**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №2**

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

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9301 |  | Серов А.В. |
| Преподаватель |  | Тутуева А. В. |

Санкт-Петербург

2021

Оглавление

[Постановка задачи 2](#_Toc74118747)

[Описание реализуемых алгоритмов и используемых структур 2](#_Toc74118748)

[Оценка временной сложности 2](#_Toc74118749)

[Unit-тесты 2](#_Toc74118750)

[Код 3](#_Toc74118751)

[Результат 8](#_Toc74118752)

[Вывод 8](#_Toc74118753)

# Постановка задачи

Реализовать кодирование и декодирование по алгоритму Хаффман входной строки, вводимой через консоль

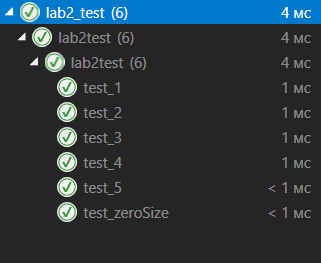
# Описание реализуемых алгоритмов и используемых структур

Построение дерева Хаффмана происходит в конструкторе класса HuffmanAlg. Для построения используется очередь с приоритетом, где приоритетом является частота повторений символа. После построения дерева, соответствия код-символ заносятся в красно-черное дерево. Для обхода по листам дерева Хаффмана используется стек. При кодировании строки в красно-черном дереве происходит поиск по символу соответствующего кода. Для декодирования происходит спуск по дереву Хаффмана.

# Оценка временной сложности

|  |  |
| --- | --- |
| **Название метода** | **Сложность** |
| HuffmanAlg() | O(n2) |
| Code() | O(n\*log(n)) |
| Decode() | O(n) |

# Unit-тесты



1. Test\_1 — тестирование при входной строке «Hello world»
2. Test\_2 — тестирование при входной строке «How are u»
3. Test\_3 — тестирование при входной строке «Where are u from»
4. Test\_4 — тестирование при входной строке «The best teacher at the Institute is Tutueva A.V.»
5. Test\_5 — тестирование при входной строке « The worst student is Alexey Serov»
6. Test\_zeroSize — тестирование при пустой входной строке

# Код

**lab\_2.cpp**

#include <iostream>

#include "HuffmanAlg.h"

int main()

{

int sizeInStr = 500;

char\* str = new char[sizeInStr];

int size;

size = 0;

char c;

do

{

cin >> noskipws >> c;

str[size] = c;

size++;

} while ((c!='.')&&((size+1) < sizeInStr));

str[size] = '\n';

double coeff = 1.0\*sizeof(char) \* size \* 8;

HuffmanAlg ForCode(str);

int sizeC;

bool\* code = ForCode.Code(size);

char\* decode = ForCode.Decode(sizeC);

for (size\_t i = 0; i < size; i++)

{

cout << code[i];

}

cout << endl;

coeff = coeff / (1 \* size);

cout << "Coefficient : " << coeff << endl;

for (size\_t i = 0; i < sizeC; i++)

{

cout << decode[i];

}

}

HuffmanAlg.h

#pragma once

#include "PriorityQueue.h"

#include "RedBlackTree.h"

#include "Stack.h"

class HuffmanAlg

{

private:

class Node

{

public:

Node(int f = 0, Node\* lC = nullptr, Node\* rC = nullptr, char l = '\n') {

frequence = f;

letter = l;

left = lC;

right = rC;

parent = nullptr;

};

~Node() {};

int frequence;

char letter;

Node\* left;

Node\* right;

Node\* parent;

};

void copyBoolArray(bool\* copy, bool\* str, int size) {

for (size\_t i = 0; i < size; i++)

copy[i] = str[i];

};

Node\* root;

char\* str;

size\_t sizeStr;

bool\* readyCode;

size\_t sizeReadyCode;

struct strArr

{

bool\* str = NULL;

int size = 0;

};

RedBlackTree<char, strArr\*> codeLetter;

public:

HuffmanAlg(char\* \_str){

if (\_str == NULL)

{

throw "null argument";

