Практическая работа № 10

СЖАТИЕ ДАННЫХ

Постановка задачи

Составить программу, реализующую кодирование строки с помощью алгоритма Хаффмана

1. Описание алгоритма

Построение кода Хаффмана сводится к построению соответствующего бинарного дерева по следующему алгоритму:

1. Составим список кодируемых символов, при этом будем рассматривать один символ как дерево, состоящее из одного элемента с весом, равным частоте появления символа в строке.
2. Из списка выберем два узла с наименьшим весом.
3. Сформируем новый узел с весом, равным сумме весов выбранных узлов, и присоединим к нему два выбранных узла в качестве детей.
4. Добавим к списку только что сформированный узел вместо двух объединенных узлов.
5. Если в списке больше одного узла, то повторим пункты со второго по пятый.

Алгоритм реализован с использованием классов Haffman и Los и одной структуры, являющейся узлом перевернутого дерева.

Методы класса Huffman:

void set_string - функция записывает строку в память.

void make_tree - функция создает дерево шифрования.

void compress – функция зашифровывает строку

string get_code – функция возвращает полученный код

Методы класса Los:

void add_ell – функция добавляет элемент в список

Node *find_ell – функция возвращает указатель на элемент по ключу

void inc_ell – функция инкрементирует поле number структуры

int count_true – функция возвращает количество записей, в которых поле is_last_layer является истиной.

int *pare_min – функция возвращает массив из 2-х id записей с наименьшими полями number.

