



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

**ОТЧЕТ**  
**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6**  
Кодирование и сжатие данных методами без потерь  
**по дисциплине**  
«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-01-21

Луковников Д.Р.

Принял преподаватель

Туманова М.Б.

Практическая

«\_\_»\_\_\_\_\_2022 г.

\_\_\_\_\_

работа выполнена

«Зачтено»

«\_\_»\_\_\_\_\_2022 г.

\_\_\_\_\_

Москва 2022

## СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ.....	3
ХОД РАБОТЫ.....	4
1.1 Постановка задачи.....	4
1.2 Ручное кодирование методом Шеннона-Фано.....	5
1.3 Сжатие данных по методу Лемпеля-Зива LZ77 .....	7
1.4 Сжатие данных по методу Лемпеля-Зива LZ78 .....	7
1.5 Кодирование методом Шеннона-Фано .....	7
1.6 Ручное кодирование методом Хаффмана .....	11
1.7 Кодирование методом Хаффмана.....	13
ТЕСТИРОВАНИЕ.....	14
ВЫВОДЫ .....	16
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	17
ПРИЛОЖЕНИЯ .....	18

## **ЦЕЛЬ РАБОТЫ**

Освоить навыки применения алгоритмов сжатия данных на примере следующих алгоритмов: Метод Хаффмана, Шеннона-Фано, метод Лемпеля-Зива LZ77, LZ78. Разработать и протестировать программы сжатия и восстановления данных.

## **ХОД РАБОТЫ**

### **1.1 Постановка задачи**

**Задание 1** Исследование алгоритмов сжатия на примерах

- Выполнить каждую задачу варианта, представив алгоритм решения в виде таблицы и указав результат сжатия.
- Описать процесс восстановления сжатого текста.
- Сформировать отчет, включив задание, вариант задания, результаты выполнения задания варианта.

**Задание 2** Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона – Фано.

- Реализовать и отладить программы.
- Сформировать отчет по разработке каждой программы в соответствии с требованиями.
- По методу Шеннона-Фано привести: постановку задачи, описать алгоритм формирования префиксного дерева и алгоритм кодирования, декодирования, код и результаты тестирования. Рассчитать коэффициент сжатия. Сравнить с результатом сжатия вашим алгоритмом с результатом любого архиватора.
- По методу Хаффмана выполнить и отобразить результаты выполнения всех требований, предъявленных в задании и оформить разработку программы: постановка, подход к решению, код, результаты тестирования.

**Индивидуальный вариант:**

Вариант представлен на рисунке 1.

3	Эне-бене, рики-таки, Буль-буль-буль, Караки-шмаки Эус- деус-краснодеус бац	0100101010010000101	лорлоралоранранлоран
---	---	---------------------	----------------------

Рисунок 1 – Данные варианта

При кодировании методом Шеннона — Фано, символы распределяются в порядке от наиболее вероятных к наименее вероятным и затем разделяются на два набора, чьи суммарные вероятности максимально приближены друг к другу. Далее формируется первый разряд кода всех символов: символы из первого набора получают двоичный "0", символы из второго — "1". Процесс деления на две части и получения следующих разрядов повторяется для полученных наборов аналогичным образом, до тех пор, пока в полученном наборе не остается по одному символу. Когда набор уменьшается до одного символа — код символа полностью сформирован. Если перефразировать, суть заключается в создании двоичного дерева для представления вероятностей появления каждого из символов. Затем они сортируются так, чтобы самые часто встречающиеся находились наверху дерева, и наоборот.

## 1.2 Ручное кодирование методом Шеннона-Фано

Данная фраза: «Эне-бене, рики-таки, Буль-буль-буль, Караки-шмаки Эус-деус-краснодеус бац»

Кодирование представлено в таблице 1, а результаты кодирования в таблице 2.

