**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**Пояснительная записка к курсовой работе**

**по дисциплине «Программирование»**

**Тема: Программирование алгоритмов с бинарными деревьями**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Быков И. В. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

# Постановка задачи

Задание на курсовую работу: Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона – демонстрация и исследование.

# Теоретический материал

* 1. ***Алгоритм Шеннона – Фано.***

Алгоритм Шеннона — Фано — один из первых алгоритмов сжатия, который впервые сформулировали американские учёные Шеннон и Роберт Фано. Алгоритм использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины. Коды Шеннона — Фано префиксные, то есть никакое кодовое слово не является префиксом любого другого. Это свойство позволяет однозначно декодировать любую последовательность кодовых слов.

Кодирование Шеннона — Фано — алгоритм префиксного неоднородного кодирования. Относится к вероятностным методам сжатия (точнее, методам контекстного моделирования нулевого порядка). Подобно алгоритму Хаффмана, алгоритм Шеннона — Фано использует избыточность сообщения, заключённую в неоднородном распределении частот символов его (первичного) алфавита, то есть заменяет коды более частых символов короткими двоичными последовательностями, а коды более редких символов — более длинными двоичными последовательностями.

Основные этапы:

* Символы первичного алфавита m1 выписывают по убыванию вероятностей.
* Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
* В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
* Полученные части рекурсивно делятся и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Когда размер подалфавита становится равен нулю или единице, то дальнейшего удлинения префиксного кода для соответствующих ему символов первичного алфавита не происходит, таким образом, алгоритм присваивает различным символам префиксные коды разной длины. На шаге деления алфавита существует неоднозначность, так как разность суммарных вероятностей p0 - p1 может быть одинакова для двух вариантов разделения (учитывая, что все символы первичного алфавита имеют вероятность больше нуля).

Алгоритм вычисления.

Код Шеннона — Фано строится с помощью дерева. Построение этого дерева начинается от корня. Всё множество кодируемых элементов соответствует корню дерева (вершине первого уровня). Оно разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Эти подмножества соответствуют двум вершинам второго уровня, которые соединяются с корнем. Далее каждое из этих подмножеств разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Им соответствуют вершины третьего уровня. Если подмножество содержит единственный элемент, то ему соответствует концевая вершина кодового дерева; такое подмножество разбиению не подлежит. Подобным образом поступаем до тех пор, пока не получим все концевые вершины. Ветви кодового дерева размечаем символами 1 и 0, как в случае кода Хаффмана.

При построении кода Шеннона — Фано разбиение множества элементов может быть произведено, вообще говоря, несколькими способами. Выбор разбиения на уровне n может ухудшить варианты разбиения на следующем уровне (n + 1) и привести к неоптимальности кода в целом. Другими словами, оптимальное поведение на каждом шаге пути ещё не гарантирует оптимальности всей совокупности действий. Поэтому код Шеннона — Фано не является оптимальным в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей. Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона — Фано, и все они могут дать различные результаты. Если построить все возможные коды Шеннона — Фано для данного распределения вероятностей, то среди них будут находиться и все коды Хаффмана, то есть оптимальные коды.

* 1. ***Алгоритм Хаффмана***

Алгоритм Хаффмана — жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью.

В отличие от алгоритма Шеннона — Фано, алгоритм Хаффмана остаётся всегда оптимальным и для вторичных алфавитов m2 с более чем двумя символами.

Этот метод кодирования состоит из двух основных этапов:

* Построение оптимального кодового дерева.
* Построение отображения код-символ на основе построенного дерева.

Идея алгоритма состоит в следующем: зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево).

* Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
* Выбираются два свободных узла дерева с наименьшими весами.
* Создается их родитель с весом, равным их суммарному весу.
* Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
* Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.
* Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Классический алгоритм Хаффмана имеет ряд существенных недостатков. Во-первых, для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот, которой пользовался кодер. Следовательно, длина сжатого сообщения увеличивается на длину таблицы частот, которая должна посылаться впереди данных, что может свести на нет все усилия по сжатию сообщения. Кроме того, необходимость наличия полной частотной статистики перед началом собственно кодирования требует двух проходов по сообщению: одного для построения модели сообщения (таблицы частот и Н-дерева), другого для собственно кодирования. Во-вторых, избыточность кодирования обращается в ноль лишь в тех случаях, когда вероятности кодируемых символов являются обратными степенями числа 2. В-третьих, для источника с энтропией, не превышающей 1 (например, для двоичного источника), непосредственное применение кода Хаффмана бессмысленно.

# Формальная постановка задачи

* 1. ***Исходные данные***

На вход подается строка (записывается в динамический массив типа char), которую требуется закодировать методами Фано – Шеннона и Хаффмана

В программе, реализующей исследование алгоритма, строка вводится из файла.

В программе, реализующей демонстрацию алгоритма, строка вводится пользователем в специальное поле (lineEdit).

* 1. ***Ограничения на исходные данные***

В программе, реализующей исследование алгоритма, ограничение на количество символов в строке: MAX\_INPUT\_LEN 10000000

В программе, реализующей демонстрацию алгоритма, ограничения на входные данные: вводятся латинские символы и цифры, длина строки не более 100 символов.

# Разработка структур данных и алгоритмов

* 1. ***Модель структур данных***

Для реализации кодирования данных методом Хаффмана и Фано-Шеннона использовались следующие структуры данных: бинарное дерево, дерево кодирования, очередь с приоритетами.

