

PRÁCTICA TEMAS 2 Y 3

CURSO 2023/24 - ENERO

ALGORITMOS

Y ESTRUCTURAS DE DATOS - I



































Cuenta Mooshak: C67 – Jorge Urbelz Alonso-Cortés
Jorge Urbelz Alonso-Cortés (titular de la cuenta)
Antonio Casas Montiel
Subgrupo: 3.3



Contenido

Prueba aceptación en Mooshak.....	3
Análisis y diseño del programa.....	4
Cuestiones generales	4
Cuestiones Tabla Hash.....	6
Cuestiones Árboles	7
Listado de código	9
Informe de desarrollo	21
Conclusiones y valoraciones personales.....	23

Prueba aceptación en Mooshak

4156	1243:00:42		G3 URBELZ ALONSO CORTES, JORGE	300	Archivo_TAR	4 Accepted
4154	1242:58:57		G3 URBELZ ALONSO CORTES, JORGE	300	Archivo_TAR	3 Wrong Answer
4133	1242:29:50		G3 URBELZ ALONSO CORTES, JORGE	300	Archivo_TAR	2 Time Limit Exceeded
3776	1222:14:06		G3 URBELZ ALONSO CORTES, JORGE	300	Archivo_TAR	2 Time Limit Exceeded
2491	884:48:30		G3 URBELZ ALONSO CORTES, JORGE	204	Archivo_TAR	3 Accepted
2175	742:26:36		G3 URBELZ ALONSO CORTES, JORGE	200	Archivo_TAR	4 Accepted
1309	403:05:59		G3 URBELZ ALONSO CORTES, JORGE	004	Archivo_TAR	3 Accepted
1306	402:49:34		G3 URBELZ ALONSO CORTES, JORGE	004	Archivo_TAR	0 Runtime Error
835	322:56:10		G3 URBELZ ALONSO CORTES, JORGE	003	C++	2 Accepted
834	322:51:28		G3 URBELZ ALONSO CORTES, JORGE	003	C++	0 Presentation Error
751	254:17:40		G3 URBELZ ALONSO CORTES, JORGE	003	C++	0 Wrong Answer
696	235:14:29		G3 URBELZ ALONSO CORTES, JORGE	002	C++	1 Accepted
685	234:30:07		G3 URBELZ ALONSO CORTES, JORGE	002	C++	0 Wrong Answer
681	234:24:53		G3 URBELZ ALONSO CORTES, JORGE	002	C++	0 Wrong Answer
284	139:40:48		G3 URBELZ ALONSO CORTES, JORGE	002	C++	0 Wrong Answer
275	136:13:51		G3 URBELZ ALONSO CORTES, JORGE	002	C++	0 Wrong Answer
274	135:50:07		G3 URBELZ ALONSO CORTES, JORGE	001	C++	1 Accepted
6038	3310:26:02		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Wrong Answer
6037	3310:24:42		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	0 Compile Time Error
6036	3310:03:02		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Time Limit Exceeded
6035	3310:01:29		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Time Limit Exceeded
6034	3309:58:28		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	0 Compile Time Error
6033	3309:49:26		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Wrong Answer
6032	3309:46:34		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	0 Compile Time Error
6031	3309:45:01		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	0 Compile Time Error
6030	3304:01:36		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	0 Time Limit Exceeded
6024	3135:07:09		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	0 Time Limit Exceeded
6023	3119:24:03		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	0 Compile Time Error
6013	2882:55:03		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Time Limit Exceeded
6012	2869:40:35		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Time Limit Exceeded
6011	2869:01:32		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Time Limit Exceeded
6010	2858:33:47		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Time Limit Exceeded
6009	2852:56:41		G3 URBELZ ALONSO CORTES, JORGE	310	Archivo_TAR	9 Wrong Answer
6008	2852:33:18		G3 URBELZ ALONSO CORTES, JORGE	307	Archivo_TAR	3 Accepted

Aclaración: el ejercicio 310 debe estar bugueado, ya que fuera de Mooshak conseguimos sacarlo adelante y aquí en cambio no es posible.