Таблица 1 – Кодирование методом Шеннона-Фано

Символ	Кол-во	1-я цифра	2-я цифра	3-я цифра	4-я цифра	5-я цифра	6-я цифра	7-я цифра	Код	Кол-во бит
-	7	0	0	0					000	21
у	6	0	1	0					010	18
к	6	0	0	1	0				0010	24
а	6	0	0	1	1				0011	24
е	5	0	1	1	0				0110	20
б	5	0	1	1	1				0111	20
	5	1	0	0	0				1000	20
и	5	1	0	0	1				1001	20
р	3	1	1	0	0				1100	12
с	4	1	0	1	0				1010	16
н	3	1	0	1	1	0			10110	15
,	3	1	0	1	1	1			10111	15
л	3	1	1	0	1	0			11010	15
ь	3	1	1	0	1	1			11011	15
э	2	1	1	1	0	0			11100	10
д	2	1	1	1	0	1			11101	10
т	1	1	1	1	1	0	0		111100	6
ш	1	1	1	1	1	0	1		111101	6
м	1	1	1	1	1	1	0		111110	6
о	1	1	1	1	1	1	1	0	1111110	7
ц	1	1	1	1	1	1	1	1	1111111	7
										307

Таблица 2 – Результаты кодирования методом Шеннона-Фано

Длина незакодированной фразы	73 * 8 = 584 бит
Закодированная фраза	307 бит

### 1.3 Сжатие данных по методу Лемпеля-Зива LZ77

Данная фраза: «0100101010010000101»

Кодирование представлено в таблице 3.

Таблица 3 – Кодирование LZ77

Исходный текст	0100101010010000101 0.1.00.10. 101. 001. 000. 01. 01
LZ-код	0.1.10.100.1001.0111.0110.0011.0011
R	2 3 4
Вводимые коды	- 10 11 100 101 110 111 1000 1001

### 1.4 Сжатие данных по методу Лемпеля-Зива LZ78

Данная фраза: «лорлоралоранранлоран»

Словарь кодирования представлен в таблице 4.

Таблица 4 – Кодирование LZ78

1	л	0л
2	о	0о
3	р	0р
4	ло	1о
5	ра	3а
6	лор	4р
7	а	0а
8	н	0н
9	ран	5н
10	лора	6а
11	н	8

Получившаяся закодированная строка: «0л0о0р1о3а4р0а0н5н6а8»

### 1.5 Кодирование методом Шеннона-Фано

Перед разбором программы разберём её эффективность, в качестве тестовой фразы используется ФИО автора повторённые 10 раз. Результаты приведены в таблице 5.

Данная фраза: «lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry romanovich»

Таблица 5 – Результаты сжатия

Размер текста	2 312 бит
После работы алгоритма	1116 бит
Коэффициент сжатия	0.482699
После сжатия архиватором	912 бит
Коэффициент сжатия архиватора	0,394463

Для хранения символов, их частот и кодов, используем структуру, листинг 1.1.

Листинг 1.1 – Структура узла

```
struct Symbol {
    char symbol;
    int frequency;
    string code;

    Symbol(char s, char f) : symbol(s), frequency(f) {};
};
```

Само по себе кодирование происходит внутри класс, конструктор принимает в качестве параметра текст, Листинг 1.2.

Листинг 1.2 – Структура кодировщика

```
class Encoder {
    string text; // Text to encode
    string encodedText; // Encoded text
    int length; // Length of text
    vector<Symbol> symbols; // Symbols
public:
    explicit Encoder(string text) : text(text), length(text.size()) {}
    ...
};
```

При запуске кодирования, сначала подсчитываются частоты символов, и происходит их сортировка по не возрастанию, затем вызывается функция



получения кодов и только после этого происходит кодирование текста, листинг 1.3.

