ACTIVIDAD TEMAS 2 Y 3

JOSE ANGEL GARCIA MARIN – 49478872F

JUAN RONDÁN RAMOS – 48854653P

Subgrupo 3.3 Noviembre / Diciembre 2024

INDICE

ANÁLISIS Y DISEÑO DEL PROBLEMA

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

Las clases definidas son las siguientes:

```
class Pagina
{
private:
    int relevancia;
    std::string url, titulo;
public:
    void leer();
    void escribir();
    std::string getURL();
    int getRelev();
    void actualizar(Pagina actualizada);
};
```

```
class DicPaginas {
   private:
     TablaHash tabla
     Arbol arbol;
   public:
     DicPaginas();
     ~DicPaginas();
     Pagina* insertar(Pagina nueva);
     Pagina* consultar(std::string url);
     int getNumElem();
     void insertar (std::string palabra, Pagina *pag);
     list<Pagina*> buscar (std::string palabra);
};
```

```
class Interprete
{
private:
    DicPaginas dic;
    void INSERTAR();
    void BUSCA_URL();
    void BUSCA_PAL();
    void AND();
    void OR();
    void AUTOCOMPLETAR();
    string normalizar(string cadena);
public:
    void Interpretar(char comando);
};

class TablaHash {
private:
    list<Pagina>* tabla;
    int nElem;
    int tam;
    void reestructurar();
    unsigned int funcion(const std::string& url);
public:
    TablaHash();
    ~TablaHash();
    Pagina* inserta(Pagina nueva);
    Pagina* consulta(string url);
    int getNumElem();
};
```

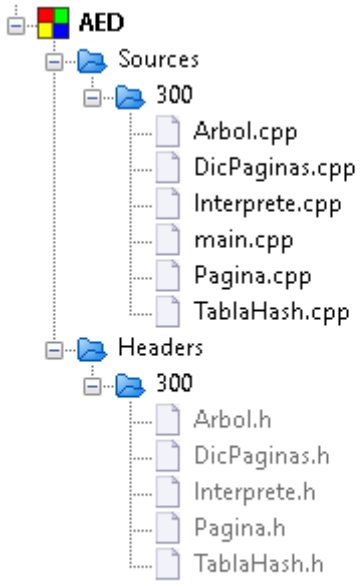
```
class Nodo {
    friend class Arbol;
    private:
        std::string palabra;
        std::list<Pagina*> referencias;
        Nodo* izq;
        Nodo* der;
        int h;

        Nodo(std::string palabra, Pagina* pag);
        ~Nodo();
        void addRef(Pagina* pag);
};
class Arbol {
    private:
        Nodo* raiz;

        int altura(Nodo* nodo);
        void rsi(Nodo*& nodo);
        void rsd(Nodo*& nodo);
        void balanceoIzq(Nodo*& nodo, std::string palabra);
        void balanceoDer(Nodo*& nodo, std::string palabra);
        bool insertar(Nodo*& nodo, std::string palabra, Pagina* pag);

    public:
        Arbol();
        ~Arbol();
        void insertar(std::string palabra, Pagina *pag);
        std::list<Pagina*> buscar(std::string palabra);
};
```

La clase Arbol utiliza la clase Nodo.

Los atributos de la clase DicPaginas son Arbol y TablaHash.

- ¿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).

Los módulos existentes son:



Los archivos .cpp únicamente dependen de su respectivo archivo .h, es decir, Arbol.cpp depende de Arbol.h, Pagina.cpp de Pagina.h …

Por otro lado, en los ficheros .h:

Pagina.h no tiene dependencias.

DicPaginas.h depende de Pagina.h, TablaHash.h y Arbol.h

Interprete.h depende de DicPaginas.h

TablaHash.h y Arbol.h depende de Pagina.h

- ¿Cómo se hace la normalización del texto?

Para la normalización del texto, se ha definido la función "normalizar", cuyo objetivo es procesar una cadena de texto con el objetivo de convertirla a una versión estándar. Esto incluye transformar letras mayúsculas a minúsculas o reemplazar caracteres especiales, como letras acentuadas, por sus equivalentes sin la tilde.

- ¿Cómo es el Makefile? ¿Contiene todas las dependencias existentes?

El Makefile que usamos para compilar tiene una serie de reglas con una estructura fija. Cada regla se compone de dos líneas. En la primera línea, se escribe primero el nombre del fichero objetivo y, tras el carácter ':', las dependencias de ese fichero. En la segunda línea, se encuentra el comando en la terminal, que requiere que existan y estén actualizadas las dependencias y que generará el fichero objetivo.

Como vemos en la última regla, el objetivo no tiene por qué ser un fichero, tenemos una regla "clean" que nos permite limpiar todos los archivos generados. El Makefile nos permite no tener que volver a compilar todos los ficheros cada vez que hacemos algún cambio sin tener que preocuparnos de comprobar que debemos recompilar en cada caso. En este caso incluimos todas las dependencias, aunque existen algunas que se resolverían indirectamente

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

Respecto a la tabla de dispersión, hemos utilizado una tabla de dispersión abierta, a la que se ha añadido una función de reestructuración que, al superar el promedio de dos elementos por cubeta, destruirá la tabla y creará una nueva con el doble de tamaño. Aunque esto no es estrictamente necesario en dispersión abierta, pues las colisiones no suponen un gran problema, a diferencia de la dispersión cerrada, hemos querido implementar esta función de cara a las operaciones de inserta y de consulta, para distribuir los elementos de una forma más uniforme y que mejore el rendimiento de las mismas.

