**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «динамическое Хаффмана»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Нгуен Ш. Х. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

**Цель работы**

Ознакомиться с динамическим алгоритмом кодирования Хаффмана. Изучить особенности его реализации на языке программирования C++. Разработать программу визуализирующую данный алгоритм.

**Задание**

Декодирование: динамическое Хаффмана

**Основные теоретические положения**

Адаптивный алгоритм Хаффмана является модификацией обычного алгоритма Хаффмана сжатия сообщений. Он позволяет не передавать таблицу кодов и ограничиться одним проходом по сообщению, как при кодировании, так и при декодировании. Суть адаптивного алгоритма состоит в том, что при каждом сопоставлении символу кода изменяется внутренний ход вычислений так, что в следующий раз этому же символу может быть сопоставлен другой код, т.е. происходит адаптация алгоритма к поступающим для кодирования символам. В адаптивном алгоритме сжатия Хаффмана используется упорядоченное бинарное дерево. Бинарное дерево называется упорядоченным, если его узлы могут быть перечислены в порядке не убывания веса. Перечисление узлов происходит по ярусам снизу-вверх и слева-направо в каждом ярусе. Узлы, имеющие общего родителя, находятся рядом на одном ярусе. Алгоритм динамического кодирования Хаффмана:

1.Элементы входного сообщения считываются побайтно.

2. Если входной символ присутствует в дереве, в выходной поток записывается код, соответствующий последовательности нулей и единиц, которыми помечены ветки дерева, при проходе от корня дерева к данному листу. Вес данного листа увеличивается на 1. Веса узлов-предков корректируются. Если дерево становится неупорядоченным – упорядочивается

3. Если очередной символ, считанный из входного сообщения при сжатии, отсутствует в дереве, в выходной поток записывается набор нулей и единиц, которыми помечены ветки бинарного дерева при движении от корня к escapeсимволу, а затем 8 бит ASCII-кода нового символа. В дерево вместо escapeсимвола добавляется ветка: родитель, два потомка. Левый потомок становится escape-символом, правый - новым добавленным в дерево символом. Веса узловпредков корректируются, а дерево при необходимости упорядочивается.

**Выполнение**

Структура MinHeapNode и MinHeap :

typedef struct MinHeapNode{

char data;

unsigned freq;

MinHeapNode \*left, \*right;

}MinHeapNode;

typedef struct MinHeap{

unsigned size;

unsigned capacity;

MinHeapNode\*\* array;

}MinHeap

Таблица 1 – Основные методы класса Huffman

|  |  |
| --- | --- |
| Метод | Назначение |
| Minheapnode\* newnode(char data, unsigned freq); | Создать новый узел |
| Minheap\* createminheap(unsigned capacity); | Создать новый minheap |
| Void swapminheapnode( minheapnode\*\* a, minheapnode\*\* b); | Поменяйте местами содержимое двух minheap |
| Void minheapify(minheap\* minheap, int idx); | Создать ветви дерева по порядку |
| Int issizeone(minheap\* minheap); | Проверьте size = 1 |
| Minheapnode\* extractmin(minheap\* minheap) ; | Принимать minheap |
| Void insertminheap( minheap\* minheap, minheapnode\* minheapnode); | Вставить minheap |
| Void buildminheap( minheap\* minheap); | Построить minheap |
| Int isleaf( minheapnode\* root); | Проверьте, является ли этот узел листом |
| Minheap\* createandbuildminheap(char data[], int freq[], int size); | Инициализировать и построить minheap |
| Minheapnode\* buildhuffmantree(string input); | Построить дерево Huffman |
| String decode(minheapnode\* root, string s,string youranswer); | Декодирование строки s |
| Void take\_data\_freq(string input, char\* data, int\* freq, int\* size); | Возвращает данные, сохраненные в data |
| String printcodes( minheapnode\* root, int arr[], int top); | Возвращает arr, содержащую строку символов 0 или 1 для отображения на экране |
| String printarr(int arr[], int n); | Возвращает arr, содержащую строку символов для отображения на экране |
| String take\_encode(minheapnode\* root, int \*arr, int top,string input); | Возвращает arr, содержащую строку символов для отображения на экране |
| String take\_encode\_3(int \*arr, int n); | Возвращает arr, содержащую строку символов для отображения на экране |
| String take\_encode\_2(minheapnode\* root, int \*arr, int top, char input); | Возвращает arr, содержащую строку символов для отображения на экране |