Análisis y diseño del programa

Cuestiones generales

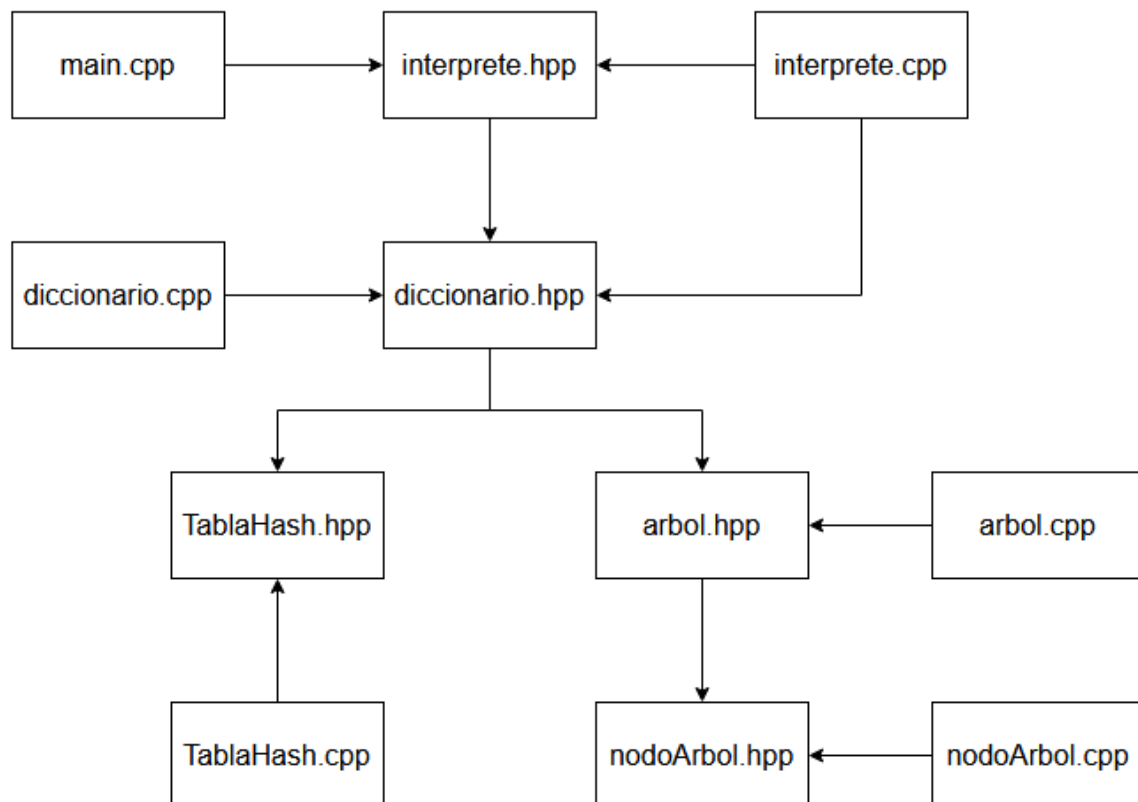
- ¿Qué clases se han definido y qué relación existe entre ellas? Incluir una representación gráfica de las clases.

Hemos definido un total de cuatro clases: “DicPalabra” corresponde con el diccionario de palabras, “Tabla Hash” trata todo el tema de las tablas hash, y tanto “ArbolPalabras” como “NodoArbol” se refieren al uso de árboles.

De alguna manera, “DicPalabra” será la clase suprema que llamará a distintos métodos de las demás clases. En cada uno de sus métodos se llama a su contraparte en las otras clases. De manera parecida, “ArbolPalabras” hace lo propio “NodoArbol”, llamando al método “alargarpal” de “NodoArbol”.

NOMBRE DE LA CLASE	DicPalabras	TablaHash	ArbolPalabras	NodoArbol
ARIBUTOS	TablaHash tabla ArbolPalabras arbol	list<string> *T int B reestructurar() int nElem	NodoArbol* raiz int nElem	char caracter NodoArbol *ptr NodoArbol *sig
MÉTODOS	DicPalabras vaciar insertar consultar alarga anagramas numElem	TablaHash ~TablaHash Hash insertar consultar vaciar numElem anagramas	ArbolPalabras ~ArbolPalabras insertar vaciar alargar numElem	NodoArbol ~NodoArbol consulta inserta PonMarca QuitaMarca HayMarca alargarpal

- ¿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).



- ¿Contiene el makefile todas las dependencias existentes?

