

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и технологий

Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

Тема “Алгоритм Хаффмана”

по дисциплине «Алгоритмы и структуры данных»

Студент гр. 13537/2

Козлова Е.А.

Руководитель

Ст. преподаватель

Самочадина Т.Н.

Санкт-Петербург

2019

СОДЕРЖАНИЕ

Оглавление

Введение, общая постановка задачи	2
Основная часть работы.....	2
1. Требования	2
2. Спецификации.....	3
3. Тест план.....	4
4. Заключение	5
5. Список источников	5
6. Приложение 1. Текст программы.....	5
7. Приложение 2. Протоколы отладки.....	24

Введение, общая постановка задачи

Идея, положенная в основу кодирования Хаффмана, основана на частоте появления символа в последовательности. Символ, который встречается в последовательности чаще всего, получает новый очень маленький код, а символ, который встречается реже всего, получает, наоборот, очень длинный код. Это нужно, так как мы хотим, чтобы, когда мы обработали весь ввод, самые частотные символы заняли меньше всего места (и меньше, чем они занимали в оригинале), а самые редкие — побольше (но так как они редкие, это не имеет значения).

Вариант 5

Общая постановка задачи:

1. Реализовать алгоритм кодирования и декодирования текста по алгоритму Хаффмана.
2. Программа должна быть написана в соответствии со стандартом программирования: C++ Programming Style Guidelines

Основная часть работы

1. Требования

Функциональные требования:

1. Файлы:

- Имена файлов должны быть указаны верно;
- Входной файл не должен быть пустым;

2. Структуры:

- Таблицы «символ - частота» и «символ - код» представлены в виде ассоциативного массива;
- Узлы хранятся в двусвязном списке;
- В результате выполнения алгоритма Хаффмана строится бинарное дерево, листьями которого являются символ и его частота, а узлами сумма частот детей этого узла;
- Коды символов хранятся в векторе типа bool;

3. Выходные данные:

- Если текст состоит из различных символов, то минимальный по весу код должен быть у символа, который встречается чаще остальных;
- Если текст состоит из одного символа, он кодируется 0;

Нефункциональные требования:

1. Для обработки ошибок создается класс Exception.
2. Используется технология ООП (классы исключений, структура Node).
3. Применяется алгоритм Хаффмана.
4. Используются структуры из STL.
5. Запись таблиц, дерева, закодированного и декодированного текста производится в текстовые файлы.

2. Спецификации

№ спецификации	Входные данные (формальное описание)	Результат (Выходные данные)
1.	Неверно указано название файла с входными данными или он не существует.	Вывод сообщения: “Заданного файла не существует!”
2.	Файл с входными данными пуст.	Вывод сообщения: “Заданный файл пуст!”
3.	В файле имеются данные и он существует.	В файл “ encoded_output ” выводятся закодированные данные, после чего в файл “ decoded_output ” записываются раскодированные данные.

3. Тест план

Спецификация	Входные данные	Результат	Подтверждение
1.1.	Ввод имени файла “input.txt”	Продолжение работы программы	+
1.2.	Ввод имени файла “dfhau/fh.doc”	Вывод сообщения: “Заданного файла не существует!”	+
2.	Пустой файл.	Вывод сообщения: “Заданный файл пуст!”	+

3.	Файл содержит символы “Hello world”	Продолжение работы программы. Вывод в файл “encoded_output ” сообщение : “10 1110 32 0111 100 010 101 000 104 001 108 10 111 110 114 0110 119 1111”. Затем вывод в файл “decoded_output ” декодированное сообщение: “Hello world”	+
----	--	---	---

4. Заключение

В процессе работы мне удалось выполнить поставленную задачу. Моя программа действительно кодирует и декодирует входные данные алгоритмом Хаффмана.

Во время выполнения данной работы я получила навыки использования бинарных деревьев и опыт разработки алгоритмов.

5. Список источников

1. <https://habr.com>
2. C/C++. Программирование на языке высокого уровня. Павловская Т.А.
3. Седжвик Р., Моргунов А.А. Алгоритмы на C++. Анализ, структуры данных, сортировка, поиск, алгоритмы на графах: М.: Вильямс, 2011.

6. Приложение 1. Текст программы

Nodetype.h

```
#include <string>
```

```

class Nodetype
{
    private:
        char mSymbol;
        int mFrequency;
        std::string mCode;
        Nodetype *mLeft;
        Nodetype *mRight;
    public:
        Nodetype();
        Nodetype(char symbol, int frequency);

        void setSymbol(char symbol);
        char getSymbol();

        void setFrequency(int frequency);
        int getFrequency();

        void setLeft(Nodetype *left);
        Nodetype *getLeft();

        void setRight(Nodetype *right);
        Nodetype *getRight();

        void increaseBitCode(std::string code);
        std::string getBitCode() const;
        void setBitCode(std::string code);

        bool isLeaf();

        friend bool operator<(Nodetype o1, Nodetype o2);
        friend bool operator>=(Nodetype o1, Nodetype o2);
};