**Оценка эффективности алгоритма**

Сложность данного алгоритма зависит от реализации функции перестройки дерева. В ходе тестирования было выяснено, что данный алгоритм имеет сложность O(n\*logn), данные тестирования в виде графика изображены на График.1.

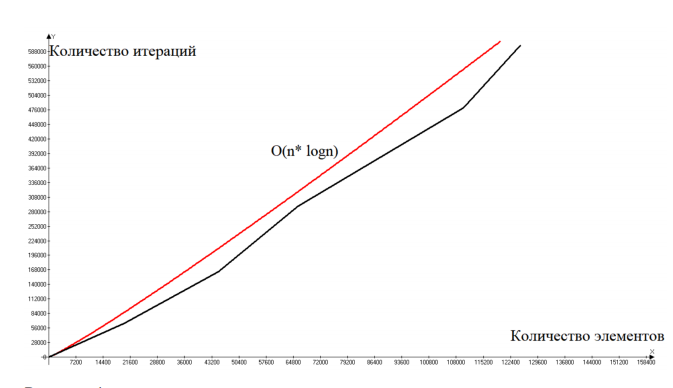
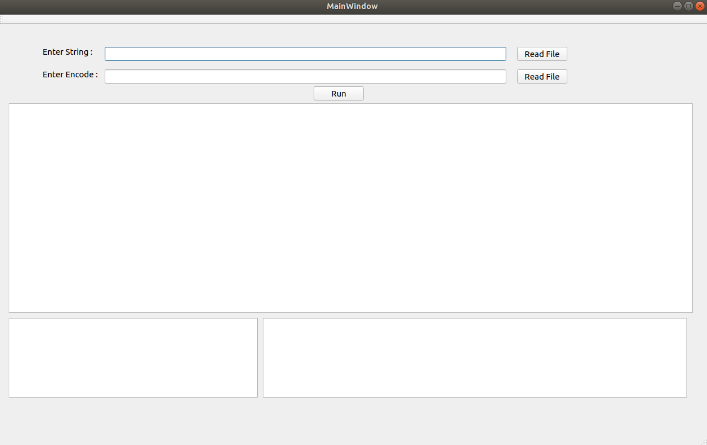
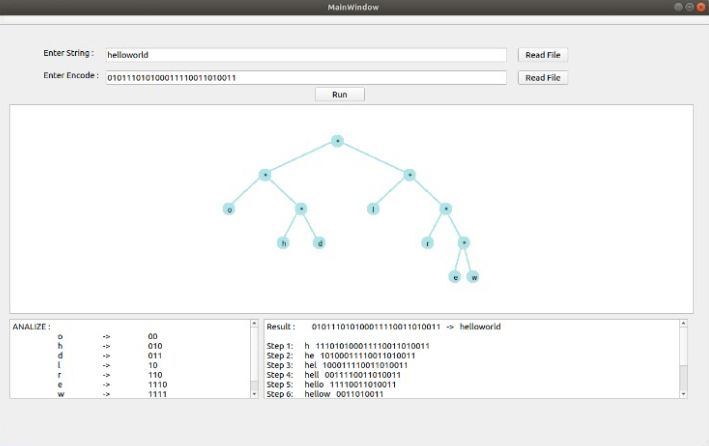
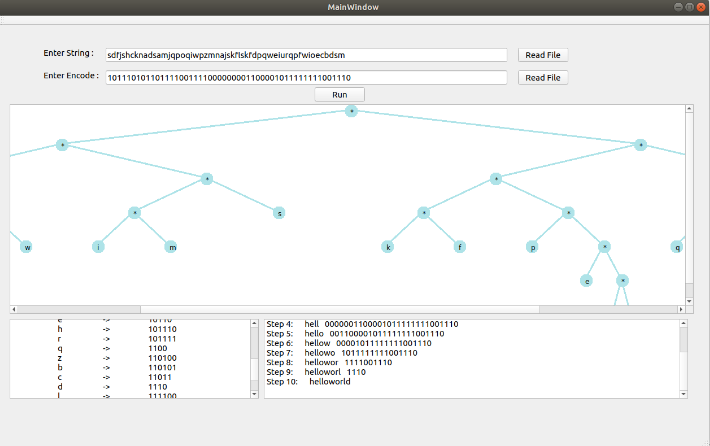