int get_id – функция возвращает id записи

void inc_id – функция инкрементирует поле counter_id в классе Los

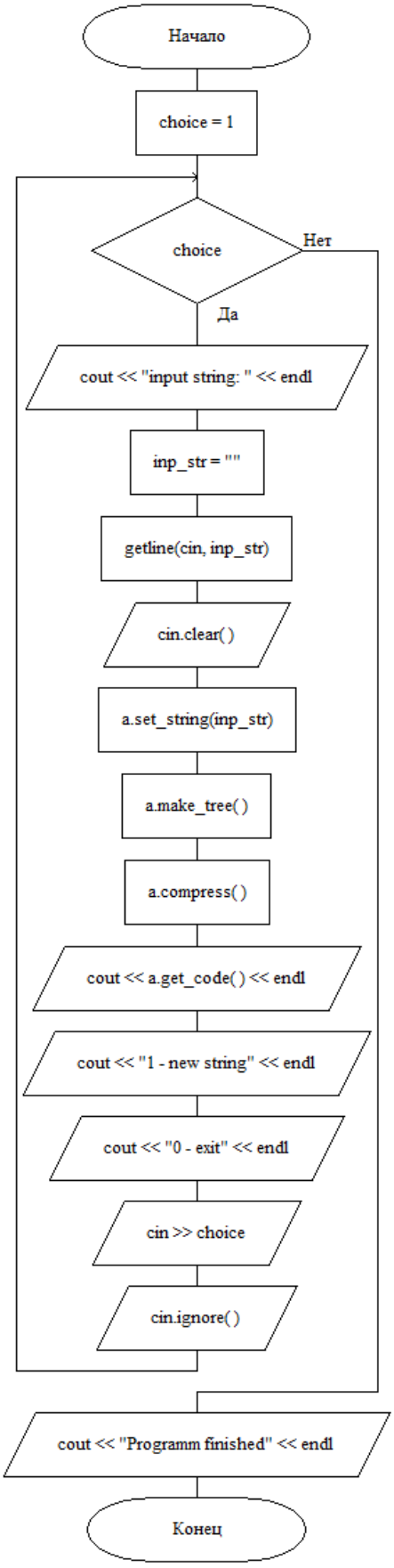

Рис.1 Схема алгоритма функции main

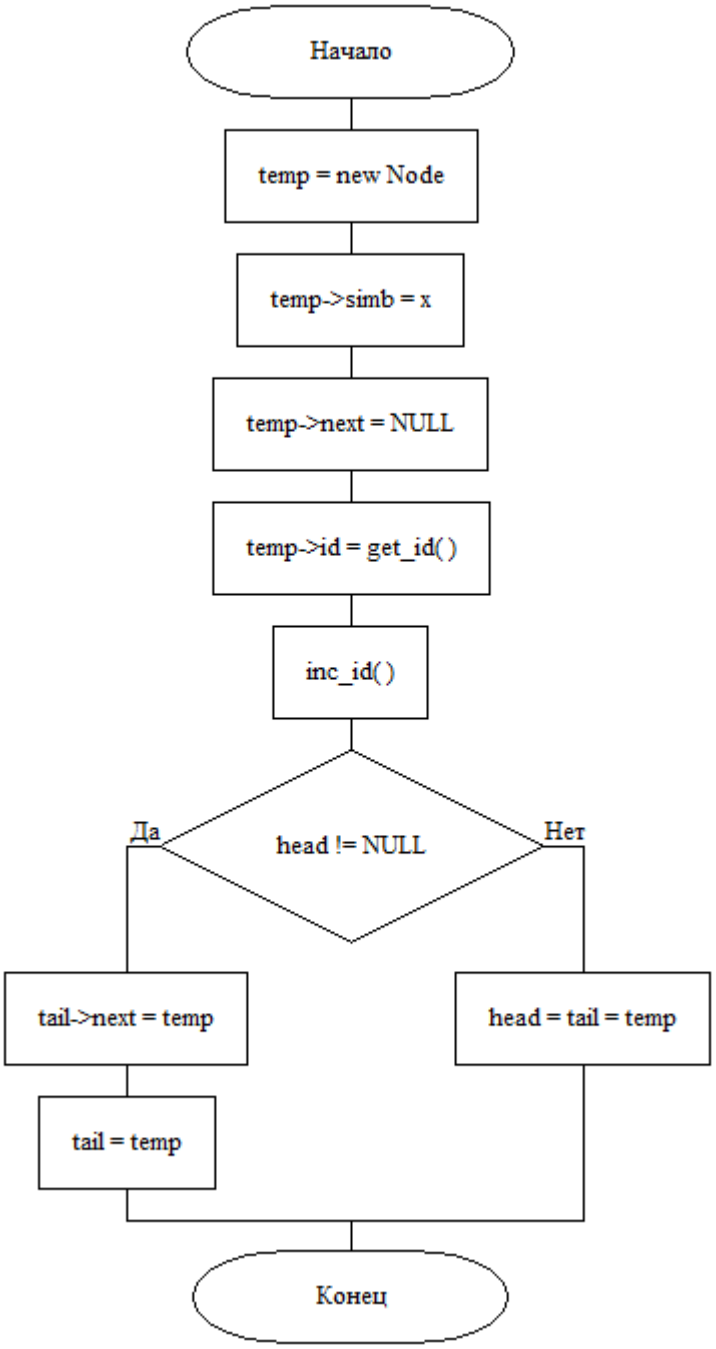

Рис.2 Схема алгоритма функции add_ell(char x)

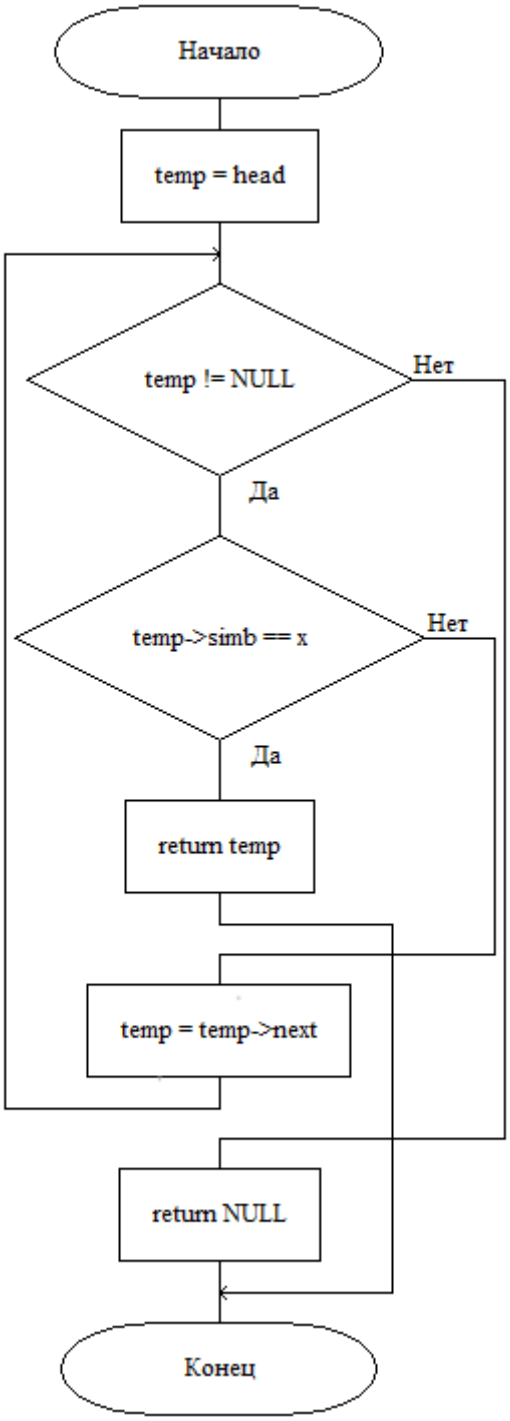

Рис.3 Схема алгоритма функции find_ell(char x)

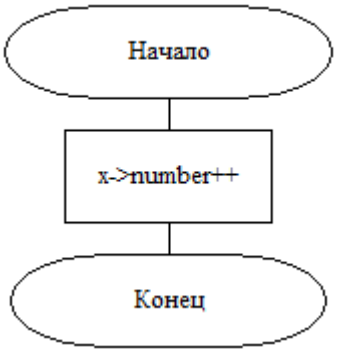

Рис.4 Схема алгоритма функции inc_ell(Node *x)