* + 1. Бинарное дерево. Реализовано в виде структуры:

struct BinaryTree

{

char symbol;

long long int weight;

BinaryTree\* left = NULL;

BinaryTree\* right = NULL;

BinaryTree(char symbol, long long int weight, BinaryTree\* left, BinaryTree\* right);

~BinaryTree();

};

* + 1. Очередь с приоритетами. Реализована в виде шаблонной структуры:

template <typename T>

struct PriorityQueueItem

{

int key;

T data;

};

template <typename T>

struct PriorityQueue

{

int size\_;

int capacity\_;

PriorityQueueItem<T>\* heap\_;

};

* + 1. Дерево кодирования. Реализовано в виде структуры:

struct CodeTree

{

char \*str;

Symbol s;

CodeTree\* parent;

CodeTree\* left;

CodeTree\* right;

};

* 1. ***Алгоритмы***

Представлено краткое описание алгоритмов кодирования, реализующих задание.

4.2.1. Кодирование алгоритмом Хаффмана:

* Разбираем строку: находим частоты каждого элемента и сортируем по убыванию частот.
* Выбираются две наименьших по частоте буквы алфавита, и создается родитель (сумма двух частот этих «листков»).
* Потомки удаляются и вместо них записывается родитель, «ветви» родителя нумеруются: левой ветви ставится в соответствие «0», правой «1».
* Шаг два повторяется до тех пор, пока не будет найден главный родитель — «корень».

4.2.2. Кодирование алгоритмом Фано-Шеннона:

* Разбираем строку: находим частоты каждого элемента и сортируем по убыванию частот.
* Находится середина, которая делит алфавит примерно на две части. Эти части (суммы частот алфавита) примерно равны. Для левой части присваивается «0», для правой «1», таким образом мы получим листья дерева
* Предыдущий повторяется до тех пор, пока мы не получим единственный элемент последовательности, т.е. листок

# Спецификация программы

* 1. ***Назначение программы***

Программа предназначена для реализации алгоритмов кодирования заданной строки алгоритмами Хаффмана и Фано – Шеннона.

* 1. ***Спецификация функций***
     1. Функции дерева кодирования:

CodeTree\* make\_leaf(const Symbol& s) – создаем лист.

CodeTree\* make\_node(long long int weight, CodeTree\* left, CodeTree\* right) – создаем узел.

bool is\_null(const CodeTree\* node) – проверка на пустое дерево.

bool is\_leaf(const CodeTree\* node) – проверка на лист.

bool is\_root(const CodeTree\* node) – проверка на корень.

char\* encode(const CodeTree\* tree, const char\* message) – функция кодирования, на вход получаем дерево, созданное в результате кодирования сообщения методом Фано-Шеннона или Хаффмана, и исходное сообщение. Возвращает закодированное сообщение.

char\* decode(const CodeTree\* tree, const char\* code) - функция декодирования, на вход получаем дерево, созданное в результате кодирования сообщения методом Фано-Шеннона или Хаффмана, и кодированное сообщение. Возвращает исходное сообщение.

void destroy(CodeTree\* tree) – удаляем дерево.

void print\_tree(CodeTree\* tree, long long int level, std::ostream& out) – вывод дерева.

* + 1. Функции кодирования методом Хаффмана.

CodeTree\* huffman(char\* message) – считаем веса символов и сортируем символы по убыванию веса, возвращаем huffman(symbols, len).

CodeTree\* huffman(const char\* keys, const int\* w, int len) – получаем дерево Хаффмана

* + 1. Функции кодирования методом Фано – Шеннона.

CodeTree\* fanno\_shannon(const char\* message) - считаем веса символов и сортируем символы по убыванию веса, возвращаем fanno\_shannon(symbols, len).

CodeTree\* fanno\_shannon(const Symbol\* symbols, long long int len) - считаем сумму всех весов, возвращаем fanno\_shannon(symbols, 0, len, sum).

CodeTree\* fanno\_shannon(const Symbol\* symbols,long long int l,long long int r,long long int sum) – получаем дерево Фано – Шеннона.

* 1. ***Диалог с пользователем***

# Тестовые данные

|  |  |  |
| --- | --- | --- |
| Исходная строка | Кодирование методом Хаффмана | Кодирование методом Фано - Шеннона |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Анализ результатов и выводы

В ходе выполнения данной курсовой работы была изучена теория по основным понятиям и приёмам кодирования информации алгоритмами Хаффмана и Фано-Шеннона, а также написана программа, выполняющая требуемые действия по кодированию информации. Были закреплены навыки работы с классами, со структурами данных: бинарное дерево, очередь с приоритетами, которые были реализованы на языке С++.

# Приложение А. Программа исследования. Листинг

CodeTree.h

#pragma once

#include <iostream>

#define MAX\_INPUT\_LEN 10000000

#define MAX\_CODE\_LEN 1000000000

struct Symbol

{

char c;

long long int weight;

};

bool symbol\_less(const Symbol& l, const Symbol& r);

bool symbol\_greater(const Symbol& l, const Symbol& r);

struct CodeTree

{

Symbol s;

CodeTree\* parent;

CodeTree\* left;

CodeTree\* right;

};

CodeTree\* make\_leaf(const Symbol& s);

CodeTree\* make\_node(long long int weight, CodeTree\* left, CodeTree\* right);

bool is\_null(const CodeTree\* node);

bool is\_leaf(const CodeTree\* node);

bool is\_root(const CodeTree\* node);

char\* encode(const CodeTree\* tree, const char\* message);

char\* decode(const CodeTree\* tree, const char\* code);

void destroy(CodeTree\* tree);

void print\_tree(CodeTree\* tree, long long int level, std::ostream& out);

fh.h

#pragma once

#include "CodeTree.h"

CodeTree\* fanno\_shannon(const char\* message);

CodeTree\* fanno\_shannon(const Symbol\* symbols, long long int len);

Huffman.h

#pragma once

#include "CodeTree.h"

CodeTree\* huffman(char\* message);

CodeTree\* huffman(const char\* keys, const int\* w, int len);

p\_queue.h

#include <utility>

template <typename T>

struct PriorityQueueItem {

int key;

T data;

};

template <typename T>

struct PriorityQueue {

int size\_;

int capacity\_;

PriorityQueueItem<T>\* heap\_;

};

template <typename T>

PriorityQueue<T>\* create\_pq(int capacity)