График 1 — сложность алгоритма

**Тестирование программы**







**Выводы**

В ходе выполнения лабораторной работы была написана программа, кодирующая заданное сообщение с помощью алгоритма динамического кодирования Хаффмана. Также был реализован интерфейс, изображающий упорядоченное бинарное дерево, который позволяет увидеть работу алгоритма на каждом шаге.

**Приложение А**

**Файл main.cpp**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**Файл Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <huffman.h>

#include <fstream>

#include <string>

#include <QGraphicsScene>

#include <QPainter>

#include <QFileDialog>

#include <QMessageBox>

#include <QString>

#include <QTextEdit>

#include "huffman.h"

#include "console.h"

using namespace std;

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_run\_clicked();

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

private:

Ui::MainWindow \*ui;

QGraphicsScene \*scene;

console cuaso;

QString input;

QString encode;

};

#endif // MAINWINDOW\_H

**Файл Mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_run\_clicked()

{

scene = new QGraphicsScene(ui->graphicsView);

scene->clear();

input = ui->lineEdit->text();

encode = ui->lineEdit\_2->text();

int flagErr = cuaso.checkErr(input,encode);

if(flagErr == 1){

QMessageBox::critical(this,"ERROR!","Text contains Russian");

return;

}

else if (flagErr == 2) {

QMessageBox::critical(this,"ERROR!","The text must contain only 0 or 1 characters");

return;

}

else if(flagErr == 3){

QMessageBox::critical(this,"ERROR!","missing input data");

return;

}

else {

string input\_2 = input.toStdString();

string encode\_2 = encode.toStdString();

int fl;

string result = cuaso.result(input\_2,encode\_2,&fl);

string analize = cuaso.analize(input\_2);

if(fl == 1){

QMessageBox::critical(this,"ERROR!","Encode incorrect!");

return;

}

ui->textEdit->setText(QString::fromStdString(analize));

ui->textEdit\_2->setText(QString::fromStdString(result));

scene = cuaso.Paint(input\_2);

ui->graphicsView->setScene(scene);

}

}

void MainWindow::on\_pushButton\_clicked()

{

QString fileName = QFileDialog::getOpenFileName(this, tr("Choose file."), QDir::homePath(), tr("\*.txt"));

if (QString::compare(fileName, QString()) != 0)

{

std::ifstream f(qPrintable(fileName), std::ios::in);

std::string out;

getline(f, out);

f.close();

ui->lineEdit->setText(QString::fromStdString(out));

}

}

void MainWindow::on\_pushButton\_2\_clicked()

{

QString fileName = QFileDialog::getOpenFileName(this, tr("Choose file."), QDir::homePath(), tr("\*.txt"));

if (QString::compare(fileName, QString()) != 0)

{

std::ifstream f(qPrintable(fileName), std::ios::in);

std::string out;

getline(f, out);

f.close();

ui->lineEdit\_2->setText(QString::fromStdString(out));

}

}

/\*

bool MainWindow::readInput(){

input = ui->lineEdit->text();

encode = ui->lineEdit\_2->text();

int fl = checkinput(input,encode);

if (input.size()==0){

QMessageBox::critical(this,"ERROR!","No massage");

return false;