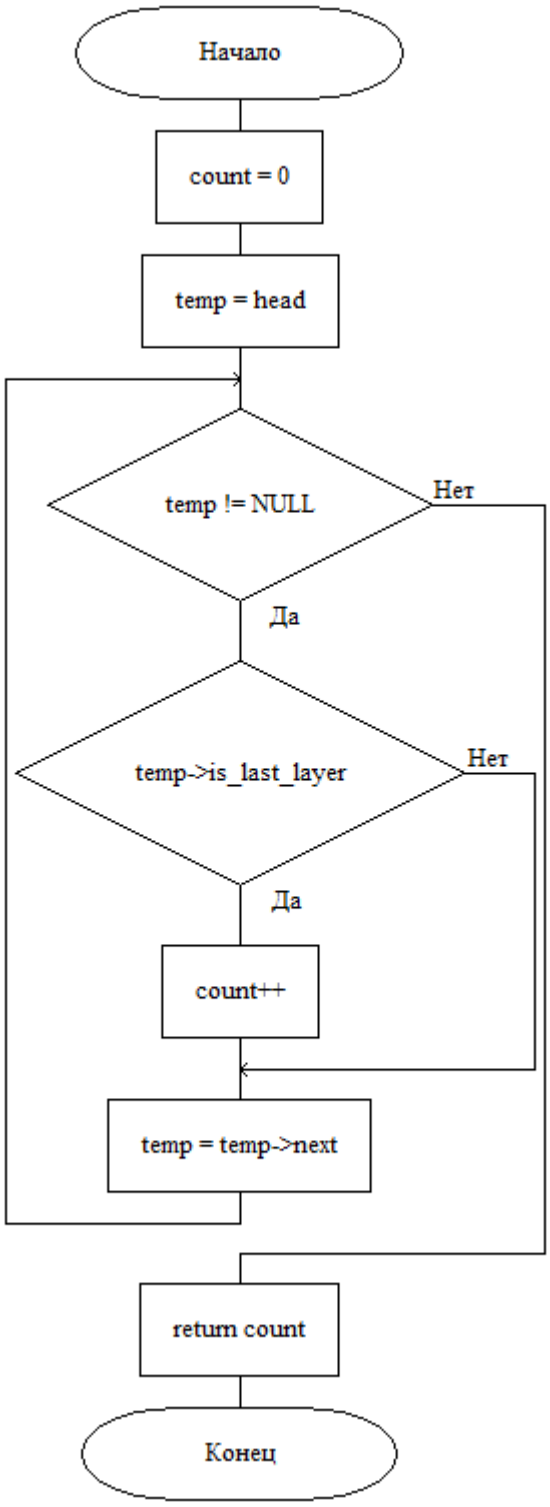

Рис.5 Схема алгоритма функции count_true()

