**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»

**вариант 1**

Студент гр. 8301 Чередилина Е.А.

Преподаватель Тутуева А.В.

Санкт-Петербург

2020

**Цель работы**

Реализовать кодирование и декодирование по алгоритму Хаффмана.

# Описание реализуемых и вспомогательных классов

Класс HoffmanCodingAlgorythm, содержит дружественный класс Node, который в свою очередь имеет три публичных поля: указатели на left (Указатель на левый элемент), right (Указатель на правый элемент) и переменную value класса Pair (вспомогательный класс состоящий из двух публичных шаблонных полей). А сам класс имеет поля Node\* treeRoot (вершина дерева) и frequencyList (список частот символов в дереве). Сам класс подразумевает собой дерево кодирования построенное на основе алгоритма кодирования Хаффмана. Класс содержит следующие методы:

* *Конструктор* – который получает на вход строку, и строит список типа Node символов и их частоты появления в строке. Потом этот список передаётся в функцию constructTree.
* constructTreе– функция построения дерева кодирования. Само добавление проходит по следующему алгоритму: сортируем список в порядке возрастания, берём первые два элемента, удаляем их из списка и создаем дерево, корнем которого становится элемент с суммой частот этих элементов и пустым полем, а листьями становятся элементы, частоты которых складывались: слева с меньшей частотой, справа – с большей. Вставляем это дерево в список так, чтобы соблюдался порядок возрастания. И так далее в цикле, пока список не будет состоять из одного элемента.
* getTreeAsCode - функция получения ассоциативного массива символов и их кодов. Для этого вызывается функция codeGen.
* codeGen – функция заполнения ассоциативного массива, ключом которого является символ, присутствующий в строке, а значением код символа (каждый символ – это элемент, не имеющий сыновей). Методом Хаффмана мы определяем код каждого элемента и записываем его в ассоциативный массив.
* decode - функция декодирования закодированной строки. Мы идём по закодированной строке и, если встречаем 0, то идём влево, иначе – вправо, и, если встречаем символ, то возвращаемся в корень, не меняя позиции продвижения по строке, тем самым декодируем закодированную строку.

# Оценка временной сложности алгоритмов

* constructTree - O(NlogN)
* getTreeAsCode– O(N)
* codeGen – O(N)
* decode - O(NlogN)

# Описание реализованных unit-тестов

Я реализовала unit-тесты проверяют правильность кодирования информации методом Хаффмана. В них мы рассматриваем такие случаи, как предложение, слова из разных символов и слова из повторяющихся символов.

# Листинг №1 – код программы

#pragma once

#include"List.h"

#include"Map.h"

#include"Pair.h"

#include<string>

class Node {

public:

Node(Pair<char, int> value = Pair<char, int>(), Node\* left = NULL, Node\* right = NULL)

:value(value), left(left), right(right) {}

Pair<char, int> value;

Node\* left;

Node\* right;

};

class HoffmanCodingAlgorythm

{

private:

Node\* treeRoot;

List<Pair<char, int>>\* frequencyList;

friend class Node;

void commitDecoding(Node\* root, string& codingString, string& decoding\_str, int& position) {

if (codingString.size() > position) {

while (root->right != NULL && root->left != NULL) {

if (codingString[position] == '0')

root = root->left;

else

root = root->right;

position++;

}

decoding\_str += root->value.first;

if (treeRoot->left == NULL && treeRoot->right == NULL)

position++;

commitDecoding(treeRoot, codingString, decoding\_str, position);

}

}

void constructTree(List<Node>\* treeNodesList) {

treeRoot = NULL;

if (treeNodesList->get\_size() <= 1)

{

treeRoot = new Node(

Pair<char, int>(treeNodesList->at(0).value.first, treeNodesList->at(0).value.second)

);