}

if (encode.size()==0){

QMessageBox::critical(this,"ERROR!","No massage");

return false;

}

if (fl == 1){

QMessageBox::critical(this,"ERROR!","Text contains Russian");

return false;

}

if (fl == 2){

QMessageBox::critical(this,"ERROR!","The text must contain only 0 or 1 characters");

return false;

}

return true;

}

\*/

**Файл huffman.cpp**

#include "huffman.h"

Huffman::Huffman()

{

}

MinHeapNode\* Huffman::newNode(char data, unsigned freq)

{

MinHeapNode\* temp = new MinHeapNode;

temp->left = temp->right = nullptr;

temp->data = data;

temp->freq = freq;

return temp;

}

MinHeap\* Huffman::createMinHeap(unsigned capacity) {

MinHeap\* minHeap = new MinHeap;

// current size is 0

minHeap->size = 0;

minHeap->capacity = capacity;

minHeap->array = (MinHeapNode\*\*)malloc(capacity \* sizeof(MinHeapNode\*));

return minHeap;

}

void Huffman::swapMinHeapNode( MinHeapNode\*\* a, MinHeapNode\*\* b){

MinHeapNode\* t = \*a;

\*a = \*b;

\*b = t;

}

void Huffman::minHeapify(MinHeap\* minHeap, int idx){

int smallest = idx;

int left = 2 \* idx + 1;

int right = 2 \* idx + 2;

if (left <static\_cast<int>(minHeap->size) && minHeap->array[left]->freq < minHeap->array[smallest]->freq)

smallest = left;

if (right < static\_cast<int>(minHeap->size) && minHeap->array[right]->freq < minHeap->array[smallest]->freq)

smallest = right;

if (smallest != idx) {

swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);

minHeapify(minHeap, smallest);

}

}

int Huffman::isSizeOne(MinHeap\* minHeap){

return (minHeap->size == 1);

}

MinHeapNode\* Huffman::extractMin(MinHeap\* minHeap) {

MinHeapNode\* temp = minHeap->array[0];

minHeap->array[0]

= minHeap->array[minHeap->size - 1];

--minHeap->size;

minHeapify(minHeap, 0);

return temp;

}

void Huffman::insertMinHeap( MinHeap\* minHeap, MinHeapNode\* minHeapNode) {

++minHeap->size;

int i = minHeap->size - 1;

while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {

minHeap->array[i] = minHeap->array[(i - 1) / 2];

i = (i - 1) / 2;

}

minHeap->array[i] = minHeapNode;

}

void Huffman::buildMinHeap( MinHeap\* minHeap){

int n = minHeap->size - 1;

int i;

for (i = (n - 1) / 2; i >= 0; --i)

minHeapify(minHeap, i);

}

int Huffman::isLeaf( MinHeapNode\* root){

return !(root->left) && !(root->right);

}

MinHeap\* Huffman::createAndBuildMinHeap(char data[], int freq[], int size){

MinHeap\* minHeap = createMinHeap(size);

for (int i = 0; i < size; ++i)

minHeap->array[i] = newNode(data[i], freq[i]);

minHeap->size = size;

buildMinHeap(minHeap);

return minHeap;

}

MinHeapNode\* Huffman::buildHuffmanTree(string input){

int len = static\_cast<int>(input.length());

char \*data = new char[len];

int \*freq = new int[len];

int size = 0;

take\_data\_freq(input,data,freq, &size);

MinHeapNode \*left, \*right, \*top;

// Step 1: Create a min heap of capacity

// equal to size. Initially, there are

// modes equal to size.

MinHeap\* minHeap = createAndBuildMinHeap(data, freq, size);

// Iterate while size of heap doesn't become 1

while (!isSizeOne(minHeap)) {

// Step 2: Extract the two minimum

// freq items from min heap

left = extractMin(minHeap);

right = extractMin(minHeap);

// Step 3: Create a new internal

// node with frequency equal to the

// sum of the two nodes frequencies.

// Make the two extracted node as

// left and right children of this new node.

