1. Problema

Comprimir un archivo de texto usando el algoritmo de Huffman.

2. Código

2.1. ArbolH.h

```
#ifndef ARBOLH_H
#define ARBOLH_H
#include "list"
#include "FibonacciHeap.h"
#include "fstream"
#include "Monticulo.h"

#define CARACTER_NULL 126
#define END_L 94

using namespace std;

class ArbolH
{
    public:
        class Nodo{
            public:
                Nodo();
                Nodo(char caracter, float valor);
                Nodo(float valor);
                Nodo * hijos[2];
                Nodo * padre;
                char caracter;
                bool buscarHijo(Nodo *& hijo);
                void _reducirNodo(string &);
                void _generarNodo(string::iterator &,string::iterator endl,map<char, Nodo *> &hojas);
                float valor;
        };
        class NodoDTO{
            public:
                NodoDTO(){nodo = nullptr;};
                NodoDTO(Nodo * nodo){this->nodo = nodo;};
                Nodo * nodo;
                bool operator == (NodoDTO second){
                    if(this->nodo->valor == second.nodo->valor) return true;
                    return false;
                };
                bool operator != (NodoDTO second){
                    if(this->nodo->valor != second.nodo->valor) return true;
                    return false;
                };
                bool operator < (NodoDTO second){
                    if(this->nodo->valor < second.nodo->valor) return true;
                    return false;
                };
                bool operator > (NodoDTO second){
                    if(this->nodo->valor > second.nodo->valor) return true;
                    return false;
                };
        };
        ArbolH();
        ArbolH(char caracter, float valor);
        void uni(vector<NodoDTO> &arboles);
        string codificar(char caracter);
        string  reducirArbol();
        void imprimirAlfabeto();
        char decodificar(ifstream &archvio, char &caracter);
        void print();
        void generarArbol(string cod);
        virtual ~ArbolH();
    protected:
    private:
        Nodo * root;
        map<char, Nodo *> hojas;
        int siz;
        void _uni(Nodo * root2);
};

void ArbolH::Nodo::_generarNodo(string::iterator &iter, string::iterator endl,map<char, Nodo *> &hojas){
    if(iter == endl) return;
    if((*iter) != '0' and (*iter) != '1'){
        this->caracter = *iter;
```

```
        iter++;
        hojas[caracter] = this;
        return;
    }
    if((*iter) == '0'){
        hijos[0] = new Nodo();
        hijos[0]->padre = this;
        iter++;
        hijos[0]->_generarNodo(iter,endI,hojas);
    }
    if((*iter) == '1'){
        hijos[1] = new Nodo();
        hijos[1]->padre = this;
        iter++;
        hijos[1]->_generarNodo(iter,endI,hojas);
    }
}

void ArbolH::generarArbol(string cod){
    auto iter = cod.begin();
    root = new Nodo();
    root->_generarNodo(iter,cod.end(),hojas);
}

void ArbolH::imprimirAlfabeto(){
    for(auto iter = hojas.begin(); iter != hojas.end(); ++iter){
        cout<<"ALFB->"<<iter->first<<endl;
    }
}

void ArbolH::Nodo::_reducirNodo(string &res){
    char a = caracter;
    if(caracter != CARACTER_NULL){
        res.insert(res.end(),caracter);
        return;
    }
    res.insert(res.end(),'0');
    hijos[0]->_reducirNodo(res);
    res.insert(res.end(),'1');
    hijos[1]->_reducirNodo(res);
}

string ArbolH::reducirArbol(){
    string res;
    if(!root) return res;
    root->_reducirNodo(res);
    return res;
}

bool ArbolH::Nodo::buscarHijo(Nodo *&hijo){
    if(hijos[0] == hijo) return false;
    return true;
}

char ArbolH::decodificar(ifstream &archivo, char &caracter){
    Nodo * temp = root;
    while(temp){
        if(temp->caracter != CARACTER_NULL){
            if(temp->caracter ==     END_L) return '\n';
            return temp->caracter;
        }
        temp = temp->hijos[caracter == '1'];
        if(!archivo.get(caracter)) return CARACTER_NULL;
    }
    return CARACTER_NULL;
}

string ArbolH::codificar(char caracter){
    if(caracter == '\n') caracter = END_L;
    if(hojas.find(caracter) == hojas.end()) return  "ERROR";
    string res;
    Nodo * temp = hojas[caracter]->padre;
    Nodo * h = hojas[caracter];
    while(temp){
        char r = '0';
        if(temp->buscarHijo(h)) r = '1';
        res.insert(res.begin(),r);
        h = temp;
        temp = temp->padre;
    }
    return res;
}

void ArbolH::print(){
    ofstream archivo("eje.dot");
    list<Nodo *> nodos;
    if(root) nodos.push_back(root);
    archivo<<"digraph{"<<endl;
    for(auto iter = nodos.begin(); iter != nodos.end(); iter++){
        archivo<<(*iter)->valor<<endl;
        if((*iter)->hijos[0]){
            archivo<<(*iter)->valor<<"->"<<(*iter)->hijos[0]->valor<<endl;
            nodos.push_back((*iter)->hijos[0]);
        }
```

```
        if((*iter)->hijos[1]){
            archivo<<(*iter)->valor<<"->"<<(*iter)->hijos[1]->valor<<endl;
            nodos.push_back((*iter)->hijos[1]);
        }
    }
    archivo<<"}";
    system("dot -Tpdf eje.dot -o eje.pdf");
}

void ArbolH::_uni(Nodo * root2){
    Nodo * nuevo = new Nodo(this->root->valor + root2->valor);
    this->root->padre = nuevo;
    root2->padre = nuevo;
    nuevo->hijos[0] = this->root;
    nuevo->hijos[1] = root2;
    root = nuevo;
}

void ArbolH::uni(vector<NodoDTO> &arboles){
    if(!root){
        root = getMin(arboles).nodo;
        if(root->caracter == '\n') root->caracter = END_L;
        hojas[root->caracter] = root;
        deleteMin(arboles);
    }
    Nodo * n = getMin(arboles).nodo;
    if(n->caracter == '\n') n->caracter = END_L;
    _uni(n);
    if(n->caracter != CARACTER_NULL) hojas[n->caracter] = n;
    deleteMin(arboles);
    minHeapinsert(arboles,NodoDTO(root));
    if(arboles.size() != 1){
        root = nullptr;
        uni(arboles);
    }
}

ArbolH::Nodo::Nodo(float valor){
    hijos[0] = nullptr;
    hijos[1] = nullptr;
    padre = nullptr;
    this->caracter = CARACTER_NULL;
    this->valor = valor;
}

ArbolH::Nodo::Nodo(char caracter, float valor){
    hijos[0] = nullptr;
    hijos[1] = nullptr;
    padre = nullptr;
    this->caracter = caracter;
    this->valor = valor;
}

ArbolH::Nodo::Nodo(){
    hijos[0] = nullptr;
    hijos[1] = nullptr;
    padre = nullptr;
    caracter = CARACTER_NULL;
    valor = -1;
}

ArbolH::ArbolH(char caracter, float valor){
    root = new Nodo(caracter,valor);
    siz = 1;
}

ArbolH::ArbolH(){
    root = nullptr;
    siz = 0;
}

ArbolH::~ArbolH(){

}

#endif // ARBOLH_H
```