```

List.h

```

#include <stdexcept>
#include <iostream>
template<class
T> class List{
private:

    class Node{
    public:
        Node(const T& object);

        Node* getNext();
        Node* getBack();          void
setNext(Node* next);            void
setBack(Node* back);
        T& getObject();

        virtual ~Node();
    private:
        T mObject;
        Node* mNext;
        Node* mBack;
    };

    int mSize;
    Node* mStart;
    Node* mFinish;
public:

```

```

List();
List(const List<T> &orig);
List<T>& operator=(const List<T> &orig);

void push_back(const T& object);
void push_front(const T& object);
void pop_front();
void pop_back();
T& front(); T&
back();
typename List<T>::Iterator begin();
typename List<T>::Iterator end();
T& at(int index) const; T&
operator[](int index) const;
int size() const;

template<class V> friend std::ostream& operator<<(std::ostream &os, const List<V>&
list);

virtual ~List();

class Iterator {
public:
    Iterator(Node* startNode, Node* start, Node* finish);
    Iterator(const Iterator &orig);

    T& operator*()const;
    Iterator& operator++();
    Iterator operator++(int);
    Iterator& operator--();
    Iterator operator--(int); bool
operator==(const Iterator &origin); bool
operator!=(const Iterator &origin);

    virtual ~Iterator();
private:
    Node*
mStart;
    Node* mCurrent;
    Node* mFinish;
};

};
template<class T>
List<T>::Node::Node(const T& object){
    mObject = object; mNext =
NULL; mBack = NULL;
}
template<class T>
typename List<T>::Node *List<T>::Node::getNext(){
    return mNext;
}
template<class T>
typename List<T>::Node *List<T>::Node::getBack(){
    return mBack;
}
template<class T>
void List<T>::Node::setNext(Node* next){
    mNext = next;
}
template<class T>
void List<T>::Node::setBack(Node* back){
    mBack = back;
}
template<class T>

```

```

T& List<T>::Node::getObject(){
    return mObject;
}
template<class T> List<T>::Node::~Node(){
}
template<class T>
List<T>::Iterator::Iterator(typename List<T>::Node* startNode, typename List<T>::Node*
start, typename List<T>::Node* finish){          mStart = start;          mCurrent =
startNode; mFinish = finish;
}
template<class T>
List<T>::Iterator::Iterator(const Iterator& orig){
    mFinish = orig.mFinish;
    mCurrent = orig.mCurrent;
    mStart = orig.mStart;
}
template<class T>
T& List<T>::Iterator::operator*() const{
    return mCurrent->getObject();
}
template<class T>
typename List<T>::Iterator& List<T>::Iterator::operator++(){
    if(mCurrent->getNext() != NULL){
        mCurrent = mCurrent->getNext();
    }else{
        mCurrent = mStart;
    }
    return *this;
}
template<class T>
typename List<T>::Iterator List<T>::Iterator::operator ++(int){
    Iterator old = *this;    if(mCurrent->getNext() !=
NULL){        mCurrent = mCurrent->getNext();
    }else{
        mCurrent = mStart;
    }
    return old;
}
template<class T>
typename List<T>::Iterator& List<T>::Iterator::operator--(){
    if(mCurrent->getBack() != NULL){
        mCurrent = mCurrent->getBack();
    }else{
        mCurrent = mFinish;
    }
    return *this;
}
template<class
T>
typename List<T>::Iterator List<T>::Iterator::operator --(int){
    Iterator old = *this;    if (mCurrent->getBack() !=
NULL){        mCurrent = mCurrent->getBack();
    } else {
        mCurrent = mFinish;
    }
    return old;
}
template<class
T>
bool List<T>::Iterator::operator==(const Iterator &origin){
    return mCurrent == origin.mCurrent;
}
template<class
T>
bool List<T>::Iterator::operator!=(const Iterator &origin){

```



```

        return mCurrent != origin.mCurrent;
    }
    template<class
    T>
    List<T>::List(){
        mStart = NULL;
        mFinish = NULL;
        mSize = 0;
    }
    template<class T>
    List<T>::List(const List<T>& orig){
        mStart = NULL;
        mFinish = NULL;
        mSize = 0;

        for(int i = 0; i < orig.size(); i++){
            this->push_back(orig[i]);
        }
    }
    template<class T>
    List<T>& List<T>::operator=(const List<T>& orig){
        mStart = NULL;
        mFinish = NULL;
        mSize = 0;

        for(int i = 0; i < orig.size(); i++){           this-
>push_back(orig[i]);
        }

        return *this;
    }
    template<class
    T>
    T& List<T>::front(){ if(mStart
    == NULL){
        throw(new std::invalid_argument("List is empty")); }
        return mStart->getObject();
    }
    template<class
    T> T&
    List<T>::back(){
        if(mFinish == NULL){
            throw(new std::invalid_argument("List is empty"));
        }
        return mFinish->getObject();
    }
    template<class
    T>
    typename List<T>::Iterator List<T>::begin(){
        return Iterator(mStart, mStart, mFinish);
    }
    template<class
    T>
    typename List<T>::Iterator List<T>::end()
    {
        return Iterator(mFinish, mStart, mFinish);
    }
    template<class
    T>
    void List<T>::push_front(const T& object){
        mSize++;

        if(mStart == NULL){
            mStart = new Node(object);
            mFinish = mStart;
            return;
        }

