidad;
   }
   return valorHash % capacidad;
}
     //CAPACIDAD INICIAL = 50001
```

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

Para evitar la sobrecarga en la tabla, antes de realizar una inserción verificamos que el factor de carga (n/B < 2) no supere 2. En ese caso se llamaría a una función de reestructuración, la cual hemos implementado para que aumente el tamaño al doble.

```
void TablaHash::redimensionar() {

   int capacidadAntigua = capacidad;
   capacidad *= 2; //doble capacidad

   TablaHash nuevaTabla(capacidad);
   for(int i = 0; i < capacidadAntigua; i++){
      list<Pagina>::iterator it = tablaHash[i].begin();
      while(it != tablaHash[i].end()){ //mientras queden paginas
          nuevaTabla.insertar(*it); //inserto en la nueva las paginas
               it++;
       }
    }
     delete[] tablaHash; // libero tabla antigua
     tablaHash = nuevaTabla.tablaHash; // reasigno nueva tabla
     nuevaTabla.tablaHash = NULL;
}
```

2.8) ¿Cómo se libera la tabla de dispersión?

La tabla de dispersión se libera en el destructor de la clase TablaHash, donde se utiliza delete[] tablaHash para liberar la memoria asignada dinámicamente a la tabla de listas enlazadas, ya que es el único atributo que emplea memoria dinámica.

9) ¿Qué tipo de árboles se han implementado y por qué?
Hemos utilizado árboles AVL por varios motivos:

- **Búsqueda rápida y ordenada:** Los árboles AVL son árboles binarios de búsqueda que garantizan un tiempo de búsqueda, inserción y eliminación de O(logn) en el peor caso.
- **Eficiencia para búsquedas ordenadas:** Es ideal para almacenar claves que requieren acceso secuencial ordenado, en este caso el las palabras del diccionario se insertan ordenadas alfabéticamente.
- **Balanceo:** Mediante rotaciones se mantiene siempre balanceado lo que mejora la eficiencia.

2.10) ¿Cómo es la definición de la clase árbol y del nodo?
Clase NODO
Un nodo está compuesto por una palabra (string) que actúa como clave para acceder a la lista de punteros a páginas donde aparece dicha palabra (clave). También almacena la altura (int) a partir del nodo (inicializada a 0 en el constructor) y dos punteros a nodos hijos inicializados en el constructor a NULL.

Clase Árbol
Un árbol tiene como atributos únicamente la raíz, que es un puntero a nodo y el número de nodos que contiene (palabras).

2.11) ¿Cómo se almacenan las páginas asociadas a cada palabra en el árbol?
En el árbol tenemos un nodo por cada palabra distinta y una lista de punteros a páginas en los que aparece dicha palabra. Esta lista se actualiza durante el proceso de inserción en el árbol. Si la palabra ya existe en el árbol, se verifica si la página correspondiente ya está en la lista; y en caso negativo se añade. Si la palabra es nueva, se crea un nodo, se inicializa la lista, y la página se incorpora como el primer elemento de la lista. Este enfoque garantiza un almacenamiento eficiente y ordenado. El primer criterio de ordenación es la relevancia, y en caso de empate se recurre a la URL.

2.12) Si se han implementado árboles AVL, ¿cómo se hace el balanceo?
El balanceo se realiza mediante la verificación de la diferencia de alturas entre los subárboles izquierdo y derecho de cada nodo después de una inserción. Si esta diferencia supera 1, se aplican rotaciones para restaurar el equilibrio.

2.13) ¿Cómo se liberan los árboles?
Hemos declarado el destructor de la clase árbol, que se encarga de liberar los nodos de manera recursiva. No es necesario llamarlo explícitamente ya que C++ lo llama automáticamente cuando el objeto del árbol sale de su ámbito o es eliminado.

2.14) ¿Se usan variables globales en el programa final?
No hemos considerado necesaria la utilización de ninguna variable global ya que todas las variables requeridas están dentro de las funciones o clases correspondientes. De esta forma evitamos errores de dependencias y tenemos un diseño más modular y limpio.

2.15) Si se han usado herramientas como ChatGPT, describir de forma detallada en qué partes y cómo se han usado.
Hemos recurrido al uso de ChatGPT porque lo consideramos una herramienta útil y beneficiosa, siempre que se emplee para resolver dudas concretas. Por ejemplo, le planteamos preguntas específicas sobre C++ (como: ¿qué devuelve la función insert de C++?), para aclarar conceptos rápidamente. Además, lo utilizamos para depurar errores en el código (por ejemplo, aprender a usar gdb) y para desarrollar un script que nos ayudase a medir los tiempos de ejecución con la orden time (véase el script en la página 9). También recurrimos a ChatGPT para incorporar una sentencia en el Makefile que eliminase los archivos de código objeto (detalles en la página 7). Finalmente, nos asistió en mejorar ligeramente la redacción de este documento. En conclusión, consideramos que es una herramienta extremadamente valiosa ya que permite aprender y resolver problemas de manera efectiva, sin la necesidad de estar horas buscando información.

3- LISTADO DEL CÓDIGO

(EL MAKEFILE SE MUESTRA EN EL APARTADO 2.4)

main.cpp