}

codeLetter = RedBlackTree<char, strArr\*>();

str = \_str;

sizeStr = 0;

for (size\_t i = 0; str[i]!='\n'; i++)

sizeStr++;

if (sizeStr > 1)

{

PriorityQueue<Node\*> PQ;

Node\* bufNode\_1;

Node\* bufNode\_2;

int chars[256] = { 0 };

for (int i = 0; str[i] != '\n'; i++)

chars[str[i] + 128]++;

int countLetter = 0;

cout << "Frequency table: " << endl;

for (int i = 0; i < 256; i++)

if (chars[i] != 0) {

countLetter++;

cout << (char)(i - 128) << " : " << chars[i] << endl;

bufNode\_1 = new Node(chars[i], nullptr, nullptr, (i - 128));

PQ.push(bufNode\_1, chars[i]);

}

int bufInt;

bufInt = PQ.tail\_priority();

bufNode\_1 = PQ.pop();

for (size\_t i = 0; (i < countLetter) && (!PQ.isEmpty()); i++)

{

bufInt += PQ.tail\_priority();

bufNode\_2 = PQ.pop();

root = new Node(bufInt, bufNode\_1, bufNode\_2);

bufNode\_1->parent = root;

bufNode\_2->parent = root;

PQ.push(root, root->frequence);

bufInt = PQ.tail\_priority();

bufNode\_1 = PQ.pop();

}

bool\* bufStr = new bool[countLetter / 2 + 1];

memset(bufStr, 0, countLetter / 2 + 1);

size\_t indexStr = 0;

unsigned int bufCodeLetter = 0;

Stack<Node\*> dft;

if (root->right != nullptr)

{

dft.push(root->right);

}

bufNode\_1 = root;

strArr\* bufStrArr = new strArr;

cout << "Code table: " << endl;

while (!dft.isEmpty())

{

while (bufNode\_1->left != nullptr)

{

bufStr[indexStr] = 0;

indexStr++;

bufNode\_1 = bufNode\_1->left;

if (bufNode\_1->right != nullptr)

{

dft.push(bufNode\_1->right);

}

}

bufStrArr = new strArr;

bufStrArr->str = new bool[indexStr];

bufStrArr->size = indexStr;

copyBoolArray(bufStrArr->str, bufStr, indexStr);

cout << bufNode\_1->letter << " : ";

for (size\_t i = 0; i < indexStr; i++)

{

cout << bufStrArr->str[i];

}

cout << endl;

codeLetter.insert(bufNode\_1->letter, bufStrArr); indexStr--;

while ((bufNode\_1->parent != nullptr) && (bufNode\_1->parent->right != dft.get\_last())) {

bufNode\_1 = bufNode\_1->parent;

indexStr--;

}

bufNode\_1 = dft.pop();

if (bufNode\_1->right != nullptr)

{

dft.push(bufNode\_1->right);

}

bufStr[indexStr] = 1;

indexStr++;

}

bufStrArr = new strArr;

bufStrArr->size = indexStr;

bufStrArr->str = new bool[indexStr];

copyBoolArray(bufStrArr->str, bufStr, indexStr);

cout << bufNode\_1->letter << " : ";

for (size\_t i = 0; i < indexStr; i++)

{

cout << bufStrArr->str[i];

}

cout << endl;

codeLetter.insert(bufNode\_1->letter, bufStrArr);

readyCode = NULL;

sizeReadyCode = 0;

delete bufStr;

}

else {

cout << "Frequency table: " <<str[0]<< " " << 1 << endl;

cout << "Code table: " << str[0] << " " << 0 << endl;

root = new Node(1, nullptr, nullptr, str[0]);

sizeReadyCode = 0;

readyCode = NULL;

strArr\* bufStrArr = new strArr;

bufStrArr->str = new bool[1];

bufStrArr->str[0] = false;

bufStrArr->size = 1;

codeLetter.insert(str[0], bufStrArr);

}

}

bool\* Code(int& codeSize) {

bool\* copyStr;

codeSize = 0;

for (int i = 0; str[i] != '\n'; i++) {

codeSize += codeLetter.find(str[i])->size;

