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: Кодировка Хаффмана

Студент гр. 8309 _____ Хваталов Д.И

Преподаватель _____ Тутуева А.В

Санкт-Петербург
2019

Исходная формулировка задания:

Реализовать кодирование и декодирование по алгоритму Хаффмана

Цель работы:

Научиться кодировать и декодировать файлы с помощью алгоритмов.

Постановка задачи:

Необходимо создать класс способный закодировать и раскодировать текстовый файл, с выводом коэффициента сжатия. К каждому публичному методу в классе необходимо создать Unit-тест, который будет проверять правильность работы.

Организация данных:

Название	Описание работы метода	Оценка временной сложности
`void encoding()`	Производит кодировку данных путем создания списка символов с их частотами, после производится сортировка и построение дерева Хаффмана и таблицы с кодом каждого элемента. Повторно открывается текстовый файл и посимвольно кодирует текст.	O ((N * K) + (Z * A) + (X * logX) + (Z + Z) + N)) Считывание файла и занесение символа в список Метод sort Вызов метода haffmanTree Вызов метода haffmanTable Вывод символов с кодом Хаффмана
`void decoding()`	Декодирует зашифрованный файл путем обхода бинарного дерева с корня до необходимого листа, цикл повторяется пока не обработается весь файл.	O(N)
`void sort()`	Сортирует список по частотам символов.	O (N*logN)
`nodeList* findSymb(char symb)`	Поиск символа в списке	O (N)
`void insertList(char symb)`	Вставка символа в список или повышение частоты имеющегося	O (N) при вставке хвоста O (1) при вставке головы
`void huffmanTree()`	Формирование дерева через список символов с частотами.	O (N*N) в стандартном формате

		O (N) при условии что все новые узлы будут либо меньше головы, либо больше последнего элемента хвоста
`void huffmanTable()`	Формирование таблицы через обход дерева в ширину.	O(N+K)
`void findCode(char symb)`	Ищем код символа	O(N)

Название Unit-теста	**Описание работы**
`TEST_METHOD(testEncodingFileNotOpen)`	Проверяем открытие файла
`TEST_METHOD(testDecodingMissingTree)`	Проверяем вызов метода раскодирования не сформировав дерева Хаффмана

Код программы

```
#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("code.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("code.txt", ios :: out);
        headList = nullptr;
        tailList = headList;
        root = nullptr;
        headCode = nullptr;
        tailCode = nullptr;

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

Контрольные примеры

Рисунок 1 Коэффициент сжатия 1.97

Рисунок 2 коэффициент сжатия 2.09

Вывод

Научились кодировать текстовые файлы с помощью алгоритма Хаффмана.