{

PriorityQueue<T>\* pq = new PriorityQueue<T>;

pq->heap\_ = new PriorityQueueItem<T>[capacity];

pq->capacity\_ = capacity;

pq->size\_ = 0;

return pq;

}

template <typename T>

int size(PriorityQueue<T>\* pq){

return pq->size\_;

}

template <typename T>

void sift\_up(PriorityQueue<T>\* pq, int index){

int parent = (index - 1) / 2;

while(parent >= 0 && pq->heap\_[index].key < pq->heap\_[parent].key) {

std::swap(pq->heap\_[index], pq->heap\_[parent]);

index = parent;

parent = (index - 1) / 2;

}

}

template <typename T>

bool push(PriorityQueue<T>\* pq, int key, const T& data){

if(pq->size\_ >= pq->capacity\_)

return false;

pq->heap\_[pq->size\_].key = key;

pq->heap\_[pq->size\_].data = data;

pq->size\_++;

sift\_up(pq, pq->size\_ - 1);

return true;

}

template <typename T>

void sift\_down(PriorityQueue<T>\* pq, int index){

int l = 2 \* index + 1;

int r = 2 \* index + 2;

int min = index;

if(l < pq->size\_ && pq->heap\_[l].key < pq->heap\_[min].key)

min = l;

if(r < pq->size\_ && pq->heap\_[r].key < pq->heap\_[min].key)

min = r;

if(min != index) {

std::swap(pq->heap\_[index], pq->heap\_[min]);

sift\_down(pq, min);

}

}

template <typename T>

T pop(PriorityQueue<T>\* pq){

std::swap(pq->heap\_[0], pq->heap\_[pq->size\_ - 1]);

pq->size\_--;

sift\_down(pq, 0);

return pq->heap\_[pq->size\_].data;

}

template <typename T>

void destroy\_pq(PriorityQueue<T>\* pq){

delete [] pq->heap\_;

delete pq;

}

CodeTree.cpp

#include "CodeTree.h"

#include <climits>

#include <cstring>

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

bool symbol\_less(const Symbol& l, const Symbol& r)

{

return l.weight < r.weight;

}

bool symbol\_greater(const Symbol& l, const Symbol& r)

{

return l.weight > r.weight;

}

CodeTree\* make\_leaf(const Symbol& s)

{

return new CodeTree{ s, nullptr, nullptr, nullptr };

}

CodeTree\* make\_node(long long int weight, CodeTree\* left, CodeTree\* right)

{

Symbol s{ 0, weight };

return new CodeTree{ s, nullptr, left, right };

}

bool is\_leaf(const CodeTree\* node)

{

return ((node->left == nullptr) && (node->right == nullptr));

}

bool is\_null(const CodeTree\* node)

{

return (node == nullptr);

}

bool is\_root(const CodeTree\* node)

{

return node->parent == nullptr;

}

static void fill\_symbols\_map(const CodeTree\* node, const CodeTree\*\* symbols\_map);

char\* encode(const CodeTree\* tree, const char\* message)

{

char\* code = new char[MAX\_CODE\_LEN];

long long int len = strlen(message);

const CodeTree\*\* symbols\_map = new const CodeTree\*[UCHAR\_MAX];

for (long long int i = 0; i < UCHAR\_MAX; ++i)

{

symbols\_map[i] = nullptr;

}

fill\_symbols\_map(tree, symbols\_map);

long long int index = 0;

char path[UCHAR\_MAX];

for (long long int i = 0; i < len; ++i)

{

const CodeTree\* node = symbols\_map[message[i] - CHAR\_MIN];

long long int j = 0;

while (!is\_root(node))

{

if (node->parent->left == node)

path[j++] = '0';

else

path[j++] = '1';

node = node->parent;

}

while (j > 0) code[index++] = path[--j];

}

if (index != 0)

code[index] = 0;

else

{

for (long long int i = 0; i < strlen(message); i++)

code[i] = '0';

code[strlen(message)] = 0;

}

delete[] symbols\_map;

return code;

}

char\* decode(const CodeTree\* tree, const char\* code)

{

char\* message = new char[MAX\_INPUT\_LEN];

long long int index = 0;

long long int len = strlen(code);

const CodeTree\* v = tree;

if (is\_leaf(v))

{

for (long long int i = 0; i < len; ++i)

{

message[index++] = v->s.c;

v = tree;

}

}

else

for (long long int i = 0; i < len; ++i)

{

if (code[i] == '0')

v = v->left;

else

v = v->right;

if (is\_leaf(v))

{

message[index++] = v->s.c;

v = tree;

}

}

message[index] = 0;

return message;

}

void destroy(CodeTree\* tree)

{

if (tree == nullptr) return;

destroy(tree->left);

destroy(tree->right);

delete tree;

tree = nullptr;

}

void fill\_symbols\_map(const CodeTree\* node, const CodeTree\*\* symbols\_map)

{

if (is\_leaf(node))

symbols\_map[node->s.c - CHAR\_MIN] = node;

else {

fill\_symbols\_map(node->left, symbols\_map);

fill\_symbols\_map(node->right, symbols\_map);

}

}

void print\_tree(CodeTree\* tree, long long int level, std::ostream& out)

{

if (tree)

{

print\_tree(tree->left, level + 1, out);

for (long long int i = 0; i < level; i++) out << " ";

out << tree->s.weight << std::endl;

print\_tree(tree->right, level + 1, out);

}

}

fh.cpp

#include "fh.h"

#include <algorithm>

#include <climits>

#include <cstring>

static long long int middle(const Symbol\* symbols, long long int l, long long int sum, long long int& lsum,long long int& rsum);

CodeTree\* fanno\_shannon(const Symbol\* symbols,long long int l,long long int r,long long int sum)

{

if (l >= r) return nullptr;

if (r - l == 1) return make\_leaf(symbols[l]);

long long int lsum, rsum;

long long int m = middle(symbols, l, sum, lsum, rsum);