}

sizeReadyCode = codeSize;

readyCode = new bool[codeSize];

size\_t indexStr=0;

for (int i = 0; str[i] != '\n'; i++) {

copyStr = codeLetter.find(str[i])->str;

for (size\_t j = 0; j< codeLetter.find(str[i])->size; j++) {

readyCode[indexStr] = copyStr[j];

indexStr++;

}

}

return readyCode;

}

char\* Decode(int& sizeStrDecode) {

char\* decodeStr = new char[sizeStr+1];

sizeStrDecode = sizeStr;

if (sizeStr==1)

{

decodeStr[0] = root->letter;

decodeStr[0] = '\n';

}

else {

size\_t indexCode = 0;

Node\* curNode = root;

size\_t indexDecode = 0;

while (indexCode < sizeReadyCode)

{

if ((readyCode[indexCode] == 0) && (curNode->left != nullptr)) {

curNode = curNode->left;

indexCode++;

}

else if ((readyCode[indexCode] == 1) && (curNode->right != nullptr)) { curNode = curNode->right;

indexCode++;

}

else {

decodeStr[indexDecode] = curNode->letter;

indexDecode++;

curNode = root;

}

}

decodeStr[indexDecode] = curNode->letter;

decodeStr[indexDecode+1] = '\n';

}

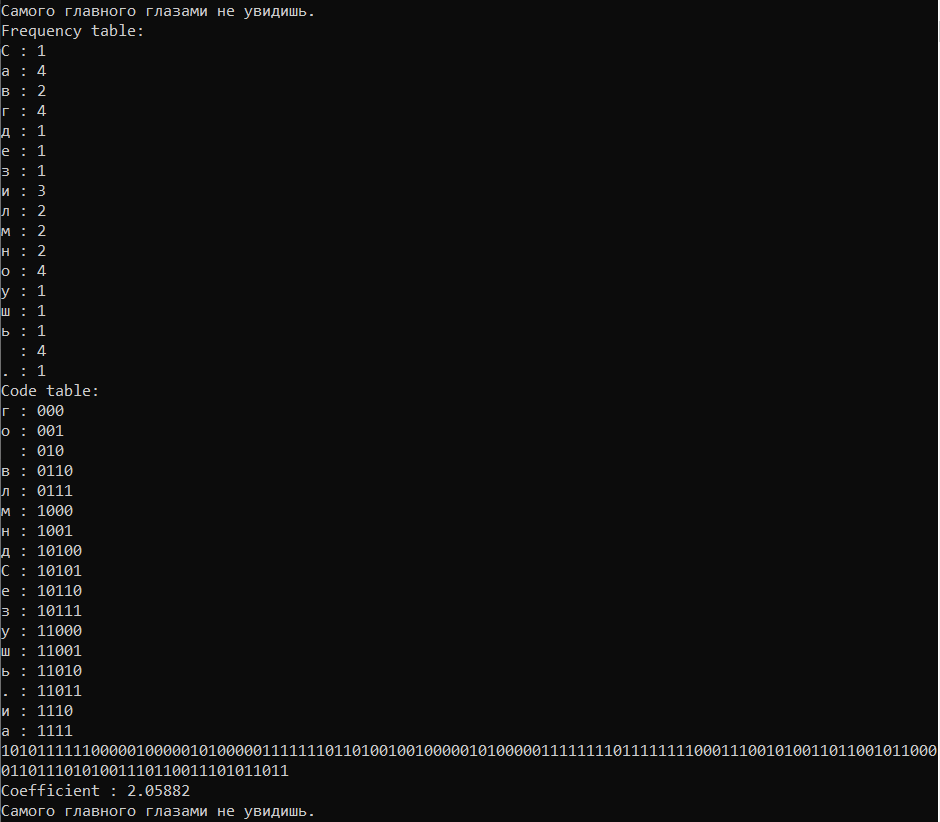
return decodeStr;

}

};

#include "PriorityQueue.cpp"

# Результат



Исходя из результатов мы видим закодированное предложение и коэффициент сжатия.

# Вывод

В ходе выполнения лабораторной работы мы изучили алгоритм хаффмана и применили его на практике.