```
#include "Interprete.h"

int main(void){
    string comando;
    DicPaginas dic;

    while(cin>>comando){
        Interprete(comando, dic);
            if(comando == "s")
                  break;
    }
    return 0;
}
```

Interprete.h

```
#ifndef INTERPRETE_H
#define INTERPRETE_H
#include "Diccionario.h"

class DicPaginas;

string normalizar(string cadena);
void Interprete (string comando, DicPaginas &dic);
void INSERTAR (DicPaginas &dic);
void BUSCAR_URL (DicPaginas &dic);
void BUSCAR_PAL (DicPaginas &dic);
void BUSCAR_AND ();
void BUSCAR_OR ();
void AUTOCOMP();
void SALIR();

#endif
```

Interprete.cpp

```
#include "Interprete.h"

string normalizar(string cadena){
	string salida = "";
	for(unsigned int i = 0; i<cadena.length(); i++){
		if (cadena[i] >= 'A' && cadena[i] <= 'Z'){
			salida.push_back(tolower(cadena[i]));
		}else if(cadena[i] == char(0xC3)){
			switch(cadena[++i]){
				case char(0xA1): case char(0x81): salida.push_back('a'); break;
				case char(0xA9): case char(0x89): salida.push_back('e'); break;
				case char(0xAD): case char(0x8D): salida.push_back('i'); break;
				case char(0xB3): case char(0x93): salida.push_back('o'); break;
				case char(0xBA): case char(0x9A): case char(0xBC): case char(0x9C):
					salida.push_back('u');
					break;

				case char(0x91): salida.push_back((char) 0XC3);
						salida.push_back((char) 0XB1);
						break;

				default: salida.push_back((char) 0XC3);
					salida.push_back(cadena[i]);
			}

		}else salida.push_back(cadena[i]);
	}
	return salida;
}

void INSERTAR (DicPaginas &dic) {

	Pagina nuevaPag;
	nuevaPag.leer();
	dic.insertarPagina(nuevaPag);

	string palabra;
	int contadorDePalabras = 0;

	Pagina* ref =  dic.consultar(nuevaPag.getURL());
	do{
		cin >> palabra;
		if (normalizar(palabra) == "findepagina"){
			break;
```

```
            }
            dic.insertarPalabra(normalizar(palabra), ref);
            contadorDePalabras++;
      }
      while(true);

      nuevaPag.escribir(dic);
      cout << contadorDePalabras << " palabras"<<endl;
}

void BUSCAR_URL (DicPaginas &dic){
      string URL;
      cin.ignore(1);
      getline(cin, URL);

      int numeroDeURLs = 0;
      Pagina* encontrada= dic.consultar(URL);

      cout << "u " << URL << endl;
      if (encontrada != NULL){
            numeroDeURLs=1;
            encontrada->escribirNum(dic, numeroDeURLs);
      }
      cout << "Total: "<< numeroDeURLs <<" resultados" << endl;