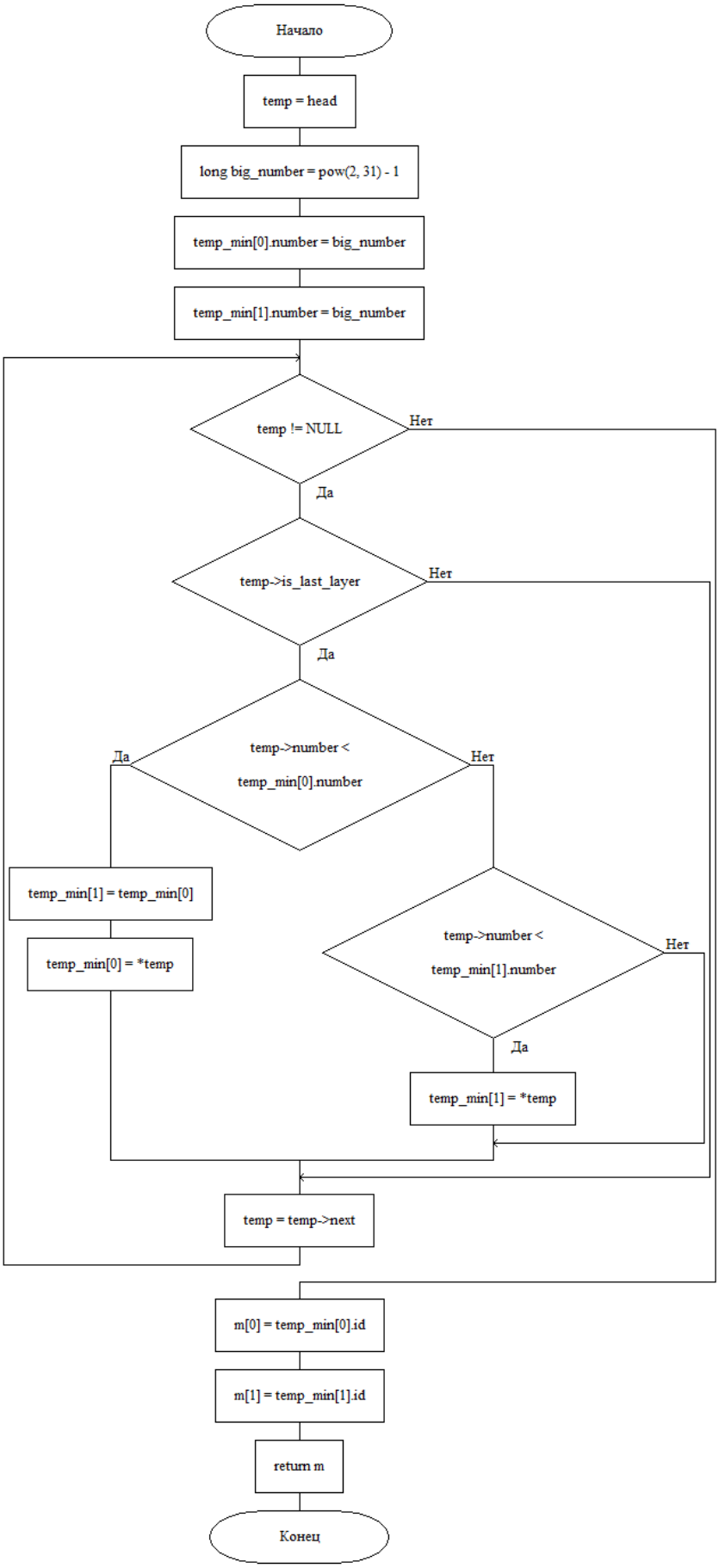

Рис.6 Схема алгоритма функции pare_min()

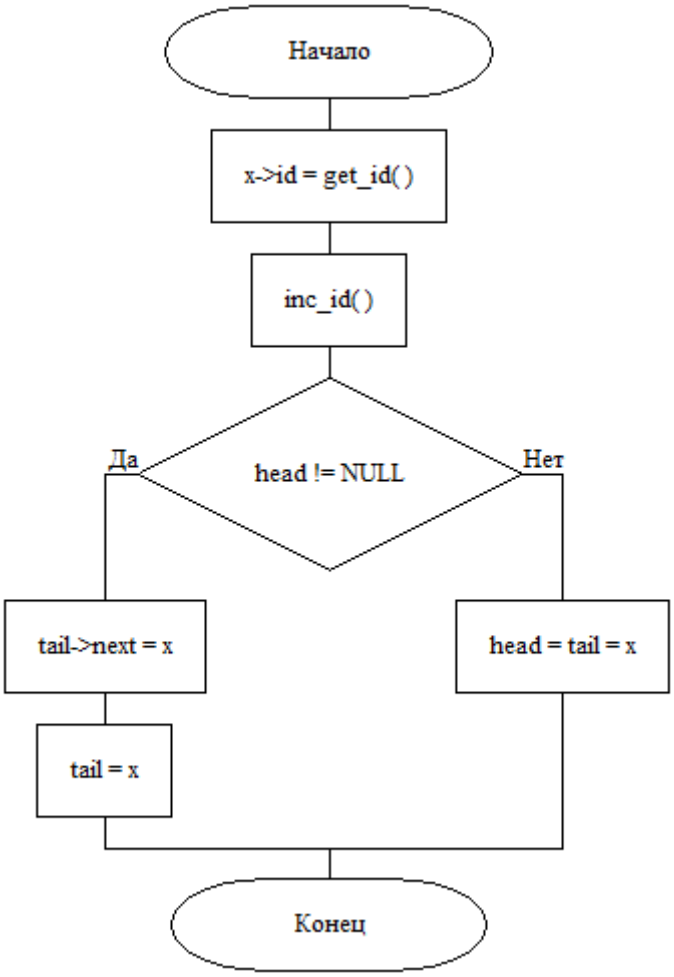

Рис.7 Схема алгоритма функции add_ell(Node *x)

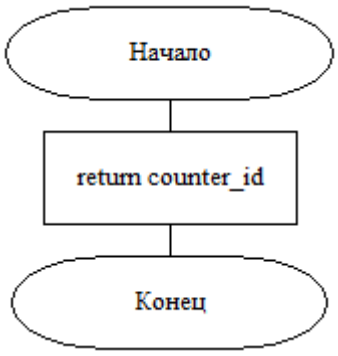

Рис.8 Схема алгоритма функции get_id()

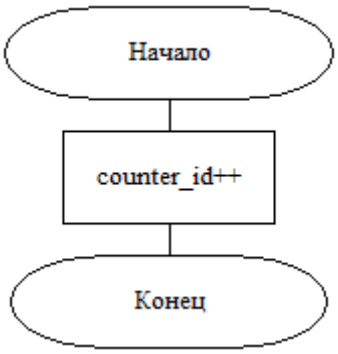

Рис.9 Схема алгоритма функции inc_id()