}

else

{

while (treeNodesList->get\_size() != 0) {

Node\* currentNode = new Node();

currentNode->left = new Node(treeNodesList->at(0));

currentNode->right = new Node(treeNodesList->at(1));

currentNode->value.second = treeNodesList->at(0).value.second + treeNodesList->at(1).value.second;

treeNodesList->pop\_front();

treeNodesList->pop\_front();

int i = 0;

for (; i < treeNodesList->get\_size() && treeNodesList->at(i).value.second < currentNode->value.second; i++);

if (treeNodesList->get\_size() != 0 && treeNodesList->get\_size() != i)

treeNodesList->insert(\*currentNode, i);

else if (treeNodesList->get\_size() == i && treeNodesList->get\_size() != 0)

treeNodesList->push\_back(\*currentNode);

if (treeNodesList->get\_size() == 0) {

treeRoot = currentNode;

}

}

}

}

void codeGen(Map<char, string>\* hoffman, Node\* root, string cur) {

if (treeRoot->left != NULL && treeRoot->right != NULL) {

if (root->left != NULL && root->right != NULL) {

codeGen(hoffman, root->left, cur + '0');

codeGen(hoffman, root->right, cur + '1');

}

else

{

hoffman->insert(root->value.first, cur);

}

}

else {

hoffman->insert(root->value.first, cur + '0');

}

}

public:

HoffmanCodingAlgorythm(string str) {

Map<char, int>\* frequencyMap = new Map<char, int>();

for (int i = 0; i < str.size(); i++) {

if(frequencyMap->contains(str[i]) == false) frequencyMap->insert(str[i], 1);

else frequencyMap->increment\_value(str[i]);

}

frequencyList = frequencyMap->getPairs();

frequencyList->sort();

List<Node>\* treeNodesList = new List<Node>();

for (int i = 0; i < frequencyList->get\_size(); i++)

treeNodesList->push\_back(Node(frequencyList->at(i)));

constructTree(treeNodesList);

}

List<Pair<char, int>>\* get\_list\_symbol() {

return frequencyList;

}

Map<char, string>\*& getTreeAsCode() {

Map<char, string>\* hoffman = new Map<char, string>();

string currentString;

codeGen(hoffman, treeRoot, currentString);

return hoffman;

}

string decode(string& coding\_str) {

string decoding\_str;

int pos = 0;

commitDecoding(treeRoot, coding\_str, decoding\_str, pos);

return decoding\_str;

}

};

# Листинг №2 – юнит тесты

#include "pch.h"

#include "CppUnitTest.h"

#include<string>

#include"..//DecodeHoffmanAlgorythm/HoffmanAlgo.h"

#include"..//DecodeHoffmanAlgorythm/Map.h"

#include"..//DecodeHoffmanAlgorythm/List.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitForHoffman

{

TEST\_CLASS(UnitForHoffman)

{

public:

TEST\_METHOD(TestCodingHoffman\_different\_symbol\_words)

{

string str = "Hello";

string coding\_str;

HoffmanCodingAlgorythm\* alg = new HoffmanCodingAlgorythm(str);

Map<char, string>\* hoffman = alg->getTreeAsCode();

int counter = 0;

List<Pair<char, int>>\* list\_symbol = alg->get\_list\_symbol();

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

coding\_str += hoffman->find(str[i]);

Assert::AreEqual(coding\_str, string("1011100110"));

}

TEST\_METHOD(TestCodingHoffman\_same\_symbol\_words)

{

string str = "fff";//

string coding\_str;

HoffmanCodingAlgorythm\* alg = new HoffmanCodingAlgorythm(str);

Map<char, string>\* hoffman = alg->getTreeAsCode();

int counter = 0;

List<Pair<char, int>>\* list\_symbol = alg->get\_list\_symbol();

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

coding\_str += hoffman->find(str[i]);

Assert::AreEqual(coding\_str, string("000"));

}

TEST\_METHOD(TestCodingHoffman\_text) {

string str = "it is string";//

string coding\_str;

HoffmanCodingAlgorythm\* alg = new HoffmanCodingAlgorythm(str);

Map<char, string>\* hoffman = alg->getTreeAsCode();

int counter = 0;

List<Pair<char, int>>\* list\_symbol = alg->get\_list\_symbol();

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

coding\_str += hoffman->find(str[i]);

Assert::AreEqual(coding\_str, string("101110010110001101110111100110010"));

}

};

}

# Вывод

При выполнении данной лабораторной работы я ознакомилась с методом кодирования Хаффмана, а также закрепила свои навыки в объектно-ориентированном программировании.