```

```

        Node* new_node = new Node(object);
        mStart->setBack(new_node);
        new_node->setNext(mStart);
        mStart = new_node;
    } template<class
T>
void List<T>::push_back(const T& object){
    mSize++;

    if(mStart == NULL){
        mStart = new Node(object);
        mFinish = mStart;
        return;
    }

    Node* new_node = new Node(object);      mFinish-
>setNext(new_node);      new_node->setBack(mFinish);
    mFinish = new_node;
} template<class T> void
List<T>::pop_front(){
    if(mStart == NULL){
        return;
    }

    mSize--;

    Node* inner_start = mStart; mStart
= mStart->getNext(); mStart-
>setBack(NULL);
    delete(inner_start);
    if(mStart == NULL)
        mFinish = NULL;
} template<class T> void
List<T>::pop_back(){
    if (mFinish == NULL) {
        return;
    }

    mSize--;

    Node* inner_back = mFinish;
    mFinish = mFinish->getBack();
    mFinish->setNext(NULL);
    delete(inner_back);

    if(mFinish == NULL)
        mStart = NULL;
}
template<class T>
T& List<T>::at(int index) const{
    int i = 0;
    Node* inner_start = mStart;

    if (index >= mSize) {
        throw(new std::invalid_argument("No such index"));
    }

    while (i < index) {
        i++;
        inner_start = inner_start->getNext();
    }
    return inner_start->getObject();
}

```

```

}
template<class T>
T& List<T>::operator [](int index) const{
    return at(index);
} template<class
T>
int List<T>::size() const{
    return mSize;
} template<class
V>
std::ostream& operator<<(std::ostream &os, const List<V>& list){
    typename List<V>::Node* inner_start = list.mStart;
    while (inner_start != NULL) {
        os << "| ";
        os << inner_start->getObject();
        os << " | ";
        inner_start = inner_start->getNext();
    }
    return os;
} template<class T>
List<T>::~~List(){ while
(mStart != NULL) {
    Node* inner_start = mStart;
    mStart = mStart->getNext();
    delete inner_start;
}
}
}

```

HuffmanEncoder.h

```

#include <string>
#include <iostream>

#include "BST.h"
#include "CodedChar.h"
class HuffmanEncoder
{
private:
    std::string mInputFile;
    std::string mOutputFile;
    BST<CodedChar> mCodedChars;

    void read();
public:
    HuffmanEncoder(const std::string &inputFile, const std::string &outputFile);
    void encode();
};

```

HuffmanDecoder.h

```

#include <string>
#include <iostream>

#include "BST.h"
#include "DeCodedChar.h"
class HuffmanDecoder
{
private:
    std::string mInputFile;
    std::string mOutputFile;
    BST<DeCodedChar> mDeCodedChars;

```

```

        void read();
public:
    HuffmanDecoder(const std::string &inputFile, const std::string &outputFile);
    void decode();

};

```

HuffmanTree.h

```

#include <string>

#include "MaxHeap.h"
#include "Nodetype.h"
class HuffmanTree
{
    private:
        static Nodetype *mRoot;
        static void makeCode(Nodetype *node, std::string bitCount);
    public:
        static void buildTree(int n, MaxHeap<Nodetype*> &pq); };

```

CodedChar.h

```

#include <string>

class CodedChar {
    private:
        char mC;
        std::string mCode;
    public:
        CodedChar(char c, std::string mCode);
        CodedChar();

        std::string getCode() const;

        friend bool operator==(const CodedChar &op1, const CodedChar &op2);
        friend bool operator<(const CodedChar &op1, const CodedChar &op2);
        friend bool operator>(const CodedChar &op1, const CodedChar &op2);
};

```

DeCodedChar.h

```

#include <string>
class DeCodedChar
{
    private:
        char mC;
        std::string mCode;
    public:
        DeCodedChar(char c, std::string mCode);
        DeCodedChar();

        char getChar() const;

        friend bool operator==(const DeCodedChar &op1, const DeCodedChar &op2);
        friend bool operator<(const DeCodedChar &op1, const DeCodedChar &op2);
        friend bool operator>(const DeCodedChar &op1, const DeCodedChar &op2);
};