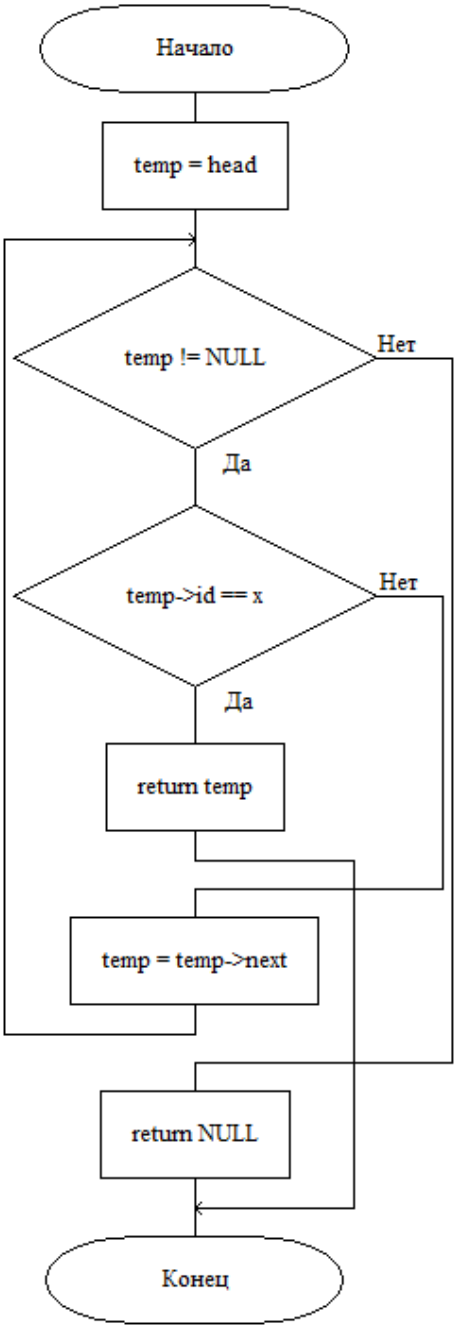

Рис.10 Схема алгоритма функции find_ell(int x)

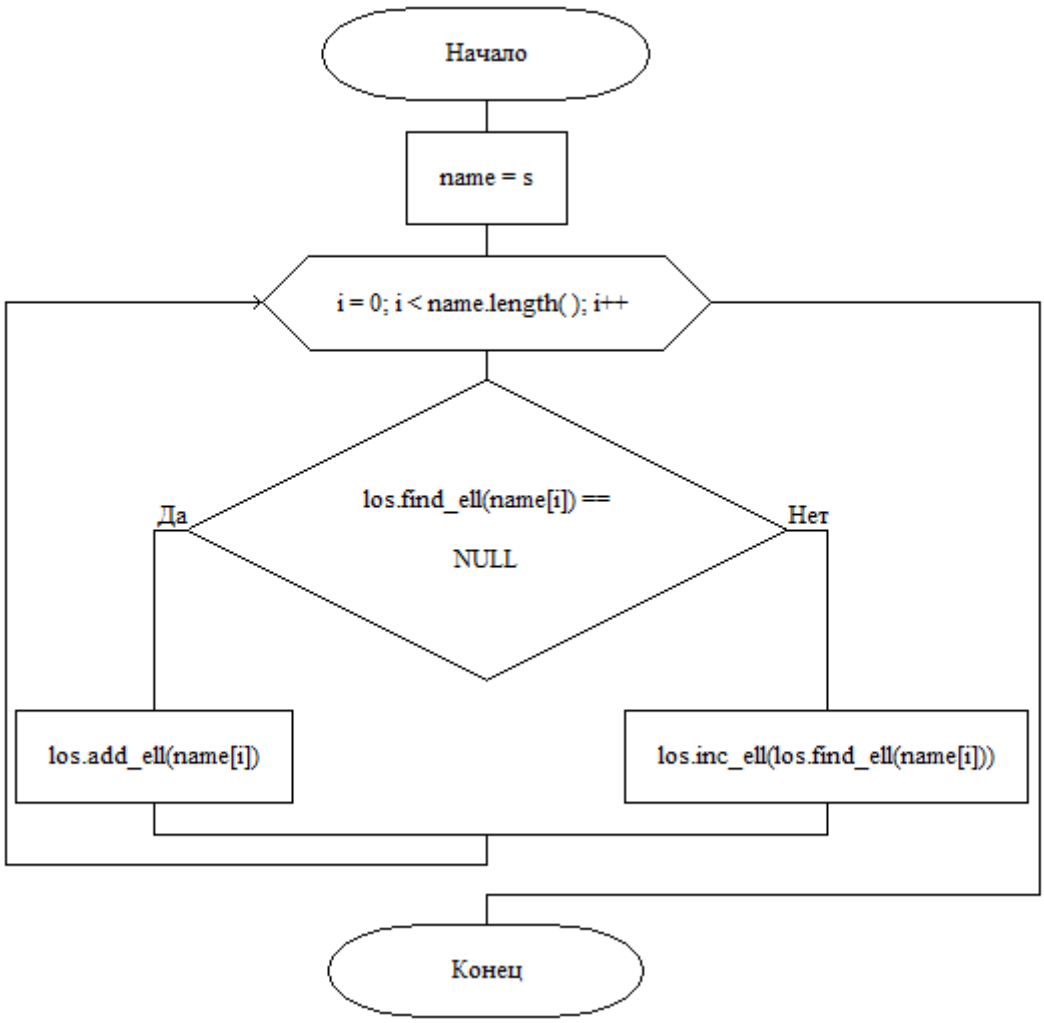

Рис.11 Схема алгоритма функции set_string(string s)