2.2. main.h

```
#include <iostream>
#include "ArbolH.h"
#include "map"

using namespace std;

void comprimir(string file){
```

```
    ifstream archivo(file);
    if(archivo.fail()){
        cout<<"EL archivo no se puede abrir"<<endl;
        return;
    }
    map<char,int> alfabeto;
    float total = 0;
    char caracter;
    while(archivo.get(caracter)){
        if(alfabeto.find(caracter) == alfabeto.end()){
            alfabeto[caracter] = 1;
        }
        else{
            int temp = alfabeto[caracter];
            alfabeto[caracter] = temp + 1;
        }
        total++;
    }
    vector<ArbolH::NodoDTO> nodos;
    for(auto iter = alfabeto.begin(); iter != alfabeto.end(); ++iter){
        cout<<"ALFA->"<<iter->first<<endl;
        cout<<iter->first<<endl;
        cout<<iter->second / total<<endl;
        minHeapinsert(nodos,ArbolH::NodoDTO(new ArbolH::Nodo(iter->first,iter->second / total)));
    }
    ArbolH arbolito;
    arbolito.uni(nodos);
    arbolito.imprimirAlfabeto();
    archivo.close();
    ifstream archivo2(file);
    ofstream salida("salida.txt");
    string ar = arbolito.reducirArbol();
    salida<<ar<<endl;
    while(archivo2.get(caracter)){
        string cod = arbolito.codificar(caracter);
        cout<<caracter<<endl;
        cout<<cod<<endl;
        salida<<cod;
    }
    archivo2.close();
    salida.close();
}

void descomprimir(string file){
    ifstream archivo("salida.txt");
    if(archivo.fail()){
        cout<<"El archvibo no puede abrirse"<<endl;
        return;
    }
    char caracter[200];
    archivo.getline(caracter,200);
    ArbolH arbolito;
    string cod(caracter);
    arbolito.generarArbol(cod);
    arbolito.imprimirAlfabeto();
    ofstream archivo2(file);
    char c = 'a';
    char ca;
    archivo.get(c);
    while(true){
        c = arbolito.decodificar(archivo,ca);
        if(c == CARACTER_NULL) break;
        cout<<c<<endl;
        archivo2<<c;
    }
    archivo2.close();
}

int main()
{
    comprimir("texto11.txt");
    descomprimir("texto2.txt");
}
```