- ¿Qué función de dispersión se ha usado? ¿Se han probado varias? Hacer una comparativa de las distintas funciones que se han probado.

La función de dispersión que se ha implementado en este código utiliza el algoritmo FNV-1 (Fowler-Noll-Vo hash), que es un método eficiente para calcular hashes de cadenas de texto. La idea base que hemos pensado es: al tener que calcular un valor hash dado una url, estas siempre comenzarán por el mismo prefijo (https://), es por ello que, para evitar esta repetición en cada una de las url's hemos implementado un método que tomará caracteres a partir de este prefijo. Tal y como el propio algoritmo requiere, se utiliza un valor inicial fijo conocido como offset basis (2166136261). Cada carácter restante de la url se incorpora al hash mediante una multiplicación por un número primo grande (BASE_FNV) y una operación XOR. Finalmente se realiza una operación modular para ajustar el valor a un índice de la tabla.

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

Como ya se ha comentado hace un par de apartados, si, hemos usado reestructuración. Al disponer de una dispersión abierta con reestructuración, estamos previniendo una sobrecarga de las cubetas, mejorando la eficiencia promedio en las operaciones de inserta y consulta. Sabemos que el costo de esta estrategia es elevado, pero creemos que, a la larga, compensará.

- ¿Cómo se libera la tabla de dispersión?

La liberación de la tabla de dispersión ocurre en el destructor de la Tabla Hash, eliminando, con un delete, todas las listas de cada una de las cubetas.

- Que tipo de árboles se han implementado y por qué?

Se ha implementado una estructura de árbol AVL. Esto se debe a que creemos que proporcionará un tiempo de ejecución menor en comparación con otros árboles y porque nos resulta más sencilla esta implementación.

- ¿Cómo es la definición de la clase árbol y del nodo?

La clase nodo es una clase con todos sus métodos y atributos privados con clase amiga Arbol. De esta manera la clase Arbol puede tener acceso a los atributos y métodos de la clase Nodo mientras que el resto de clases no podrá acceder. Esto es lo lógico puesto que la clase árbol es por así decirlo una clase interfaz, pues lo único que contiene es un puntero al nodo raíz del árbol para permitir su acceso al resto de clases a través de métodos.

En un principio, además de esto, se definió la clase Nodo como una clase interna para para agrupar lógica relacionada y mejorar la organización del código. No obstante, decidimos sacarlo de la clase Árbol puesto que hacía que el código se viera demasiado compacto.

- Cómo se almacenan las páginas asociadas a cada palabra en el árbol?

Para almacenar las páginas asociadas a cada palabra, se almacenan en una lista los punteros a Pagina junto a la palabra (en el nodo). Además, para optimizar la búsqueda, sobre todo en las funciones AND y OR donde el tiempo pasa de tener un coste de

ejecución exponencial respecto al tamaño de las listas a ser lineal, y mostrar los resultados en el orden apropiado. De esta manera, cada vez que insertamos una página, añadimos su puntero a la lista de referencias de todas sus palabras.

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

El balanceo funciona de igual forma a como se hacía en teoría de forma manual. El balanceo se aplicará tras una inserción de un nodo para garantizar una altura óptima.

Los tipos de desbalances Izquierda-Izquierda (II) se solucionarán con una rotación simple a la izquierda (RSI). Los Izquierda-Derecha (ID), se solucionarán con una rotación doble a la izquierda (RDI). Los Derecha-Derecha (DD) se solucionarán con una rotación simple a la derecha (RSD). Y los Derecha-Izquierda (DI) que se soluciona con una rotación doble a la derecha (RDD).

- ¿Cómo se liberan los árboles?

Para liberar los árboles, el proceso comienza con la llamada al destructor ~Arbol(). Este destructor invoca delete sobre el nodo raíz del árbol, lo que desencadena el destructor ~Nodo() de la raíz. Este destructor se encarga de liberar recursivamente todos los nodos del árbol, invocando a delete sobre los subárboles izquierdo y derecho. Finalmente, el nodo raíz es liberado cuando se completa la ejecución de ~Nodo().

- ¿Se usan variables globales en el programa final?

No, solo se han usado variables locales.

- Si se han usado herramientas como ChatGPT, describir de forma detallada en qué partes y cómo se han usado.

Aunque no es un factor decisivo en la práctica, hemos utilizado ChatGPT para explorar sobre el uso de algunas herramientas avanzadas del entorno de Linux que nos ayudan a depurar errores (como valgrind) y a validar los resultados (como meld). Por un lado, valgrind nos ha sido de gran ayuda para solucionar algunos problemas de memoria (como una violación del segmento de datos en el ejercicio 300 que nos dió muchos dolores de cabeza). Además, meld nos fue muy útil para comparar la salida de nuestro programa con la salida esperada para la entrada de prueba proporcionada, lo cual nos ayudó a solucionar

algunos problemas de salida (por un lado pudimos apreciar con mayor facilidad que las páginas en las que se encontraba una palabra tenían un orden incorrecto y además nos ayudó a encontrar un error debido a un despiste consistente en olvidar normalizar la palabra antes de buscarlas).

