Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Алгоритмы и структуры данных»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

«Сортировка слиянием на основе стека»

Выполнили:

Дынина Е.А., студент группы N3249

_____
(подпись)

Проверил:

Ерофеев С.А.

_____
(отметка о выполнении)

_____
(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Цель работы — разработать программу сортировки нисходящим слиянием для чисел из файла, используя стек на базе связного списка. Записать результат в красно-черное дерево.

Для достижения поставленной цели необходимо решить следующие задачи:

- Изучить структуру стека и красно-черного дерева;
- Создать блок-схему алгоритма с обработкой ошибок;
- Написать программу на языке C++ с интерфейсом для пользователей.

ТЕХНИЧЕСКАЯ ЗАДАЧА

Стек — абстрактный тип данных, экземпляр которого представляет собой список, в котором вставка и удаление элементов выполняются с одного конца — вершины стека. Базовые операции с очередью:

- Добавление элементов;
- Удаление элементов;
- Чтение головного элемента.

Сортировка слиянием заключается в рекурсивном разбиении стека на две равные части. Каждая часть сортируется отдельно той же сортировкой. После две отсортированные части сливаются в один упорядоченный стек, сравнивая элементы по очереди.

Красно-черное дерево — двоичное дерево поиска, в котором баланс осуществляется на основе «цвета» узла дерева, который принимает только два значения: «красный» и «черный». Свойства:

- Каждый узел промаркирован красным или чёрным цветом;
- Корень и конечные узлы (листья) дерева — чёрные;
- У красного узла родительский узел — чёрный;
- Все простые пути из любого узла x до листьев содержат одинаковое количество чёрных узлов;
- Чёрный узел может иметь чёрного родителя.

Входные данные

На вход в программу через указанный заранее файл подается стек с числами типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$).

Промежуточные данные

В ходе работы программы задействованы промежуточные переменные, указанные в таблице 1. Во время рекурсивной сортировки используются стек left, right и result с числами типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$).

Таблица 1 – Промежуточные данные

Название переменной	**Тип в C++**	**Диапазон типа**	**Значение**
file_path	string	—	Путь к файлу с числами

num	long double	$[-1{,}2 \cdot 10^{4932}, 1{,}2 \cdot 10^{4932}]$	Переменная, в которую считываем числа из файла
cur	int	$[0, 2147483647]$	Позиция указателя
half	int	$[0, 2147483647]$	Половина стека
*size_*	int	$[0, 2147483647]$	Размер стека
i	int	$[0, 2147483647]$	Переменная для реализации перебора в цикле
x	double	$[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$	Переменная для добавления в стек и удаления чисел из него
indent	string	—	Оформление вывода элементов красно-черного дерева
last	bool	{0,1}	Переменная для определения левого или правого поддерева
key	double	$[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$	Переменная для добавления чисел в стек

Выходные данные

В результате работы программы в консольное приложение выводится отсортированный стек и красно-черное дерево с числами типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$).

Используемые функции

- **ifstream —** открывает файл для чтения;
- **.is_open() —** проверяет, открыт ли файл;
- **numeric_limits <тип>::lowest —** возвращает наименьшее отрицательное конечное значение типа;
- **numeric_limits <тип>::max —** возвращает максимальное конечное значение типа;
- **.clear() —** сбрасывает состояние потока ifstream:
- **.tellg() —** возвращает текущую позицию указателя в файле;
- **.seekg() —** устанавливает указатель на определенную позицию в файле;
- **.close()** — закрывает файл;
- **delete** — освобождает память;

- **merge(stack &s, stack &left, stack &right) —** сравнивает две части стека, добавляет сначала меньшие значения в стек, а потом оставшиеся;
- **merge_sort(stack &s) —** делит стек на две части и проводит сортировку;

Функции класса stack

- **.push(double x) —** добавление элемента x в стек;
- **.pop() —** удаление и возвращение элемента из стека;
- **.empty() —** проверка стека на пустоту;
- **.size() —** выводит размер стека;
- **.print() —** выводит все элементы стека.