}

void BUSCAR_PAL (DicPaginas &dic){
      string palabra;
      cin >> palabra;

      string normalizada = normalizar(palabra);
      list<Pagina*>lista = dic.buscarPalabra(normalizada);
      cout << "b " << normalizada << endl;

      int paginasAsociadas = 0;

      for(Pagina *pagina : lista) {
            paginasAsociadas++;
            pagina->escribirNum(dic, paginasAsociadas);
      }

      cout<< "Total: " << paginasAsociadas << " resultados" << endl;

}
```

```
void BUSCAR_AND (){

	string palabras;
	getline(cin, palabras);
	cout << "a" << normalizar(palabras) << endl<< "Total: 0 resultados" << endl;
}

void BUSCAR_OR (){

	string palabras;
	getline(cin, palabras);
	cout << "o" << normalizar(palabras) << endl<< "Total: 0 resultados" << endl;
}

void SALIR(){
	cout <<"Saliendo..."<<endl;
}

void AUTOCOMP(){
	string palabra;
	getline(cin, palabra);
	cout << "p" << normalizar(palabra) << endl<< "Total: 0 resultados" << endl;
}

void Interprete (string comando, DicPaginas &dic){
	if (comando == "i"){
		INSERTAR(dic);
	}else if (comando == "u"){
		BUSCAR_URL(dic);
	}else if (comando == "b"){
		BUSCAR_PAL(dic);
	}else if (comando == "a"){
		BUSCAR_AND();
	}else if (comando == "o"){
		BUSCAR_OR();
	}else if (comando == "p"){
		AUTOCOMP();
	}else if (comando == "s"){
		SALIR();
	}
}
```

Diccionario.h

```
#ifndef _DICCIONARIO_H
#define _DICCIONARIO_H

#include "TablaHash.h"
#include "ArbolAVL.h"

class DicPaginas {
private:
        TablaHash tabla;
        ArbolAVL arbol;
public:
            void insertarPagina(Pagina nueva);
            Pagina* consultar (string URL);
        int contadorPaginas ();

        void insertarPalabra(string palabra, Pagina *pag);
            list<Pagina*> buscarPalabra(string palabra);
};

#endif
```

Diccionario.cpp

```
#include "Diccionario.h"

int DicPaginas::contadorPaginas(){
      return tabla.getNumElem();
}

void DicPaginas::insertarPagina(Pagina nueva){
      tabla.insertar(nueva);
}

void DicPaginas::insertarPalabra(string palabra, Pagina* nueva){
      arbol.insertarRaiz(palabra, nueva);
}

Pagina * DicPaginas::consultar(string URL){
    return tabla.consultar(URL);
}

list<Pagina*>  DicPaginas::buscarPalabra(string palabra){
      return arbol.buscar(palabra);
}
```

TablaHash.h

```
#ifndef _TABLAHASH_H
#define _TABLAHASH_H

#include "Pagina.h"

class TablaHash {
  private:
    list<Pagina> *tablaHash;
    int capacidad;
    int numElem;

    unsigned int hash(string url);

public:
    TablaHash();
    TablaHash(int cap);
    ~TablaHash();
    void insertar(Pagina nueva);
    void redimensionar();
    Pagina* consultar(string url);
    int getNumElem();
};
#endif
```

TablaHash.cpp

```
#include "TablaHash.h"

unsigned int TablaHash::hash(string url){
        unsigned int valorHash = 0;
        int suma =17;
       for (unsigned int i = 0; i < url.length(); i++) {
                valorHash = (valorHash * suma ^ (unsigned char) url[i])% capacidad;
        }
        return valorHash % capacidad;
}


TablaHash::TablaHash(){
        capacidad = 100001;
        tablaHash = new list<Pagina>[capacidad];
        numElem = 0;
```

```
}

TablaHash::TablaHash(int cap){
        capacidad = cap;
        tablaHash = new list<Pagina>[capacidad];
        numElem = 0;
}