```

MaxHeap.h

```

#include <stdexcept>

template<typename T>
class MaxHeap{
public:
    MaxHeap(int capacity = 10);

    bool isEmpty();
    const T &top();
    void push(const T& input);
    void pop();

    virtual
~MaxHeap(); private:
    T* mHeap; int mHeapSize;
    int mCapacity;
};

template<typename T>
MaxHeap<T>::MaxHeap(int capacity){
    if(capacity < 1) throw std::invalid_argument("Capacity must be >= 1");
    mCapacity = capacity;    mHeapSize = 0;
    mHeap = new T[capacity + 1];
}

template<typename T>
bool MaxHeap<T>::isEmpty(){
    return (mHeapSize == 0);
}

template<typename T>
const T& MaxHeap<T>::top(){
    return mHeap[1];
}

template<typename T>
void MaxHeap<T>::push(const T& input){
    if(mHeapSize == mCapacity){
        T* newHeap = new T[mCapacity * 2 + 1];
        for(int i = 1; i <= mHeapSize; i++){
            newHeap[i] = mHeap[i];
        }
        mHeap = newHeap;
        mCapacity *= 2;
    }

    int current = ++mHeapSize;    while(current != 1
&& *mHeap[current / 2] < *input){
        mHeap[current] = mHeap[current / 2];
        current /= 2;
    }
    mHeap[current] = input;
}

template<typename T>
void MaxHeap<T>::pop(){
    if(isEmpty()) throw std::invalid_argument("Heap is empty. Cannot delete.");
    T last = mHeap[mHeapSize];
    mHeapSize--;

    int current = 1;    int
child = 2;    while(child <=
mHeapSize){
        if(child < mHeapSize && *mHeap[child] < *mHeap[child + 1]) child++;

        if(*last >= *mHeap[child]) break;

        mHeap[current] = mHeap[child];
        current = child;
        child *= 2;
    }
}

```

```

    }
    mHeap[current] = last;
} template<typename
T>
MaxHeap<T>::~~MaxHeap()
{
    delete[] mHeap;
}

```

BST.h

```

#include <cstdlib>
#include <stdexcept>
#include <iostream>
template<class
T>
class BST {
private:
    class Node {
private:
        T object;
        Node *right;
        Node *left;
        Node *parent;
public:
        Node(const T& object);

        void setObject(const T& object);
        T& getObject();
        void setRight(Node* right);
        Node* getRight();
        void setLeft(Node* left);
        Node* getLeft();
        void setParent(Node* parent);
        Node* getParent();

    };
    Node* root;

    void inorder(Node *start);
public:
    BST();

    void remove(const T& object);
    void insert(const T& object);
    T find(const T& object);
    const T& sup(const T& object);
    void inorder();
};
template<class T>
BST<T>::Node::Node(const T& object){
    this->object = T(object);
    right = NULL;    left =
    NULL;
    parent = NULL;
} template<class
T>
void BST<T>::Node::setObject(const T& object){
    this->object = T(object);
}
template<class T> T&
BST<T>::Node::getObject(){
    return this->object;
}

```

```

} template<class
T>
void BST<T>::Node::setRight(Node *right){      this-
>right = right;
} template<class
T>
typename BST<T>::Node* BST<T>::Node::getRight(){
    return right;
}
template<class T>
void BST<T>::Node::setLeft(Node *left){
    this->left = left;
}
template<class T>
typename BST<T>::Node* BST<T>::Node::getLeft(){
    return left;
}
template<class T>
void BST<T>::Node::setParent(Node *parent){
    this->parent = parent;
}
template<class T>
typename BST<T>::Node* BST<T>::Node::getParent(){
    return parent;
}
template<class T> BST<T>::BST(){
    root = NULL;
}
template<class T>
void BST<T>::insert(const T& object){
    if(root == NULL){
        root = new Node(object);
        return;
    }
    Node* new_node = new Node(object);
    Node* x = root;
    Node* y = NULL;
    while(x != NULL){
        y = x;
        if(x->getObject() < object){
            x = x->getRight();
        }else{
            x = x->getLeft();
        }
    }
    if(y->getObject() < object){      y-
>setRight(new_node);
        new_node->setParent(y);
    }
    if(y->getObject() > object){      y-
>setLeft(new_node);
        new_node->setParent(y);
    }
}
template<class T>
const T& BST<T>::sup(const T& object){  if(root == NULL) throw
std::invalid_argument("BST is empty, Cannot find your elemet");

    Node* x = root;
    Node* y = NULL;
    while(x != NULL){
        y = x;

```

```

        if(x->getObject() < object){
            x = x->getRight();
        }else if(x->getObject() == object){
            break;
        }
        else{
            x = x->getLeft();
        }
    }
    x = NULL;
    if(y->getObject() == object){
        if(y->getRight() == NULL){
            do{
x = y;
y = y->getParent();
            }while(x == y->getRight());
        }else{
            x = y->getRight();
            y = x;
            while(x != NULL){
                x = x->getLeft();
            }
        }
    }else{
        throw std::invalid_argument("404 Not Found");
    }
    if(y != NULL){
        return y->getObject();
    }else{
        throw std::invalid_argument("404 Not Found");
    }
}

template<class T>
void BST<T>::remove(const T& object){
    if(root == NULL) throw std::invalid_argument("BST is empty, Cannot delete.");

    Node* x = root;    Node* y =
    NULL;    bool isLeft = false;
    while(x != NULL){
        if(x->getObject() < object){
            y = x;
            x = x->getRight();
            isLeft = false;
        }else if(x->getObject() == object){
            break;
        }else{
            y = x;
            x = x->getLeft();
            isLeft = true;
        }
    }
    if(x->getObject() == object){
        if(!x->getRight() && !x->getLeft()){
            delete x;
        }
        if(y != NULL){
            (isLeft) ? y->setLeft(NULL) : y->setRight(NULL);
        }
    }else if(!x->getRight() && x->getLeft()){
        if(y != NULL){
            (isLeft) ? y->setLeft(x->getLeft()) : y->setRight(x->getLeft());
        }
        delete x;
    }else if(x->getRight() && !x->getLeft()){
        if(y != NULL){