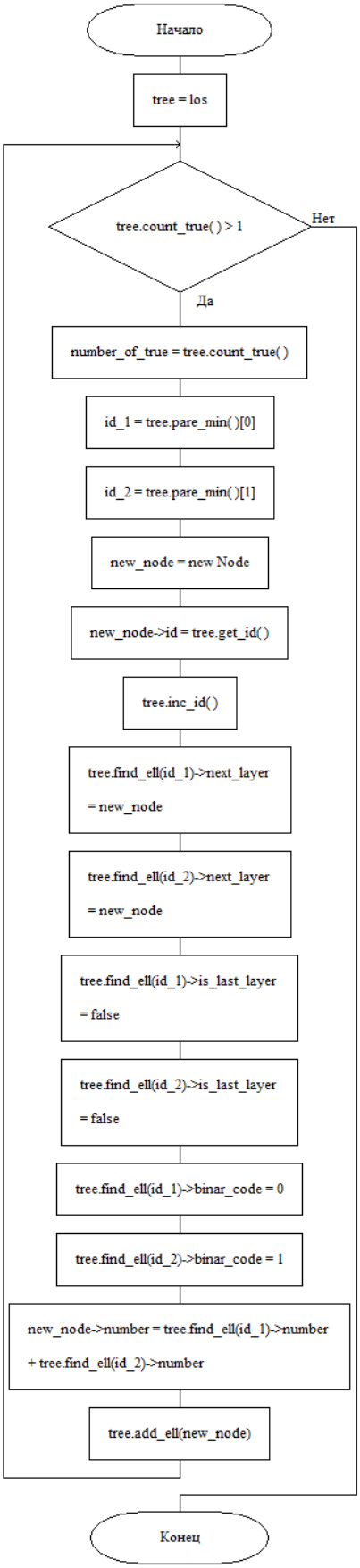

Рис.12 Схема алгоритма функции make_tree()

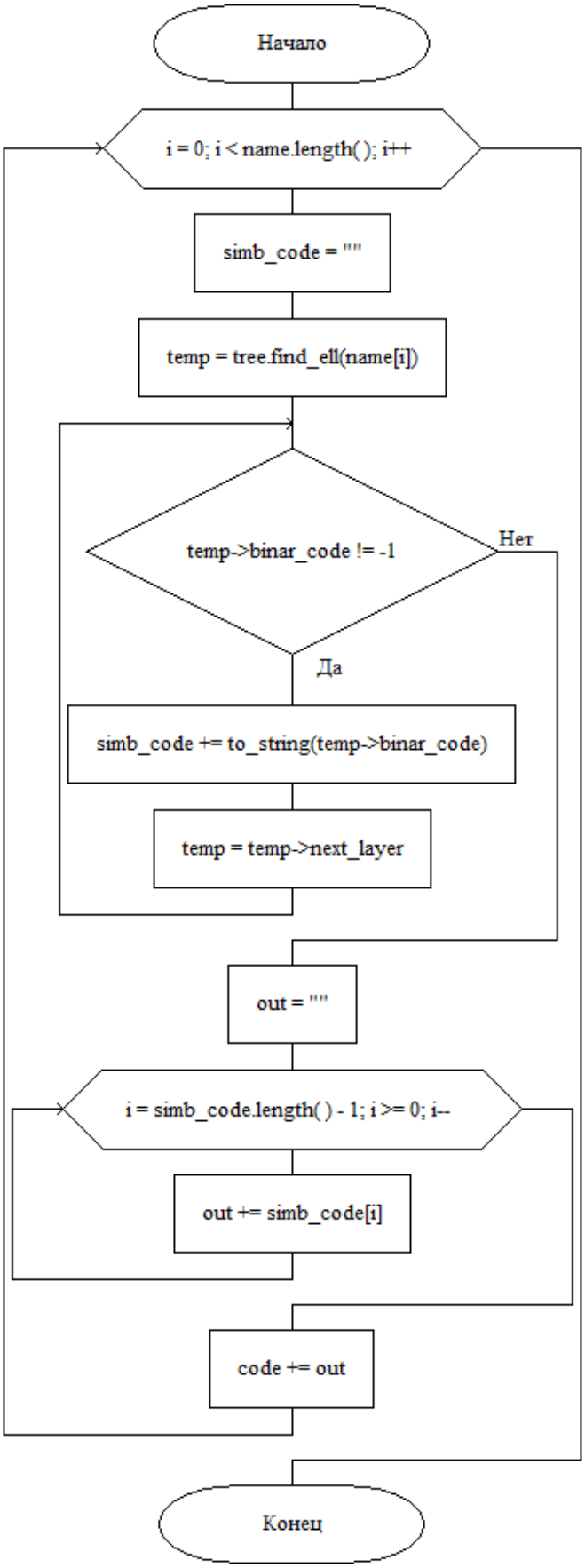

Рис.13 Схема алгоритма функции compress()