3. Ejemplo

El archivo que vamos a comprimir es el siguiente:

```
Hola mundo, me llamo Chris.
Este es una archivo un poquito mas grande que se va a comprimir.
Ojala el programa este lo suficiente muy bien hecho para soportar el archivo.
Saludos; :)
```

El archivo resultante es el siguiente:

```
00000000000000000000000000000000 1y1d1C1g1f1E1;1q1:1)1b1O1j1S1v1.1H1,1^1t1c1h1p1n1u1l1m1i1s1r1o1e1a
000000000000000010010000000110000000000000000000000000000000000000000100000000100000000010000000000000
000000000000000000010010000000000000000010000000000000000000000000000000000000000000101000000000000000000000
000000000000000000010000000110000001001000000000000000000000000000000000000000000000000000000000000000
010000000000001000100000010000100000000000000000100000000000000100000000000000000000000000000100001000000
000000010100000000000000000000000000000000001000010000000000000000000000000000000000000000000001000000000
110000000000000000000000000000000000010001000000000000100000000000100000100000000000000000001001000000000
000000000000000000000000000000000001000000000100000000000000000000000000000000000000000000001001000000000
000000000000000010000000001000001000000000000010010000000000000000000000000000000000000000001100001000000
000000000000000000000000000000000000000000000000000000000100011000000000100000000000000000000000000000000
010100000000000000000000000000000000000000000000000000000000000001000000000101000000000000000000000000000
000000000010100000000000000000000000000000000000000000000000000000001100000000000000000000000000000000010
000000000000000000000000000000000000000000000001001000000010000000000100010000010000001000001000100000000
000000000100000000000000010000000000000000000000100000000000000000000110000000110000000000000000000000000
000000000001000000010000000000000000000000000000000000000000000001000100100000000000000000000000000000001
000110000000110000000000000000000000000000000100001000000000000001010000000000000000000000000000000000000
000000001001000000000000000000000000000000000000100000000100000000000000000000000000000001000001000000
000001000001010000000000100000000000000010100000000000000000000000000000000000000000010000000001000000000000
000000000000000000001000000000000000000000000000000000000000000000000000000001000001010000000000001000000
000000000000000000000000000000000000000010100000000000001000000000001001000000000000000000000000000000000
000000000001100011000000000000000000000000000000000000000010010000000000100100010000000000000011000100000
000000000000000000000000000000010000000100000000000000000000000000000000000100010000000000000010000000000000
000001000000000000000000100100000000000000000010000000000000001000000000000000000011000000010000000100
000000000000000000000000000000100100001000000000000000000000000000000010000000000000000000000000000000000000
0000000000000000000001000000000000000000000000001000000000000001
```