// Add this node to the min heap

// '$' is a special value for internal nodes, not used

top = newNode('\*', left->freq + right->freq);

top->left = left;

top->right = right;

insertMinHeap(minHeap, top);

}

// Step 4: The remaining node is the

// root node and the tree is complete.

return extractMin(minHeap);

}

string Huffman::printArr(int \*arr, int n)

{

string out;

int i;

for (i = 0; i < n; ++i){

if(arr[i] == 1 )

out.append(1,'1');

else if(arr[i] == 0)

out.append(1,'0');

}

out+= "\n";

return out;

}

string Huffman::printCodes( MinHeapNode\* root, int \*arr, int top){

string out;

if (root->left) {

arr[top] = 0;

out += printCodes(root->left, arr, top + 1);

}

if (root->right) {

arr[top] = 1;

out += printCodes(root->right, arr, top + 1);

}

if (isLeaf(root)) {

out+= '\t';

out += root->data;

out +='\t';

out += "->";

out += '\t';

out += printArr(arr, top);

}

return out;

}

string Huffman::decode(MinHeapNode\* root, string s, int \*fl)

{

\*fl = 1;

string ans = "";

string step ;

int id = 1;

struct MinHeapNode\* curr = root;

int len = static\_cast<int>(s.size());

for (int i=0; i < len; i++)

{

if (s[i] == '0')

curr = curr->left;

else

curr = curr->right;

if (curr->left==nullptr and curr->right==nullptr)

{

if(i == len -1)

\*fl = 0;

step += "Step " + std::to\_string(id) + ": ";

id++;

ans += curr->data;

step += ans;

step += " ";

step.append(s,i+1,len-i-1);

step += '\n';

curr = root;

}

}

string buffer = "Result :";

buffer += '\t';

buffer += s + " -> ";

ans = ans + '\n';

ans = buffer + ans;

ans += '\n';

ans += step;

return ans;

}

void Huffman::take\_data\_freq(string input, char\* data, int\* freq, int\* size){

int id = 1;

data[0] = input[0];

for(int i = 0; i <static\_cast<int>(input.length()) ; i++){

int flag = 0;

for(int j = 0; j < id; j++){

if(data[j] == input[i]){

flag = 1;

break;

}

}

if(flag == 0){

data[id] = input[i];

id++;

}

}

\*size = id;

data[id] = '\0';

freq[id] = '\0';

for(int i = 0; i < id; i++){

freq[i] = count(input.begin(),input.end(),data[i]);

}

return;

}

**Файл huffman.h**

#ifndef HUFFMAN\_H

#define HUFFMAN\_H

#include <iostream>

#include <cstdlib>

#include <string>

#include <algorithm>

using namespace std;

#define MAX\_TREE\_HT 100

typedef struct MinHeapNode{

char data;

unsigned freq;

MinHeapNode \*left, \*right;

}MinHeapNode;

typedef struct MinHeap{

unsigned size;

unsigned capacity;

MinHeapNode\*\* array;

}MinHeap;

class Huffman

{

private:

public:

Huffman();

MinHeapNode\* newNode(char data, unsigned freq);

MinHeap\* createMinHeap(unsigned capacity);

void swapMinHeapNode( MinHeapNode\*\* a, MinHeapNode\*\* b);

void minHeapify(MinHeap\* minHeap, int idx);

int isSizeOne(MinHeap\* minHeap);

MinHeapNode\* extractMin(MinHeap\* minHeap) ;

void insertMinHeap( MinHeap\* minHeap, MinHeapNode\* minHeapNode);

void buildMinHeap( MinHeap\* minHeap);

int isLeaf( MinHeapNode\* root);

MinHeap\* createAndBuildMinHeap(char data[], int freq[], int size);

MinHeapNode\* buildHuffmanTree(string input);

string decode(MinHeapNode\* root, string s, int \*fl);

void take\_data\_freq(string input, char\* data, int\* freq, int\* size);

string printCodes( MinHeapNode\* root, int arr[], int top);

string printArr(int arr[], int n);

