

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
<code>void encoding()</code>	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.	$O((N * K) + (Z * A) + (X * \log X) + (Z + Z + N))$ Считывание файла и занесение символа в список Метод sort Вызов метода haffmanTree Вызов метода haffmanTable Вывод символов с кодом Хаффмана
<code>void decoding()</code>	Декодирует зашифрованный файл путем обхода бинарного дерева с корня до необходимого листа, цикл повторяется пока не обработается весь файл.	$O(N)$
<code>void sort()</code>	Сортирует список по частотам символов.	$O(N * \log N)$
<code>nodeList* findSymb(char symb)</code>	Поиск символа в списке	$O(N)$
<code>void insertList(char symb)</code>	Вставка символа в список или повышение частоты имеющегося	$O(N)$ при вставке хвоста $O(1)$ при вставке головы
<code>void huffmanTree()</code>	Формирование дерева через список символов с частотами.	$O(N * N)$ в стандартном формате

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

Название Unit-теста	Описание работы
<code>TEST_METHOD(testEncodingFileNotOpen)</code>	Проверяем открытие файла
<code>TEST_METHOD(testDecodingMissingTree)</code>	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

```

Консоль отладки Microsoft Visual Studio
Enter file link - C:\Users\hvv200\Desktop\testFileBin.txt
110 - code t - symb
111 - code - symb
1000 - code n - symb
1010 - code f - symb
1110 - code e - symb
1001 - code o - symb
1001 - code i - symb
1010 - code c - symb
1011 - code a - symb
1100 - code s - symb
1101 - code h - symb
10010 - code w - symb
11110 - code l - symb
100110 - code T - symb
100111 - code u - symb
101101 - code p - symb
111110 - code r - symb
111111 - code y - symb
1011000 - code j - symb
1011001 - code , - symb
1011100 - code k - symb
1011101 - code - symb
1011110 - code . - symb
1011111 - code M - symb
The first half of life consists of the capacity to enjoy without the chance, the last half consists of the chance without the capacity. Mark Twain
compression ratio = 1.97647
C:\Users\hvv200\Desktop\SCIENCE\Programs\4 sem - 2 prog\HuffmanEncoding\Debug\HuffmanEncoding.exe (процесс 13564) завершил работу с кодом 0.
Если автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно.

```

Рисунок 1 Коэффициент сжатия 1.97


```
Выбрать Консоль отладки Microsoft Visual Studio
Enter file link - C:\Users\hv200\Desktop\testFileBin.txt
000 - code o - symb
100 - code s - symb
110 - code t - symb
111 - code - symb
0110 - code u - symb
1010 - code i - symb
1011 - code r - symb
01000 - code h - symb
01001 - code f - symb
01011 - code h - symb
01110 - code a - symb
01111 - code u - symb
001000 - code , - symb
001001 - code w - symb
001010 - code o - symb
001011 - code n - symb
001100 - code l - symb
001101 - code - symb
001110 - code - symb
001111 - code A - symb
010100 - code T - symb
010101 - code d - symb
The roots of education are bitter, but the fruit is sweet. Aristotele
compression ratio =2.02166
C:\Users\hv200\Desktop\SCIENCE\Programma\4 sem - 2 prog\HuffmanEncoding\Debug\HuffmanEncoding.exe (процесс 1252) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрывать консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно.
```

Рисунок 2 коэффициент сжатия 2.09

Вывод

Научились кодировать текстовые файлы с помощью алгоритма Хаффмана.