*Листинг 1.3 – Начало кодирования*

```
void encode() {
    // Get symbols
    for (int i = 0; i < length; i++) {
        bool found = false;
        for (auto &symbol: symbols)
            if (symbol.symbol == text[i]) { symbol.frequency++; found = true;
break; }
        if (!found) symbols.emplace_back(text[i], 1);
    }
    // Sort symbols by frequency
    sort(symbols.begin(), symbols.end(), [](const Symbol &a, const Symbol &b){
        return a.frequency > b.frequency;
    });
    if (symbols.empty()) return;
    // Get codes
    codes(0, symbols.size() - 1);

    // Encode text
    for (int i = 0; i < length; i++)
        for (auto &symbol: symbols)
            if (symbol.symbol == text[i]) { encodedText += symbol.code;
break; }
}
```

Функция codes строит виртуальное бинарное дерево, принцип её работы рекурсивный, пока не найдём до «листа», листинг 1.4.

*Листинг 1.4 – Построение бинарного дерева*

```
void codes(int lb, int rb) {
    /*
     * Recursive function for getting codes
     */
    if (rb - 1 == lb) { symbols[lb].code += "0"; symbols[rb].code += "1";
return; }
    if (lb == rb) return;

    int different = INT_MAX, mb = rb, ls = 0, rs = 0;
    while (mb > lb) {
        ls += symbols[mb].frequency; rs = 0;
        for (int i = mb - 1; i >= lb; i--) rs += symbols[i].frequency;
        if (abs(ls - rs) < different) different = abs(ls - rs);
        else break;
        mb--;
    }
    for (int i = lb; i <= rb; i++) {
        if (i <= mb) symbols[i].code += "0";
        else symbols[i].code += "1";
    }

    codes(lb, mb);
    codes(mb + 1, rb);
}
```

Осталась последняя значимая функция, а именно декодирование, листинг 1.5.

*Листинг 1.5 – Поиск кратчайшего пути*

```
string decode(const string& basicString) {
    string decodedText;
    for (int i = 0; i < basicString.size(); i++) {
        for (auto &symbol: symbols) {
            if (symbol.code == basicString.substr(i, symbol.code.size())) {
                decodedText += symbol.symbol;
                i += symbol.code.size() - 1;
                break;
            }
        }
    }
    return decodedText;
}
```

## 1.6 Ручное кодирование методом Хаффмана

Данная фраза: «lukovnikov dmitry romanovich»

Таблица вероятностей представлена в таблице 5.

Таблица 5 – Таблица вероятностей

Символ	Вхождения	Вероятность
о	4	0,14
v	3	0,11
i	3	0,11
k	2	0,07
n	2	0,07
	2	0,07
m	2	0,07
r	2	0,07
l	1	0,04
u	1	0,04
d	1	0,04
t	1	0,04
y	1	0,04
a	1	0,04
c	1	0,04
h	1	0,04

По данной таблицы построено дерево, рисунок 1.