LISTADO DE EJERCICIOS Y CÓDIGO

Los ejercicios que hemos resuelto son los siguientes:

- 001. Separando las palabras. Envío 173.
- 002. Normalizar el texto. Envío 765.
- 003. Intérprete de comandos. Envío 1385.
- 004. Diccionario de páginas con listas. Envío 1526.
- 200. Diccionario con tablas de dispersión. Envío 2780.
- 300. Diccionarios de palabras con árboles. Envío 3944.
- 301. Búsqueda de palabras con AND. Envío 4190.
- 302. Búsqueda de palabras con OR. Envío 4191.

Total: 8 ejercicios resueltos

001. SEPARANDO LAS PALABRAS. ENVIO 173

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

int main() {
    string palabra;
    int contador = 0;
    while (cin >> palabra) {
        cout << ++contador << ". " << palabra << endl;
    }
    return 0;
}
```

002. NORMALIZAR EL TEXTO. ENVIO 765

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

string normalizar(string cadena)
{
   string palabraNormalizada;
   size_t i = 0;

   while (i<cadena.length())
   {
      char byte1 = cadena[i];
      if (byte1 == '\xC3')
      {
         char byte2 = cadena[i+1];
         switch (byte2)
         {
         case '\xA1':
         case '\x81':
            palabraNormalizada.push_back('a');
            break;

         case '\xA9':
         case '\x89':
            palabraNormalizada.push_back('e');
            break;

         case '\xAD':
         case '\x8D':
            palabraNormalizada.push_back('i');
            break;

         case '\xB3':
         case '\x93':
            palabraNormalizada.push_back('o');
            break;

         case '\xBA':
         case '\x9A':
         case '\xBC':
         case '\x9C':
            palabraNormalizada.push_back('u');
            break;
```

```
            case '\xB1':
            case '\x91':
               palabraNormalizada.push_back('\xC3');
               palabraNormalizada.push_back('\xB1');
               break;

            default:
               palabraNormalizada.push_back('\xC3');
               palabraNormalizada.push_back(byte2);  /
               break;
            }
            i = i+2;
         }
         else
         {
            palabraNormalizada.push_back(tolower(byte1));
            i++; //letra codificada en 1 byte
         }
      }
      return palabraNormalizada;
}


int main ()
{
   string palabra;
   int contador = 1;

   while (cin >> palabra)
   {
      cout << contador << ". " << palabra << " -> " <<
normalizar(palabra)<<endl;
      contador++;
   }
   return 0;
}
```

003. INTÉRPRETE DE COMANDOS. ENVÍO 1385

```
#include <iostream>
#include <vector>
#define findepagina "findepagina"
using namespace std;

int paginas_insertadas;

*** Aquí va la función normalizar del ejercicio 002 ***

void INSERTAR()
{
   int relevancia;
   string URL;
   string titulo;
   cin >> relevancia >> URL;
   cin.ignore();
   getline(cin, titulo);
   int npalabras = 0;
   string palabra;
   while(cin >> palabra && normalizar(palabra) != findepagina)
   {
      npalabras++;
   }
   cout << ++paginas_insertadas << ". " << URL << ", " << titulo << ", Rel. "
<< relevancia << endl;
   cout << npalabras << " palabras" << endl;
}

void BUSCA_URL()
{
   string url;
   cin >> url;
   cout << "u " << url << endl;
   cout << "Total: 0 resultados" << endl;
}

void BUSCA_PAL()
{
   string palabra;
   cin >> palabra;
   cout << "b " << normalizar(palabra) << endl;
   cout << "Total: 0 resultados" << endl;
}
```

```
void AND()
{
   string palabra;
   vector<string> palabras;

   while (cin.peek() != '\n' && cin >> palabra)
   {
      palabras.push_back(palabra);
   }

   cout << "a";
   for (string palabra : palabras)
   {
      cout << " " << normalizar(palabra);
   }
   cout << endl;

   cout << "Total: 0 resultados" << endl;
}

void OR()
{
   string palabra;
   vector<string> palabras;

   while (cin.peek() != '\n' && cin >> palabra)
   {
      palabras.push_back(palabra);
   }

   cout << "o";
   for (string palabra : palabras)
   {
      cout << " " << normalizar(palabra);
   }
   cout << endl;

   cout << "Total: 0 resultados" << endl;
}

void AUTOCOMPLETAR()
{
   string prefijo;
   cin >> prefijo;
   cout << "p " << normalizar(prefijo) <<endl;
   cout << "Total: 0 resultados" << endl;
}
```

```
void INTERPRETE(char comando) {

        switch (comando)
        {
        case 'i': // insertar
            INSERTAR();
            break;
        case 'u': // búsqueda por url
            BUSCA_URL();
            break;
        case 'b': //búsqueda por palabra
            BUSCA_PAL();
            break;
        case 'a': //and
            AND();
            break;
        case 'o': //or
            OR();
            break;
        case 'p': //autocompletar
            AUTOCOMPLETAR();
            break;
        case 's': // salir
            cout << "Saliendo..." << endl;
            exit(0);
            break;
        default:
            cout << "Comando incorrecto, intentelo de nuevo" << endl;
            break;
        }
}

int main ()
{
    char comando;

    while (cin >> comando) {
        INTERPRETE(comando);
    }

    return 0;
}
```