TablaHash::~TablaHash(){
        delete[] tablaHash;
}

void TablaHash::insertar(Pagina nueva) {

    if ( (double) numElem / capacidad > 2){
            redimensionar();
    }

      int posicion = hash(nueva.getURL());

      list<Pagina>::iterator iterador = tablaHash[posicion].begin();

        while(iterador != tablaHash[posicion].end() && iterador->getURL() < nueva.getURL()){
                iterador++;
        }

        if(iterador == tablaHash[posicion].end() || iterador->getURL() != nueva.getURL()){
                tablaHash[posicion].insert(iterador, nueva);
                numElem++;
        }
        else
        {
                iterador->setTitulo(nueva.getTitulo());
                iterador->setRelevancia(nueva.getRelevancia());
        }

}


void TablaHash::redimensionar() {

      int capacidadAntigua = capacidad;
      capacidad *= 2;

      TablaHash nuevaTabla(capacidad);
```

```
      for(int i = 0; i < capacidadAntigua; i++){
            list<Pagina>::iterator it = tablaHash[i].begin();
            while(it != tablaHash[i].end()){
                  nuevaTabla.insertar(*it);
                  it++;
            }
      }
      delete[] tablaHash;
      tablaHash = nuevaTabla.tablaHash;
      nuevaTabla.tablaHash = NULL;
}


Pagina* TablaHash::consultar(string url){
    Pagina* p = NULL;
    int posicion = hash(url);
        list<Pagina>::iterator iterador = tablaHash[posicion].begin();

      while(iterador != tablaHash[posicion].end() && iterador->getURL() < url){
            iterador++;
      }

      if(iterador != tablaHash[posicion].end() && iterador->getURL() == url){
            p=&(*iterador);

      }
      return p;
}

int TablaHash::getNumElem(){
        return numElem;
}
```

ÁrbolAVL.h

```
#ifndef _ARBOL_H
#define _ARBOL_H

#include "NodoAVL.h"
#include "Pagina.h"

class ArbolAVL{
private:
    NodoAVL * raiz;
    int nElem;
    int altura(NodoAVL * nodo);
    void insertar (NodoAVL *&A, string palabra, Pagina* pag);
    void insertarPorIzquierda(NodoAVL *&A, string palabra, Pagina *pag);
    void insertarPorDerecha(NodoAVL *&A, string palabra, Pagina *pag);
    void insertaPagina(NodoAVL *&A, Pagina *pag);
    void insertarAux(NodoAVL *&A, Pagina *pag);
    void RDI(NodoAVL *& A);
    void RSI(NodoAVL *& A);
    void RSD(NodoAVL *& A);
    void RDD(NodoAVL *& A);
public:
    ArbolAVL();
    ~ArbolAVL();
    void insertarRaiz (string palabra, Pagina *pag);
    list<Pagina*> buscar (string palabra);
    int numElem (void);
};

#endif
```

ÁrbolAVL.cpp

```
#include "ArbolAVL.h"

ArbolAVL::ArbolAVL(){
      this->raiz = NULL;
      this->nElem = 0;
}

ArbolAVL::~ArbolAVL(){
      delete raiz;
}
```

```
void ArbolAVL::insertarRaiz (string palabra, Pagina *pag){
      insertar(raiz, palabra, pag);
}

list<Pagina*> ArbolAVL::buscar(string palabra) {
      NodoAVL* actual = raiz;
    while (actual != NULL) {
        if (palabra == actual->palabra) {
            return actual->paginas;
        } else if (palabra < actual->palabra) {
            actual = actual->izq;
        } else {
            actual = actual->der;
        }
    }

    return list<Pagina*>();
}

int ArbolAVL::numElem (void) {
      return this->nElem;
}

int ArbolAVL::altura(NodoAVL * nodo){
      if(nodo == NULL)
            return -1;
      else
            return nodo->altura;
}

void ArbolAVL::insertar (NodoAVL *&A, string palabra, Pagina *pag){

      if (A == NULL){
            A = new NodoAVL();
            A->palabra = palabra;
            A->paginas.push_back(pag);
            nElem++;
      }else if(palabra < A->palabra){
            insertarPorIzquierda(A, palabra, pag);
      }else if (palabra > A->palabra){
            insertarPorDerecha(A, palabra, pag);
      }else if (pag != NULL){
                  insertaPagina(A, pag);
      }
}
```