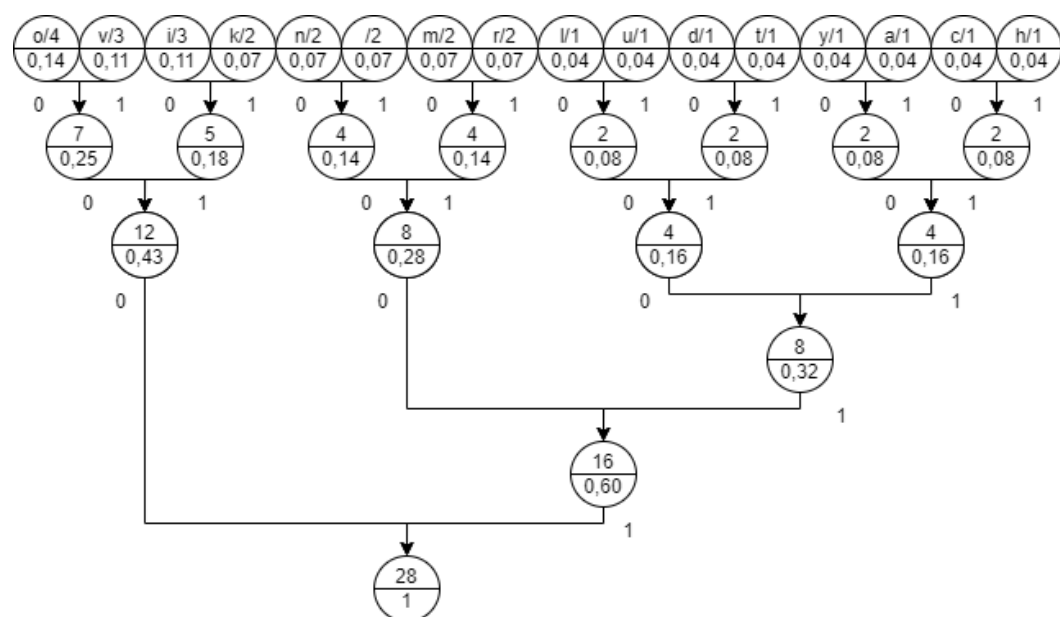


Рисунок 1 – Дерево Хаффмана

Таблица кодов представлена в таблице 6.

Таблица 6 – Таблица кодов

Символ	Код
o	000
v	001
i	010
k	011
n	1000
	1001
m	1010
r	1011
l	11000
u	11001
d	11010
t	11011
y	11100
a	11101
c	11110
h	11111

Закодированная фраза: «11000 11001 011 000 001 1000 010 011 000 001 1001 11010 1010 010 11011 1011 11100 1001 1011 000 1010 11101 1000 000 001 010 11110 11111»

## 1.7 Кодирование методом Хаффмана

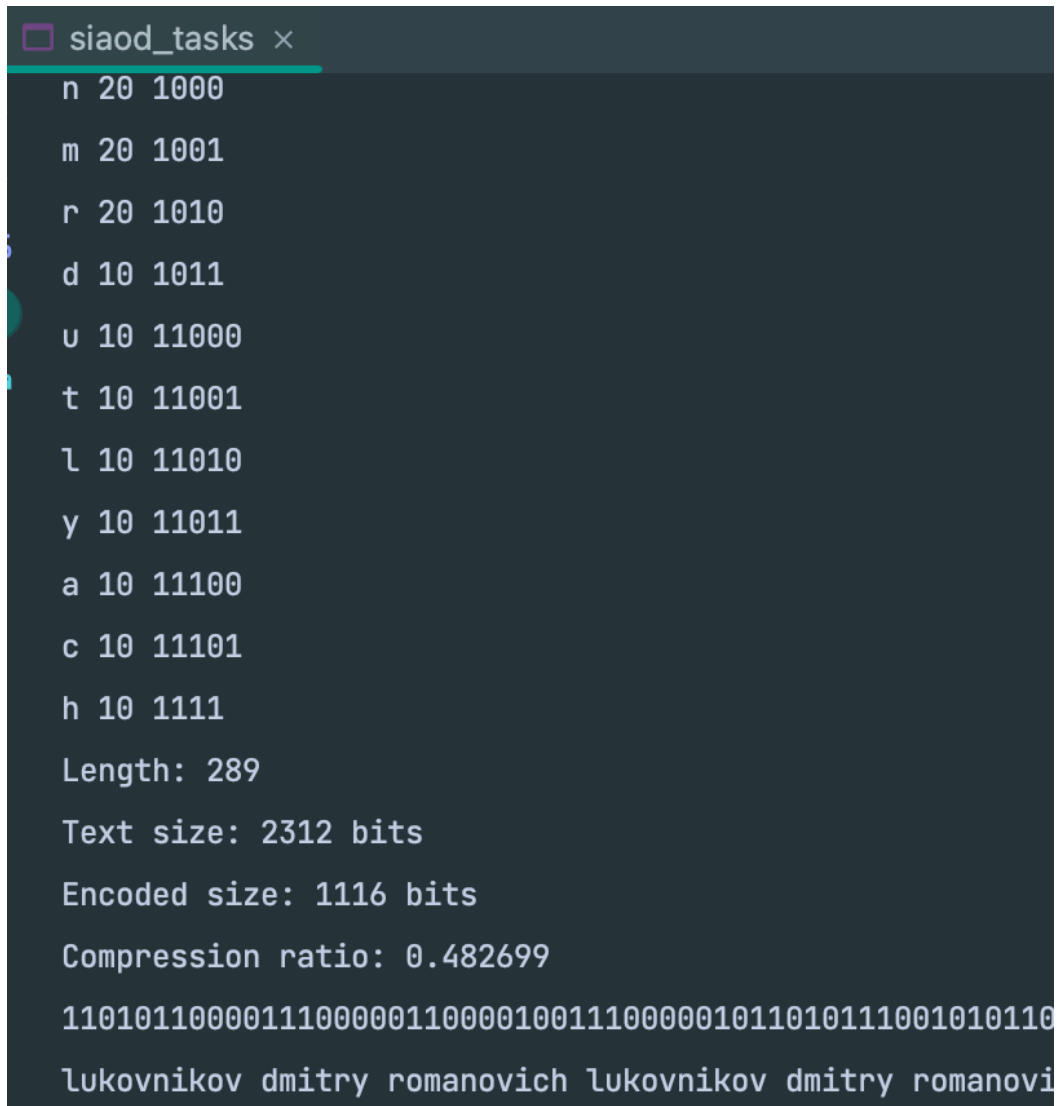
Основная идея программы такая же, как и в алгоритме Шеннона-Фано, за исключением принципе составления кодов, функция представлена в листинге 1.6, (предварительно было построено дерево).