```



```

        (isLeft) ? y->setLeft(x->getRight()) : y->setRight(x-
>getRight());
    }
    delete x;
}
else{
    T sup_value = sup(x->getObject());
    remove(sup_value);
    x->setObject(sup_value);
}
}
}
template<class T>
T BST<T>::find(const T& object){
    Node* x = root;    while(x !=
NULL){
        if(x->getObject() <
object){
            x = x-
>getRight();
        }else if(x->getObject() == object){
            return x->getObject();
        }else{
            x = x->getLeft();
        }
    }
    return object;
}
template<class T>
void BST<T>::inorder(){
    inorder(root);
}
template<class T>
void BST<T>::inorder(typename BST<T>::Node* start){
    if(start){
        inorder(start->getLeft());
        std::cout << start->getObject() << std::endl;
        inorder(start->getRight());
    }
}
}

```

BitStream.h

```

#ifndef BITSTREAM_H
#define BITSTREAM_H

#include <vector>
#include <string>
#include <iostream>

#define CHAR_BIT 8

class BitStream {
public:
    virtual ~BitStream();
    virtual bool isGood() const = 0;
    virtual bool isEOF() const = 0;
    virtual std::ios & GetStream() const { return m_stream; }
    virtual unsigned int GetPaddingLength() const { return m_paddingBitLength; }
    virtual void SetPaddingLength(unsigned int padd){ m_paddingBitLength = padd; }
}

protected:
    BitStream(std::ios &stream);

```

```

        std::ios & m_stream;          unsigned
int m_paddingBitLength;              unsigned
int m_currentBitNum;                 unsigned char
m_currentByte;                       private:
        BitStream();
        BitStream(const BitStream& copy);
};
class OutputBitStream: public BitStream
{ public:
    OutputBitStream(std::ostream & stream);
virtual ~OutputBitStream();

    virtual void InsertBits(std::string bits);
virtual void InsertBit(char bit);      virtual
void InsertBit(unsigned int bit);     virtual
void Flush();
    virtual bool isGood() const { return m_stream.good();
}    virtual bool isEOF() const { return m_stream.eof(); }

    virtual std::ostream & GetStream() const { return
dynamic_cast<std::ostream*>(m_stream); }
    friend std::ostream & operator<< (std::ostream &, const OutputBitStream &);
private:
    OutputBitStream();
    OutputBitStream(const OutputBitStream& copy);

};
class InputBitStream: public BitStream
{ public:
    InputBitStream(std::istream & stream);
virtual ~InputBitStream();

    virtual std::string GetNextBit();
    virtual bool isGood()
const;    virtual bool isEOF()
const;

    virtual std::istream & GetStream() const { return
dynamic_cast<std::istream*>(m_stream); } private:
    virtual void Consume();
    InputBitStream();
    InputBitStream(const InputBitStream& copy);
};

#endif

```

Nodetype. Cpp

```

#include "Nodetype.h"
#include <string>

Nodetype::Nodetype()
{
    mFrequency = 0;
    mLeft = NULL;
    mRight = NULL;
}

Nodetype::Nodetype(char symbol, int frequency)
{

```

```

        mSymbol = symbol;
        mFrequency = frequency;
        mLeft = NULL;
        mRight = NULL;
    }
    void Nodetype::setSymbol(char symbol)
    {
        mSymbol = symbol;
    }
    char Nodetype::getSymbol()
    {
        return mSymbol;
    }
    void Nodetype::setFrequency(int frequency)
    {
        mFrequency = frequency;
    }
    int Nodetype::getFrequency()
    {
        return mFrequency;
    }
    void Nodetype::setLeft(Nodetype* left)
    {
        mLeft = left;
    }

    Nodetype* Nodetype::getLeft()
    {
        return mLeft;
    }
    void Nodetype::setRight(Nodetype* right)
    {
        mRight = right;
    }

    Nodetype* Nodetype::getRight()
    {
        return mRight;
    }
    void Nodetype::increaseBitCode(std::string
code)
    {
        mCode += code;
    }
    std::string Nodetype::getBitCode()
const
    {
        return mCode;
    }
    void Nodetype::setBitCode(std::string
code)
    {
        mCode = code;
    }
    bool
Nodetype::isLeaf() {
        return (!mLeft && !mRight);
    }
    bool operator<(Nodetype o1, Nodetype o2)
    {
        return (o1.mFrequency > o2.mFrequency);
    }
    bool operator>=(Nodetype o1, Nodetype o2)
    {
        return (o1.mFrequency <= o2.mFrequency);
    }