CodeTree\* ltree = fanno\_shannon(symbols, l, m + 1, lsum);

CodeTree\* rtree = fanno\_shannon(symbols, m + 1, r, rsum);

CodeTree\* node = make\_node(sum, ltree, rtree);

ltree->parent = node;

rtree->parent = node;

return node;

}

CodeTree\* fanno\_shannon(const Symbol\* symbols, long long int len)

{

long long int sum = 0;

for (long long int i = 0; i < len; ++i)

sum += symbols[i].weight;

return fanno\_shannon(symbols, 0, len, sum);

}

CodeTree\* fanno\_shannon(const char\* message)

{

Symbol symbols[UCHAR\_MAX];

for (long long int i = 0; i < UCHAR\_MAX; ++i) {

symbols[i].c = i + CHAR\_MIN;

symbols[i].weight = 0;

}

int size = strlen(message);

for (long long int i = 0; i < size; ++i)

symbols[message[i] - CHAR\_MIN].weight++;

std::sort(symbols, symbols + UCHAR\_MAX, symbol\_greater);

long long int len = 0;

while (symbols[len].weight > 0 && len < UCHAR\_MAX) len++;

return fanno\_shannon(symbols, len);

}

long long int middle(const Symbol\* symbols, long long int l, long long int sum, long long int& lsum, long long int& rsum)

{

long long int m = l;

lsum = symbols[m].weight;

rsum = sum - lsum;

long long int delta = lsum - rsum;

while (delta + symbols[m + 1].weight < 0) {

m++;

lsum += symbols[m].weight;

rsum -= symbols[m].weight;

delta = lsum - rsum;

}

return m;

}

Huffman.cpp

#include "huffman.h"

#include "p\_queue.h"

#include <functional>

#include <algorithm>

#include <climits>

#include <cstring>

#include <iostream>

CodeTree\* huffman(const Symbol\* symbols, int len){

PriorityQueue<CodeTree\*>\* queue = create\_pq<CodeTree\*>(len);

for(int i = 0; i < len; ++i)

push(queue, symbols[i].weight, make\_leaf(symbols[i]));

while(size(queue) > 1) {

CodeTree\* ltree = pop(queue);

CodeTree\* rtree = pop(queue);

int weight = ltree->s.weight + rtree->s.weight;

CodeTree\* node = make\_node(weight, ltree, rtree);

ltree->parent = node;

rtree->parent = node;

push(queue, weight, node);

}

CodeTree\* result = pop(queue);

destroy\_pq(queue);

return result;

}

CodeTree\* huffman(char\* message) {

Symbol symbols[UCHAR\_MAX];

for(int i = 0; i < UCHAR\_MAX; ++i) {

symbols[i].c = i + CHAR\_MIN;

symbols[i].weight = 0;

}

long long int size = strlen(message);

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

symbols[message[i] - CHAR\_MIN].weight++;

std::sort(symbols, symbols + UCHAR\_MAX, symbol\_greater);

int len = 0;

while(symbols[len].weight > 0 && len < UCHAR\_MAX) len++;

return huffman(symbols, len);

}

main.cpp

#include "fh.h"

#include "huffman.h"

#include <fstream>

#include <chrono>

#include<iterator>

using namespace std;

using namespace std::chrono;

int main()

{

CodeTree\* ct\_shf;// = nullptr;

CodeTree\* ct\_huff;

char \*c;

std::ifstream fin("input.dat");

std::ofstream fout\_encode\_huffman("huffman.dat");

std::ofstream fout\_encode\_shannon\_fano("shf.dat");

std::ofstream fout\_decode\_huffman("huffman\_decode.dat");

std::ofstream fout\_decode\_shannon\_fano("shf\_decode.dat");

c = new char[MAX\_INPUT\_LEN];

char \*\_encode\_shf, \*\_decode\_shf;

char \*\_encode\_huff, \*\_decode\_huff;

long long int i = 1;

c[0] = fin.get();

while(fin.peek()!=EOF)

{

c[i] = fin.get();

i++;

}

ct\_shf = fanno\_shannon(c);

ct\_huff = huffman(c);

auto begin\_enc\_shf = std::chrono::high\_resolution\_clock::now();

\_encode\_shf = encode(ct\_shf, c);

auto end\_enc\_shf = std::chrono::high\_resolution\_clock::now();

auto begin\_dec\_shf = std::chrono::high\_resolution\_clock::now();

\_decode\_shf = decode(ct\_shf, \_encode\_shf);

auto end\_dec\_shf = std::chrono::high\_resolution\_clock::now();

auto begin\_enc\_huff = std::chrono::high\_resolution\_clock::now();

\_encode\_huff = encode(ct\_huff, c);

auto end\_enc\_huff = std::chrono::high\_resolution\_clock::now();

auto begin\_dec\_huff = std::chrono::high\_resolution\_clock::now();

\_decode\_huff = decode(ct\_huff, \_encode\_huff);

auto end\_dec\_huff = std::chrono::high\_resolution\_clock::now();

std::cout << "Algorithm speed (in nanoseconds):" << std::endl;

std::cout << "Encoding sh\_f: " << std::chrono::duration\_cast<std::chrono::nanoseconds>(end\_enc\_shf - begin\_enc\_shf).count() << std::endl;

std::cout << "Decoding sh\_f: " << std::chrono::duration\_cast<std::chrono::nanoseconds>(end\_dec\_shf - begin\_dec\_shf).count() << std::endl;

std::cout << "Encoding huff: " << std::chrono::duration\_cast<std::chrono::nanoseconds>(end\_enc\_huff - begin\_enc\_huff).count() << std::endl;

std::cout << "Decoding huff: " << std::chrono::duration\_cast<std::chrono::nanoseconds>(end\_dec\_huff - begin\_dec\_huff).count() << std::endl;

//std::cout << "Tree:\n";