004. DICCIONARIO DE PÁGINAS CON LISTAS. ENVÍO 1526

Pagina.h:

```
#include <iostream>

#ifndef _PAGINA
#define _PAGINA

class Pagina
{
private:
int relevancia;
std::string url, titulo;
public:
void leer();
void escribir();
std::string getURL();
};
#endif
```

Página.cpp:

```
#include "Pagina.h"
using namespace std;
void Pagina::leer()
{
   cin >> relevancia >> url;
   cin.ignore();
   getline(cin, titulo);
}
void Pagina::escribir()
{
      cout << url << ", " << titulo << ", Rel. " << relevancia << endl;
}
std::string Pagina::getURL()
{
   return url;
}
```

Interprete.h:

```
#ifndef INTERPRETE_H
#define INTERPRETE_H
#include<string>
#include<iostream>
#include "DicPaginas.h"
using namespace std;
class Interprete
{
private:
   DicPaginas dic;
   void INSERTAR();
   void BUSCA_URL();
   void BUSCA_PAL();
   void AND();
   void OR();
   void AUTOCOMPLETAR();
   string normalizar(string cadena);
public:
   void Interpretar(char comando);
};
#endif
```

Interprete.cpp:

*** Solo se han modificado las funciones insertar y busca_por_url ***

```
void Interprete::INSERTAR()
{
    Pagina nueva;
    nueva.leer();
    dic.insertar(nueva);
    int npalabras = 0;
    string palabra;
    while(cin >> palabra && normalizar(palabra) != findepagina)
    {
        npalabras++;
    }
    cout << dic.getNumElem() << ". ";
    nueva.escribir();
    cout << npalabras << " palabras" << endl;
}
void Interprete::BUSCA_URL() {
    string url;
    cin >> url;
    Pagina* pagina = dic.consultar(url);
    cout << "u " << url << endl;
    if (pagina != nullptr) {
        cout << "1. ";
        pagina->escribir();
        cout << "Total: 1 resultados" << endl;
    } else {
        cout << "Total: 0 resultados" << endl;
    }
}
```

DicPaginas.h:

```
#ifndef _DICCIONARIO
#define _DICCIONARIO

#include <list>
#include <iostream>
#include "Pagina.h"

class DicPaginas {
  private:
    std::list<Pagina> lista;
    int NumElem;
  public:
    DicPaginas ();
    void insertar (Pagina nueva);
    Pagina* consultar (std::string url);
    int getNumElem ();
};
```

DicPaginas.cpp:

```
#include "DicPaginas.h"
#include "Pagina.h"
#include <list>
#include <iostream>
DicPaginas::DicPaginas () {
   NumElem = 0 ;
}
void DicPaginas::insertar (Pagina nueva) {
   std::list<Pagina>::iterator it = lista.begin();
   while ((it!=lista.end()) && (it->getURL() < nueva.getURL())) it++;
   if ((it == lista.end()) || (it->getURL() != nueva.getURL()))
   {
      lista.insert(it, nueva);
      NumElem++;
   } else {
       *it = nueva;
   }
}
Pagina* DicPaginas::consultar (std::string url) {
   Pagina * pagina = NULL;
   std::list<Pagina>::iterator it = lista.begin();
   while ((it!=lista.end()) && (it->getURL() < url)) it++;
   if ((it != lista.end()) && (it->getURL() == url))
      pagina = &(*it);

   return pagina;
}
int DicPaginas::getNumElem () {
   return NumElem;
}
```

Main:

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

int main ()
{
   Interprete interprete;
   char comando;
   while (cin >> comando) {
      interprete.Interpretar(comando);
   }
}
```

Makefile:

```
#programa a
a.out: Pagina.o DicPaginas.o Interprete.o main.o
	g++ Pagina.o DicPaginas.o Interprete.o main.o

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

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

#Interprete
Interprete.o: Interprete.cpp Interprete.h Pagina.h DicPaginas.h
	g++ -c Interprete.cpp

#main
main.o: main.cpp Pagina.h DicPaginas.h Interprete.h
	g++ -c main.cpp

#limpiar
clean:
	rm -f Pagina.o DicPaginas.o Interprete.o main.o a.out
```

200. DICCIONARIO CON TABLAS DE DISPERSIÓN. ENVÍO 2780

*** Pagina.h y Pagina.cpp importados de 004 ***

*** Interprete.h e Interprete.cpp importados de 004 ***

DicPaginas.h:

```
#ifndef _DICCIONARIO
#define _DICCIONARIO
#include "Pagina.h"
#include "TablaHash.h"
class DicPaginas {
  private:
    TablaHash tabla;
  public:
    DicPaginas();
    void insertar(Pagina nueva);
    Pagina* consultar(std::string url);
    int getNumElem();
};
#endif // _DICCIONARIO
```

DicPaginas.cpp:

```
#include "DicPaginas.h"
DicPaginas::DicPaginas() {}
void DicPaginas::insertar(Pagina nueva) {
   tabla.inserta(nueva);
}
Pagina* DicPaginas::consultar(std::string url) {
   return tabla.consulta(url);
}
int DicPaginas::getNumElem() {
   return tabla.getNumElem();
}
```

TablaHash.h:

```
#ifndef TABLAHASH_H
#define TABLAHASH_H
#include <string>
#include <list>
#include "Pagina.h"
using namespace std;
class TablaHash {
private:
   list<Pagina>* tabla;
   int nElem;
   int tam;
   void reestructurar();
   unsigned int funcion(const std::string& url);
public:
   TablaHash();
   ~TablaHash();
   void inserta(Pagina nueva);
   Pagina* consulta(string url);
   int getNumElem();
};

#endif
```

TablaHash.cpp:

```
#include "TablaHash.h"
#include <functional>
#define BASE_FNV 16777619
#define MAX_ELEM 2 //con relacion a tam
#define TAM 10007 //tamaño inicial

TablaHash::TablaHash() {
    nElem = 0;
    tam = TAM;
    tabla = new std::list<Pagina>[tam];
}

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

unsigned int TablaHash::funcion(const std::string& url) {
    size_t pos = url.find("://");
    pos = (pos == std::string::npos) ? 0 : pos + 3;
    unsigned int hash = 2166136261;
    while (pos < url.length()) {
        hash *= BASE_FNV;
        hash ^= static_cast<unsigned int>(url[pos]);
            pos++;
    }
    return hash % tam;
}

void TablaHash::inserta(Pagina nueva) {
    int indice = funcion(nueva.getURL());
```

```
    std::list<Pagina>::iterator it = tabla[indice].begin();
    while ((it!=tabla[indice].end()) && (it->getURL() < nueva.getURL())) it++;
    if ((it == tabla[indice].end()) || (it->getURL() != nueva.getURL()))
    {
        tabla[indice].insert(it, nueva);
        nElem++;
        if (nElem > MAX_ELEM * tam)
        {
            reestructurar();
        }
    }
    else
    {
        *it = nueva;
    }
}

Pagina* TablaHash::consulta(string url) {
    Pagina * pagina = nullptr;
    int indice = funcion(url);
    std::list<Pagina>::iterator it = tabla[indice].begin();
    while ((it!=tabla[indice].end()) && (it->getURL() < url)) it++;
    if ((it != tabla[indice].end()) && (it->getURL() == url))
        pagina = &(*it);
    return pagina; // Retorna nullptr si no se encuentra el objeto
}

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

```
void TablaHash::reestructurar() {
    const int tam_anterior = tam;
    tam *= 2;
    list<Pagina>* nuevaTabla = new list<Pagina>[tam];
    for (int i = 0; i < tam_anterior; i++) {
        for (auto& pagina : tabla[i]) {
            int indice = funcion(pagina.getURL());
            std::list<Pagina>::iterator it = nuevaTabla[indice].begin();
            while ((it!=nuevaTabla[indice].end()) && (it->getURL() <
pagina.getURL())) it++;
            nuevaTabla[indice].insert(it, pagina);
        }
    }
    delete[] tabla;
    tabla = nuevaTabla;
}
```

Main:

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

int main ()
{
   Interprete interprete;
   char comando;
   while (cin >> comando) {
      interprete.Interpretar(comando);
   }
}
```

Makefile:

```
#programa a
a.out: Pagina.o TablaHash.o DicPaginas.o Interprete.o main.o
	g++ Pagina.o TablaHash.o DicPaginas.o Interprete.o main.o

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

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

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

#Interprete
Interprete.o: Interprete.cpp Interprete.h Pagina.h TablaHash.h DicPaginas.h
	g++ -c Interprete.cpp

#main
main.o: main.cpp Pagina.h TablaHash.h DicPaginas.h Interprete.h
	g++ -c main.cpp

#limpiar
clean:
	rm -f Pagina.o TablaHash.o DicPaginas.o Interprete.o main.o a.out
```

300. DICCIONARIOS DE PALABRAS CON ÁRBOLES. ENVÍO 3944

*** TablaHash.h y TablaHash.cpp importados de 200 ***

Pagina.h:

```
#include <iostream>
#ifndef _PAGINA
#define _PAGINA
class Pagina
{
private:
   int relevancia;
   std::string url, titulo;

public:
   void leer();
   void escribir();
   std::string getURL();
   int getRelev();
   void actualizar(Pagina actualizada);
};
#endif
```

Pagina.cpp:

```
#include "Pagina.h"
using namespace std;
void Pagina::leer() {
    cin >> relevancia >> url;
    cin.ignore();
    getline(cin, titulo);
}
void Pagina::escribir() {
        cout << url << ", " << titulo << ", Rel. " << relevancia << endl;
}
std::string Pagina::getURL() {
    return url;
}
int Pagina::getRelev() {
    return relevancia;
}
void Pagina::actualizar(Pagina actualizada) {
    url = actualizada.url;
    titulo = actualizada.titulo;
}
```

DicPaginas.h:

```
#ifndef _DICCIONARIO
#define _DICCIONARIO

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

class DicPaginas {
  private:
    TablaHash tabla;
    Arbol arbol;
  public:
    DicPaginas();
    ~DicPaginas();
    Pagina* insertar(Pagina nueva);
    Pagina* consultar(std::string url);
    int getNumElem();
    void insertar (std::string palabra, Pagina *pag);
    list<Pagina*> buscar (std::string palabra);
};

#endif // _DICCIONARIO
```

DicPaginas.cpp:

```
#include "DicPaginas.h"

DicPaginas::DicPaginas() {}

DicPaginas::~DicPaginas() {}

Pagina* DicPaginas::insertar(Pagina nueva) {
   return tabla.inserta(nueva);
}

Pagina* DicPaginas::consultar(std::string url) {
   return tabla.consulta(url);
}

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

void DicPaginas::insertar (string palabra, Pagina *pag) {
   arbol.insertar(palabra, pag);
}

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

Interperete.h:

```
#ifndef INTERPRETE_H
#define INTERPRETE_H
#include<string>
#include<iostream>
#include "DicPaginas.h"

using namespace std;

class Interprete
{
private:
   DicPaginas dic;
   void INSERTAR();
   void BUSCA_URL();
   void BUSCA_PAL();
   void AND();
   void OR();
   void AUTOCOMPLETAR();
   string normalizar(string cadena);
public:
   void Interpretar(char comando);
};


#endif
```

Interprete.cpp:

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

#define findepagina "findepagina"

using namespace std;

*** Importar funcion normalizar de 200 ***
void Interprete::INSERTAR() {
    Pagina nueva;
    nueva.leer();
    Pagina* referencia = dic.insertar(nueva);
    int npalabras = 0;
    string palabra;
    while(cin >> palabra && normalizar(palabra) != findepagina)
    {
          dic.insertar(normalizar(palabra), referencia);
       npalabras++;
    }
    cout << dic.getNumElem() << ". ";
    nueva.escribir();
    cout << npalabras << " palabras" << endl;
}

*** Importar funcion BUSCA_URL(), BUSCA_PAL(), OR(), AUTOCOMPLETAR(),
Interpretar() de 200 ***
```

Arbol.h:

```
#ifndef _ARBOL_AVL
#define _ARBOL_AVL

#include "Pagina.h"
#include <iostream>
#include <string>
#include <list>

class Nodo {
   friend class Arbol;
   private:
      std::string palabra;
      std::list<Pagina*> referencias;
      Nodo* izq;
      Nodo* der;
      int h;

      Nodo(std::string palabra, Pagina* pag);
      ~Nodo();
      void addRef(Pagina* pag);
};

class Arbol {
   private:
      Nodo* raiz;

      int altura(Nodo* nodo);
      void rsi(Nodo*& nodo);
      void rsd(Nodo*& nodo);
      void balanceoIzq(Nodo*& nodo, std::string palabra);
```

```
        void balanceoDer(Nodo*& nodo, std::string palabra);
        bool insertar(Nodo*& nodo, std::string palabra, Pagina* pag);

    public:
        Arbol();
        ~Arbol();
        void insertar(std::string palabra, Pagina *pag);
        std::list<Pagina*> buscar(std::string palabra);
};

#endif
```

Arbol.cpp:

```
#include "Arbol.h"


Nodo::Nodo(std::string palabra, Pagina* pag): palabra(palabra) {
   izq = NULL;
   der = NULL;
   h = 0;
   referencias.push_back(pag);
}


Nodo::~Nodo() {
   delete izq;
   delete der;
}


void Nodo::addRef(Pagina* pag) {
   std::list<Pagina*>::iterator it = referencias.begin();
   while ((it!=referencias.end()) && (((*it)->getRelev() > pag->getRelev() || ((*it)->getRelev() == pag->getRelev()) && ((*it)->getURL() < pag->getURL()))))
it++;
   if ((it == referencias.end()) || (*it != pag))
   {
      referencias.insert(it, pag);
   }
}


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


Arbol::~Arbol() {
   delete raiz;
```

```
}

int Arbol::altura(Nodo* nodo) {
    if (nodo == NULL) {
        return -1;
    }
    return nodo->h;
}

void Arbol::rsi(Nodo*& nodo) {
    Nodo* temp = nodo->izq;
    nodo->izq = temp->der;
    temp->der = nodo;
    nodo->h = 1 + std::max(altura(nodo->izq), altura(nodo->der));
    temp->h = 1 + std::max(altura(temp->izq), altura(temp->der));
    nodo = temp;
}

void Arbol::rsd(Nodo*& nodo) {
    Nodo* temp = nodo->der;
    nodo->der = temp->izq;
    temp->izq = nodo;
    nodo->h = 1 + std::max(altura(nodo->izq), altura(nodo->der));
    temp->h = 1 + std::max(altura(temp->izq), altura(temp->der));
    nodo = temp;
}

void Arbol::balanceoIzq(Nodo*& nodo, std::string palabra) {
    if ((altura(nodo->izq) - altura(nodo->der)) > 1) {
        if (palabra < nodo->izq->palabra) {
            rsi(nodo);
```