El código del Makefile sigue la lógica de los includes. Si por ejemplo, “interprete.o” no se relaciona con “TablaHash.o”, no es necesario incluirlo en las dependencias. A continuación, le muestro una imagen del código del Makefile:

```

a.out: diccionario.o interprete.o arbol.o nodoArbol.o TablaHash.o main.o
    g++ diccionario.o interprete.o arbol.o nodoArbol.o TablaHash.o main.o

diccionario.o: diccionario.cpp diccionario.hpp arbol.hpp TablaHash.hpp
    g++ -g -c diccionario.cpp

interprete.o: interprete.cpp interprete.hpp diccionario.hpp
    g++ -g -c interprete.cpp

arbol.o: arbol.cpp arbol.hpp nodoArbol.hpp
    g++ -g -c arbol.cpp

nodoArbol.o: nodoArbol.cpp nodoArbol.hpp
    g++ -g -c nodoArbol.cpp

TablaHash.o: TablaHash.cpp TablaHash.hpp
    g++ -g -c TablaHash.cpp

main.o: main.cpp diccionario.hpp interprete.hpp arbol.hpp nodoArbol.hpp TablaHash.hpp
    g++ -g -c main.cpp
  
```

- ¿Se usan variables globales en el programa final?

La única variable global que utilizamos es “STR_ESPEC”. Esta variable global corresponde a la clase “interprete.hpp” y “interprete.cpp”. Cuando queremos normalizar una palabra, esto es convertirla a mayúsculas, decidimos introducir todos los caracteres especiales (á, é, í, ó, ú, Á, É, Í, Ó, Ú) como una única cadena, y así recorrerlo de uno en uno de manera más sencilla.

Cuestiones Tabla Hash

- ¿Qué tipo de tablas de dispersión se ha usado y por qué? Justificar la decisión
Hemos decidido implementar una tabla de dispersión abierta con tamaño variable. A la hora de generar el diseño del programa creímos conveniente seguir este camino. Una tabla de dispersión abierta es aquella que tiene una lista con una serie de elementos por cubetas. Nos pareció más sencillo hacerlo de esta manera. Haber utilizado dispersión cerrada hubiese supuesto asignar una función de redispersión, haciéndolo más dificultoso.
- ¿Qué función de dispersión se ha usado? ¿Se han probado varias? Explicar.
Hemos implementado una función de dispersión básica (puede producir colisiones), pero que a fin de cuentas funciona. La función “Hash” suma los valores de los caracteres ASCII de cada letra en la palabra. Esta fue la función de dispersión quizás para un conjunto de datos distinto sería terrible, pero para nuestro caso es correcta.
Por otro lado, estuvimos probando otra función de dispersión que hacía uso del algoritmo FNV-1a, conocido por ser simple y eficaz. Sin embargo, nos generó problemas a la hora de implementar anagramas. Tuvimos una tutoría con el profesor donde se nos aconsejaba el uso de la primera función descrita.
- ¿Cómo se ha resuelto el juego adicional con tablas de dispersión?
Como hemos comentado, parte de nuestro proyecto se ha basado en el que hicimos hace dos años. Por aquel entonces, también había que hacer algunos juegos adicionales, entre los que se encontraba anagramas. Sabiendo esto, decidimos apostar por este juego, pero cambiando gran parte del código anterior ya que era erróneo. Para que os hagáis una idea, hace dos años decidimos crear una clase llamada anagrama, la cual hacía lo mismo que la clase TablaHash, pero añadiendo su implementación especial. Nos dimos cuenta de que hacer esto no era del todo eficiente. No es necesario crear una clase nueva. En vez de eso, nos propusimos añadir la función de “anagramas” a la clase TablaHash. La función “anagramas” busca anagramas de una palabra dada en la tabla. Para ello, utiliza la técnica de ordenar los caracteres para comparar si dos palabras son anagramas entre sí.
- Si se hace reestructuración de las tablas, explicar cómo y justificar la decisión
Nuestra clase TablaHash se encarga de reestructurar la tabla mediante la función “reestructurar”. A ella solo se accede cuando insertamos una palabra en la tabla. Para ello, consideramos oportuno que, si el número de elementos es mayor que el doble del número de cubetas actual, debíamos reestructurar la tabla. Dentro de la función

“reestructurar”, se duplica el tamaño de la tabla, recalcula los índices para las palabras y las reorganiza en la nueva tabla. Además, ordena las listas en la nueva tabla para mejorar el rendimiento de búsquedas (mantiene un factor de carga bajo y reduce la posibilidad de colisiones).