//print\_tree(ct\_shf, 0, std::cout);

fout\_encode\_shannon\_fano << \_encode\_shf << std::endl;

fout\_decode\_shannon\_fano << \_decode\_shf << std::endl;

fout\_encode\_huffman << \_encode\_huff << std::endl;

fout\_decode\_huffman << \_decode\_huff << std::endl;

destroy(ct\_shf);

destroy(ct\_huff);

delete[] c;

fout\_encode\_huffman.close();

fout\_encode\_shannon\_fano.close();

fout\_decode\_huffman.close();

fout\_decode\_shannon\_fano.close();

fin.close();

return 0;

}

# Приложение Б. Программа визуализации. Листинг.

binarytree.cpp

#include "binarytree.h"

BinaryTree::BinaryTree(char symbol, long long int weight, BinaryTree\* left, BinaryTree\* right)

{

this->symbol = symbol;

this->weight = weight;

this->left = left;

this->right = right;

}

BinaryTree::~BinaryTree()

{

if(left != NULL)

delete left;

if(right != NULL)

delete right;

}

binarytree.h

#pragma once

// Описание структуры Бинарное дерево

#include <iostream>

#include <vector>

#include <climits>

#include <cstring>

#include <algorithm>

#include <functional>

#include <cstdlib>

#include <cmath>

#include <QMainWindow>

#include <QWidget>

#include <QLineEdit>

#include <QPushButton>

#include <QLabel>

#include <QTextStream>

struct BinaryTree

{

char symbol;

long long int weight;

BinaryTree\* left = NULL;

BinaryTree\* right = NULL;

BinaryTree(char symbol, long long int weight, BinaryTree\* left, BinaryTree\* right);

~BinaryTree();

};

codetree.cpp

#include "codetree.h"

bool symbol\_less(const Symbol& l, const Symbol& r)

{

return l.weight < r.weight;

}

bool symbol\_greater(const Symbol& l, const Symbol& r)

{

return l.weight > r.weight;

}

CodeTree\* make\_leaf(const Symbol& s)

{

return new CodeTree{ "-",s, NULL, NULL, NULL };

}

CodeTree\* make\_node(int weight, CodeTree\* left, CodeTree\* right)

{

Symbol s{ '-', weight };

return new CodeTree{ "-", s,NULL, left, right };

}

bool is\_leaf(const CodeTree\* node)

{

return node->left == NULL && node->right == NULL;

}

bool is\_root(const CodeTree\* node)

{

return node->parent == NULL;

}

static void fill\_symbols\_map(const CodeTree\* node, const CodeTree\*\* symbols\_map);

char\* encode( CodeTree\* tree, char\* message)

{

char\* code = new char[MAX\_CODE\_LEN];

const CodeTree\*\* symbols\_map = new const CodeTree\*[UCHAR\_MAX];

for(int i = 0; i < UCHAR\_MAX; ++i)

{

symbols\_map[i] = NULL;

}

fill\_symbols\_map(tree, symbols\_map);

int len = strlen(message);

int index = 0;

char path[MAX\_CODE\_LEN];

for(int i = 0; i < len; ++i)

{

const CodeTree\* node = symbols\_map[message[i] - CHAR\_MIN];

int j = 0;

while(!is\_root(node))

{

if(node->parent->left == node)

path[j++] = '0';

else

path[j++] = '1';

node = node->parent;

}

while(j > 0)

code[index++] = path[--j];

}

if (index != 0)

code[index] = 0;

else

{

for (unsigned i = 0; i < strlen(message); i++)

code[i] = '0';

code[strlen(message)] = 0;

}

delete [] symbols\_map;

return code;

}

char\* decode(const CodeTree\* tree, char\* code)

{

char\* message = new char[MAX\_CODE\_LEN];

int index = 0;

int len = strlen(code);

const CodeTree\* v = tree;

if (is\_leaf(v))

{

for (int i = 0; i < len; ++i)

{

message[index++] = v->s.c;

v = tree;

}

}

else

for (int i = 0; i < len; ++i)

{

if (code[i] == '0')

v = v->left;

else

v = v->right;

if (is\_leaf(v))

{

message[index++] = v->s.c;

v = tree;

}

}

message[index] = 0;

return message;

}

void destroy(CodeTree\* tree)

{

if(tree == NULL) return;

destroy(tree->left);

destroy(tree->right);

delete tree;

tree = NULL;

}

void fill\_symbols\_map(const CodeTree\* node, const CodeTree\*\* symbols\_map)

{

if(is\_leaf(node))

symbols\_map[node->s.c - CHAR\_MIN] = node;

else

{

fill\_symbols\_map(node->left, symbols\_map);

fill\_symbols\_map(node->right, symbols\_map);

}

}

Trees::Trees(CodeTree \*tree,int weight, char symbol)

{

this->tree = tree;

this->weight = weight;

this->symbol = symbol;

this->root = makeBinaryTree(tree);

}

Trees::~Trees()

{

if(root != NULL)delete root;

}

BinaryTree\* Trees::makeBinaryTree(CodeTree\* root)

{

if(root == NULL)

return NULL;

return new BinaryTree(root->s.c, root->s.weight, makeBinaryTree(root->left), makeBinaryTree(root->right));

}

codetree.h

#pragma once

// Структуры: символ, Кодовое дерево, Trees - для создания бинарного дерева

#include "binarytree.h"

#define MAX\_CODE\_LEN 1000000

struct Symbol

{

char c;

int weight;

};

bool symbol\_less(const Symbol& l, const Symbol& r);

bool symbol\_greater(const Symbol& l, const Symbol& r);

struct CodeTree

{

char \*str;

Symbol s;

CodeTree\* parent;

CodeTree\* left;

CodeTree\* right;

};

CodeTree\* make\_leaf(const Symbol& s);