```
void ArbolAVL::insertarPorIzquierda(NodoAVL *&A, string palabra, Pagina *pag){
      insertar(A->izq, palabra, pag);

      if(altura(A->izq) - altura(A->der) > 1){
            if(palabra < A->izq->palabra)
                  RSI(A);
            else
                  RDI(A);
      }else{
            A->altura = 1 + max(altura(A->izq), altura(A->der));
      }
}



void ArbolAVL::insertarPorDerecha(NodoAVL *&A, string palabra, Pagina *pag) {
    insertar(A->der, palabra, pag);

    if (altura(A->der) - altura(A->izq) > 1) {
        if (palabra > A->der->palabra)
            RSD(A);
        else
            RDD(A);
    } else {
        A->altura = 1 + max(altura(A->izq), altura(A->der));
    }
}



void ArbolAVL::insertaPagina(NodoAVL *&A, Pagina *pag){
            list<Pagina*>::iterator iterador = A->paginas.begin();
            while (iterador != A->paginas.end() && (*iterador != NULL) &&
(*iterador)->getURL() != pag->getURL()) {
                  iterador++;
            }

            if (iterador == A->paginas.end() ) {
                  insertarAux(A, pag);
            }
}


void ArbolAVL::insertarAux (NodoAVL *&A, Pagina *pag) {
            list<Pagina*>::iterator iterador = A->paginas.begin();
```

```
            while (iterador != A->paginas.end() && (*iterador != NULL) &&
(*iterador)->getRelevancia() > pag->getRelevancia()) {
                  iterador++;
            }


            if (iterador != A->paginas.end() && (*iterador != NULL) &&
(*iterador)->getRelevancia() == pag->getRelevancia()) {
                  while (iterador != A->paginas.end() && (*iterador != NULL) &&
(*iterador)->getRelevancia() == pag->getRelevancia() && (*iterador)->getURL() <
pag->getURL()) {
                        iterador++;
                  }
            }

            A->paginas.insert(iterador, pag);

}

void ArbolAVL::RSI(NodoAVL *& A){
    NodoAVL* B = A->izq;
    A->izq = B->der;
    B->der = A;
    A->altura = 1 + max(altura(A->izq), altura(A->der));
    B->altura = 1 + max(altura(B->izq), A->altura);
    A = B;
}

void ArbolAVL::RSD(NodoAVL*& A){
    NodoAVL* B = A->der;
    A->der = B->izq;
    B->izq = A;
    A->altura = 1 + max(altura(A->izq), altura(A->der));
    B->altura = 1 + max(altura(B->der), A->altura);
    A = B;
}

void ArbolAVL::RDI(NodoAVL*& A){
      RSD(A->izq);
    RSI(A);
}

void ArbolAVL::RDD(NodoAVL*& A){
    RSI(A->der);
    RSD(A);
}
```

NodoAVL.h

```
#ifndef _NODO_H
#define _NODO_H
#include "Pagina.h"

class ArbolAVL;

class NodoAVL{

friend class ArbolAVL;

private:
      string palabra;
      list<Pagina*> paginas;
      int altura;
      NodoAVL* izq;
      NodoAVL* der;
public:
      NodoAVL();
      ~NodoAVL();
};
#endif
```

Nodo.cpp

```
#include "NodoAVL.h"
NodoAVL::NodoAVL(){
    this->altura = 0;
    this->izq = NULL;
    this->der = NULL;
}

// DESTRUCTOR
NodoAVL::~NodoAVL() {
    delete izq;
    delete der;
}
```

Página.h

