

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА
по дисциплине «Алгоритмы и структуры данных»
Тема: Демонстрация статического кодирования и декодирования
текстового документа методами Хаффмана и Фано-Шеннона

Студент гр. 9304

Атаманов С. Д.

Преподаватель

Филатов А. Ю.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Атаманов С. Д.

Группа 9304

Тема работы: Демонстрация статического кодирования и декодирования текстового документа методами Хаффмана и Фано-Шеннона

Исходные данные:

Для работы программы требуются компилятор языка C++, поддерживающий C++17 стандарт

Содержание пояснительной записки:

- Аннотация
- Содержание
- Введение
- Формальная постановка задачи
- Описание алгоритма
- Описание структур данных
- Описание функций
- Описание интерфейса пользователя
- Тестирование
- Заключение
- Список использованных источников
- Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 27 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 28.12.2020

Студент

Атаманов С. Д.

Преподаватель

Филатов А. Ю.

АННОТАЦИЯ

Для выполнения курсовой работы была разработана программа на языке программирования C++, которая осуществляет демонстрацию алгоритмов статического кодирования Хаффмана и Фано-Шеннона.

SUMMARY

To perform the course work, a program was developed in the C++ programming language, which demonstrates the Huffman and Fano-Shannon static coding algorithms.

СОДЕРЖАНИЕ

	Введение	5
1.	Формальная постановка задачи	6
2.	Описание алгоритма	7
3.	Описание структур данных	8
4.	Описание функций	9
5.	Описание интерфейса пользователя	10
6.	Тестирование	11
	Заключение	15
	Список использованных источников	16
	Приложение А. Исходный код программы	17

ВВЕДЕНИЕ

Алгоритмы сжатия данных представляют собой алгоритмические преобразования данных, производимое с целью уменьшения занимаемого им объёма. В данной работе будут рассмотрены два статических алгоритма: Хаффмана и Фано-Шеннона.

Цель работы: реализовать алгоритмы сжатия, используя язык программирования C++. Осуществить демонстрацию работы алгоритмов на примере текстового документа, закодировав и декодировав его обратно.

1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Реализовать визуализацию статического кодирования и декодирования Хаффмана и Шано-Феннона.

Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

2. ОПИСАНИЕ АЛГОРИТМА

Программа реализует алгоритмы кодирования Хаффмана и Фано-Шеннона.

Фано-Шеннон:

Работа алгоритма основывается на Бинарном дереве. Алгоритм получает таблицу символов с частотой их появления в тексте(далее — весом). Все символы складываются в единую строку, их веса складываются. В узлы дерева записываются отдельные символы или строка состоящая из них. Затем, если это строка, она разделяется по весу напополам, в левое поддерево записывается строка с меньшим весом, в правое с большим. Так продолжается до тех пор, пока в листьях не будут отдельные символы. После по дереву проходятся, добавляя к каждому коду левого поддерева каждого узла 0, а к правому 1. В результате у каждого символа получается уникальный код который тем короче, чем чаще встречается символ и наоборот, тем длинее, чем символ встречается реже.

Хаффман:

Работа алгоритма также основывается на Бинарном дереве. Алгоритм получает на вход строку с весом всех символов в тексте, а также таблицу символов с их весом. Далее заполняется вектор строк, помещая каждый символ в отдельную ячейку. Затем берутся первые два элемента вектора, объединяются, создавая узел, с помощью отдельной функции. Данный узел затем помещается в отдельный вектор. Объединенный узел помещается обратно в исходный вектор. Данный вектор потом отсортировывается по весам. Данные действия повторяются до тех пор, пока в исходном векторе не останется 1 узла. В результате получается бинарное дерево, узлами которого являются строки с их весами, а листьями – отдельные символы с их весами. Чтобы получить коды символов, по дереву проходятся, добавляя к каждому коду левого поддерева каждого узла 0, а к правому 1. В результате у каждого символа получается уникальный код который тем короче, чем чаще встречается символ и наоборот, тем длинее, чем символ встречается реже.

3. ОПИСАНИЕ СТРУКТУР ДАННЫХ

1. Класс BinTreeNode – узел бинарного дерева.

Класс содержит следующие публичные поля:

`std::shared_ptr<BinTreeNode> left` и `right`, которые хранят умный указатель на левое и правое поддерево, `std::pair<std::string, int> data`, которое хранит значение узла в паре строка – её вес. Также класс содержит публичные методы `getShannonTree()`, `getHuffmanTree()`, которые позволяют получить соответственно дерево Фано-Шеннона и Хаффмана. Метод `makeNode()` позволяет создать узел `BinTreeNode` из различных входных данных. Структура класса представлена на рисунке 1.

```
class BinTreeNode {
public:
    std::shared_ptr<BinTreeNode> left{nullptr};
    std::shared_ptr<BinTreeNode> right{nullptr};
    std::pair<std::string, int> data;

    std::shared_ptr<BinTreeNode>
    getShannonFanoTree(std::pair<std::string, int> stringWithWeight, std::map<char, int> usingSymbols,
                      std::map<char, std::string> &codes, std::string code);

    std::shared_ptr<BinTreeNode>
    getHuffmanTree(std::pair<std::string, int> stringWithWeight, std::map<char, int> usingSymbols);

    std::shared_ptr<BinTreeNode>
    makeNode(std::shared_ptr<BinTreeNode> leftNode, std::shared_ptr<BinTreeNode> rightNode,
             std::pair<std::string, int> leftFutureNode, std::pair<std::string, int> rightFutureNode);
}
```

Рисунок 1 – Структура класса BinTreeNode

4. ОПИСАНИЕ ФУНКЦИЙ

Функция `HuffmanComparator()` – используется для сортировки вектора в качестве компаратора.

Функция `getListOfElem()` – позволяет получить словарь символов, используемых в кодируемом тексте вместе с их весами.

Функция `getStringWithWeigh()` – позволяет получить из словаря символов цельную, лексикографически отсортированную строку, вместе с её весом.

Функция `getCodesFromHuffman()` – позволяет получить словарь с кодировкой каждого символа текста.

Функция `printTree()` – позволяет распечатать деревья как Шано-Феннона, так и Хаффмана.

Функция `HuffmanStringComparator()` – используется для сортировки строки в качестве компаратора.

Функция `printHuffman()` – распечатывает на экран ход алгоритма Хаффмана.

Функция `printCodeTable()` – распечатывает на экран таблицу символов.

Функция `getCodesFromFile()` – берет из текстового документа символы и их коды для декодирования.

Функция `encode()` – выполняет кодирование, вывод демонстрационной информации на экран и декодирование.

5. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Программа принимает на вход строку, которую нужно закодировать. Строка должна содержать символы ASCII таблицы. Количество символов: больше 1.

После ввода строки программа выполняет кодирование и декодирование двумя способами: методом Хаффмана и методом Шано-Феннона. Демонстрационная информация выводится на экран и сопровождается пояснениями. Более никаких действий от пользователя не требуется.

6. ТЕСТИРОВАНИЕ

Для тестирования были выбраны несколько предложений, которые будут закодированы. Для их удобного прогона был написан bash script.