};

#endif // HUFFMAN\_H

**Файл console.cpp**

#include "console.h"

console::console()

{

}

int console::checkErr(QString input, QString encode){

if(input.size() == 0 || encode.size() == 0)

return 3;

static QString russian = RUSSIAN;

foreach(const QChar & ch, russian) {

if(input.contains(ch)) {

return 1;

}

}

for(int i = 0; i < encode.length(); i++){

if(encode[i] != '0' && encode[i] != '1')

return 2;

}

return 0;

}

int console::CountDeep(MinHeapNode \*root)

{

if (root == nullptr)

return 0;

int cl = CountDeep(root->left);

int cr = CountDeep(root->right);

return 1 + ((cl>cr)?cl:cr);

}

QGraphicsScene\* console::Paint(string data){

QGraphicsScene \*scene = new QGraphicsScene;

string input = data;

MinHeapNode \*root = huffman.buildHuffmanTree(input);

int deeptree = CountDeep(root);

scene->clear();

QPen pen;

QColor color;

color.setRgb(174, 227, 232);

pen.setColor(color);

QBrush brush (color);

QFont font("Helvetica [Cronyx]", 9, 10,false);

pen.setWidth(3);

int wDeep = static\_cast<int>(pow(2, deeptree));

int hDelta = 60;

int wDelta = 8;

int width = (wDelta\*wDeep)/2;

treePainter(scene, root, width/2, hDelta, wDelta, hDelta, pen, brush, font, wDeep);

return scene;

}

void console::treePainter(QGraphicsScene \*&scene, MinHeapNode \*root, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth)

{

if (root == nullptr)

return ;

string out;

out += root->data;

QGraphicsTextItem \*textItem = new QGraphicsTextItem;

textItem->setPos(w, h);// set toa do (x;y) cua nut

textItem->setPlainText(QString::fromStdString(out));

textItem->setFont(font);

scene->addEllipse(w-wDelta/2, h, wDelta\*5/2, wDelta\*5/2, pen, brush); // Tạo hình tròn của các nút

if (root->left != nullptr)

scene->addLine(w+wDelta/2, h+wDelta, w-(depth/2)\*wDelta+wDelta/2, h+hDelta+wDelta, pen);

if (root->right != nullptr)

scene->addLine(w+wDelta/2, h+wDelta, w+(depth/2)\*wDelta+wDelta/2, h+hDelta+wDelta, pen);

scene->addItem(textItem);

treePainter(scene, root->left, w-(depth/2)\*wDelta, h+hDelta, wDelta, hDelta, pen, brush, font, depth/2);

treePainter(scene, root->right, w+(depth/2)\*wDelta, h+hDelta, wDelta, hDelta, pen, brush, font, depth/2);

return ;

}

string console::analize(string data){

string out;

out += "ANALIZE : \n";

int ch[MAX\_TREE\_HT], top = 0;

out += huffman.printCodes(huffman.buildHuffmanTree(data),ch,top);

return out;

}

string console::result(string input,string encode, int \*fl){

return huffman.decode(huffman.buildHuffmanTree(input),encode,fl);

}

**Файл console.h**

#ifndef CONSOLE\_H

#define CONSOLE\_H

#include "huffman.h"

#include <QGraphicsScene>

#include <QGraphicsView>

#include <cmath>

#include <QGraphicsTextItem>

#include <QString>

#include <QMessageBox>

#include <QLineEdit>

#define MAX\_TREE\_HT 100

#define RUSSIAN "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцчшщъыьэюя"

class console

{

public:

console();

QGraphicsScene \*Paint(string data);

void treePainter(QGraphicsScene \*&scene, MinHeapNode \*root, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth);

int checkErr(QString input, QString encode);

int CountDeep(MinHeapNode \*root);

string analize(string data);

string result(string input,string encode, int\* fl);

Huffman huffman;

void free(MinHeap \* a, MinHeapNode\* b);

};

#endif // CONSOLE\_H