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»

Тема: Алгоритмы кодирования

Вариант: 1

Студент гр. 8309 Носов А.С.

Преподаватель Тутуева А.В.

Санкт-Петербург

2020

Оглавление

Постановка задачи. Описание реализуемого класса и методов

Научится кодировать и декодировать файлы с помощью алгоритмов. Реализовать кодирования и декодирование по алгоритму Хаффмина. Необходимо создать класс, способный закодировать и раскодировать текстовый файл с выводом коэффициента сжатия. К каждому публичному методу в классе необходимо создать Unit-тест.

Описание методов:

1. void encoding(): Кодировка данных путем создания списка символов с их частотами, после производится сортировка и построение дерева Хаффмина и таблицы с кодом каждого элемента.
2. void decoding(): Декодирует зашифрованный файл путем обхода двоичного дерева с корды до необходимого листа.
3. void sort(): Сортировка списка по частотам символов.
4. nodeList* findSymb(char symb): Поиск символа в списке.
5. void insertList(char symb): Вставка символа в список или повышение частоты имеющегося.
6. void huffmanTree(): Формирование дерева через список дерева с частотами.
7. void huffmanTable(): Формирование таблицы через обход дерева в ширину.
8. void findCode(char symb): Поиск кода символа.

Оценка временной сложности каждого метода

Временная сложность оценена с помощью модульных тестов.

1. void encoding(): O((N*K)+(A*B)+(C*logD)+(A+A)+N)
2. void decoding(): O(N)
3. void sort(): O(N*logN)
4. nodeList* findSymb(char symb): O(N)
5. void insertList(char symb): 1) O(1) – при вставке головы; 2) O(N) – при вставке хвоста
6. void huffmanTree(): 1) O(N*N) – в стандартном формате; 2) O(N) – при условии, что все новые узлы будут либо меньше, либо больше последнего элемента хвоста.
7. void huffmanTable(): O(N+K)
8. void findCode(char symb): O(N)

Описание реализованных unit-тестов

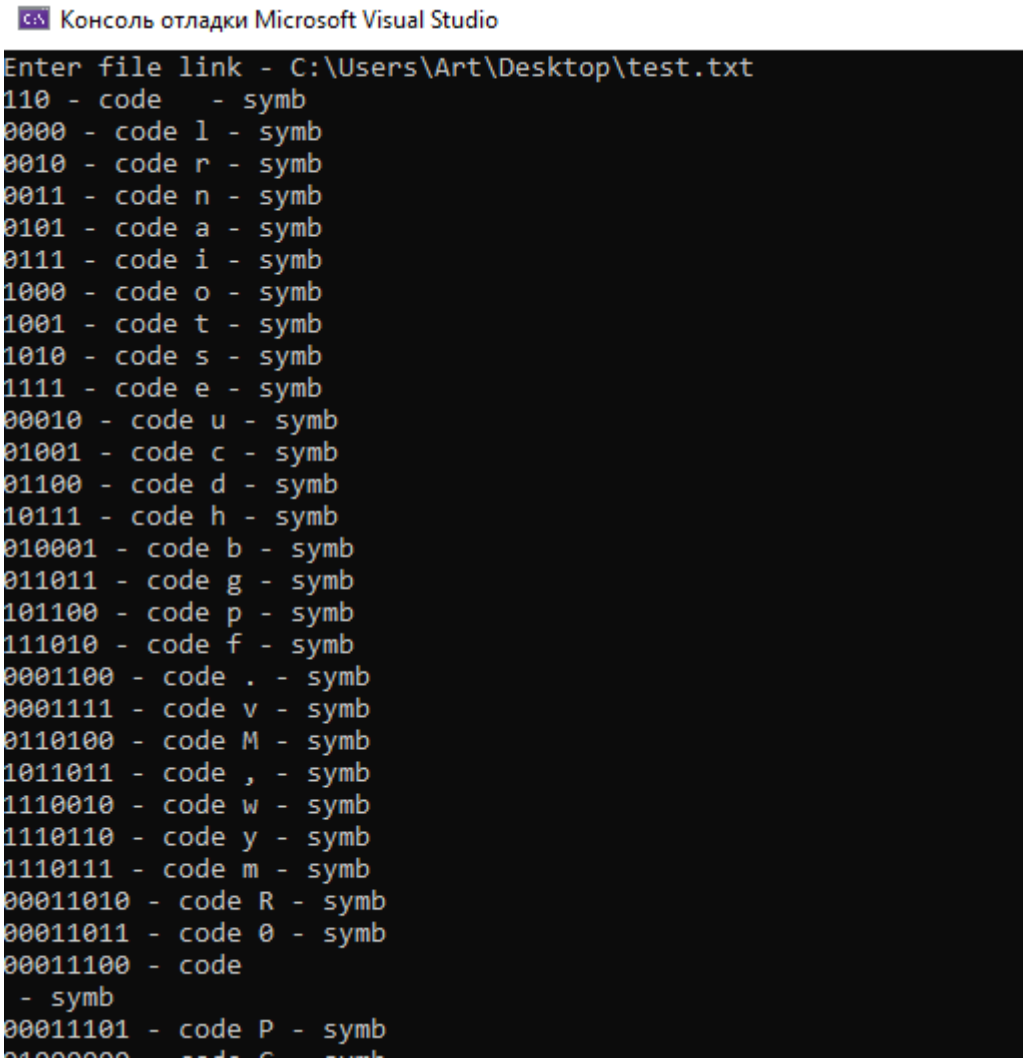
Модульные тесты нужны для сравнительно эффективного тестирования программы, удобный как для разработчика, так и другого пользователя.