CodeTree\* make\_node(int weight, CodeTree\* left, CodeTree\* right);

bool is\_leaf(const CodeTree\* node);

bool is\_root(const CodeTree\* node);

char\* encode(CodeTree\* tree, char\* message);

char\* decode(const CodeTree\* tree, char\* code);

void destroy(CodeTree\* tree);

struct Trees

{

BinaryTree\* root = NULL;

CodeTree \*tree;

int weight;

char symbol;

Trees(CodeTree \*tree, int weight, char symbol);

~Trees();

BinaryTree\* makeBinaryTree(CodeTree\* root);

};

fh.cpp

#include "fh.h"

static long long int middle(const Symbol\* symbols, long long int l, long long int sum, long long int& lsum, long long int& rsum);

CodeTree\* fanno\_shannon(std::vector<Trees\*> &trees, const Symbol\* symbols, long long int l, long long int r, long long int sum)

{

if (l >= r)

return nullptr;

if (r - l == 1)

return make\_leaf(symbols[l]);

long long int lsum, rsum;

long long int m = middle(symbols, l, sum, lsum, rsum);

CodeTree\* ltree = fanno\_shannon(trees, symbols, l, m + 1, lsum);

CodeTree\* rtree = fanno\_shannon(trees, symbols, m + 1, r, rsum);

CodeTree\* node = make\_node(sum, ltree, rtree);

trees.push\_back(new Trees(node, node->s.weight, node->s.c));

ltree->parent = node;

rtree->parent = node;

return node;

}

CodeTree \*fanno\_shannon(std::vector<Trees\*> &trees, const Symbol\* symbols, long long int len)

{

long long int sum = 0;

for (long long int i = 0; i < len; ++i)

sum += symbols[i].weight;

return fanno\_shannon(trees, symbols, 0, len, sum);

}

CodeTree\* fanno\_shannon(std::vector<Trees\*> &trees, char\* message)

{

if (\*message == '\0')

return 0;

trees.clear();

trees.push\_back(new Trees(NULL, 0, '-'));

if (\*message == '\0')

return 0;

Symbol symbols[UCHAR\_MAX];

for(long long int i = 0; i < UCHAR\_MAX; ++i)

{

symbols[i].c = i + CHAR\_MIN;

symbols[i].weight = 0;

}

long long int size = strlen(message);

for(long long int i = 0; i < size; ++i)

symbols[message[i] - CHAR\_MIN].weight++;

std::sort(symbols, symbols + UCHAR\_MAX, symbol\_greater);

long long int len = 0;

while(symbols[len].weight > 0 && len < UCHAR\_MAX) len++;

return fanno\_shannon(trees, symbols, len);

}

long long int middle(const Symbol\* symbols, long long int l, long long int sum, long long int& lsum, long long int& rsum)

{

long long int m = l;

lsum = symbols[m].weight;

rsum = sum - lsum;

long long int delta = lsum - rsum;

while (delta + symbols[m + 1].weight < 0)

{

m++;

lsum += symbols[m].weight;

rsum -= symbols[m].weight;

delta = lsum - rsum;

}

return m;

}

fh.h

#pragma once

// Запись в дерево строки по алгоритму Фано - Шеннона

#include "codetree.h"

CodeTree\* fanno\_shannon(std::vector<Trees\*> &trees, char\* message);

CodeTree\* fanno\_shannon(std::vector<Trees\*> &trees, const Symbol\* symbols, long long int len);

Huffman.cpp

#include "huffman.h"

#include "p\_queue.h"

CodeTree \*huffman(std::vector<Trees\*> &trees, const Symbol\* symbols, int len)

{

PriorityQueue<CodeTree\*>\* queue = create\_pq<CodeTree\*>(len);

for(int i = 0; i < len; ++i)

push(queue, symbols[i].weight, make\_leaf(symbols[i]));

while(size(queue) > 1)

{

CodeTree \* tmp[size(queue)];

CodeTree \*tree;

int k = size(queue);

while (size(queue) > 0)

{

tree = pop(queue);

trees.push\_back(new Trees(tree,tree->s.weight,tree->s.c));

tmp[size(queue)] = tree;

}

for (int i = 0;i<k;i++)

{

push(queue, tmp[i]->s.weight, tmp[i]);

}

CodeTree\* ltree = pop(queue);

CodeTree\* rtree = pop(queue);

int weight = ltree->s.weight + rtree->s.weight;

CodeTree\* node = make\_node(weight, ltree, rtree);

ltree->parent = node;

rtree->parent = node;

push(queue, weight, node);

}

CodeTree\* result = pop(queue);

trees.push\_back(new Trees(result,result->s.weight,result->s.c));

destroy\_pq(queue);

return result;

}

CodeTree\* huffman(std::vector<Trees\*> &trees, char\* message)

{

if (\*message == '\0') return 0;

trees.clear();

trees.push\_back(new Trees(NULL,0,'\*'));

if (\*message == '\0') return 0;

Symbol symbols[UCHAR\_MAX];

for(int i = 0; i < UCHAR\_MAX; ++i)

{

symbols[i].c = i + CHAR\_MIN;

symbols[i].weight = 0;

}

int size = strlen(message);

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

symbols[message[i] - CHAR\_MIN].weight++;

std::sort(symbols, symbols + UCHAR\_MAX, symbol\_greater);

int len = 0;

while(symbols[len].weight > 0 && len < UCHAR\_MAX) len++;

return huffman(trees,symbols, len);

}

Huffman.h

#pragma once

// Запись в дерево строки по алгоритму Хаффмана

#include "codetree.h"

CodeTree\* huffman(std::vector<Trees\*> &trees, char\* message);

CodeTree\* huffman(std::vector<Trees\*> &trees, const Symbol\* symbols, int len);

main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

mainwindow.cpp

#include "mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :QMainWindow(parent){

setWindowTitle("Coding by Shannon-Fano and Huffman methods by Bykov I.V. 6383 group");

setMinimumSize(1000, 600);

widget\_ = new TreeWidget(this);

setCentralWidget(widget\_);

}

mainWindow.h

#pragma once

/\* Курсовая работа по дисциплине Программирование.