- ¿Cómo se libera la tabla de dispersión?
La manera de liberar la tabla es muy sencilla. La función “~TablaHash” simplemente elimina la tabla: delete[] T;

Cuestiones Árboles

- ¿Qué tipo de árboles se han implementado y por qué?
Hemos implementado un árbol trie por listas. Cuando estudiamos la teoría nos parecía el tipo de árbol más interesante para implementar. Si lo comparamos con el árbol trie por arrays nos dimos cuenta de que resultaba un gran ahorro de memoria a la hora de implementarlo. Por otro lado, realizar un árbol AVL nos parecía demasiado complejo en el tema de rotaciones al momento de insertar un nuevo nodo. Quizás, las listas no sean el método más eficaz o rápido, pero nos decantamos por ella debido a la facilidad que implica.
- ¿Cómo es la definición del tipo árbol y del tipo nodo?
La definición de la clase “ArbolPalabras” crea un nodo del árbol denominado raíz. Desde aquí nos encargaremos de ir llenando el árbol de nodos con caracteres. En la clase “NodoArbol” nos situamos en la creación de un nodo. Como este nodo comienza vacío, no le colocaremos ningún carácter aún. Además, sus dos punteros (“ptr” y “sig”) apuntan a NULL, ya que no existen otros caracteres dentro del nodo a los que apuntar y aún no presentan ningún hijo.
- ¿Cómo se ha resuelto el juego adicional con árboles?
El juego adicional que decidimos hacer fue el de alargar palabras. Este consiste en dado un prefijo, primero buscar en el árbol si ese prefijo existe. Si no existe, el juego acaba. Por el contrario, si existe, debemos buscar la palabra más larga que contiene ese prefijo. Si dos palabras tienen la misma longitud, nos quedamos con la primera alfabéticamente.
Debemos reconocer que la implementación de este ejercicio nos ha llevado más tiempo del que esperábamos. El carácter “Ü” nos supuso un pequeño quebradero de cabeza, debido a que cuenta como dos caracteres. Decidimos crear dos métodos, uno en cada clase nueva (uno en la clase “ArbolPalabras” y otro en la clase “NodoArbol”). El primero de los métodos recibe de diccionario el prefijo y se encarga de buscarlo en el árbol. Aquí crearemos la salida final, ya sea vacía (no se encuentra el prefijo) o la palabra más larga con este. Una vez hemos encontrado el prefijo, nos introducimos en el método de la clase del nodo. Aquí buscaremos recursivamente entre los nodos del árbol todas las palabras que empiecen con el prefijo. A la hora de realizar este método nos fijamos en la función “ListarTodas” que aparece en las transparencias del tema 3.

- ¿Cómo se liberan los árboles?

Al tener dos nuevas clases tenemos dos métodos de eliminación. Por un lado, en la clase “NodoArbol” encontramos el método “~NodoArbol()”, el cual se encarga de eliminar los punteros “sig” y “ptr” del nodo. De manera similar se hace en la clase “ArbolPalabras” con el método “~ArbolPalabras()”, aunque en este caso borra el nodo raíz del árbol y por consiguiente el árbol en sí.

Listado de código

```
***MAIN***
#include "interprete.hpp"
#include <iostream>

using namespace std;
void procesar(string comando);

int main (void){
    string comando;
    while (cin>>comando){
        procesar(comando);
    }
}

***INTERPRETE***
**HPP**
#include "diccionario.hpp"

#ifndef INTERPRETE_H_INCLUDED
#define INTERPRETE_H_INCLUDED

const string STR_ESPEC="áéíóúÁÉÍÓÚ";
#endif

**CPP**
#include "interprete.hpp"
#include "diccionario.hpp"

#include <iostream>
#include <string>
#include <list>
#include <algorithm>
#include <cstring>
#include <unordered_map>
#include <algorithm>
#include <chrono>

using namespace std;
DicPalabras dic;
string normalizar (string cad){
    string salida="";
```

```

int letras[10][2]= { {0xA1, 0x41}, {0xA9, 0x45}, {0xAD, 0x49},
                    {0xB3, 0x4F}, {0xBA, 0x55}, {0x81, 0x41},
                    {0x89, 0x45}, {0x8D, 0x49}, {0x93, 0x4F},
                    {0x9A, 0x55}};

for(unsigned i=0; i<cad.length(); i++){
    if (cad[i] >= 'a' && cad[i] <= 'z')
        salida+=toupper(cad[i]);
    else if (cad[i]==(char)0xC3){
        if (cad[i+1]==(char)0xBC || cad[i+1]==(char)0x9C){
            salida+=(char)0xC3;
            salida+=(char)0x9C;
            i++;
        } else if (cad[i+1]==(char)0xB1 || cad[i+1]==(char)0x91){
            salida+=(char)0xC3;
            salida+=(char)0x91;
            i++;
        } else if (STR_ESPEC.find(cad[i+1])!=string::npos){
            for (int j= 0; j<10; j++){
                if (cad[i+1]==(char)letras[j][0]){
                    salida+=(char)letras[j][1];
                    i++;
                    break;
                }
            }
        } else{
            salida+=cad[i];
        }
    } else {salida+=cad[i];}
}
return salida;
}

void PARTIDAS(){
    string palabra;
    cout << "Partidas: ";

    cin>>palabra;

    if(palabra!="</partidas>"){
        cout << normalizar(palabra);
        while(cin>>palabra){
            if(palabra=="</partidas>")
                break;
            cout << " " << normalizar(palabra);
        }
    }
    cout << endl << "No implementado" << endl;
}