*Листинг 1.6 – Получение кодов*

```
void codes(Node *node, string code) {  
    /*  
     * Recursive function for getting codes  
     */  
    if (node->left == nullptr || node->right == nullptr) { node->code = code;  
return; }  
    codes(node->left, code + "0");  
    codes(node->right, code + "1");  
}
```

## ТЕСТИРОВАНИЕ

Проверка работоспособности алгоритма Шеннона-Фано, рисунок 2.



```
siaod_tasks x
n 20 1000
m 20 1001
r 20 1010
d 10 1011
u 10 11000
t 10 11001
l 10 11010
y 10 11011
a 10 11100
c 10 11101
h 10 1111
Length: 289
Text size: 2312 bits
Encoded size: 1116 bits
Compression ratio: 0.482699
1101011000011100000110000100111000001011010111001010110
lukovnikov dmitry romanovich lukovnikov dmitry romanovi
```

Рисунок 2 – Вывод программы

Для тестирования работоспособности алгоритма Хаффмена были использованы ФИО автора, рисунок 3.

```
/Users/minusd/CLionProjects/siaod-tasks/cmake-build-debug/siaod_tasks
Encoded text: 1000110000000101010011100100010101001101001111010011001011001110101100101110111100011110101000111111111110
Length: 28
Text size: 224 bits
Encoded length: 108
Compression ratio: 0.482143
Decoded text: lukovnikov dmitry romanovich
Process finished with exit code 0
```

Рисунок 3 – Алгоритм Хаффмена

## **ВЫВОДЫ**

При выполнении работы были получены навыки кодирования и декодирования информации различными алгоритмами. Так же были реализованы некоторые программы и



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием С++. 2-е изд., 2016.
2. Документация по языку С++ [Электронный ресурс]. URL: <https://docs.microsoft.com/ruru/cpp/cpp/> (дата обращения 01.12.2022).
3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: <https://online-edu.mirea.ru/course/view.php?id=4020> (дата обращения 01.12.2022).
4. Видеоурок - Метод Шеннона-Фано [Электронный ресурс] URL: <https://youtu.be/orbJosR-Cqk> (дата обращения 01.12.2022)