```

HuffmanEncoder. Cpp

```
#include <fstream>
#include <iostream>
#include <sstream>

#include "HuffmanTree.h"
#include "HuffmanEncoder.h"
#include "Bitstream.h"
#include "Nodetype.h"
#include "CodedChar.h"
#include "List.h"
#include "MaxHeap.h"

HuffmanEncoder::HuffmanEncoder(const std::string &inputFile, const std::string &outputFile)
{
    mInputFile = inputFile;
    mOutputFile = outputFile;
}

void HuffmanEncoder::read()
{
    std::ifstream inputFileStream(mInputFile.c_str(), std::ios::in);
    std::ofstream outputFileStream(mOutputFile.c_str(), std::ios::out);
    int chars[256];    char currentChar;    List<Nodetype*> nodes;
    MaxHeap<Nodetype*> pq;

    for (int i = 0; i < 255; i++)
        chars[i] = 0;
    std::cout << "---input---" << std::endl;

    while (inputFileStream.good()) {
        inputFileStream.get(currentChar);
        if (inputFileStream.eof())
            break;
        std::cout << (int) currentChar << " " << currentChar << std::endl;
        chars[(int)currentChar]++;
    }

    for (int i = 0; i < 256; i++)        if (chars[i] >
0)            nodes.push_back(new Nodetype(i, chars[i]));

    for (int i = 0; i < nodes.size(); i++)
        pq.push(nodes[i]);

    HuffmanTree::buildTree(nodes.size(), pq);

    for (int i = 0; i < nodes.size(); i++) {
        CodedChar codedChar = CodedChar(nodes[i]-
>getBitCode());
        outputFileStream << (int) nodes[i]->getSymbol() << " " << nodes[i]-
>getBitCode() << std::endl;
        mCodedChars.insert(codedChar);
        delete nodes[i];
    }
    outputFileStream << "-1 -" << std::endl;

    outputFileStream.close();
    inputFileStream.close();
}

void HuffmanEncoder::encode()
{
    read();
```

```

std::ifstream inputFileStream(mInputFile.c_str(), std::ios::in);
std::stringstream tempStream;    char currentChar;
OutputBitStream outBitStream(tempStream);

std::cout << "----output----" << std::endl;

while (inputFileStream.good()) {
inputFileStream.get(currentChar);
    if (inputFileStream.eof())
        break;
    CodedChar codedChar = CodedChar(currentChar, "");
    std::string binaryCode = mCodedChars.find(codedChar).getCode();
    std::cout << currentChar << " : " << binaryCode << std::endl;
    outBitStream.InsertBits(binaryCode);
}

std::ofstream outputFileStream(mOutputFile.c_str(), std::ios::app);
outBitStream.Flush();
outputFileStream << outBitStream.GetPaddingLength() << std::endl;
outputFileStream << outBitStream;
outputFileStream.close();
}

```

HuffmanDecoder. Cpp

```

#include <fstream>
#include <iostream>
#include <sstream>

#include "HuffmanTree.h"
#include "HuffmanDecoder.h"
#include "Bitstream.h"
#include "DeCodedChar.h"
#include "List.h"

HuffmanDecoder::HuffmanDecoder(const std::string &inputFile, const std::string &outputFile)
{
    mInputFile = inputFile;
    mOutputFile = outputFile;
}

void HuffmanDecoder::decode()
{
    std::ifstream inputFileStream(mInputFile.c_str(), std::ios::in);
    std::ofstream outputFileStream(mOutputFile.c_str(), std::ios::out);
    std::stringstream tempStream;
    int currentAscii;
    char currentChar;
    std::string currentCode;

    std::cout << "----input----" << std::endl;

    do {
        inputFileStream >> currentAscii >> currentCode;
        std::cout << currentAscii << " " << currentCode << std::endl;
        mDeCodedChars.insert(DeCodedChar(currentAscii, currentCode));
    } while (currentAscii != -1);

    int padding;
    inputFileStream >> padding;

    inputFileStream.ignore(1, '\n');

    currentCode = "";
}

```

```

InputBitStream inBitStream(inputFileStream);
inBitStream.SetPaddingLength(padding);

std::cout << "----output----" << std::endl;

while (inBitStream.isGood()) {
    currentCode += inBitStream.GetNextBit();
    currentChar = mDeCodedChars.find(DeCodedChar(-1, currentCode)).getChar();

    if (currentChar != -1) {        outputFileStream <<
currentChar;
                                std::cout << currentChar << " " << currentCode << std::endl;
                                currentCode = "";
                                }

                                if (inBitStream.isEOF())
                                    break;
                                }

    outputFileStream.close();
}

```

HuffmanTree. Cpp

```

#include <string>

#include "MaxHeap.h"
#include "HuffmanTree.h"
#include "Nodetype.h"

Nodetype *HuffmanTree::mRoot = NULL;

void HuffmanTree::buildTree(int n, MaxHeap<Nodetype*> &pq)
{
    for (int i = 0; i < n - 1; i++) {
        Nodetype *p = pq.top();
        pq.pop();
        Nodetype *q = pq.top();
        pq.pop();
        Nodetype *r = new Nodetype();
        r->setLeft(p);
        r->setRight(q);
        r->setFrequency(p->getFrequency() + q->getFrequency());
        pq.push(r);
    }
    mRoot = pq.top();
    pq.pop(); makeCode(mRoot,
    "");
}

void HuffmanTree::makeCode(Nodetype *node, std::string bitCount)
{
    node->increaseBitCode(bitCount);
    Nodetype *left = node->getLeft();
    Nodetype *right = node->getRight();
    if (left != NULL) {
        left->setBitCode(node->getBitCode());
        HuffmanTree::makeCode(left, "0");
    }
    if (right != NULL) {
        right->setBitCode(node->getBitCode());
        HuffmanTree::makeCode(right, "1");
    }
}