```

```

}

void ALOCADO(){
    string palabra;
    cin >> palabra;
    cout << "Alocado: " << normalizar(palabra) << endl <<
        "No implementado" << endl;
}

void CESAR(){
    string palabra;
    cin >> palabra;
    cout << "César: " << normalizar(palabra) << endl <<
        "No implementado" << endl;
}

void JUANAGRAMA(){
    string palabra;
    cin >> palabra;
    string normalizada = normalizar(palabra);
    cout << "Juanagrama: " << normalizada << " ->";
    list<string> juanagrama = dic.anagramas(normalizada);
    list<string>::iterator iter = juanagrama.begin();
    if(iter==juanagrama.end()) cout << "";
    else cout << " " << *iter;
    cout << endl;
}

void SACO(){
    string palabra1, palabra2;
    cin >> palabra1 >> palabra2;
    cout << "Saco: " << normalizar(palabra1) << " " << normalizar(palabra2) <<
        endl << "No implementado" << endl;
}

void CONSOME(){
    string palabra;
    cin >> palabra;
    cout << "Consomé: " << normalizar(palabra) << endl <<
        "No implementado" << endl;
}

void ALARGA(){
    string palabra;
    cin >> palabra;
    string normalizada=normalizar(palabra);
    cout << "Alarga: " << normalizada << " ->";
    dic.alarga(normalizada);
}

```

```

}

void INSERTAR(){
    string palabra;
    int M=0;

    while(cin>>palabra){
        if(palabra=="</insertar>")
            break;
        dic.insertar(normalizar(palabra)); //Insertamos la palabra normalizada
        M++;
    }
    cout << "Insertando: " << M << " palabras" << endl <<
        "Total diccionario: " << dic.numElem() << " palabras" << endl;
}

void VACIAR(){
    dic.vaciar();
    cout << "Vacando" << endl << "Total diccionario: " <<
        dic.numElem() << " palabras" << endl;
}

void BUSCAR(){
    string palabra;
    cin >> palabra;
    if (dic.consultar(normalizar(palabra))) { //La buscamos normalizada porque así es como se guarda
        cout << "Buscando: " << normalizar(palabra) << " -> " <<
            "Encontrada" << endl;
    } else {
        cout << "Buscando: " << normalizar(palabra) << " -> " <<
            "No encontrada" << endl;
    }
}

void EXIT(){
    cout << "Saliendo..." << endl;
    exit(0);
}

void procesar(string comando){
    if (comando=="<insertar>") INSERTAR();
    else if (comando=="<vaciar>") VACIAR();
    else if (comando=="<buscar>") BUSCAR();
    else if (comando=="<partidas>") PARTIDAS();
    else if (comando=="<alocado>") ALOCADO();
    else if (comando=="<césar>") CESAR();
    else if (comando=="<juanagra>") JUANAGRAMA();
    else if (comando=="<saco>") SACO();
}