## **ПРИЛОЖЕНИЯ**

Приложение А – Исходный код алгоритма Шеннона-Фано

Приложение Б – Исходный код алгоритма Хаффмана

## Приложение А

### Исходный код алгоритма Шеннона-Фано

#### Листинг А.1 – *main.cpp*

```
/*
 * Algorithm of Shannon-Fano
 */

#include <iostream>
#include <utility>
#include <vector>

using namespace std;

struct Symbol {
    char symbol;
    int frequency;
    string code;

    Symbol(char s, char f) : symbol(s), frequency(f) {};
};

class Encoder {
    string text; // Text to encode
    string encodedText; // Encoded text
    int length; // Length of text
    vector<Symbol> symbols; // Symbols
public:
    explicit Encoder(string text) : text(text), length(text.size()) {}

    void encode() {
        // Get symbols
        for (int i = 0; i < length; i++) {
            bool found = false;
            for (auto &symbol: symbols)
                if (symbol.symbol == text[i]) { symbol.frequency++; found =
true; break; }
            if (!found) symbols.emplace_back(text[i], 1);
        }

        // Sort symbols by frequency
        sort(symbols.begin(), symbols.end(), [](const Symbol &a, const Symbol
&b) {
            return a.frequency > b.frequency;
        });

        if (symbols.empty()) return;

        // Get codes
        codes(0, symbols.size() - 1);

        // Encode text
        for (int i = 0; i < length; i++)
            for (auto &symbol: symbols)
                if (symbol.symbol == text[i]) { encodedText += symbol.code;
break; }
        }

        void codes(int lb, int rb) {
            /*
             * Recursive function for getting codes
            */
        }
    };
};
```

*Продолжение Листинг А.1*

```
a      */
      if (rb - 1 == lb) { symbols[lb].code += "0"; symbols[rb].code += "1";
return; }
      if (lb == rb) return;

      int different = INT_MAX, mb = rb, ls = 0, rs = 0;
      while (mb > lb) {
          ls += symbols[mb].frequency; rs = 0;
          for (int i = mb - 1; i >= lb; i--) rs += symbols[i].frequency;
          if (abs(ls - rs) < different) different = abs(ls - rs);
          else break;
          mb--;
      }
      for (int i = lb; i <= rb; i++) {
          if (i <= mb) symbols[i].code += "0";
          else symbols[i].code += "1";
      }

      codes(lb, mb);
      codes(mb + 1, rb);
  }

  void printTable() {
      /*
       * Print table of symbols and codes
       */
      for (auto &symbol: symbols) {
          cout << symbol.symbol << " " << symbol.frequency << " " <<
symbol.code << endl;
      }
  }

  void statistics() {
      cout << "Length: " << length << endl
           << "Text size: " << length * 8 << " bits" << endl
           << "Encoded size: " << encodedText.size() << " bits" << endl
           << "Compression ratio: " << (double) encodedText.size() /
(length * 8) << endl;
  }

  string getEncodedText() {
      return encodedText;
  }

  string decode(const string& basicString) {
      string decodedText;
      for (int i = 0; i < basicString.size(); i++) {
          for (auto &symbol: symbols) {
              if (symbol.code == basicString.substr(i, symbol.code.size()))
{
                  decodedText += symbol.symbol; i += symbol.code.size() -
1; break;
              }
          }
      }
      return decodedText;
  }
};

int main() {
```

*Продолжение Листинг А.1*

```
string text = "lukovnikov dmitry romanovich lukovnikov dmitry romanovich  
lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry  
romanovich lukovnikov dmitry romanovich lukovnikov dmitry romanovich  
lukovnikov dmitry romanovich lukovnikov dmitry romanovich lukovnikov dmitry  
romanovich";  
Encoder encoder(text);  
encoder.encode();  
encoder.printTable();  
encoder.statistics();  
cout << encoder.getEncodedText() << endl;  
cout << encoder.decode(encoder.getEncodedText()) << endl;  
return 0;  
}
```