```

```

        if (!node->isLeaf())
            delete node;
    }

```

CodedChar. Cpp

```

#include <string>

#include "CodedChar.h"

CodedChar::CodedChar(char c, std::string code)
{
    mC = c;
    mCode = code;
}

CodedChar::CodedChar()
{
    mC = ' ';
    mCode = "";
}

std::string CodedChar::getCode() const
{
    return mCode;
}

bool operator==(const CodedChar &op1, const CodedChar
&op2)
{
    return (op1.mC == op2.mC);
}

bool operator<(const CodedChar &op1, const CodedChar &op2)
{
    return (op1.mC < op2.mC);
}

bool operator>(const CodedChar &op1, const CodedChar &op2)
{
    return (op1.mC > op2.mC);
}

```

DeCodedChar. Cpp

```

#include <string>

#include "DeCodedChar.h"

DeCodedChar::DeCodedChar(char c, std::string code)
{
    mC = c;
    mCode = code;
}

DeCodedChar::DeCodedChar()
{
    mC = ' ';
    mCode = "";
}

char DeCodedChar::getChar() const
{
    return mC;
}

bool operator==(const DeCodedChar &op1, const DeCodedChar &op2)
{

```

```

        return (op1.mCode == op2.mCode);
    }
    bool operator<(const DeCodedChar &op1, const DeCodedChar &op2)
    {
        return (op1.mCode < op2.mCode);
    }
    bool operator>(const DeCodedChar &op1, const DeCodedChar &op2)
    {
        return (op1.mCode > op2.mCode);
    }
}

```

Main.cpp

```

#include <iostream>
#include "HuffmanTree.h"
#include "HuffmanDecoder.h"
#include "HuffmanEncoder.h"

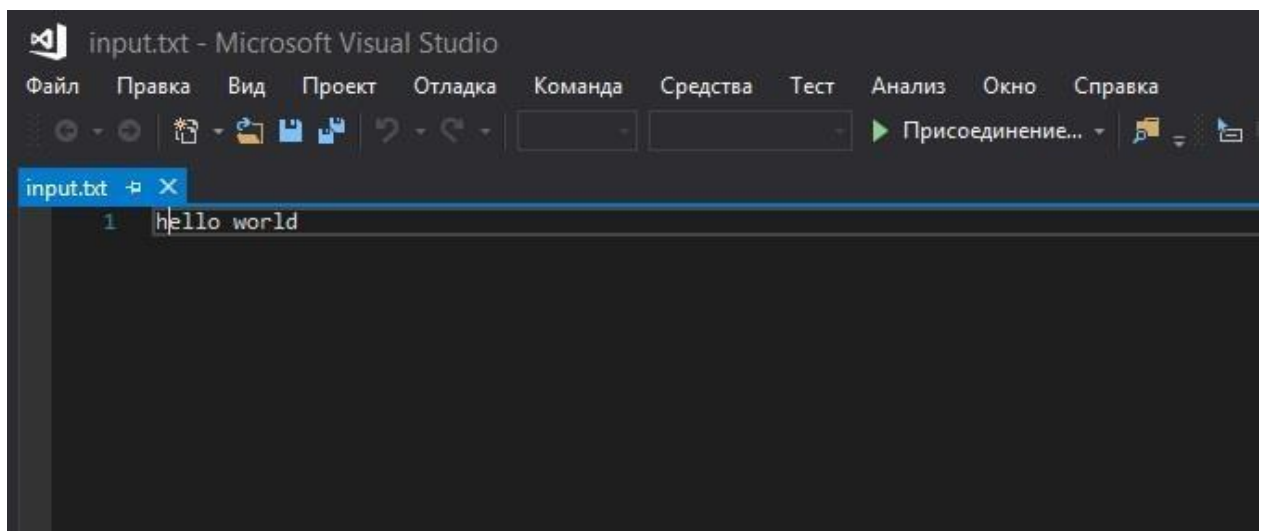
int main(int argc, char *argv[])
{
    std::cout << "---Huffman Encoder---" <<
    std::endl;

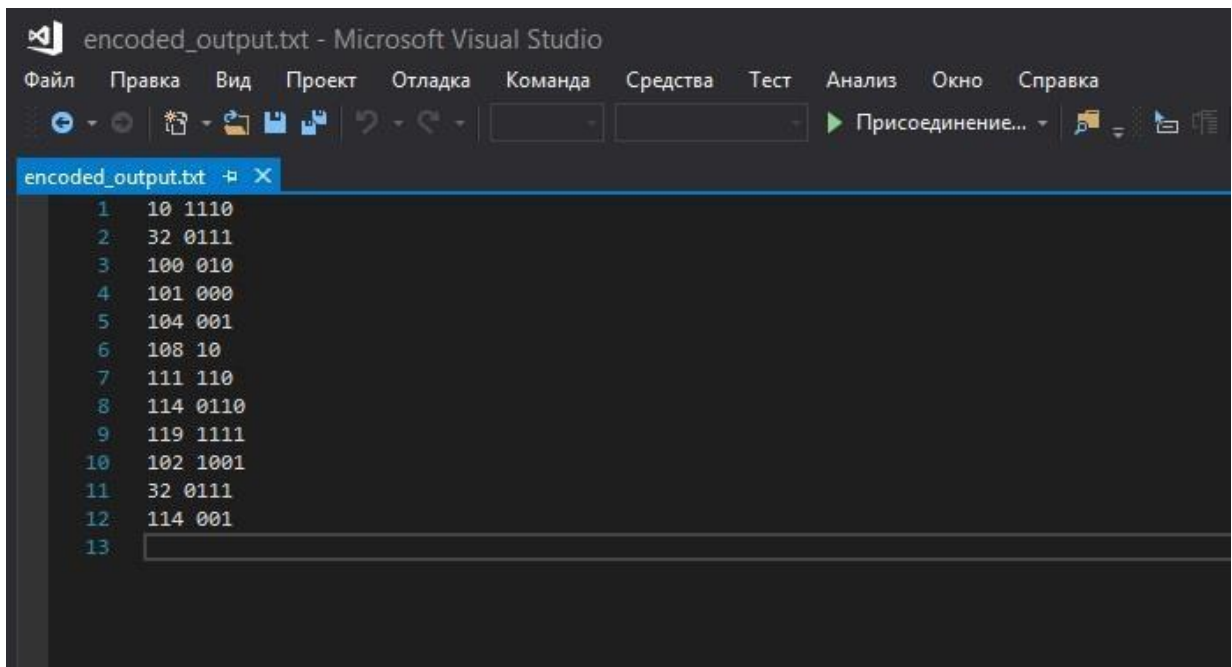
    HuffmanEncoder hman_1("input.txt", "encoded_output.txt");
    hman_1.encode();

    std::cout << "---Huffman Decoder---" << std::endl;
    HuffmanDecoder hman_2("encoded_output.txt", "decoded_output.txt");
    hman_2.decode();
}