```
#ifndef _PAGINA_H
#define _PAGINA_H

#include <list>
#include <iostream>
#include <string>
using namespace std;

class DicPaginas;

class Pagina {
private:
        string URL;
        string titulo;
        int relevancia;

public:

        void escribir(DicPaginas& dic);
        string leer();
        void escribirNum(DicPaginas & dic, int indice);

        string getURL();
        string getTitulo();
        int getRelevancia();

        void setURL(string nuevaURL);
        void setTitulo(string nuevoTitulo);
        void setRelevancia(int nuevaRelevancia);
};
#endif
```

Página.cpp

```
#include "Pagina.h"
#include "Diccionario.h"
#include "Interprete.h"

string Pagina::leer(){
        int relevanciaLeida;
        std::cin>>relevanciaLeida;
        setRelevancia(relevanciaLeida);
```

```
        cin.ignore();

        string urlLeida;
        urlLeida=normalizar(urlLeida);
        std::getline(cin, urlLeida);
        setURL(urlLeida);

        string tituloLeido;
        std::getline(cin, tituloLeido);
        setTitulo(tituloLeido);
        return getURL();
}

void Pagina::escribir(DicPaginas & dic){
        cout <<dic.contadorPaginas()<<". "<<getURL()<<", "<<getTitulo()<<", Rel.
"<<getRelevancia()<<endl;
}


void Pagina::escribirNum(DicPaginas & dic, int indice){
        cout <<indice<<". "<<getURL()<<", "<<getTitulo()<<", Rel.
"<<getRelevancia()<<endl;
}

string Pagina::getURL(){
        return URL;
}

string Pagina::getTitulo(){
        return titulo;
}

int Pagina::getRelevancia(){
        return relevancia;
}

void Pagina::setTitulo(string nuevoTitulo){
        titulo = nuevoTitulo;
}

void Pagina::setRelevancia(int nuevaRelevancia){
        relevancia = nuevaRelevancia;
}

void Pagina::setURL(string nuevaURL){
        URL = nuevaURL;
```

```
}
```

4- INFORME DE DESARROLLO

FECHA INICIO: 08/10/24
FECHA FIN: 05/12/24

Semana	Análisis	Diseño	Implementación	Validación	Total
1	20	15	40	15	90
2	30	60	80	20	190
3	30	50	90	30	200
4	15	30	60	10	115
5	25	30	120	25	200
6	30	75	200	30	335
7	60	60	360	45	525
8	45	40	300	300	685
9	120	90	720	360	1290
TOTAL(min)	375	450	1970	835	3.630
Media(%)	10,3306%	12,3967%	54,2700%	23,0028%	100%

***HORAS APROXIMADAS* = 3630/60 = 60.5**

5- CONCLUSIONES Y VALORACIONES PERSONALES

Este trabajo nos ha producido sentimientos encontrados. Por un lado nos ha parecido frustrante sobre todo en la última etapa (ejercicio 300). Esto se debe a que hemos lidiado con el mítico *Segmentation fault (core dumped)*, lo que nos ha llevado a horas de revisión del código. Finalmente, lo conseguimos solucionar verificando que los punteros son *distintos de null* antes de utilizarlos. Al mismo tiempo este proyecto nos ha parecido motivador y gratificante, ya que hemos puesto en *práctica conceptos teóricos* como las tablas hash y los árboles. De esta forma hemos comprendido cómo estas estructuras de datos pueden aplicarse en escenarios reales (buscador web). Hemos encontrado especialmente interesante y útil ver cómo estas estructuras de datos que estudiamos con Ginés son capaces de garantizar la eficiencia y organización en el manejo de información.

Otro punto a destacar es la *relación con otras asignaturas*. Programar en C++ nos ha servido para consolidar los conocimientos de asignaturas como Introducción a la Programación y Tecnología de la Programación. Al mismo tiempo, hemos aplicado conceptos de Programación Orientada a Objetos, profundizando en el uso de clases. Hemos fortalecido así tanto conocimientos como habilidades técnicas.

Para acabar nos gustaría resaltar que ha sido una gran primera toma de contacto del *desarrollo software en equipo*. En este caso hemos disfrutado del trabajo por parejas puesto que intentamos avanzar el proyecto siempre que fuese posible de manera conjunta mediante reuniones virtuales.