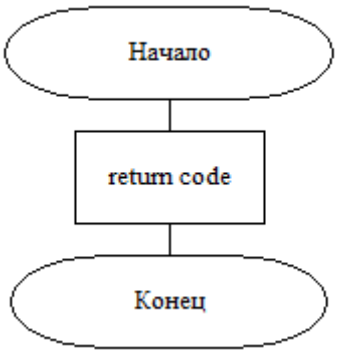

Рис.14 Схема алгоритма функции get_code()

1. Реализация алгоритма

Текст исходного кода программы

main.cpp

```cpp
#include <iostream>
#include "Huffman.h"
#include <string>


int main() {

    int choice = 1;
    while (choice) {
        Huffman a;
        cout << "input string: " << endl;
        string inp_str = "";
        getline(cin, inp_str);
        cin.clear();
        a.set_string(inp_str);
        a.make_tree();
        a.compress();
        cout << a.get_code() << endl;
        cout << "1 - new string" << endl;
        cout << "0 - exit" << endl;
        cin >> choice;
        cin.ignore();


    }
    cout << "Programm finished" << endl;


}
```

Los.h

```
#ifndef PR_10_LOS_H
#define PR_10_LOS_H

#include <iostream>

struct Node {
    Node *next = NULL;
    int id = NULL;
    char simb = '_';
    int number = 1;
    Node *next_layer = NULL;
    int binar_code = -1;
    bool is_last_layer = true;

};

class Los {
private:
    Node *head, *tail;
    int counter_id = 0;
public:

    Los() : head(NULL), tail(NULL) {};

    void add_ell(char x);

    void add_ell(Node *x);

    Node *find_ell(char x);

    Node *find_ell(int x);

    void inc_ell(Node *x);

    int count_true();

    int *pare_min();

    int get_id();

    void inc_id();


};


#endif
```

Los.cpp

```
#include "Los.h"
#include <iostream>


void Los::add_ell(char x) {
    Node *temp = new Node;
    temp->simb = x;
    temp->next = NULL;
    temp->id = get_id();
```

```
    inc_id();

    if (head != NULL) {
        tail->next = temp;
        tail = temp;
    } else head = tail = temp;
};

Node *Los::find_ell(char x) {
    Node *temp = head;
    while (temp != NULL) {
        if (temp->simb == x) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

void Los::inc_ell(Node *x) {
    x->number++;
}

int Los::count_true() {
    int count = 0;
    Node *temp = head;
    while (temp != NULL) {
        if (temp->is_last_layer) count++;
        temp = temp->next;

    }
    return count;
}


int *Los::pare_min() {
    int m[2];
    Node *temp = head;
    static Node temp_min[2];
    unsigned long big_number = pow(2, 31) - 1;
    temp_min[0].number = big_number;
    temp_min[1].number = big_number;

    while (temp != NULL) {
        if (temp->is_last_layer) {
            if (temp->number < temp_min[0].number) {
                temp_min[1] = temp_min[0];
                temp_min[0] = *temp;
            } else if (temp->number < temp_min[1].number) { temp_min[1] =
*temp; }

        }
        temp = temp->next;
    }
    m[0] = temp_min[0].id;
    m[1] = temp min[1].id;


    return m;


}

void Los::add_ell(Node *x) {
```

```
    x->id = get_id();
    inc_id();
    if (head != NULL) {
        tail->next = x;
        tail = x;
    } else head = tail = x;
}

int Los::get_id() {
    return counter_id;
}

void Los::inc_id() {
    counter_id++;
}

Node *Los::find_ell(int x) {
    Node *temp = head;
    while (temp != NULL) {
        if (temp->id == x) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}
```

Haffman.h

```
#ifndef PR_10_HUFFMAN_H
#define PR_10_HUFFMAN_H

#include <iostream>
#include <list>
#include "Los.h"

using namespace std;

class Huffman {
private:
    Los los;
    Los tree;
    string name = "A";
    string code = "";

public:
    void set_string(string s);

    void make_tree();

    void compress();

    string get_code();


};


#endif
```