```
        } else {
            rsd(nodo->izq);
            rsi(nodo);
        }
    } else {
        nodo->h = 1 + std::max(altura(nodo->izq),altura(nodo->der));
    }
}

void Arbol::balanceoDer(Nodo*& nodo, std::string palabra) {
    if ((altura(nodo->der) - altura(nodo->izq)) > 1) {
        if (palabra > nodo->der->palabra) {
            rsd(nodo);
        } else {
            rsi(nodo->der);
            rsd(nodo);
        }
    } else {
        nodo->h = 1 + std::max(altura(nodo->izq),altura(nodo->der));
    }
}

bool Arbol::insertar(Nodo*& nodo, std::string palabra, Pagina* pag) {
    bool nodoAnadido = false;
    if (nodo == NULL) {
        nodo = new Nodo(palabra, pag);
        nodoAnadido = true;
    } else if (palabra == nodo->palabra) {
        nodo->addRef(pag);
    } else if (palabra < nodo->palabra) {
        if (insertar(nodo->izq, palabra, pag)) balanceoIzq(nodo, palabra);
```

```
    } else if (palabra > nodo->palabra) {
        if (insertar(nodo->der, palabra, pag)) balanceoDer(nodo, palabra);
    }
    return nodoAnadido;
}

void Arbol::insertar(std::string palabra, Pagina* pag) {
    insertar(raiz, palabra, pag);
}

std::list<Pagina*> Arbol::buscar(std::string palabra) {
    std::list<Pagina*> resultado;
    Nodo* nodo = raiz;
    while (nodo && (nodo->palabra != palabra)) {
        if (palabra < nodo->palabra) nodo = nodo->izq;
        else nodo = nodo->der;
    }
    if (nodo) resultado = nodo->referencias;
    return resultado;
}
```

Main:

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

int main ()
{
   Interprete interprete;
   char comando;
   while (cin >> comando) {
      interprete.Interpretar(comando);
   }
}
```

Makefile:

```
#programa a
a.out: Pagina.o TablaHash.o Arbol.o DicPaginas.o Interprete.o main.o
	g++ Pagina.o TablaHash.o Arbol.o DicPaginas.o Interprete.o main.o
#Pagina
Pagina.o: Pagina.cpp Pagina.h
	g++ -c Pagina.cpp
#TablaHash
TablaHash.o : TablaHash.cpp TablaHash.h Pagina.h
	g++ -c TablaHash.cpp
#Arbol
Arbol.o : Arbol.cpp Arbol.h Pagina.h
	g++ -c Arbol.cpp
#DicPaginas
DicPaginas.o: DicPaginas.cpp DicPaginas.h TablaHash.h Arbol.h Pagina.h
	g++ -c DicPaginas.cpp
#Interprete
Interprete.o: Interprete.cpp Interprete.h Pagina.h TablaHash.h Arbol.h
DicPaginas.h
	g++ -c Interprete.cpp
#main
main.o: main.cpp Pagina.h TablaHash.h Arbol.h DicPaginas.h Interprete.h
	g++ -c main.cpp
#limpiar
clean:
	rm -f P
```

301. BÚSQUEDA DE PALABRAS CON AND. ENVÍO 4190

Importado de 300. Única modificación en el fichero Interprete.cpp, en la función AND().

```
void Interprete::AND() {
    string palabra;
    vector<string> palabras;

    while (cin.peek() != '\n' && cin >> palabra) {
        palabras.push_back(normalizar(palabra));
    }

    cout << "a";
    for (string palabra : palabras) {
        cout << " " << palabra;
    }
    cout << endl;
    int nPal = palabras.size();
    list<Pagina*> tabla_referencias[nPal];
    std::list<Pagina*>::iterator its[nPal];
    std::list<Pagina*>::iterator ends[nPal];
    for (int i = 0; i < nPal; ++i) {
        tabla_referencias[i] = dic.buscar(palabras[i]);
        its[i] = tabla_referencias[i].begin();
        ends[i] = tabla_referencias[i].end();
    }

int hayElem = true;
    list<Pagina*> referencias;
    while (hayElem) {
        Pagina* mejorPagina = NULL;
```

```
        for (int i = 0; i < nPal; ++i) {
            if (its[i] != ends[i]) {
                Pagina* actual = *its[i];
                if (mejorPagina == NULL | |
                    actual->getRelev() > mejorPagina->getRelev() | |
                    (actual->getRelev() == mejorPagina->getRelev() && actual-
>getURL() < mejorPagina->getURL())) {
                    mejorPagina = actual;
                }
            }
        }
        if (mejorPagina) {
            bool estaEnTodas = true;
            for (int i = 0; i<nPal; i++) {
                if (*its[i]==mejorPagina) ++its[i];
                else estaEnTodas = false;
            }
            if (estaEnTodas) referencias.push_back(mejorPagina);
        } else {
            hayElem = false;
        }
    }


    int i = 0;
    for(std::list<Pagina*>::iterator it = referencias.begin(); it!=referencias.end() ;
it++) {
        std::cout << ++i << ". " ;
        (*it)->escribir();
    }
    cout << "Total: " << i <<" resultados" << endl;
}
```