Выполнил Быков И. В. 6383

Вариант 1. Кодирование и декодирование методами Фано-Шеннона и Хаффмана

Визуалицация метода.

\*/

#include "treewidget.h"

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

private:

TreeWidget \*widget\_;

};

p\_queue.h

#pragma once

// Реализация очереди с приоритетом

#include <utility>

template <typename T>

struct PriorityQueueItem

{

int key;

T data;

};

template <typename T>

struct PriorityQueue

{

int size\_;

int capacity\_;

PriorityQueueItem<T>\* heap\_;

};

template <typename T>

PriorityQueue<T>\* create\_pq(int capacity)

{

PriorityQueue<T>\* pq = new PriorityQueue<T>;

pq->heap\_ = new PriorityQueueItem<T>[capacity];

pq->capacity\_ = capacity;

pq->size\_ = 0;

return pq;

}

template <typename T>

int size(PriorityQueue<T>\* pq)

{

return pq->size\_;

}

template <typename T>

void sift\_up(PriorityQueue<T>\* pq, int index)

{

int parent = (index - 1) / 2;

while(parent >= 0 && pq->heap\_[index].key < pq->heap\_[parent].key)

{

std::swap(pq->heap\_[index], pq->heap\_[parent]);

index = parent;

parent = (index - 1) / 2;

}

}

template <typename T>

bool push(PriorityQueue<T>\* pq, int key, const T& data)

{

if(pq->size\_ >= pq->capacity\_)

return false;

pq->heap\_[pq->size\_].key = key;

pq->heap\_[pq->size\_].data = data;

pq->size\_++;

sift\_up(pq, pq->size\_ - 1);

return true;

}

template <typename T>

void sift\_down(PriorityQueue<T>\* pq, int index)

{

int l = 2 \* index + 1;

int r = 2 \* index + 2;

int min = index;

if(l < pq->size\_ && pq->heap\_[l].key < pq->heap\_[min].key)

min = l;

if(r < pq->size\_ && pq->heap\_[r].key < pq->heap\_[min].key)

min = r;

if(min != index) {

std::swap(pq->heap\_[index], pq->heap\_[min]);

sift\_down(pq, min);

}

}

template <typename T>

T pop(PriorityQueue<T>\* pq)

{

std::swap(pq->heap\_[0], pq->heap\_[pq->size\_ - 1]);

pq->size\_--;

sift\_down(pq, 0);

return pq->heap\_[pq->size\_].data;

}

template <typename T>

void destroy\_pq(PriorityQueue<T>\* pq)

{

delete [] pq->heap\_;

delete pq;

}

treeview.cpp

#include "treeview.h"

#include "treewidget.h"

#include <QPainter>

#include <QFontMetrics>

#include <sstream>

#include <cmath>

#include <QTextStream>

#include <mainwindow.h>

TreeView::TreeView(QWidget \*parent): QWidget(parent), tree(NULL)

{

}

void TreeView::setTree(BinaryTree\* tree)

{

this->tree = tree;

update();

}

void TreeView::paintEvent(QPaintEvent \*)

{

QPainter painter(this);

drawNode(&painter, tree, 0, 0);

}

void TreeView::drawNode(QPainter \*painter, BinaryTree\* root, int x, int y)

{

if(root == NULL)

return;

painter->save();

QPoint p(width() / pow(2, y) \* x + width() / pow(2, y+1), nodeHeight \* y);

QRect rec(p.x() - nodeWidth/4, p.y(), nodeWidth/2, nodeHeight/2);

painter->setPen(Qt::red);

painter->drawEllipse(rec);

painter->setPen(Qt::blue);

QFont font = painter->font() ;

font.setPointSize(10);

painter->setFont(font);

std::stringstream str;

str<<root->symbol<<std::endl;

str<<root->weight;

QString text = fontMetrics().elidedText(

QString::fromLatin1(str.str().c\_str()),

Qt::ElideLeft, rec.width());

painter->drawText(rec, Qt::AlignCenter, text);

painter->setPen(Qt::red);

if(root->left != NULL)

{

painter->drawLine(p.x(), p.y() + nodeHeight / 2, p.x() - width()/pow(2, y+2), p.y() + nodeHeight);

}

if(root->right != NULL)

{

painter->drawLine(p.x(), p.y() + nodeHeight / 2, p.x() + width()/pow(2, y+2), p.y() + nodeHeight);

}

painter->restore();

if(root->left != NULL)

drawNode(painter, root->left, 2\*x, y+1);

if(root->right != NULL)

drawNode(painter, root->right, 2\*x + 1, y+1);

}

treeview.h

#pragma once

// Прорисовка дерева

#include "codetree.h"

class TreeView : public QWidget

{

Q\_OBJECT

public:

explicit TreeView(QWidget \*parent = 0);

void setTree(BinaryTree\* data);

void setTree(CodeTree\* data);

protected:

void paintEvent(QPaintEvent \*event) override;

private:

void drawNode(QPainter \*painter, BinaryTree\* root, int x, int y);

BinaryTree\* tree;

const int nodeWidth{ 80 };

const int nodeHeight{ 80 };

int dynamic = -1;

};

treewidget.cpp

#include "treewidget.h"

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QRegExpValidator>

#include "huffman.h"

#include "fh.h"

TreeWidget::TreeWidget(QWidget \*parent) : QWidget(parent)

{

view\_ = new TreeView(this);

view\_->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding); // задаем размры элемента

QLabel \*label = new QLabel("Enter message: "); // задаем текст лейбла

button\_huffman = new QPushButton("Huffman");

button\_shennon\_fano = new QPushButton("Shennon-Fano");

// button\_huffman->setEnabled(false);