Пример работы программы представлен на рисунке 2.

```
neither@neither: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/sw$ ./sw
Hello! My name is Sergey!!!
Исходные символы в кодируемом тексте и их веса:
(4), l(1), )(3), H(1), S(1), a(1), e(4), g(1), l(1), l(2), n(1), n(1), o(1), r(1), s(1), y(2)
строка, полученная сложением всех символов, посещая сортировки, и сложении их весов
e)lylHMSaglnnors(27)

Построение дерева Шано-Феннона
Head: e)lylHMSaglnnors(27)

Шаг 1 Узел со строкой e)lylHMSaglnnors и весом 27 делится на узел со строкой 'e)' и весом 13, который идет в левое поддерево, а также узел со строкой 'ylHMSaglnnors' и весом 14, который идет в правое поддерево
|
|--- L: 'e)'(13) Code = '0'
|
|--- R: 'ylHMSaglnnors'(14) Code = '1'

Шаг 2 Узел со строкой 'e)' и весом 13 делится на узел со строкой 'l)' и весом 5, который идет в левое поддерево, а также узел со строкой 'e' и весом 8, который идет в правое поддерево
|
|--- L: 'l)'(5) Code = '00'
|
|--- R: 'e'(8) Code = '01'

Шаг 3 Узел со строкой 'l)' и весом 5 делится на узел со строкой 'l' и весом 2, который идет в левое поддерево, а также узел со строкой ')' и весом 3, который идет в правое поддерево
|
|--- L: 'l'(2) Code = '000'
|
|--- R: ')' (3) Code = '001'

Шаг 4 Узел со строкой 'e' и весом 8 делится на узел со строкой 'l' и весом 4, который идет в левое поддерево, а также узел со строкой 'e' и весом 4, который идет в правое поддерево
|
|--- L: 'l'(4) Code = '010'
|
|--- R: 'e'(4) Code = '011'

Шаг 5 Узел со строкой 'ylHMSaglnnors' и весом 14 делится на узел со строкой 'yHMSa' и весом 7, который идет в левое поддерево, а также узел со строкой 'glnnors' и весом 7, который идет в правое поддерево
|
|--- L: 'yHMSa'(7) Code = '10'
|
|--- R: 'glnnors'(7) Code = '11'

Шаг 6 Узел со строкой 'yHMSa' и весом 7 делится на узел со строкой 'MSa' и весом 3, который идет в левое поддерево, а также узел со строкой 'yH' и весом 4, который идет в правое поддерево
|
|--- L: 'MSa'(3) Code = '100'
|
|--- R: 'yH'(4) Code = '101'

Шаг 7 Узел со строкой 'MSa' и весом 3 делится на узел со строкой 'a' и весом 1, который идет в левое поддерево, а также узел со строкой 'MS' и весом 2, который идет в правое поддерево
|
|--- L: 'a'(1) Code = '1000'
|
|--- R: 'MS'(2) Code = '1001'

Шаг 8 Узел со строкой 'MS' и весом 2 делится на узел со строкой 'M' и весом 1, который идет в левое поддерево, а также узел со строкой 'S' и весом 1, который идет в правое поддерево
|
|--- L: 'M'(1) Code = '10010'
|
|--- R: 'S'(1) Code = '10011'

Шаг 9 Узел со строкой 'yH' и весом 4 делится на узел со строкой 'y' и весом 2, который идет в левое поддерево, а также узел со строкой 'H' и весом 2, который идет в правое поддерево
|
|--- L: 'y'(2) Code = '1010'
|
|--- R: 'H'(2) Code = '1011'

Шаг 10 Узел со строкой 'H' и весом 2 делится на узел со строкой 'l' и весом 1, который идет в левое поддерево, а также узел со строкой 'n' и весом 1, который идет в правое поддерево
|
|--- L: 'l'(1) Code = '10110'
|
|--- R: 'n'(1) Code = '10111'

Шаг 11 Узел со строкой 'glnnors' и весом 7 делится на узел со строкой 'ors' и весом 3, который идет в левое поддерево, а также узел со строкой 'glnn' и весом 4, который идет в правое поддерево
|
|--- L: 'ors'(3) Code = '110'
|
|--- R: 'glnn'(4) Code = '111'

Шаг 12 Узел со строкой 'ors' и весом 3 делится на узел со строкой 's' и весом 1, который идет в левое поддерево, а также узел со строкой 'or' и весом 2, который идет в правое поддерево
|
|--- L: 's'(1) Code = '1100'
|
|--- R: 'or'(2) Code = '1101'

Шаг 13 Узел со строкой 'or' и весом 2 делится на узел со строкой 'o' и весом 1, который идет в левое поддерево, а также узел со строкой 'r' и весом 1, который идет в правое поддерево
|
|--- L: 'o'(1) Code = '11010'
|
|--- R: 'r'(1) Code = '11011'

Шаг 14 Узел со строкой 'glnn' и весом 4 делится на узел со строкой 'gl' и весом 2, который идет в левое поддерево, а также узел со строкой 'nn' и весом 2, который идет в правое поддерево
|
|--- L: 'gl'(2) Code = '1110'
|
|--- R: 'nn'(2) Code = '1111'

Шаг 15 Узел со строкой 'gl' и весом 2 делится на узел со строкой 'g' и весом 1, который идет в левое поддерево, а также узел со строкой 'l' и весом 1, который идет в правое поддерево
|
|--- L: 'g'(1) Code = '11100'
|
|--- R: 'l'(1) Code = '11101'

Шаг 16 Узел со строкой 'nn' и весом 2 делится на узел со строкой 'n' и весом 1, который идет в левое поддерево, а также узел со строкой 'n' и весом 1, который идет в правое поддерево
|
|--- L: 'n'(1) Code = '11110'
|
|--- R: 'n'(1) Code = '11111'

Полученное дерево:
Head: e)lylHMSaglnnors(27)
|
|--- L: 'e)'(13) Code = '0'
|
|--- R: 'ylHMSaglnnors'(14) Code = '1'
|
|--- L: 'l)'(5) Code = '00'
|
|--- R: 'e'(8) Code = '01'
|
|--- L: 'l'(2) Code = '000'
|
|--- R: ')' (3) Code = '001'
|
|--- L: 'l'(4) Code = '010'
|
|--- R: 'e'(4) Code = '011'
|
|--- L: 'yHMSa'(7) Code = '10'
|
|--- R: 'glnnors'(7) Code = '11'
|
|--- L: 'MSa'(3) Code = '100'
|
|--- R: 'yH'(4) Code = '101'
|
|--- L: 'a'(1) Code = '1000'
|
|--- R: 'MS'(2) Code = '1001'
|
|--- L: 'M'(1) Code = '10010'
|
|--- R: 'S'(1) Code = '10011'
|
|--- L: 'y'(2) Code = '1010'
|
|--- R: 'H'(2) Code = '1011'
|
|--- L: 'l'(1) Code = '10110'
|
|--- R: 'n'(1) Code = '10111'
|
|--- L: 'ors'(3) Code = '110'
|
|--- R: 'glnn'(4) Code = '111'
|
|--- L: 's'(1) Code = '1100'
|
|--- R: 'or'(2) Code = '1101'
|
|--- L: 'o'(1) Code = '11010'
|
|--- R: 'r'(1) Code = '11011'
|
|--- L: 'gl'(2) Code = '1110'
|
|--- R: 'nn'(2) Code = '1111'
|
|--- L: 'g'(1) Code = '11100'
|
|--- R: 'l'(1) Code = '11101'
|
|--- L: 'n'(1) Code = '11110'
|
|--- R: 'n'(1) Code = '11111'
```

Рисунок 2 — Пример работы программы

```
Полученная таблица кодов для символов:
|--- L: 's'(1) Code = '1100'
|--- R: 'or'(2) Code = '1101'

Шаг 13 Узел со строкой or и весом 2 делится на узел со строкой 'o' и весом 1, который идет в левое поддерево, а также узел со строкой 'r' и весом 1, который идет в правое поддерево
|
|--- L: 'o'(1) Code = '11010'
|
|--- R: 'r'(1) Code = '11011'

Шаг 14 Узел со строкой glnn и весом 4 делится на узел со строкой 'gl' и весом 2, который идет в левое поддерево, а также узел со строкой 'nn' и весом 2, который идет в правое поддерево
|
|--- L: 'gl'(2) Code = '1110'
|
|--- R: 'nn'(2) Code = '1111'

Шаг 15 Узел со строкой gl и весом 2 делится на узел со строкой 'g' и весом 1, который идет в левое поддерево, а также узел со строкой 'l' и весом 1, который идет в правое поддерево
|
|--- L: 'g'(1) Code = '11100'
|
|--- R: 'l'(1) Code = '11101'

Шаг 16 Узел со строкой nn и весом 2 делится на узел со строкой 'n' и весом 1, который идет в левое поддерево, а также узел со строкой 'n' и весом 1, который идет в правое поддерево
|
|--- L: 'n'(1) Code = '11110'
|
|--- R: 'n'(1) Code = '11111'

Полученное дерево:
Head: e)lylHMSaglnnors(27)
|
|--- L: 'e)'(13) Code = '0'
|
|--- R: 'ylHMSaglnnors'(14) Code = '1'
|
|--- L: 'l)'(5) Code = '00'
|
|--- R: 'e'(8) Code = '01'
|
|--- L: 'l'(2) Code = '000'
|
|--- R: ')' (3) Code = '001'
|
|--- L: 'l'(4) Code = '010'
|
|--- R: 'e'(4) Code = '011'
|
|--- L: 'yHMSa'(7) Code = '10'
|
|--- R: 'glnnors'(7) Code = '11'
|
|--- L: 'MSa'(3) Code = '100'
|
|--- R: 'yH'(4) Code = '101'
|
|--- L: 'a'(1) Code = '1000'
|
|--- R: 'MS'(2) Code = '1001'
|
|--- L: 'M'(1) Code = '10010'
|
|--- R: 'S'(1) Code = '10011'
|
|--- L: 'y'(2) Code = '1010'
|
|--- R: 'H'(2) Code = '1011'
|
|--- L: 'l'(1) Code = '10110'
|
|--- R: 'n'(1) Code = '10111'
|
|--- L: 'ors'(3) Code = '110'
|
|--- R: 'glnn'(4) Code = '111'
|
|--- L: 's'(1) Code = '1100'
|
|--- R: 'or'(2) Code = '1101'
|
|--- L: 'o'(1) Code = '11010'
|
|--- R: 'r'(1) Code = '11011'
|
|--- L: 'gl'(2) Code = '1110'
|
|--- R: 'nn'(2) Code = '1111'
|
|--- L: 'g'(1) Code = '11100'
|
|--- R: 'l'(1) Code = '11101'
|
|--- L: 'n'(1) Code = '11110'
|
|--- R: 'n'(1) Code = '11111'
```