Я использовал предустановленный в Visual Studio шаблон для написания unit-тестов. Созданная мною среда для каждого теста была примерно одна и та же, в основном отличаясь лишь наличием того или иного метода.

Ключевым элементов всех моих тестов было наличие сравнение практического и теоретического результата с помощью Assert::AreEqual().

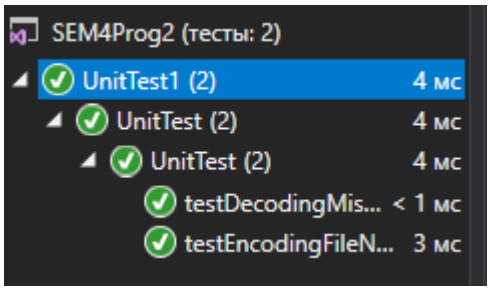
Проект с юнит-тестами был реализован в одном решении с основной программой

Пример работы



```
1110001110 - code x - symb
1110001111 - code 5 - symb
1110011110 - code   - symb
11100111110 - code 9 - symb
11100111111 - code B - symb
Researchers from Monash, Swinburne and RMIT universities have successfully tested
l chipтАФcapable of downloading 1000 high definition movies in a split second.
Published in the prestigious journal Nature Communications, these findings have th
but also the possibility for this home-grown technology to be rolled out across th
In light of the pressures being placed on the world's internet infrastructure, re
l Corcoran (Monash), Distinguished Professor Arnan Mitchell (RMIT) and Professor D
 single light source.

compression ratio =1.73847
C:\Users\Art\source\repos\SEM4Prog2\Debug\SEM4Prog2.exe (процесс 16820) завершает
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр
Чтобы закрыть это окно, нажмите любую клавишу…
```