```

```

    else if (comando=="<consomé>") CONSOME();
    else if (comando=="<alarga>") ALARGA();
    else if (comando=="<exit>") EXIT();
}

***DICCIONARIO***
**HPP**
#include "arbol.hpp"
#include "TablaHash.hpp"

#ifdef _DICCIONARIO_H_INCLUDED
#define _DICCIONARIO_H_INCLUDED

using namespace std;

class DicPalabras {
private:
    TablaHash tabla;
    ArbolPalabras arbol;
public:
    DicPalabras ();
    void vaciar (void) { tabla.vaciar(); arbol.vaciar(); }
    void insertar (string palabra) { tabla.insertar(palabra); arbol.insertar(palabra); }
    bool consultar (string palabra) { return tabla.consultar(palabra); }
    void alarga (string palabra) { arbol.alargar(palabra); }
    list<string> anagramas(string palabra) { return tabla.anagramas(palabra); }
    int numElem (void) { return tabla.numElem(); }
};
#endif

**CPP**
#include "diccionario.hpp"
DicPalabras::DicPalabras() : tabla({})

***TABLA HASH***
**HPP**
#ifdef __TABLA_HASH_H_INCLUDED
#define __TABLA_HASH_H_INCLUDED

#include <list>
#include <string>

using namespace std;
class TablaHash {
private:
    list<string> *T;

```

```

    int B;
    void reestructurar();
    int nElem;
public:
    TablaHash();
    ~TablaHash();
    unsigned long Hash(string palabra);
    void insertar (string palabra);
    bool consultar (string palabra);
    void vaciar (void);
    int numElem (void) { return nElem; }

    list<string> anagramas(string palabra);
};
#endif

**CPP**
#include "TablaHash.hpp"

#include <list>
#include <string>
#include <algorithm>
#include <unordered_map>

using namespace std;

void TablaHash::reestructurar(){
    unsigned long t;
    int tamprevio = B;
    B = B*2;
    list<string> *nuevaLista = new list<string>[B];

    for(int i = 0; i<tamprevio; i++){
        list<string>::iterator iter = T[i].begin();
        while(iter!=T[i].end()){
            string palabra = *iter;
            t = Hash(palabra)%B;
            list<string>::iterator nuevoIter = nuevaLista[t].begin();
            while(nuevoIter!=nuevaLista[t].end()&&*nuevoIter<palabra)nuevoIter++;
            if(nuevoIter==nuevaLista[t].end()| *nuevoIter!=palabra){
                nuevaLista[t].insert(nuevoIter, palabra);
            }iter++;
        }
    }
    delete[] T;
    T = nuevaLista;
}

```

```

TablaHash::TablaHash(){
    B = 1000;
    T = new list<string>[B];
    nElem = 0;
}

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

void TablaHash::vaciar(void){
    for(unsigned int i=0; i < B; i++) T[i].clear();
    nElem = 0;
}

unsigned long TablaHash::Hash(string palabra){
    long resultado=3*(abs(int(palabra[0])));
    for (unsigned i=1;i<palabra.length();i++){
        resultado+= 3*(abs(int(palabra[i])));
    }
    resultado+=B;
    return resultado;
}

bool TablaHash::consultar (string palabra){
    unsigned long t = Hash(palabra)%B;
    list<string>::iterator iter = T[t].begin();
    while(iter!=T[t].end() && (*iter<palabra)) iter++;
    if(iter==T[t].end() || *iter!=palabra ) return false;
    return true;
}

void TablaHash::insertar (string palabra){
    unsigned long t = Hash(palabra)%B;
    list<string>::iterator iter = T[t].begin();
    while (iter!=T[t].end() && *iter<palabra) iter++;
    if (iter==T[t].end() || *iter!=palabra){
        nElem++;
        T[t].insert(iter, palabra);
    }
    if(nElem > 2*B) reestructurar();
}

list<string> TablaHash::anagramas(string palabra) {
    unsigned long t = Hash(palabra)%B;
    list<string>::iterator iter = T[t].begin();
    list<string> nuevaLista;

```

```

unordered_map<char, int> frecuenciaPalabra;
for (char c : palabra)
    frecuenciaPalabra[c]++;

while (iter != T[t].end()) {
    unordered_map<char, int> frecuenciaAnagrama;
    bool esAnagrama = true;
    for (char c : *iter) frecuenciaAnagrama[c]++;
    for (auto &p : frecuenciaPalabra) {
        if (frecuenciaAnagrama[p.first] != p.second) {
            esAnagrama = false;
            break;
        }
    }

    if (esAnagrama) nuevaLista.push_back(*iter);
    iter++;
}
return nuevaLista;
}

```

ARBOL

HPP

#include "nodoArbol.hpp"