Рисунок 3 — Пример работы программы

```
Activities Terminal
neither@neither: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/cw
полученная таблица кодов для символов:
(010, l(10110), )(001, H(10111), M(10010), S(10011), a(1000), e(011), g(11100), l(11101), l(000), m(11110), n(11111), o(11010), r(11011), s(1100), y(1010)
.....
Ход алгоритма Хаффмана
S(1), r(1), o(1), n(1), m(1), l(1), g(1), a(1), S(1), M(1), H(1), l(1), y(2), l(2), )(3), e(4), (4)
Шаг 1 первые два узла: 'S'(1) и 'r'(1) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'rs' с весом 2
o(1), n(1), m(1), l(1), g(1), a(1), S(1), M(1), H(1), l(1), y(2), l(2), rs(2), )(3), e(4), (4)
Шаг 2 первые два узла: 'o'(1) и 'n'(1) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'no' с весом 2
m(1), l(1), g(1), a(1), S(1), M(1), H(1), l(1), y(2), l(2), rs(2), no(2), )(3), e(4), (4)
Шаг 3 первые два узла: 'm'(1) и 'l'(1) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'ln' с весом 2
g(1), a(1), S(1), M(1), H(1), l(1), y(2), l(2), rs(2), no(2), ln(2), )(3), e(4), (4)
Шаг 4 первые два узла: 'g'(1) и 'a'(1) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'ag' с весом 2
S(1), M(1), H(1), l(1), y(2), l(2), rs(2), no(2), ln(2), ag(2), )(3), e(4), (4)
Шаг 5 первые два узла: 'S'(1) и 'M'(1) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'MS' с весом 2
H(1), l(1), y(2), l(2), rs(2), no(2), ln(2), ag(2), MS(2), )(3), e(4), (4)
Шаг 6 первые два узла: 'H'(1) и 'l'(1) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'lH' с весом 2
y(2), l(2), rs(2), no(2), ln(2), ag(2), MS(2), lH(2), )(3), e(4), (4)
Шаг 7 первые два узла: 'y'(2) и 'l'(2) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'ly' с весом 4
rs(2), no(2), ln(2), ag(2), MS(2), lH(2), )(3), e(4), (4), ly(4)
Шаг 8 первые два узла: 'rs'(2) и 'no'(2) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'nors' с весом 4
ln(2), ag(2), MS(2), lH(2), )(3), e(4), (4), ly(4), nors(4)
Шаг 9 первые два узла: 'ln'(2) и 'ag'(2) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'agln' с весом 4
MS(2), lH(2), )(3), e(4), (4), ly(4), nors(4), agln(4)
Шаг 10 первые два узла: 'MS'(2) и 'lH'(2) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'lHMS' с весом 4
)(3), e(4), (4), ly(4), nors(4), agln(4), lHMS(4)
```

Рисунок 4 — Пример работы программы

```
Activities Terminal
neither@neither: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/cw
)(3), e(4), (4), ly(4), nors(4), agln(4), lHMS(4)
Шаг 11 первые два узла: ')('(3) и 'e'(4) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'je' с весом 7
(4), ly(4), nors(4), agln(4), lHMS(4), je(7)
Шаг 12 первые два узла: ' '(4) и 'ly'(4) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел ' ly' с весом 8
nors(4), agln(4), lHMS(4), je(7), ly(8)
Шаг 13 первые два узла: 'nors'(4) и 'agln'(4) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'aglnnors' с весом 8
lHMS(4), je(7), ly(8), aglnnors(8)
Шаг 14 первые два узла: 'lHMS'(4) и 'je'(7) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел 'l)HMSe' с весом 11
ly(8), aglnnors(8), l)HMSe(11)
Шаг 15 первые два узла: ' ly'(8) и 'aglnnors'(8) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел ' aglInnorsy' с весом 16
l)HMSe(11), aglInnorsy(16)
Шаг 16 на последнем шаге оставшиеся два узла: l)HMSaeglInnorsy(27) и aglInnorsy(16) складываются вместе: складываются сами строки, сортируются в лексикографическом порядке, а также складываются их веса.
Получается узел l)HMSaeglInnorsy с весом 27, который и является головой полученного дерева
l)HMSaeglInnorsy(27)
Дерево Хаффмана, полученное при работе алгоритма Хаффмана
Head: SMHl)je ylsronniga(27)
|--- L: SMHl)je(11) Code = '0'
|   |--- L: SMHl'(4) Code = '00'
|   |   |--- L: SM'(2) Code = '000'
|   |   |   |--- L: S'(1) Code = '0000'
|   |   |   |   |--- R: H'(1) Code = '0001'
|   |   |   |   |--- R: H'(2) Code = '001'
|   |   |   |   |   |--- L: H'(1) Code = '0010'
|   |   |   |   |   |--- R: l'(1) Code = '0011'
|   |   |   |--- R: je'(7) Code = '01'
|   |   |   |   |--- L: )( '(3) Code = '010'
|   |   |   |   |--- R: e'(4) Code = '011'
|   |   |--- R: ylsronniga(16) Code = '1'
|   |   |   |--- L: yl'(8) Code = '10'
|   |   |   |   |--- L: ' '(4) Code = '100'
|   |   |   |   |   |--- R: yl'(4) Code = '101'
|   |   |   |   |   |   |--- L: y'(2) Code = '1010'
|   |   |   |   |   |   |--- R: l'(2) Code = '1011'
|   |   |   |   |--- R: sronniga(8) Code = '11'
|   |   |   |   |   |--- L: sron'(4) Code = '110'
|   |   |   |   |   |--- L: sr'(2) Code = '1100'
|   |   |   |   |   |--- L: 'a'(1) Code = '11000'
```

Рисунок 5 — Пример работы программы

```
Activities Terminal
neither@neither: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/св

--- R: 'ylsronniga'(16) Code = '1'
--- L: 'yl'(8) Code = '10'
--- L: 'l'(4) Code = '100'
--- R: 'yl'(4) Code = '101'
--- L: 'y'(2) Code = '1010'
--- R: 'l'(2) Code = '1011'
--- R: 'sronniga'(8) Code = '11'
--- L: 'sron'(4) Code = '110'
--- L: 'sr'(2) Code = '1100'
--- L: 's'(1) Code = '11000'
--- R: 'r'(1) Code = '11001'
--- R: 'on'(2) Code = '1101'
--- L: 'o'(1) Code = '11010'
--- R: 'n'(1) Code = '11011'
--- R: 'niga'(4) Code = '111'
--- L: 'nl'(2) Code = '1110'
--- L: 'n'(1) Code = '11100'
--- R: 'l'(1) Code = '11101'
--- R: 'ga'(2) Code = '1111'
--- L: 'g'(1) Code = '11110'
--- R: 'a'(1) Code = '11111'

Полученная таблица кодов для символов:
(100), l(0011), )(010), N(0010), M(0001), S(0000), a(1111), e(011), g(1110), l(1101), n(1100), n(1011), o(1010), r(11001), s(11000), y(1010)
Шифрование:
Шифрование происходит путем обращения к кодам, созданным ранее. Берется символ, её код ищется в таблице и заменяется в тексте на этот символ
С помощью таблицы Фано-Шеннона:
Hello! My name is Sergey)))
Шаг 1: Текущий символ: H ,его код: 10111
10111ello! My name is Sergey)))
Шаг 2: Текущий символ: e ,его код: 011
1011011llo! My name is Sergey)))
Шаг 3: Текущий символ: l ,его код: 000
1011011000lo! My name is Sergey)))
Шаг 4: Текущий символ: l ,его код: 000
101101100000lo! My name is Sergey)))
Шаг 5: Текущий символ: o ,его код: 1010
10110110000001010lo! My name is Sergey)))
Шаг 6: Текущий символ: ! ,его код: 1010
101101100000010101010lo! My name is Sergey)))
Шаг 7: Текущий символ: ,его код: 010
101101100000010101010010My name is Sergey)))
Шаг 8: Текущий символ: M ,его код: 10010
1011011000000101010100101010My name is Sergey)))
Шаг 9: Текущий символ: y ,его код: 1010
10110110000001010101001010101010 name is Sergey)))
Шаг 10: Текущий символ: ,его код: 010
1011011000000101010100101001010101010name is Sergey)))
Шаг 11: Текущий символ: n ,его код: 11111
101101100000010101010010100101010101111name is Sergey)))
Шаг 12: Текущий символ: a ,его код: 1000
10110110000001010101001010010101011111000ne is Sergey)))
Шаг 13: Текущий символ: n ,его код: 11110
1011011000000101010100101001010101111100011110e is Sergey)))
Шаг 14: Текущий символ: e ,его код: 011
1011011000000101010100101001010101111100011110011 is Sergey)))
Шаг 15: Текущий символ: ,его код: 010
1011011000000101010100101001010101111100011100101010 is Sergey)))
Шаг 16: Текущий символ: l ,его код: 1101
1011011000000101010100101001010101111100011100101010101s Sergey)))
Шаг 17: Текущий символ: s ,его код: 1100
101101100000010101010010100101010111110001110010101011011100 Sergey)))
Шаг 18: Текущий символ: ,его код: 010
101101100000010101010010100101010111110001110010101011011100010Sergey)))
Шаг 19: Текущий символ: S ,его код: 10011
1011011000000101010100101001010101111100011100101010111000100011ergey)))
Шаг 20: Текущий символ: e ,его код: 011
101101100000010101010010100101010111110001110010101011100010100111rgey)))
Шаг 21: Текущий символ: r ,его код: 11011
101101100000010101010010100101010111110001110010101011100010100110111011gey)))
Шаг 22: Текущий символ: g ,его код: 11100
1011011000000101010100101001010101111100011100101010111000101001110111100ey)))
Шаг 23: Текущий символ: e ,его код: 011
10110110000001010101001010010101011111000111001010101110001010011011110111100011y)))
Шаг 24: Текущий символ: y ,его код: 1010
10110110000001010101001010010101011111000111001010101110001010011011101111000111010)))
Шаг 25: Текущий символ: ) ,его код: 001
10110110000001010101001010010101011111000111001010101110001010011011101111000111010001)))
Шаг 26: Текущий символ: ) ,его код: 001
10110110000001010101001010010101011111000111001010101110001010011011101111000111010001001)))
Шаг 27: Текущий символ: ) ,его код: 001
101101100000010101010010100101010111110001110010101011100010100110111101111000111010001001001)))
```