```

7. Приложение 2. Протоколы отладки



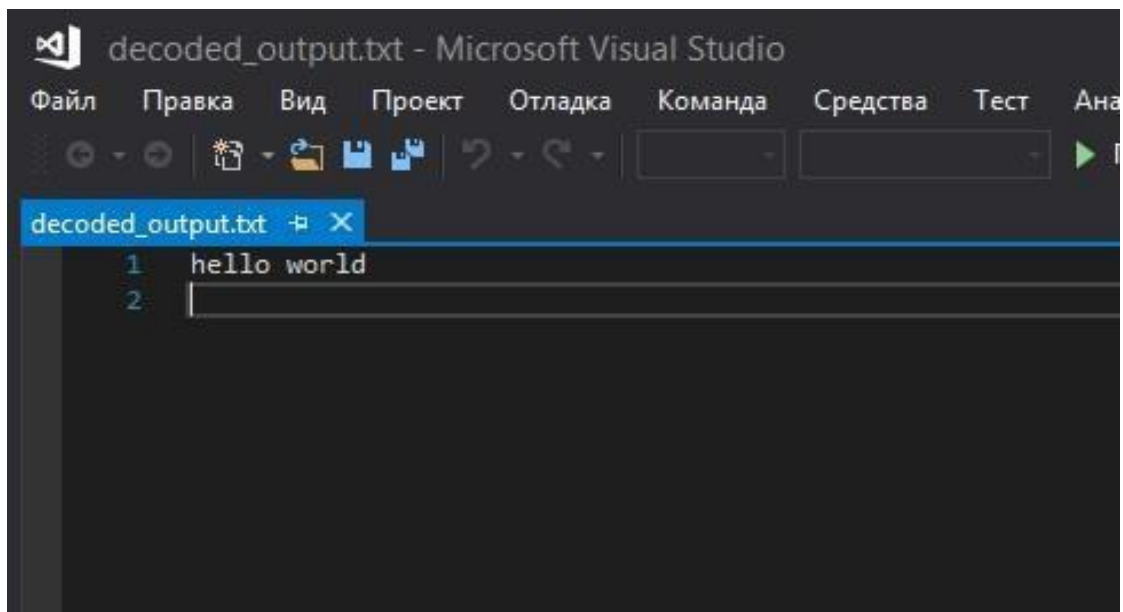


encoded_output.txt - Microsoft Visual Studio

Файл Плавка Вид Проект Отладка Команда Средства Тест Анализ Окно Справка

encoded_output.txt

```
1 10 1110
2 32 0111
3 100 010
4 101 000
5 104 001
6 108 10
7 111 110
8 114 0110
9 119 1111
10 102 1001
11 32 0111
12 114 001
13
```



decoded_output.txt - Microsoft Visual Studio

Файл Плавка Вид Проект Отладка Команда Средства Тест Ана

decoded_output.txt

```
1 hello world
2
```