#ifndef ARBOL_H_INCLUDED

#define ARBOL_H_INCLUDED

#include <string>

using namespace std;

class ArbolPalabras {

private:

NodoArbol* raiz;

int nElem;

public:

ArbolPalabras();

~ArbolPalabras();

void insertar(string palabra);

void vaciar(void);

void alargar(string palabra);

int numElem(void) { return nElem; }

};

#endif

CPP

#include "arbol.hpp"

#include <iostream>


```

using namespace std;

ArbolPalabras::ArbolPalabras() {
    raiz = new NodoArbol();    //Crea un nodo denominado raiz, está vacío
}

ArbolPalabras::~~ArbolPalabras() {
    delete raiz;              //Borra raíz, y por tanto el árbol
}

void ArbolPalabras::insertar(string palabra) {
    NodoArbol* nodo = raiz;
    for(int i = 0; i<palabra.length();i++) {
        if (nodo->consulta(char(palabra[i]))==NULL) nodo->inserta(char(palabra[i]));
        nodo=nodo->consulta(char(palabra[i]));
    }
    if (!nodo->HayMarca()){
        nodo->PonMarca();
        nElem++;
    }
}

void ArbolPalabras::vaciar() {
    delete raiz;              //Vacía el árbol actual borrando la raíz de este
    nElem=0;                  //El número de palabras del árbol vuelve a ser 0
    raiz = new NodoArbol();    //Ahora, se genera un nuevo árbol vacío
}

void ArbolPalabras::alargar(string prefijo){
    NodoArbol* nodo = raiz;    //Nos situamos en la raiz
    string palcompleta="";      //Esto será lo que imprimamos

    for(int i = 0; i<prefijo.length();i++) { //Recorremos el prefijo
        if (nodo->consulta(prefijo[i])==NULL){ //Si el caracter no está en el árbol:
            cout << "" << endl;              // devuelve vacío, no se puede alargar
            return;                            //Nos salimos, ya que ha impreso
        }else{
            nodo = nodo->consulta(prefijo[i]); //Nos situamos en el lugar donde está el caracter
            palcompleta+=prefijo[i];          //Añadimos el caracter a la palabra a imprimir
        }
    }
    string mejorpal="";
    nodo->alargarpal(palcompleta, mejorpal); //Alargamos nodo actual a partir del prefijo

    cout << " " << mejorpal << endl;
}

```

```

***NODO ARBOL***
**HPP**
#ifdef NODOARBOL_H_INCLUDED
#define NODOARBOL_H_INCLUDED

#include <string>

using namespace std;
class NodoArbol {
    private:
        char character;
        NodoArbol *ptr, *sig;

    public:
        NodoArbol();
        NodoArbol(char car, NodoArbol* sig, NodoArbol* ptr);
        ~NodoArbol();

        NodoArbol* consulta(char character);
        void inserta(char character);
        void PonMarca();
        void QuitaMarca();
        bool HayMarca();
        void alargarpal(string &palActual, string &mejorPalabra);
};
#endif

**CPP**
#include "nodoArbol.hpp"

#include <iostream>
#include <cstring>
#include <queue>

using namespace std;
NodoArbol::NodoArbol() {
    character=' ';
    ptr = NULL;
    sig = NULL;
}

NodoArbol::NodoArbol(char car, NodoArbol* sig, NodoArbol* ptr){
    this->character = car;
    this->sig = sig;
    this->ptr = ptr;
}

NodoArbol::~~NodoArbol() {