Рисунок 6 — Пример работы программы

```
Activities Terminal
neither@neither: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/св

Шаг 9: Текущий символ: y ,его код: 1010
10110110000001010101001010010101010 name is Sergey)))
Шаг 10: Текущий символ: ,его код: 010
101101100000010101010010100101010101010name is Sergey)))
Шаг 11: Текущий символ: n ,его код: 11111
1011011000000101010100101001010101011111name is Sergey)))
Шаг 12: Текущий символ: a ,его код: 1000
10110110000001010101001010010101011111000ne is Sergey)))
Шаг 13: Текущий символ: n ,его код: 11110
1011011000000101010100101001010101111100011110e is Sergey)))
Шаг 14: Текущий символ: e ,его код: 011
1011011000000101010100101001010101111100011110011 is Sergey)))
Шаг 15: Текущий символ: ,его код: 010
1011011000000101010100101001010101111100011100101010 is Sergey)))
Шаг 16: Текущий символ: l ,его код: 1101
1011011000000101010100101001010101111100011100101010101s Sergey)))
Шаг 17: Текущий символ: s ,его код: 1100
101101100000010101010010100101010111110001110010101011011100 Sergey)))
Шаг 18: Текущий символ: ,его код: 010
101101100000010101010010100101010111110001110010101011011100010Sergey)))
Шаг 19: Текущий символ: S ,его код: 10011
10110110000001010101001010010101011111000111001010101110001010011ergey)))
Шаг 20: Текущий символ: e ,его код: 011
10110110000001010101001010010101011111000111001010101110001010011011rgey)))
Шаг 21: Текущий символ: r ,его код: 11011
1011011000000101010100101001010101111100011100101010111000101001101110111gey)))
Шаг 22: Текущий символ: g ,его код: 11100
101101100000010101010010100101010111110001110010101011100010100110111011100ey)))
Шаг 23: Текущий символ: e ,его код: 011
10110110000001010101001010010101011111000111001010101110001010011011110111100011y)))
Шаг 24: Текущий символ: y ,его код: 1010
10110110000001010101001010010101011111000111001010101110001010011011101111000111010)))
Шаг 25: Текущий символ: ) ,его код: 001
10110110000001010101001010010101011111000111001010101110001010011011101111000111010001)))
Шаг 26: Текущий символ: ) ,его код: 001
10110110000001010101001010010101011111000111001010101110001010011011101111000111010001001)))
Шаг 27: Текущий символ: ) ,его код: 001
101101100000010101010010100101010111110001110010101011100010100110111101111000111010001001001)))
```

Рисунок 7 — Пример работы программы

```
Activities Terminal
dek 28 16:26
neyther@neyther: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/cw

Окончательная закодированная строка:
10111011000000110101011001010010100101111100011100110101101110001010011011101111000111010001001001

-----

С помощью таблицы Хаффмана:
Hello! My name is Sergey)))

Шаг 1: Текущий символ: H ,его код: 0010
0010Hello! My name is Sergey)))

Шаг 2: Текущий символ: e ,его код: 011
0010011!lo! My name is Sergey)))

Шаг 3: Текущий символ: l ,его код: 1011
0010011101!lo! My name is Sergey)))

Шаг 4: Текущий символ: l ,его код: 1011
00100111011101!lo! My name is Sergey)))

Шаг 5: Текущий символ: o ,его код: 11010
0010011101110111010! My name is Sergey)))

Шаг 6: Текущий символ: ! ,его код: 0011
00100111011101110100011 My name is Sergey)))

Шаг 7: Текущий символ: , ,его код: 100
00100111011101110100011100My name is Sergey)))

Шаг 8: Текущий символ: M ,его код: 0001
001001110111011101000111000001y name is Sergey)))

Шаг 9: Текущий символ: y ,его код: 1010
0010011101110111010001110000011010 name is Sergey)))

Шаг 10: Текущий символ: , ,его код: 100
001001110111011101000111000001101010name is Sergey)))

Шаг 11: Текущий символ: n ,его код: 11011
0010011101110111010001110000011010100111011ame is Sergey)))

Шаг 12: Текущий символ: a ,его код: 11111
0010011101110111010001110000011010100110111111ame is Sergey)))

Шаг 13: Текущий символ: m ,его код: 11100
0010011101110111010001110000011010100111111111100e is Sergey)))

Шаг 14: Текущий символ: e ,его код: 011
001001110111011101000111000001101010011111111100011 is Sergey)))

Шаг 15: Текущий символ: , ,его код: 100
0010011101110111010001110000011010100110111111111000111001s Sergey)))

Шаг 16: Текущий символ: t ,его код: 11101
00100111011101110100011100000110101001101111111110001110011101s Sergey)))
```

Рисунок 8 — Пример работы программы

```
Activities Terminal
dek 28 16:27
neyther@neyther: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/cw

Шаг 17: Текущий символ: s ,его код: 11000
0010011101110111010001110000011010100110111111111000111001110111000 Sergey)))

Шаг 18: Текущий символ: , ,его код: 100
0010011101110111010001110000011010100110111111111000111001110111000100Sergey)))

Шаг 19: Текущий символ: S ,его код: 0000
00100111011101110100011100000110101001101111111110001110011101110001000000Sergey)))

Шаг 20: Текущий символ: e ,его код: 011
0010011101110111010001110000011010100110111111111000111001110011000000011rgey)))

Шаг 21: Текущий символ: r ,его код: 11001
001001110111011101000111000001101010011011111111100011100111001100000001111001gey)))

Шаг 22: Текущий символ: g ,его код: 11110
00100111011101110100011100000110101001101111111110001110011101110001000000011110011110ey)))

Шаг 23: Текущий символ: e ,его код: 011
0010011101110111010001110000011010100110111111111000111001110111000100000001110011110011y)))

Шаг 24: Текущий символ: y ,его код: 1010
00100111011101110100011100000110101001101111111110001110011101110001000000011100111100111010)))

Шаг 25: Текущий символ: ) ,его код: 010
00100111011101110100011100000110101001101111111110001110011101110001000000011100111100111010010)))

Шаг 26: Текущий символ: ) ,его код: 010
00100111011101110100011100000110101001101111111110001110011101110001000000011100111100111010010010)))

Шаг 27: Текущий символ: ) ,его код: 010
10111011000000011010101100101001010010111110001110011010110111000101001101111000111010001001001

Окончательная закодированная строка:
101110110000000110101011001010010100101111100011100110101101110001010011011101111000111010001001001

-----

Декодирование:
Декодирование происходит в порядке, обратном кодированию: код заменяется соответствующим символом
С помощью таблицы Шеннона-Фано:
101110110000000110101011001010010101111100011100110101110001010011011101111000111010001001001

Шаг 1: Текущий код: '10111', символ, соответствующий коду: 'H'
He1100000011010101100101001010100101111100011100110011011100010100110111011110001110001001001

Шаг 2: Текущий код: '011', символ, соответствующий коду: 'e'
He000000110101011001010010100101111100011100110011011101110001010011011101111000111010001001001

Шаг 3: Текущий код: '000', символ, соответствующий коду: 'l'
He100011010101001010010100101111100011100110101101110001010011011101111000111010001001001

Шаг 4: Текущий код: '000', символ, соответствующий коду: 'l'
He11101010100101001010100101111100011100110101101110001010011011101111000111010001001001

Шаг 5: Текущий код: '11010', символ, соответствующий коду: 'o'
```