Haffman.cpp

```
#include "Huffman.h"

#include <string>

void Huffman::set_string(string s) {
    name = s;
    for (int i = 0; i < name.length(); i++) {
        if (los.find_ell(name[i]) == NULL) {
            los.add_ell(name[i]);
        } else {
            los.inc_ell(los.find_ell(name[i]));
        }
    }

}

void Huffman::make_tree() {
    tree = los;
    while (tree.count_true() > 1) {
        int number_of_true = tree.count_true();

        int id_1 = tree.pare_min()[0];
        int id_2 = tree.pare_min()[1];
        Node *new_node = new Node;
        new_node->id = tree.get_id();
        tree.inc_id();
        tree.find_ell(id_1)->next_layer = new_node;
        tree.find_ell(id_2)->next_layer = new_node;
        tree.find_ell(id_1)->is_last_layer = false;
        tree.find_ell(id_2)->is_last_layer = false;
        tree.find_ell(id_1)->binar_code = 0;
        tree.find_ell(id_2)->binar_code = 1;
        new_node->number = tree.find_ell(id_1)->number + tree.find_ell(id_2)-
>number;
        tree.add_ell(new_node);

    }
}


void Huffman::compress() {

    for (int i = 0; i < name.length(); i++) {
        string simb_code = "";
        Node *temp = tree.find_ell(name[i]);
        while (temp->binar_code != -1) {
            simb_code += to_string(temp->binar_code);
            temp = temp->next_layer;
        }
        string out = "";
        for (int i = simb_code.length() - 1; i >= 0; i--)
            out += simb_code[i];
        code += out;
    }


}

string Huffman::get_code() {
```

```
    return code;
}
```

2. Тестирование программы

```
input string:
aaabbcccc
11111110100000
1 - new string
0 - exit
1
input string:
Mironov Aleksey
01100111100000110010011010101111001101010111011110100000
1 - new string
0 - exit
0
Programm finished
```

Рис.15 Скриншот шифрования двух строк (1 – произвольная, 2 - Фамилия, имя)

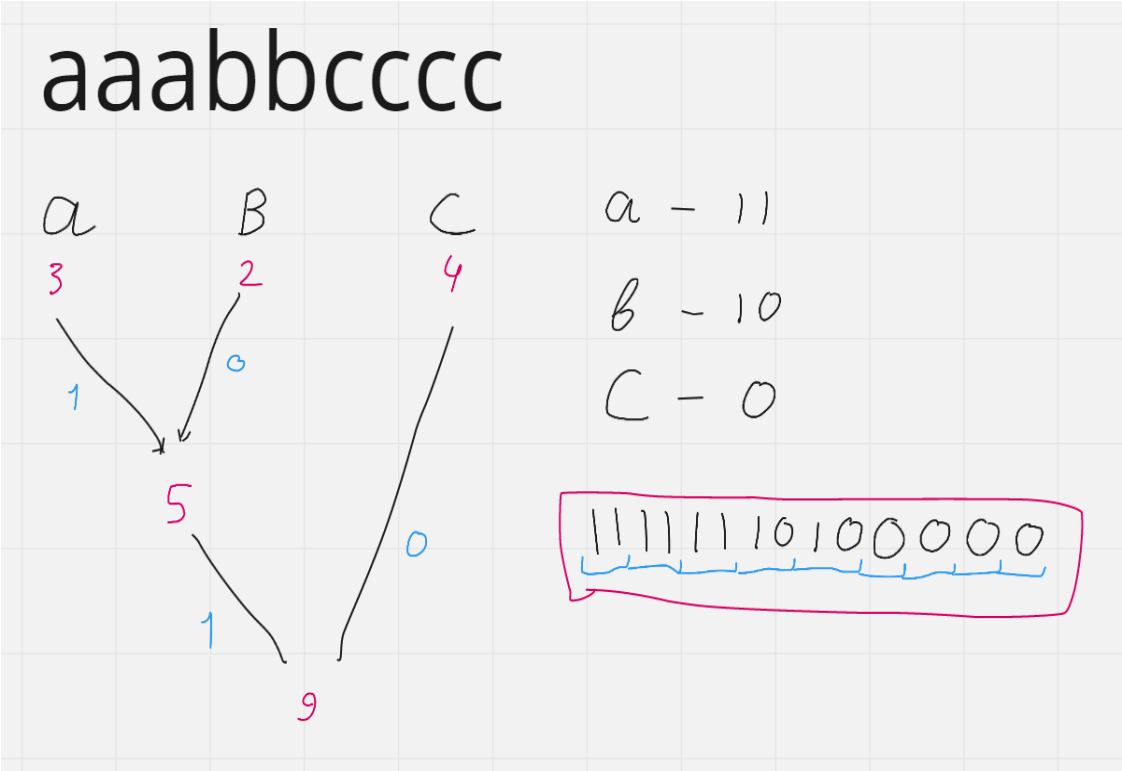

Рис.16 Скриншот ручной проверки алгоритма на строке из рис.1

3. Выводы

1. В ходе данной работы были изучены принципы работы сжатия данных
2. На практике изучен и реализован алгоритм Хаффмана.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).
5. Описание алгоритма Хаффмана. интернет-ресурс: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Хаффмана (Дата обращения – 02.11.2020).