## Приложение Б

### Исходный код алгоритма Хаффмана

Листинг Б.1 – *main.cpp*

```
/*
 * Algorithm of Huffman
 */

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

using namespace std;

struct Node {
    /*
     * Node of Huffman tree
     */
    Node *left, *right;
    int frequency;
    char symbol;
    string code;

    Node(int frequency, char symbol) : frequency(frequency), symbol(symbol) {
        left = right = nullptr;
    }
};

class Encoder {
    string text; // Text to encode
    string encodedText; // Encoded text
    int length; // Length of text

    vector<Node *> nodes; // Nodes of Huffman tree
    Node *root; // Root of Huffman tree

public:
    explicit Encoder(const string &text) : text(text), length(text.length())
    {}

    void encode() {
        /*
         * Encoding text
         */

        // Get frequency of each symbol
        for (int i = 0; i < length; i++) {
            bool found = false;
            for (auto &node: nodes)
                if (node->symbol == text[i]) { node->frequency++; found =
true; break; }
            if (!found) nodes.emplace_back(new Node(1, text[i]));
        }

        // Sort nodes by frequency
        sort(nodes.begin(), nodes.end(), [](const Node *a, const Node *b) {
            return a->frequency > b->frequency;
        });

        if (nodes.empty()) return;
    }
};
```

```

// Build Huffman tree
while (nodes.size() > 1) {
    Node *left = nodes.back(); nodes.pop_back();
    Node *right = nodes.back(); nodes.pop_back();
    Node *parent = new Node(left->frequency + right->frequency, 0);
    parent->left = left;
    parent->right = right;
    nodes.emplace_back(parent);
    sort(nodes.begin(), nodes.end(), [](const Node *a, const Node *b)
{
        return a->frequency > b->frequency;
    });
}
root = nodes.back(); nodes.pop_back();

// Get codes
codes(root, "");

// Encode text
for (int i = 0; i < length; i++)
    encodedText += getCode(text[i], root);
}

void codes(Node *node, string code){
    /*
     * Recursive function for getting codes
     */
    if (node->left == nullptr || node->right == nullptr) { node->code =
code; return; }
    codes(node->left, code + "0");
    codes(node->right, code + "1");
}

string getCode(char symbol, Node *cur) {
    /*
     * Find code of symbol
     */
    if (cur->symbol == symbol) return cur->code;
    if (cur->left != nullptr) {
        string code = getCode(symbol, cur->left);
        if (code != "") return code;
    }
    if (cur->right != nullptr) {
        string code = getCode(symbol, cur->right);
        if (code != "") return code;
    }
    return "";
}

string decode(string basicString){
    /*
     * Decoding text
     */
    string decodedText;
    Node *node = root;
    for (char i : basicString){
        if (i == '0') node = node->left;
        else node = node->right;
    }
}

```

*Продолжение Листинг Б.1*

```
        if (node->left == nullptr && node->right == nullptr){
            decodedText += node->symbol;
            node = root;
        }
    }
    return decodedText;
}

void statistics() {
    /*
     * Print statistics
     */
    cout << "Length: " << length << endl
         << "Text size: " << length * 8 << " bits" << endl
         << "Encoded length: " << encodedText.length() << endl
         << "Compression ratio: " << (double) encodedText.length() /
(length * 8) << endl;
}

const string &getEncodedText() const {
    return encodedText;
}

~Encoder(){
    /*
     * Destructor
     */
    delete root;
}

};

int main() {
    string text = "lukovnikov dmitry romanovich";
    Encoder encoder(text);
    encoder.encode();
    cout << "Encoded text: " << encoder.getEncodedText() << endl;
    encoder.statistics();
    cout << "Decoded text: " << encoder.decode(encoder.getEncodedText()) <<
endl;
    return 0;
}
```