Рисунок 9 — Пример работы программы

```
Activities Terminal dek 28 16:27 neyther@neyther: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/св

Шаг 5: Текущий код: '11010', символ, соответствующий коду: 'o'
Hello! 1010110010100101001011111100011100110101101110001010011011101111000111010001001001

Шаг 6: Текущий код: '10110', символ, соответствующий коду: 'l'
Hello! 0101001010010101111100011100110101101110001010011011101111000111010001001001

Шаг 7: Текущий код: '010', символ, соответствующий коду: ' '
Hello! 10010100101111100011100110101101110001010011011101111000111010001001001

Шаг 8: Текущий код: '10010', символ, соответствующий коду: 'M'
Hello! M10100101111100011100110101101110001010011011101111000111010001001001

Шаг 9: Текущий код: '1010', символ, соответствующий коду: 'y'
Hello! My01011111000111001101011101110001010011011101111000111010001001001

Шаг 10: Текущий код: '010', символ, соответствующий коду: ' '
Hello! My 11111000111001101011101110001010011011101111000111010001001001

Шаг 11: Текущий код: '11111', символ, соответствующий коду: 'n'
Hello! My n100011100110101101110001010011011101111000111010001001001

Шаг 12: Текущий код: '1000', символ, соответствующий коду: 'a'
Hello! My na111001101011101110001010011011101111000111010001001001

Шаг 13: Текущий код: '11110', символ, соответствующий коду: 'n'
Hello! My nam010101101110001010011011101111000111010001001001

Шаг 14: Текущий код: '011', символ, соответствующий коду: 'e'
Hello! My name0101101110001010011011101111000111010001001001

Шаг 15: Текущий код: '010', символ, соответствующий коду: ' '
Hello! My name 1110110001010011011101111000111010001001001

Шаг 16: Текущий код: '11101', символ, соответствующий коду: 'l'
Hello! My name l110001010011011101111000111010001001001

Шаг 17: Текущий код: '1100', символ, соответствующий коду: 's'
Hello! My name ls01010011011101111000111010001001001

Шаг 18: Текущий код: '010', символ, соответствующий коду: ' '
Hello! My name ls 10011011101111000111010001001001

Шаг 19: Текущий код: '10011', символ, соответствующий коду: 'S'
Hello! My name ls S011101111000111010001001001

Шаг 20: Текущий код: '011', символ, соответствующий коду: 'e'
Hello! My name ls Se1101111000111010001001001

Шаг 21: Текущий код: '11011', символ, соответствующий коду: 'r'
Hello! My name ls Ser1100011010001001001

Шаг 22: Текущий код: '11100', символ, соответствующий коду: 'g'
Hello! My name ls Serg0111010001001001

Шаг 23: Текущий код: '011', символ, соответствующий коду: 'e'
Hello! My name ls Serge0110000001
```

Рисунок 10 — Пример работы программы

```
Activities Terminal dek 28 16:27 neyther@neyther: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/св

Шаг 24: Текущий код: '1010', символ, соответствующий коду: 'y'
Hello! My name ls Sergey001001001

Шаг 25: Текущий код: '001', символ, соответствующий коду: '}'
Hello! My name ls Sergey}001001

Шаг 26: Текущий код: '001', символ, соответствующий коду: '}'
Hello! My name ls Sergey})001

Шаг 27: Текущий код: '001', символ, соответствующий коду: '}'
Hello! My name ls Sergey)))

Окончательный вариант:
Hello! My name ls Sergey)))

С помощью таблицы Хаффмана:
001001101110111010001110000010101001101111111100011100110111000100000011100111100111010010010010

Шаг 1: Текущий код: '0010', символ, соответствующий коду: 'N'
Hello! N0110110111010001110000010101001101111111100011100110111000100000011100111100111010010010010

Шаг 2: Текущий код: '011', символ, соответствующий коду: 'e'
Hello! Ne101110111010001110000010101001101111111100011100110111000100000011100111100111010010010010

Шаг 3: Текущий код: '1011', символ, соответствующий коду: 'l'
Hello! Nel01111010001110000010101001101111111100011100110111000100000011100111100111010010010010010

Шаг 4: Текущий код: '1011', символ, соответствующий коду: 'l'
Hello! Nell1010001110000010101001101111111100011100110111000100000011100111100111010010010010010

Шаг 5: Текущий код: '11010', символ, соответствующий коду: 'o'
Hello! Nello01110000010101001101111111100011100110111000100000011100111100111010010010010010

Шаг 6: Текущий код: '0011', символ, соответствующий коду: 'i'
Hello! Nellooi0000110101001101111111100011100110111000100000011100111100111010010010010010

Шаг 7: Текущий код: '100', символ, соответствующий коду: ' '
Hello! Nellooi 000110101001101111111100011100110111000100000011100111100111010010010010010

Шаг 8: Текущий код: '0001', символ, соответствующий коду: 'M'
Hello! M10101001101111111100011100110111000100000011100111100111010010010010010

Шаг 9: Текущий код: '1010', символ, соответствующий коду: 'y'
Hello! My100110111111110001110011011100010000001110011110011100111010010010010010

Шаг 10: Текущий код: '100', символ, соответствующий коду: ' '
Hello! My 11011111111100011100110111000100000011100111100111010010010010010

Шаг 11: Текущий код: '11011', символ, соответствующий коду: 'n'
Hello! My n111111100011100110111000100000011100111100111010010010010010

Шаг 12: Текущий код: '11111', символ, соответствующий коду: 'a'
Hello! My na11100011001110111000100000011100111100111010010010010010

Шаг 13: Текущий код: '11100', символ, соответствующий коду: 'g'
Hello! My name11100011001110111000100000011100111100111010010010010010
```

Рисунок 11 — Пример работы программы

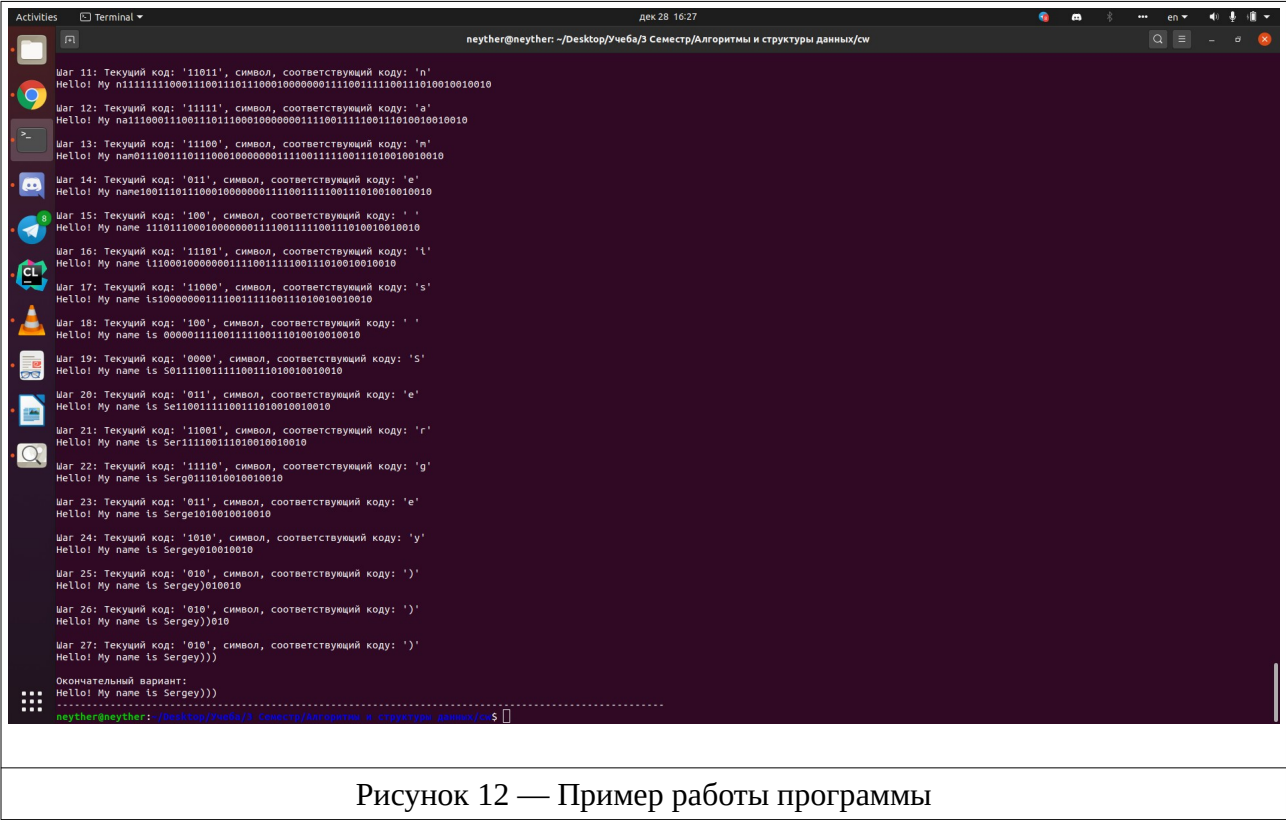


Рисунок 12 — Пример работы программы

Результаты тестирования представлены в таблице 1.