302. BÚSQUEDA DE PALABRAS CON OR. ENVÍO 4191

Importado de 300. Única modificación en el fichero Interprete.cpp, en la función OR().

```
void Interprete::OR() {
    string palabra;
    vector<string> palabras;
    while (cin.peek() != '\n' && cin >> palabra) {
        palabras.push_back(normalizar(palabra));
    }
    cout << "o";
    for (string palabra : palabras) {
        cout << " " << palabra;
    }
    cout << endl;
    int nPal = palabras.size();
    list<Pagina*> tabla_referencias[nPal];
    std::list<Pagina*>::iterator its[nPal];
    std::list<Pagina*>::iterator ends[nPal];
    for (int i = 0; i < nPal; ++i) {
        tabla_referencias[i] = dic.buscar(palabras[i]);
        its[i] = tabla_referencias[i].begin();
        ends[i] = tabla_referencias[i].end();
    }

    int hayElem = true;
    list<Pagina*> referencias;
    while (hayElem) {
        Pagina* mejorPagina = NULL;
        for (int i = 0; i < nPal; ++i) {
            if (its[i] != ends[i]) {
                Pagina* actual = *its[i];
```

```
            if (mejorPagina == NULL ||
                actual->getRelev() > mejorPagina->getRelev() ||
                (actual->getRelev() == mejorPagina->getRelev() && actual->getURL() < mejorPagina->getURL())) {
                mejorPagina = actual;
            }
        }
    }
    if (mejorPagina) {
        referencias.push_back(mejorPagina);
        for (int i = 0; i<nPal; i++) {
            if (*its[i]==mejorPagina) ++its[i];
        }
    } else {
        hayElem = false;
    }
  }
  int i = 0;
  for(std::list<Pagina*>::iterator it = referencias.begin(); it!=referencias.end() ; it++) {
    std::cout << ++i << ". " ;
    (*it)->escribir();
  }
  cout << "Total: " << i <<" resultados" << endl;
}
```

INFORME DE DESARROLLO

Para la realización de este proyecto hemos decidido trabajar juntos todo el tiempo. Para las fases iniciales de los ejercicios (Análisis y diseño) hemos trabajado presencialmente puesto que en estos momentos la comunicación y el intercambio de puntos de vista ha sido clave. Por otro lado, para la implementación hemos alternado trabajo presencial (en los momentos clave) con telemático (para hacer coincidir nuestra agenda con más facilidad).

Además, para acelerar la implementación nos repartimos la implementación de las funciones. De esta manera, el código que uno implementa el otro se encarga de revisarlo y de refactorizar en caso de que sea necesario. Así logramos ir más rápido, pero sin perder la oportunidad de combinar nuestro conocimiento y, lo que es más importante, tener en cuenta cómo funciona cada una.

Por último, para la validación hemos trabajado totalmente de manera telemática puesto que esto nos permite buscar errores paralelamente y, cada vez que uno de nosotros encuentra un problema, nos paramos a resolverlo en conjunto

A continuación, se muestra una tabla con el tiempo aproximado dedicado a cada ejercicio.

Ejercicio	Análisis	Diseño	Implementación	Validación	TOTAL
001	0,00	0,50	0,50	0,00	1,00
002	0,50	0,50	1,50	4,00	6,50
003	1,50	1,50	2,00	0,50	5,50
004	1,00	2,00	4,50	2,50	10,00
200	2,00	6,50	7,00	1,50	17,00
300	2,50	8,50	10,50	12,00	33,50
301	0,00	1,00	1,50	1,50	4,00
302	0,00	0,50	0,50	0,50	1,50
TOTAL	7,50	21,00	28,00	22,50	79,00
MEDIAS	9%	27%	35%	28%	1,00

CONCLUSIONES Y VALORACIONES PERSONALES

Durante el desarrollo de este proyecto, hemos tenido la oportunidad de profundizar en algunos aspectos clave de la programación en c++, especialmente en lo relacionado con la gestión de memoria, la eficiencia de los programas y la interacción entre cabeceras y clases. Este enfoque práctico ha sido crucial para consolidar conocimientos teóricos y enfrentarnos a desafíos reales. Si bien ha sido un trabajo exigente, tanto en términos de dificultad como de tiempo, especialmente en el ejercicio 300, en el manejo de estructuras de árboles, creemos que ha merecido totalmente la pena, pues ha representado una oportunidad para mejorar nuestras habilidades y resolución de problemas.

Otro de los aspectos clave, ha sido aprender a manejar de una forma más efectiva los errores que surgían durante la implementación, que no han sido pocos. Esto nos ha permitido aprender a gestionar los propios errores y a realizar una buena depuración del código

Sin embargo, sabemos que aún hay margen de mejora, particularmente en el diseño de estructuras más complejas y en la optimización de código, por lo que debemos seguir trabajando.

En definitiva, este proyecto ha representado un desafío y una gran experiencia de aprendizaje. Hemos conseguido crear una estructura de datos más sofisticada, teniendo una buena relación entre rendimiento y uso de memoria, lo cual puede ser clave en programas cuyo rendimiento o memoria es limitado. Estamos muy satisfechos con los resultados obtenidos y entusiasmados por enfrentarnos a nuevos retos.