```

```

        delete ptr;
        delete sig;
    }

void NodoArbol::inserta(char car) {
    NodoArbol* nodotemp = this; //Nos situamos en el nodo
    //Mientras la siguiente posición del nodo no esté vacía y no el carácter sea distinto, avanzamos por
    el nodo
    while(nodotemp->sig!=NULL && nodotemp->sig->caracter<car) nodotemp=nodotemp->sig;
    //Si la siguiente posición del nodo está vacía: (esto quiere decir que el caracter aún no ha sido escrito)
    if(nodotemp->sig==NULL || nodotemp->sig->caracter!=car)
        nodotemp->sig = new NodoArbol(car, nodotemp->sig, new NodoArbol());
}

NodoArbol* NodoArbol::consulta(char car) {
    NodoArbol* nodotemp = this->sig;
    while(nodotemp!=NULL && nodotemp->caracter<car) nodotemp=nodotemp->sig;
    if (nodotemp!=NULL && nodotemp->caracter == car) return nodotemp->ptr;
    else return NULL;
}

void NodoArbol::PonMarca() {
    this->caracter='$'; //Pone marca de fin de palabra
}

void NodoArbol::QuitaMarca() {
    this->caracter=' '; //Quita $
}

bool NodoArbol::HayMarca() {
    return this->caracter=='$'; //Busca final de palabra
}

void NodoArbol::alargarpal(string &palActual, string &mejorPalabra) {
    if (this->HayMarca()) {
        int contAct=0;
        int contMej=0;
        for(int i=0; i<palActual.length(); i++){
            if (palActual[i]==(char)0xC3) contAct+=1;
        }
        for(int i=0; i<mejorPalabra.length(); i++){
            if (mejorPalabra[i]==(char)0xC3) contMej+=1;
        }
        if (palActual.length()-contAct > mejorPalabra.length()-contMej ||
            (palActual.length()-contAct==mejorPalabra.length()-contMej && palActual < mejorPalabra)){
            mejorPalabra = palActual;
        }
    }
}

```

```
NodoArbol* nodotemp = this->sig;
while (nodotemp != nullptr) {
    palActual += nodotemp->caracter;
    NodoArbol* ptrtemp = nodotemp->ptr;
    ptrtemp->alargarpal(palActual, mejorPalabra);
    palActual.pop_back();
    nodotemp = nodotemp->sig;
}
}
```

Informe de desarrollo

Antes de todo, queremos dejar en claro que somos estudiantes de segunda matrícula. Es por ello por lo que no empezamos desde el principio, sino que tomamos de base el proyecto sin finalizar del año pasado. Simplemente adaptamos algunas partes a como debíamos hacerlo esta vez.

Tanto el juego anagramas que usa la tabla hash como todo lo relacionado con árboles es completamente nuevo, y es lo que estuvimos haciendo este año. Debido a esto, el informe de desarrollo compete menos horas de las normales para hacer el trabajo desde cero.

Como podéis comprobar, los últimos meses hemos dedicado una ingente cantidad de tiempo a la validación de la parte final de la práctica, la cual no éramos capaces de resolver por nosotros mismos.

Día/Mes	Análisis	Diseño	Implementación	Validación	TOTAL
15/10	10	15	20	10	55
17/10	10	20			30
18/10		18	40		58
19/10			10	60	70
“	10	20			30
21/10		12	30	10	52
23/10				25	25
24/10	10	40			50
26/10			20	30	50
08/11	10	10			20
09/11			20	5	25
11/11	10	30	40		80
15/11				60	60
22/01				120	120
28/01				180	180
04/02			120	240	360
10/2				240	240
17/2				120	120
24/2				240	240
1/3				60	60
9/3				50	50
10/3				80	80
23/3				45	45
31/3				75	75
13/4				30	30
19/4				40	40

TOTAL (minutos)	60	155	300	1720	2235
MEDIAS (porcentaje)	2,68%	6,93%	13,42%	76,96%	100%

Conclusiones y valoraciones personales

En conclusión, la realización de esta práctica sobre AED1 nos ha permitido consolidar los conocimientos teóricos adquiridos en clase. La aplicación práctica de los conceptos de búsqueda e inserción nos han servido para una comprensión más profunda de ciertos algoritmos junto con árboles.

A lo largo del desarrollo de la práctica, nos enfrentamos a diversos problemas, siendo uno de los más destacados la implementación de un diccionario que incluyera correctamente las tildes. Este problema resalta la importancia de abordar no solo los aspectos algorítmicos, sino también los detalles prácticos y las peculiaridades del lenguaje de programación utilizado.

A pesar de los obstáculos encontrados, consideramos que esta entrega ha contribuido significativamente a mejorar nuestras habilidades de programación en C++. La necesidad de traducir conceptos teóricos en código funcional ha fortalecido nuestra capacidad para diseñar y estructurar clases de manera efectiva. Asimismo, la resolución de problemas específicos nos ha brindado una buena experiencia en la aplicación de algoritmos y estructuras de datos en situaciones del mundo real.

En resumen, esta práctica no solo ha servido como un ejercicio técnico, sino también como un medio para profundizar en el entendimiento de los conceptos y técnicas abordados en el curso. A pesar de las dificultades encontradas, consideramos que hemos logrado alcanzar los objetivos propuestos y que la experiencia adquirida será de gran utilidad en futuros proyectos y desarrollos.