Таблица 1. Тестовые данные

Входные данные	Описание теста
Hello!	Короткий текст
My name is Sergey)	Текст подлиннее
Three pizzas, 4 cheeses... Is that seven cheese turns out?	Текст из двух предложений

ЗАКЛЮЧЕНИЕ

Алгоритм кодирования Хаффмана является одним из первых алгоритмов эффективного кодирования информации. Не смотря на его возраст, он до сих пор используется при кодировании, к примеру при сжатии фото- и видеоизображений (JPEG, MPEG).

Алгоритм Фано-Шеннона является одним из первых алгоритмов сжатия. Он имеет большое сходство с алгоритмом Хаффмана. Главной его идеей является замена часто встречающиеся символы более короткими кодами, а редко встречающиеся – более длинными кодами. Код является префиксным, что позволяет однозначно определить один закодированный символ, код которого не является префиксом любого другого кода.

В ходе выполнения работы была разработана программа на языке программирования C++, которая реализует алгоритмы Хаффмана и Фано-Шеннона, а также выполняет демонстрацию работы этих алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Код Хаффмана. URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0
2. Алгоритм Шеннона-Фано. URL: https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D0%B5%D0%BD%D0%BD%D0%BE%D0%BD%D0%B0_%E2%80%94%D0%A4%D0%B0%D0%BD%D0%BE
3. Алгоритм Хаффмана. URL: <https://habr.com/ru/post/144200/>

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
Файл main.cpp
#include "functions.h"

int main(int argc, char* argv[]){
    std::string encodeString;
    std::string encodedShannon, encodedHuffman;
    std::shared_ptr<BinTreeNode> shannonTree, huffmanTree;
    std::getline(std::cin, encodeString);

    auto iterBeg = encodeString.begin();
    encode(iterBeg, encodedShannon, encodedHuffman, shannonTree, huffmanTree);
    return 0;
}

Файл function.cpp
#include "functions.h"

static bool HuffmanComparator(std::pair<std::string, int> a1, std::pair<std::string, int> a2){
    if(a1.second >= a2.second)
        return false;
    else
        return true;
}

void getListOfElem(std::map<char, int>& map, std::string::iterator iterator){ // Мана
используемых символов
    if(*iterator == '\0')
        return;
    while(*iterator != '\0') {
        if (map.find(*iterator) != map.end()) {
            map[*iterator]++;
        } else {
            map.insert({*iterator, 1});
        }
        iterator++;
    }
}

void getStringWithWeigh(std::map<char, int> stringMap, std::pair<std::string, int>&
weightString){ // Строка с весами(отсортированная)
    auto iterBeg = stringMap.begin();
    auto iterEnd = stringMap.end();
    std::pair<char, int> max;
    std::map<char, int> finder;
    int flag;
    while(1){
        flag = 0;
        for(;iterBeg != iterEnd;iterBeg++){
            if(iterBeg->second > max.second && finder.find((*iterBeg).first) == finder.end()){
                max.first = (*iterBeg).first;
                max.second = (*iterBeg).second;
                flag = 1;
            }
        }
        iterBeg = stringMap.begin();
        if(flag == 1){
            weightString.first += max.first;
        }
    }
}
```

```

        weightString.second += max.second;
        finder.insert({max.first, max.second});
        max.second = 0;
        continue;
    }
    break;
}
}

void getCodesFromHuffman(std::shared_ptr<BinTreeNode> huffmanTree, std::map<char,
std::string>& codesHuffman, std::string code){//Заполнение карты кодов Хаффмана
    if(huffmanTree == nullptr){
        return;
    }
    else{
        if(huffmanTree->data.first.length() == 1)
            codesHuffman.insert({huffmanTree->data.first[0], code});
        else{
            getCodesFromHuffman(huffmanTree->left, codesHuffman, code+'0');
            getCodesFromHuffman(huffmanTree->right, codesHuffman, code+'1');
        }
    }
}

void printTree(std::shared_ptr<BinTreeNode> tree, int level, std::string code, bool debug, bool
left, int& step){
    if(level == 0) {
        std::cout << "Head: " << tree->data.first << "(" << tree->data.second << ")" << "\n";
        if (debug)
            std::cout << "\n";
    }
    else {
        for (int i = 1; i < level; i++) {
            std::cout << "|t";
        }
        if(left) {
            std::cout << "|--- L: \" << tree->data.first << \"(\" << tree->data.second << ") Code =
\" << code << \"\n";
            if (level == 1 && debug)
                std::cout << "\n";
        }
        else {
            std::cout << "|--- R: \" << tree->data.first << \"(\" << tree->data.second << ") Code
= \" << code << \"\n";
            if(debug)
                std::cout << "\n";
        }
    }
    if(debug){
        if(tree->left && tree->right) {
            step++;
            std::cout << "Шаг " << step;
            std::cout << " Узел со строкой " << tree->data.first << " и весом " << tree-
>data.second;
            std::cout << " делится на узел со строкой \" << tree->left->data.first << \"\ и весом
" << tree->left->data.second << ", который идет в левое поддерево";
            std::cout << ", а также узел со строкой \" << tree->right->data.first << \"\ и весом "
<< tree->right->data.second << ", который идет в правое поддерево\n";
        }
    }
}

```

```

    }
    if (tree->left) {
        printTree(tree->left, level + 1, code + '0', debug, true, step);
    }
    if (tree->right) {
        printTree(tree->right, level + 1, code + '1', debug, false, step);
    }
}

static bool HuffmanStringComparator(char a1, char a2){
    if(a1 >= a2)
        return false;
    else
        return true;
}

void printHuffman(std::pair<std::string, int> stringWithWeight, std::map<char, int>
usingSymbols, bool debug, int& step){
    std::cout << "\n-----\n\n";
    std::cout << "Ход алгоритма Хаффмана\n";
    std::pair<std::string, int> temp;
    std::vector<std::pair<std::string, int>> huffmanString;
    for(int i=1;i<=stringWithWeight.first.length();i++){
        temp = std::make_pair(stringWithWeight.first[stringWithWeight.first.length()-i],
usingSymbols[stringWithWeight.first[stringWithWeight.first.length()-i]]);
        huffmanString.push_back(temp);
    }
    while(huffmanString.size() != 1){
        temp = std::make_pair(huffmanString[0].first+huffmanString[1].first,
huffmanString[0].second+huffmanString[1].second);
        std::sort(temp.first.begin(), temp.first.end(), HuffmanStringComparator);

        for(int i=0;i<huffmanString.size();i++){
            if(i == huffmanString.size() - 1)
                std::cout << huffmanString[i].first << "(" << huffmanString[i].second << "\n";
            else
                std::cout << huffmanString[i].first << "(" << huffmanString[i].second << "), ";
        }
        std::cout << "\n";
        if(debug && huffmanString.size() != 2){
            std::cout << "Шаг " << ++step;
            std::cout << " первые два узла: \' " << huffmanString[0].first << "\'(" <<
huffmanString[0].second << ") и \' " << huffmanString[1].first << "\'(" <<
huffmanString[1].second << ")";
            std::cout << " складываются вместе: складываются сами строки, сортируются в
лексикографическом порядке, а также складываются их веса.\n";
            std::cout << "Получается узел \' " << temp.first << \' с весом " << temp.second <<
"\n\n";
        }
        if(huffmanString.size() == 2) {
            huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
            huffmanString.push_back(temp);
            std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator);
            break;
        }
        else {
            huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
            huffmanString.push_back(temp);
            std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator);
        }
    }
}

```

```

    }
}
if(debug){
    std::cout << "Шаг " << ++step;
    std::cout << " на последнем шаге оставшиеся два узла: " << huffmanString[0].first <<
    "(" << huffmanString[0].second << ")" и " << huffmanString[1].first << "(" <<
    huffmanString[1].second << ")";
    std::cout << " складываются вместе: складываются сами строки, сортируются в
    лексикографическом порядке, а также складываются их веса.\n";
    std::cout << "Получается узел " << temp.first << " с весом " << temp.second << ",
    который и является головой полученного дерева\n\n";
}
std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator);
std::cout << huffmanString[0].first << "(" << huffmanString[0].second << ")\n\n";
}

void printCodeTable(std::map<char, std::string> codes){
    auto checker = codes.end();
    checker--;
    for(auto i = codes.begin(); i != codes.end(); i++) {
        if (i == checker)
            std::cout << i->first << "(" << i->second << ")\n";
        else
            std::cout << i->first << "(" << i->second << "), ";
    }
}

std::map<std::string, char> getCodesFromFile(const std::string& fileName){
    std::map<std::string, char> codes;
    char symbol;
    std::string temp;
    std::string code;
    auto iterBeg = temp.begin();
    std::ifstream file;
    file.open(fileName);
    if(!file.is_open()){
        std::cout << "Error: Undeclared codeFile";
        exit(EXIT_FAILURE);
    }
    while(!file.eof()){
        std::getline(file, temp);
        iterBeg = temp.begin();
        symbol = *iterBeg;
        iterBeg += 2;
        while(*iterBeg != ';') {
            code += *iterBeg;
            iterBeg++;
        }
        codes.insert({code, symbol});
        code.clear();
        temp.clear();
    }
    file.close();
    return codes;
}

void encode(std::string::iterator encodeString, std::string& shannonString, std::string&
huffmanString, std::shared_ptr<BinTreeNode>& shannonTree, std::shared_ptr<BinTreeNode>&
huffmanTree) {
    std::map<char, int> map; //Мапа с используемыми символами и их весами
    std::map<char, std::string> codesShannon, codesHuffman; //Коды для расшифровки

```