Листинг

Head.h

```
#pragma once
#include <iostream>
#include <fstream>

using namespace std;

class EncodingHuffman
{
public:
	EncodingHuffman(string fileLink);
	~EncodingHuffman();

	void encoding()
	{
		if (!file.is_open()) {
			throw domain_error("Domain error");
		}
		char symb;
		while (!file.eof())
		{
			symb = file.get();
			insertList(symb);
		}
		file.close();
		sort();
		huffmanTree();
		huffmanTable();
		file.open(saveFile);
		while (!file.eof())
		{
			symb = file.get();
			findCode(symb);
		}
		file.close();
		encFile.close();

	}

	void decoding()
	{
		if (root == nullptr) throw out_of_range("Missing Decoding Tree");
		encFile.open("PROG2.txt", ios::in);
		nodeTree * bypass = root;
		float tempBit = 0;
		float tempByte = 0;
		while (!encFile.eof())
		{
			if (encFile.get() == '1')
			{
				bypass = bypass->right;
				if (bypass->symb != NULL)
				{
					cout << bypass->symb;
					bypass = root;
					tempByte++;
				}
```

```
                        }
                        else
                        {
                                bypass = bypass->left;
                                if (bypass->symb != NULL)
                                {
                                        cout << bypass->symb;
                                        bypass = root;
                                        tempByte++;
                                }
                        }
                        tempBit++;
                }
                tempBit = tempBit / 8;
                float compression = tempByte / tempBit;
                cout << endl << "compression ratio =" << compression;
        }

private:

        string saveFile;

        struct nodeTree
        {
                int freq = 0;
                char symb = NULL;
                nodeTree* left = nullptr;
                nodeTree* right = nullptr;
        };

        nodeTree* root;

        struct nodeList
        {
                nodeList* next = nullptr;
                nodeList* prev = nullptr;
                nodeTree* link = nullptr;
        };

        ifstream file;
        fstream encFile;
        nodeList* headList;
        nodeList* tailList;

        struct nodeCodeQueue
        {
                nodeTree* link = nullptr;
                string code = "";
                nodeCodeQueue* next = nullptr;
        };
        nodeCodeQueue* headCode;
        nodeCodeQueue* tailCode;

        void sort()
        {
                for (nodeList* first = headList; first != nullptr; first = first->next)
                {
                        for (nodeList* second = first; second != nullptr; second =
second->next)
                        {
                                if (second->link->freq < first->link->freq)
                                {
```

```
                        nodeTree* save = first->link;
                        first->link = second->link;
                        second->link = save;
                    }
                }
            }
        }

nodeList* findSymb(char symb)
{
        nodeList* tail = headList;
        while (tail != nullptr)
        {
                if (tail->link->symb == symb)
                {
                        break;
                }
                else
                {
                        tail = tail->next;
                }
        }
        return tail;
}


void insertList(char symb)
{
        if (headList == nullptr)
        {
                headList = new nodeList;
                headList->link = new nodeTree;
                headList->link->symb = symb;
                headList->link->freq = 1;
                tailList = headList;
        }
        else
        {
                nodeList* pos = findSymb(symb);
                if (pos == nullptr)
                {
                        tailList->next = new nodeList;
                        tailList->next->link = new nodeTree;
                        tailList->next->prev = tailList;
                        tailList = tailList->next;
                        tailList->link->symb = symb;
                        tailList->link->freq = 1;
                }
                else
                {
                        pos->link->freq++;
                }
        }
}

void huffmanTree()
{
        nodeTree* comb;

        for (nodeList* tail; headList->next != nullptr;)
        {
                comb = new nodeTree;
```

```
                comb->left = headList->link;
                comb->right = headList->next->link;
                comb->freq = comb->left->freq + comb->right->freq;
                nodeList* nnode = new nodeList;
                nnode->link = comb;
                if (nnode->link->freq >= tailList->link->freq)
                {
                        tailList->next = nnode;
                        nnode->prev = tailList;
                        tailList = tailList->next;
                }
                else
                {
                        tail = headList;
                        while (tail->link->freq < nnode->link->freq)
                        {
                                tail = tail->next;
                        }
                        nnode->next = tail;
                        nnode->prev = tail->prev;
                        nnode->prev->next = nnode;
                        tail->prev = nnode;
                }
                headList = headList->next->next;
                delete headList->prev->prev;
                delete headList->prev;
                headList->prev = nullptr;
        }
        root = headList->link;
}

void huffmanTable()
{
        nodeCodeQueue* headQueue = new nodeCodeQueue;
        headQueue->link = root;
        nodeCodeQueue* tail = headQueue;
        string elm;
        for (; headQueue != nullptr;)
        {
                if (headQueue->link->left != nullptr)
                {
                        tail->next = new nodeCodeQueue;
                        tail->next->code = headQueue->code + "0";
                        tail = tail->next;
                        tail->link = headQueue->link->left;
                }
                if (headQueue->link->right != nullptr)
                {
                        tail->next = new nodeCodeQueue;
                        tail->next->code = headQueue->code + "1";
                        tail = tail->next;
                        tail->link = headQueue->link->right;
                }

                if (headQueue->link->symb != NULL)
                {
                        if (headCode == nullptr)
                        {
                                headCode = headQueue;
                                headQueue = headQueue->next;
                                headCode->next = nullptr;
                                tailCode = headCode;
```

```
                        }
                        else
                        {
                                tailCode->next = headQueue;
                                tailCode = tailCode->next;
                                headQueue = headQueue->next;
                                tailCode->next = nullptr;
                        }
                }
                else
                {
                        nodeCodeQueue* del = headQueue;
                        headQueue = headQueue->next;
                        delete del;
                }
        }
        for (nodeCodeQueue* i = headCode; i != nullptr; i = i->next)
        {
                cout << i->code << " - code " << i->link->symb << " - symb\n";
        }
}

void findCode(char symb)
{
        nodeCodeQueue* tail = headCode;
        while (tail->link->symb != symb)
        {
                tail = tail->next;
        }
        encFile << tail->code;
}
};

EncodingHuffman::EncodingHuffman(string fileLink)
{
        saveFile = fileLink;
        file.open(fileLink, ios::in);
        encFile.open("PROG2.txt", ios::out);
        headList = nullptr;
        tailList = headList;
        root = nullptr;
        headCode = nullptr;
        tailCode = nullptr;

}

EncodingHuffman::~EncodingHuffman()
{
}
```

main.cpp

```
#include "Head.h"

int main()
{
        string fileLink;
        cout << "Enter file link - ";
        cin >> fileLink;
        EncodingHuffman myEncoding(fileLink);
        try
```

```
	{
		myEncoding.encoding();
		myEncoding.decoding();
	}
	catch (domain_error error)
	{
		cout << error.what();
	}

	return 0;
}
```

UnitTest1.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "Head.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest
{
	TEST_CLASS(UnitTest)
	{
	public:

		TEST_METHOD(testEncodingFileNotOpen)
		{
			string test = "C:/Users\Art\Desktop";
			try
			{
				EncodingHuffman myEncoding(test);
				myEncoding.encoding();
			}
			catch (domain_error error)
			{
				Assert::AreEqual("Domain error", error.what());
			}


		}

		TEST_METHOD(testDecodingMissingTree)
		{
			string test = "C:/Users\Art\Desktop";
			try
			{
				EncodingHuffman myEncoding(test);
				myEncoding.decoding();
			}
			catch (out_of_range error)
			{
				Assert::AreEqual("Missing Decoding Tree", error.what());
			}


		}
	};
}
```