// button\_shennon\_fano->setEnabled(false);

lineEdit\_ = new QLineEdit(); // однострочный текст

QRegExp exp("[a-zA-Z0-9]{0,100}"); // ограничение на ввод входных данных

lineEdit\_->setValidator(new QRegExpValidator(exp, this)); // контроль вводимых даннных

connect(lineEdit\_, SIGNAL(textChanged(QString)), this, SLOT(buttonEnabledHuff()));

connect(lineEdit\_, SIGNAL(textChanged(QString)), this, SLOT(buttonEnabledShennon\_Fano()));

next\_tree=new QPushButton("Next tree");

back\_tree=new QPushButton("Previous tree");

step = new QLabel("");

QHBoxLayout \*layoutCode = new QHBoxLayout;

QLabel \*codeInf = new QLabel("");

code = new QLabel("");

code->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);

message = new QLabel("");

message->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);

layoutCode->addWidget(codeInf);

layoutCode->addWidget(code);

layoutCode->addWidget(message);

QHBoxLayout \*layout = new QHBoxLayout;

layout->addWidget(label);

layout->addWidget(lineEdit\_);

layout->addWidget(button\_huffman);

layout->addWidget(button\_shennon\_fano);

QHBoxLayout \*infLayout=new QHBoxLayout;

infLayout->addWidget(back\_tree);

infLayout->addWidget(next\_tree);

infLayout->addWidget(step);

QVBoxLayout \*mainLayout = new QVBoxLayout;

mainLayout->addLayout(layout);

mainLayout->addLayout(infLayout);

mainLayout->addLayout(layoutCode);

mainLayout->addWidget(view\_);

setLayout(mainLayout);

connect(button\_huffman, &QPushButton::clicked, this, &TreeWidget::insert\_huff);

connect(button\_shennon\_fano, &QPushButton::clicked, this, &TreeWidget::insert\_shf);

connect(next\_tree, &QPushButton::clicked, this, &TreeWidget::next);

connect(back\_tree, &QPushButton::clicked, this, &TreeWidget::back);

}

void TreeWidget::buttonEnabledHuff()

{

button\_huffman->setEnabled(lineEdit\_->hasAcceptableInput());

}

void TreeWidget::buttonEnabledShennon\_Fano()

{

button\_shennon\_fano->setEnabled(lineEdit\_->hasAcceptableInput());

}

int TreeWidget::solve(int s)

{

int D = 1 + 8 \* s;

return (int) (-1 + sqrt(D)) / 2;

}

void TreeWidget::next()

{

if (stateIndex +2 > trees.size())

return;

view\_->setTree(trees[++stateIndex]->root);

step->setText(QString::asprintf("Tree = %d", stateIndex));

update();

}

void TreeWidget::back()

{

if(stateIndex<=1)return;

view\_->setTree(trees[--stateIndex]->root);

//treeIndex--;

step->setText(QString::asprintf("Tree = %d", treeIndex));

update();

}

void TreeWidget::insert\_huff()

{

next\_tree->show();

back\_tree->show();

stateIndex = 1;

step->setText(QString::asprintf("Tree = 0"));

char\* str = new char[256];

str[0]='\0';

strcpy(str,lineEdit\_->text().toLatin1().data());

huffman(trees,str);

levels = solve(trees.size() - 1);

view\_->setTree(trees[stateIndex]->root);

char\* \_encode = encode(huffman(trees,str)/\*huffman(str)\*/, str);

code->setText(QString::asprintf("Encode: %s", \_encode));

char\* \_decode = decode(huffman(trees,str)/\*huffman(str)\*/, \_encode);

message->setText(QString::asprintf("Decode: %s", \_decode));

delete[] str;

update();

}

void TreeWidget::insert\_shf()

{

next\_tree->show();

back\_tree->show();

stateIndex = 1;

step->setText(QString::asprintf("Tree = 0"));

char\* str = new char[256];

str[0]='\0';

strcpy(str,lineEdit\_->text().toLatin1().data());

fanno\_shannon(trees,str);

levels = solve(trees.size() - 1);

view\_->setTree(trees[stateIndex]->root);

char\* \_encode = encode(fanno\_shannon(trees,str)/\*fanno\_shannon(str)\*/,str);

code->setText(QString::asprintf("Encode: %s", \_encode));

char\* \_decode = decode(fanno\_shannon(trees,str)/\*fanno\_shannon(str)\*/, \_encode);

message->setText(QString::asprintf("Decode: %s", \_decode));

delete[] str;

update();

}

treewidget.h

#pragma once

// Создание окна и обработка функций в визуальном режиме

#include "treeview.h"

class TreeWidget : public QWidget

{

Q\_OBJECT

public:

explicit TreeWidget(QWidget \*parent = 0);

int solve(int s);

void fill\_beginers();

bool dynamic;

void buttonEnabledShennon\_Fano();

void buttonEnabledHuff();

public slots:

void insert\_huff();

void insert\_shf();

void next();

void back();

private:

TreeView \*view\_;

QLineEdit \*lineEdit\_;

QPushButton \*button\_huffman;

QPushButton \*button\_shennon\_fano;

QLabel \*code ;

QLabel \*message;

QLabel \*step;

QPushButton \*next\_tree;

QPushButton \*back\_tree;

std::vector<Trees\*> trees;

int stateIndex = 0;

int treeIndex = 0;

int \*beginers;

int levels;

};

Оглавление

[1. Постановка задачи 2](#_Toc501313010)

[2. Теоретический материал 2](#_Toc501313011)

[3. Формальная постановка задачи 6](#_Toc501313012)

[4. Разработка структур данных и алгоритмов 6](#_Toc501313013)

[5. Спецификация программы 8](#_Toc501313014)

[6. Тестовые данные 9](#_Toc501313015)

[7. Анализ результатов и выводы 10](#_Toc501313016)

[Приложение А. Программа исследования. Листинг 11](#_Toc501313017)

[Приложение Б. Программа визуализации. Листинг. 22](#_Toc501313018)