```

std::pair<std::string, int> stringWithWeight; //Строка с общим весом символов
std::string code; //Служебная переменная для передачи в функцию
std::string outputShannon, outputHuffman; //Строка для вывода
int step = 0;

//Получение деревьев и файлов с кодами
getListOfElem(map, encodeString);
getStringWithWeigh(map, stringWithWeight);
shannonTree = shannonTree->getShannonFanoTree(stringWithWeight, map, codesShannon,
code);
huffmanTree = huffmanTree->getHuffmanTree(stringWithWeight, map);
getCodesFromHuffman(huffmanTree, codesHuffman, code);

//Вывод деревьев и "пояснения за алгоритмы
//-----
std::cout << "Исходные символы в кодируемом тексте и их веса:\n";
auto checker = map.end();
checker--;
for (auto i = map.begin(); i != map.end(); i++) {
    if (i == checker)
        std::cout << i->first << "(" << i->second << ")\n";
    else
        std::cout << i->first << "(" << i->second << "), ";
}
std::cout << "Строка, полученная сложением всех символов, поседующей сортировки,
и сложении их весов\n";
std::cout << stringWithWeight.first << "(" << stringWithWeight.second << ")\n\n";
std::cout << "Построение дерева Шано-Феннона\n";
printTree(shannonTree, 0, code, true, false, step);
std::cout << "Полученное дерево:\n";
step = 0;
printTree(shannonTree, 0, code, false, false, step);
std::cout << "Полученная таблица кодов для символов:\n";
printCodeTable(codesShannon);
step = 0;
printHuffman(stringWithWeight, map, true, step);
std::cout << "Дерево Хаффмана, полученное при работе алгоритма Хаффмана\n";
step = 0;
printTree(huffmanTree, 0, code, false, false, step);
std::cout << "\nПолученная таблица кодов для символов:\n";
printCodeTable(codesHuffman);
std::cout
    << "-----\n";
//-----

std::cout << "Шифрование:\n";
std::cout
    << "Шифрование происходит путем обращения к кодам, созданным ранее.
Берется символ, её код ищется в таблице и заменяется в тексте на этот символ\n";
std::string demonstrate;
std::cout << "\nС помощью таблицы Фано-Шеннона:\n";
step = 0;
auto temp = encodeString;
auto save = encodeString;
for (; *encodeString != '\0'; encodeString++) {
    temp = encodeString;
    for (; *temp != '\0'; temp++)
        demonstrate += *temp;
    std::cout << demonstrate << "\n\n";
    std::cout << "Шаг " << ++step << ": Текущий символ: " << *encodeString << ",его
код: " << codesShannon[*encodeString] << "\n";

```

```

        outputShannon += codesShannon[*encodeString];
        demonstrate.clear();
        demonstrate += outputShannon;
    }
    std::cout << outputShannon << "\n\n";
    std::cout << "Окончательная закодированная строка:\n";
    std::cout << outputShannon << "\n\n";
    std::cout
        << "-----\n";

    std::cout << "\nC помощью таблицы Хаффмана:\n";
    encodeString = save;
    demonstrate.clear();
    step = 0;
    for (; *encodeString != '\0'; encodeString++) {
        temp = encodeString;
        for (; *temp != '\0'; temp++)
            demonstrate += *temp;
        std::cout << demonstrate << "\n\n";
        std::cout << "Шаг " << ++step << ": Текущий символ: " << *encodeString << ", его
код: " << codesHuffman[*encodeString] << "\n";
        outputHuffman += codesHuffman[*encodeString];
        demonstrate.clear();
        demonstrate += outputHuffman;
    }
    std::cout << outputShannon << "\n\n";
    std::cout << "Окончательная закодированная строка:\n";
    std::cout << outputShannon << "\n\n";
    std::cout
        << "-----\n";

    //Формирование файлов с кодами для декодирования
    std::ofstream outputShannonFile, outputHuffmanFile;
    outputShannonFile.open("./CodesShannon.txt");
    outputHuffmanFile.open("./CodesHuffman.txt");
    auto iterBeg = codesShannon.begin();
    for (; iterBeg != codesShannon.end(); iterBeg++)
        outputShannonFile << iterBeg->first << ":" << iterBeg->second << ";\n";
    iterBeg = codesHuffman.begin();
    for (; iterBeg != codesHuffman.end(); iterBeg++)
        outputHuffmanFile << iterBeg->first << ":" << iterBeg->second << ";\n";
    outputShannonFile.close();
    outputHuffmanFile.close();

    std::cout << "\nДекодирование:\n";
    std::map<std::string, char> decodesShannon = getCodesFromFile("./CodesShannon.txt");
    std::map<std::string, char> decodesHuffman = getCodesFromFile("./CodesHuffman.txt");
    std::cout << "Декодирование происходит в порядке, обратном кодированию: код
заменяется соответствующим символом\n";

    std::cout << "C помощью таблицы Шеннона-Фано:\n";
    step = 0;
    auto decoderIter = outputShannon.begin();
    demonstrate.clear();
    std::string output;
    for (; decoderIter != outputShannon.end(); decoderIter++) {
        temp = decoderIter;
        for (; *temp != '\0'; temp++)
            demonstrate += *temp;
        while (true) {
            code += *decoderIter;

```



```

        if (decodesShannon.find(code) != decodesShannon.end())
            break;
        else {
            decodelter++;
            continue;
        }
    }
    std::cout << demonstrate;
    std::cout << "\n\nШаг " << ++step << ": Текущий код: \' " << code << "\', символ,
соответствующий коду: \' " << decodesShannon[code] << "\n\n";
    output += decodesShannon[code];
    demonstrate.clear();
    demonstrate += output;
    code.clear();
}
std::cout << output << "\n\n";
std::cout << "Окончательный вариант:\n" << output << "\n";
std::cout
    << "-----\n";

std::cout << "\nС помощью таблицы Хаффмена:\n";
decodelter = outputHuffman.begin();
demonstrate.clear();
output.clear();
step = 0;
for (; decodelter != outputHuffman.end(); decodelter++) {
    temp = decodelter;
    for(*temp != '\0'; temp++)
        demonstrate += *temp;
    while (true) {
        code += *decodelter;
        if (decodesHuffman.find(code) != decodesHuffman.end())
            break;
        else {
            decodelter++;
            continue;
        }
    }
    std::cout << demonstrate;
    std::cout << "\n\nШаг " << ++step << ": Текущий код: \' " << code << "\', символ,
соответствующий коду: \' " << decodesHuffman[code] << "\n\n";
    output += decodesHuffman[code];
    demonstrate.clear();
    demonstrate += output;
    code.clear();
}
std::cout << output << "\n\n";
std::cout << "Окончательный вариант:\n" << output << "\n";
std::cout
    << "-----\n";
}

```

```

Файл function.h:
#ifndef CW_FUNCTIONS_H
#define CW_FUNCTIONS_H

#include <iostream>
#include <fstream>
#include <string>
#include <memory>
#include <map>

```

```

#include <algorithm>
#include <cmath>

#include "BinTreeNode.h"

static bool HuffmanComparator(std::pair<std::string, int> a1, std::pair<std::string, int> a2);

void getListOfElem(std::map<char, int>& map, std::string::iterator iterator);

void getStringWithWeigh(std::map<char, int> stringMap, std::pair<std::string, int>&
weightString);

void getCodesFromHuffman(std::shared_ptr<BinTreeNode> huffmanTree, std::map<char,
std::string>& codesHuffman, std::string code);

void printTree(std::shared_ptr<BinTreeNode> tree, int level, std::string code, bool debug, bool
left);

static bool HuffmanStringComparator(char a1, char a2);

void printHuffman(std::pair<std::string, int> stringWithWeight, std::map<char, int>
usingSymbols, bool debug);

void printCodeTable(std::map<char, std::string> codes);

std::map<std::string, char> getCodesFromFile(const std::string& fileName);

void encode(std::string::iterator encodeString, std::string& shannonString, std::string&
huffmanString, std::shared_ptr<BinTreeNode>& shannonTree, std::shared_ptr<BinTreeNode>&
huffmanTree);

#endif //CW_FUNCTIONS_H
Файл BinTreeNode.cpp:
#include "BinTreeNode.h"

static bool HuffmanComparator1(std::pair<std::string, int> a1, std::pair<std::string, int> a2){
    if(a1.second >= a2.second)
        return false;
    else
        return true;
}

std::shared_ptr<BinTreeNode> BinTreeNode::getShannonFanoTree(std::pair<std::string, int>
stringWithWeight, std::map<char, int> usingSymbols, std::map<char, std::string>& codes,
std::string code) {
    std::shared_ptr<BinTreeNode> tree = std::make_shared<BinTreeNode>();
    std::pair<std::string, int> left;
    std::pair<std::string, int> right;
    tree->data = stringWithWeight;
    if (stringWithWeight.first.length() == 1) {
        codes.insert({tree->data.first[0], code});
        return tree;
    }
    while (true) {
        if ((left.second + usingSymbols[stringWithWeight.first[0]]) > (stringWithWeight.second)
&& left.second == 0) {
            right.first += stringWithWeight.first[0];
            right.second += usingSymbols[stringWithWeight.first[0]];
            stringWithWeight.second -= usingSymbols[stringWithWeight.first[0]];

```

```

        stringWithWeight.first.erase(0, 1);
        left.first = stringWithWeight.first;
        left.second = stringWithWeight.second;
        break;
    } else if ((left.second + usingSymbols[stringWithWeight.first[0]]) >
(stringWithWeight.second))
        break;
    else {
        left.first += stringWithWeight.first[0];
        left.second += usingSymbols[stringWithWeight.first[0]];
        stringWithWeight.second -= usingSymbols[stringWithWeight.first[0]];
        stringWithWeight.first.erase(0, 1);
    }
}
if (right.second == 0) {
    right.first = stringWithWeight.first;
    right.second = stringWithWeight.second;
}

if (left.second > right.second)
    std::swap(left, right);
if (left.second != 0) {
    tree->left = std::make_shared<BinTreeNode>();
    tree->left = getShannonFanoTree(left, usingSymbols, codes, code + '0');
}
if (right.second != 0) {
    tree->right = std::make_shared<BinTreeNode>();
    tree->right = getShannonFanoTree(right, usingSymbols, codes, code + '1');
}
return tree;
}

std::shared_ptr<BinTreeNode> BinTreeNode::getHuffmanTree(std::pair<std::string, int>
stringWithWeight, std::map<char, int> usingSymbols) {
    //Инициализация переменных
    std::shared_ptr<BinTreeNode> head; //Созданный узел
    std::map<std::string, std::shared_ptr<BinTreeNode>> tempNodes; //Мапа
нераспределенных узлов
    std::map<std::string, int> tempLeft, tempRight; //Мапы левых и правых сыновей
    std::vector<std::pair<std::string, int>> huffmanString; // Основная строка с весами
Хаффмана
    std::pair<std::string, int> temp, emptyPairLeft, emptyPairRight; // Текущая пара

    //Заполнение строки Хаффмана
    for (int i = 1; i <= stringWithWeight.first.length(); i++) {
        temp = std::make_pair(stringWithWeight.first[stringWithWeight.first.length() - i],
            usingSymbols[stringWithWeight.first[stringWithWeight.first.length() - i]]);
        huffmanString.push_back(temp);
    }

    //Основной блок
    while (huffmanString.size() != 1) {

        //Если длина первых двух элементов равна по 1
        if (huffmanString[0].first.length() == 1 && huffmanString[1].first.length() == 1) {
            head = makeNode(nullptr, nullptr, huffmanString[0], huffmanString[1]);
            tempNodes.insert({huffmanString[0].first + huffmanString[1].first, head});
            temp = std::make_pair(huffmanString[0].first + huffmanString[1].first,
                huffmanString[0].second + huffmanString[1].second);
            huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
            huffmanString.push_back(temp);
        }
    }
}

```

```

        std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator1);
    }

    //Если длина первого больше 1, а второго == 1
    else if (huffmanString[0].first.length() > 1 && huffmanString[1].first.length() == 1) {
        head = makeNode(tempNodes[huffmanString[0].first], nullptr, emptyPairLeft,
huffmanString[1]);
        tempNodes.erase(huffmanString[1].first);
        tempNodes.insert({huffmanString[0].first + huffmanString[1].first, head});
        temp = std::make_pair(huffmanString[0].first + huffmanString[1].first,
            huffmanString[0].second + huffmanString[1].second);
        huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
        huffmanString.push_back(temp);
        std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator1);
    }

    //Если длина первого - 1, а второго больше 1
    else if (huffmanString[0].first.length() == 1 && huffmanString[1].first.length() > 1) {
        head = makeNode(nullptr, tempNodes[huffmanString[1].first], huffmanString[0],
emptyPairRight);
        tempNodes.erase(huffmanString[1].first);
        tempNodes.insert({huffmanString[0].first + huffmanString[1].first, head});
        temp = std::make_pair(huffmanString[0].first + huffmanString[1].first,
            huffmanString[0].second + huffmanString[1].second);
        huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
        huffmanString.push_back(temp);
        std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator1);
    }

    //Если длина обоих больше 1
    else {
        head = makeNode(tempNodes[huffmanString[0].first],
tempNodes[huffmanString[1].first], emptyPairLeft,
            emptyPairRight);
        tempNodes.erase(huffmanString[0].first);
        tempNodes.erase(huffmanString[1].first);
        tempNodes.insert({huffmanString[0].first + huffmanString[1].first, head});
        temp = std::make_pair(huffmanString[0].first + huffmanString[1].first,
            huffmanString[0].second + huffmanString[1].second);
        huffmanString.erase(huffmanString.begin(), huffmanString.begin() + 2);
        huffmanString.push_back(temp);
        std::sort(huffmanString.begin(), huffmanString.end(), HuffmanComparator1);
    }
}
return head;
}

std::shared_ptr<BinTreeNode> BinTreeNode::makeNode(std::shared_ptr<BinTreeNode>
leftNode, std::shared_ptr<BinTreeNode> rightNode, std::pair<std::string, int> leftFutureNode,
std::pair<std::string, int> rightFutureNode) {
    std::shared_ptr<BinTreeNode> node = std::make_shared<BinTreeNode>();
    std::pair<std::string, int> empty;
    std::string temp;
    if (leftNode && rightNode) {
        node->left = leftNode;
        node->right = rightNode;
        node->data.first = node->left->data.first + node->right->data.first;
        node->data.second = node->left->data.second + node->right->data.second;
        return node;
    }
    if (leftNode && rightFutureNode.second != 0) {

```

```

        node->left = leftNode;
        node->right = makeNode(nullptr, nullptr, empty, rightFutureNode);
        node->data.first = node->left->data.first + node->right->data.first;
        node->data.second = node->left->data.second + node->right->data.second;
        return node;
    }
    if (leftFutureNode.second != 0 && rightNode) {
        node->left = makeNode(nullptr, nullptr, leftFutureNode, empty);
        node->right = rightNode;
        node->data.first = node->left->data.first + node->right->data.first;
        node->data.second = node->left->data.second + node->right->data.second;
        return node;
    }
    if (leftFutureNode.second != 0 && rightFutureNode.second != 0) {
        node->left = makeNode(nullptr, nullptr, leftFutureNode, empty);
        node->right = makeNode(nullptr, nullptr, empty, rightFutureNode);
        node->data.first = node->left->data.first + node->right->data.first;
        node->data.second = node->left->data.second + node->right->data.second;
        return node;
    }
    if (leftFutureNode.second != 0) {
        node->data.first = leftFutureNode.first;
        node->data.second = leftFutureNode.second;
        return node;
    }
    if (rightFutureNode.second != 0) {
        node->data.first = rightFutureNode.first;
        node->data.second = rightFutureNode.second;
        return node;
    }
}

```

Файл BinTreeNode.h:

```

#ifndef CW_BINTREENODE_H
#define CW_BINTREENODE_H

```

```

#include <iostream>
#include <fstream>
#include <string>
#include <memory>
#include <map>
#include <algorithm>

```

```

class BinTreeNode {

```

```

public:

```

```

    std::shared_ptr<BinTreeNode> left{nullptr};
    std::shared_ptr<BinTreeNode> right{nullptr};
    std::pair<std::string, int> data;

```

```

    std::shared_ptr<BinTreeNode> getShannonFanoTree(std::pair<std::string, int>
stringWithWeight, std::map<char, int> usingSymbols, std::map<char, std::string> &codes,
std::string code);

```

```

    std::shared_ptr<BinTreeNode> getHuffmanTree(std::pair<std::string, int> stringWithWeight,
std::map<char, int> usingSymbols);

```

```

    std::shared_ptr<BinTreeNode> makeNode(std::shared_ptr<BinTreeNode> leftNode,
std::shared_ptr<BinTreeNode> rightNode, std::pair<std::string, int> leftFutureNode,
std::pair<std::string, int> rightFutureNode);
};

```

```
static bool HuffmanComparator1(std::pair<std::string, int> a1, std::pair<std::string, int> a2);  
#endif //CW_BINTREENODE_H
```