Функции класса RedBlackTree

- **.insertFix(NodePtr k) —** согласование свойств красно-черного дерева после добавления элемента;
- **.printHelper() —** реализация интерфейса графа красно-черного дерева;
- **.leftRotate(NodePtr x) —** левый поворот с элементом x;
- **.rightRotate(NodePtr x) —** правый поворот с элементом x;
- **.insert(int key) —** добавление элемента со значением key в дерево;
- **.printTree() —** выводит красно-черное дерево.

БЛОК-СХЕМА

Рассмотрим блок-схему придуманного алгоритма, где реализована сортировка нисходящим слиянием на основе стека и учтены всевозможные ошибки.

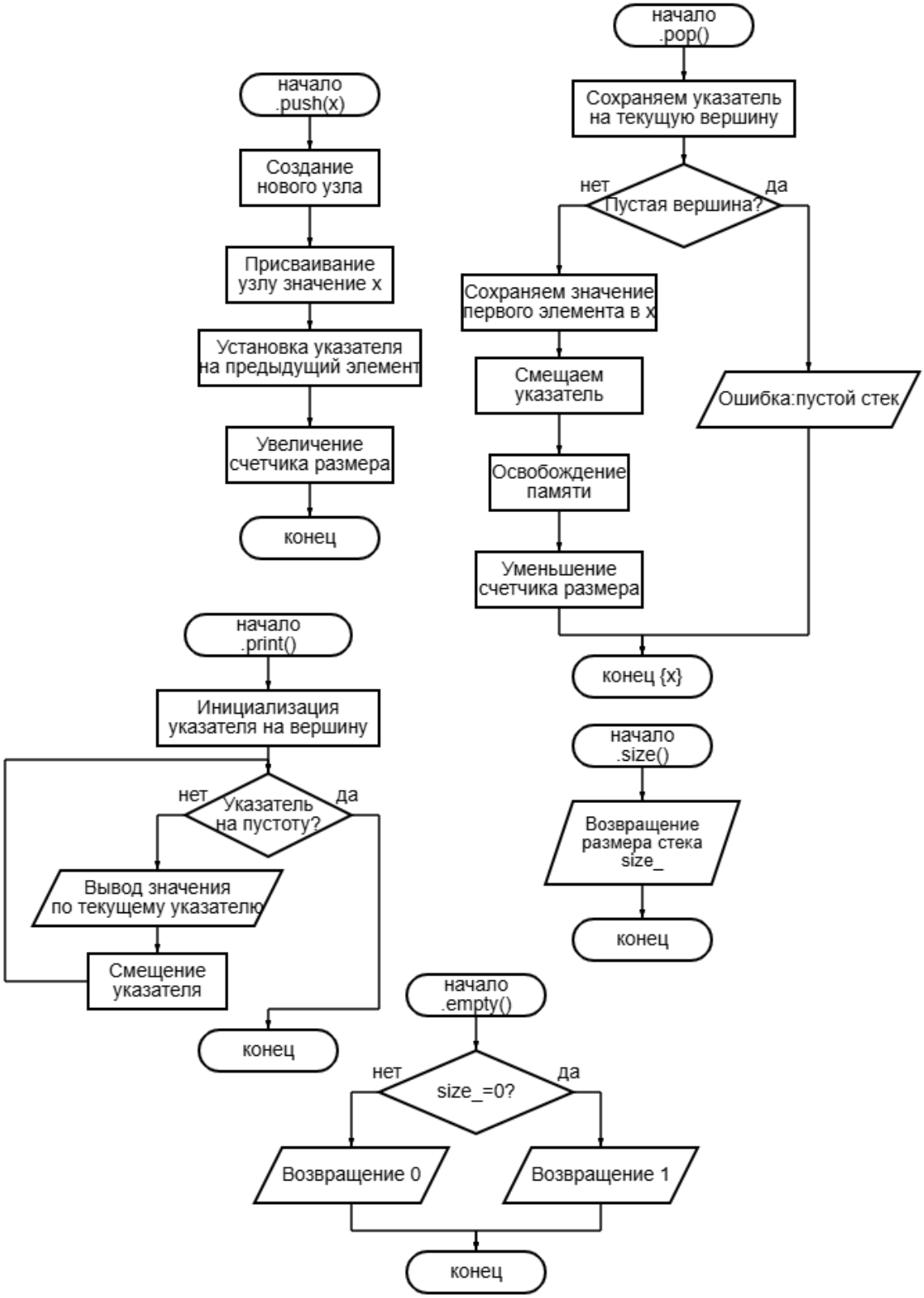

Рисунок 1 – Блок-схема функций класса stack

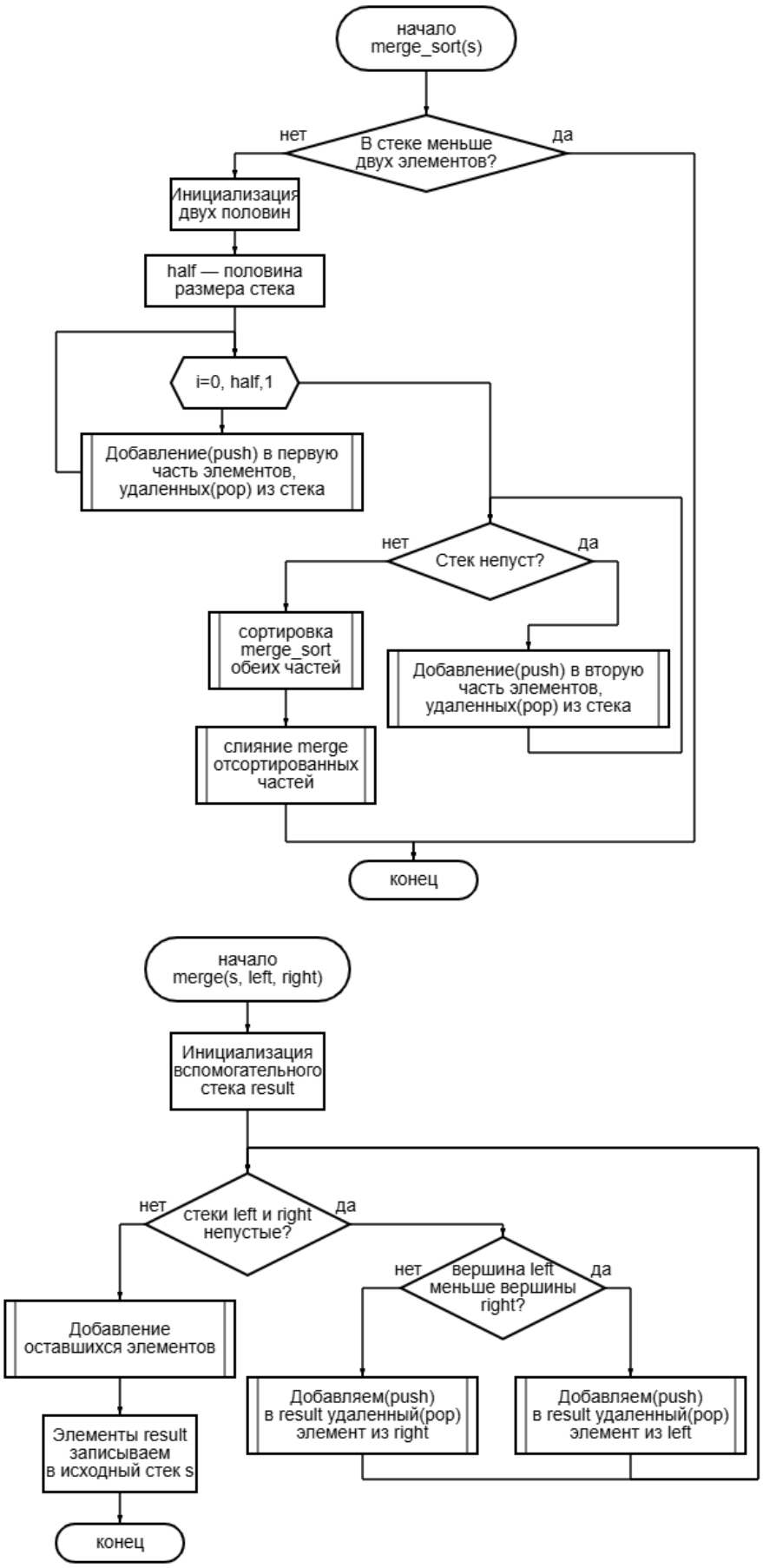

Рисунок 2 – Блок-схема функций слияния и сортировки

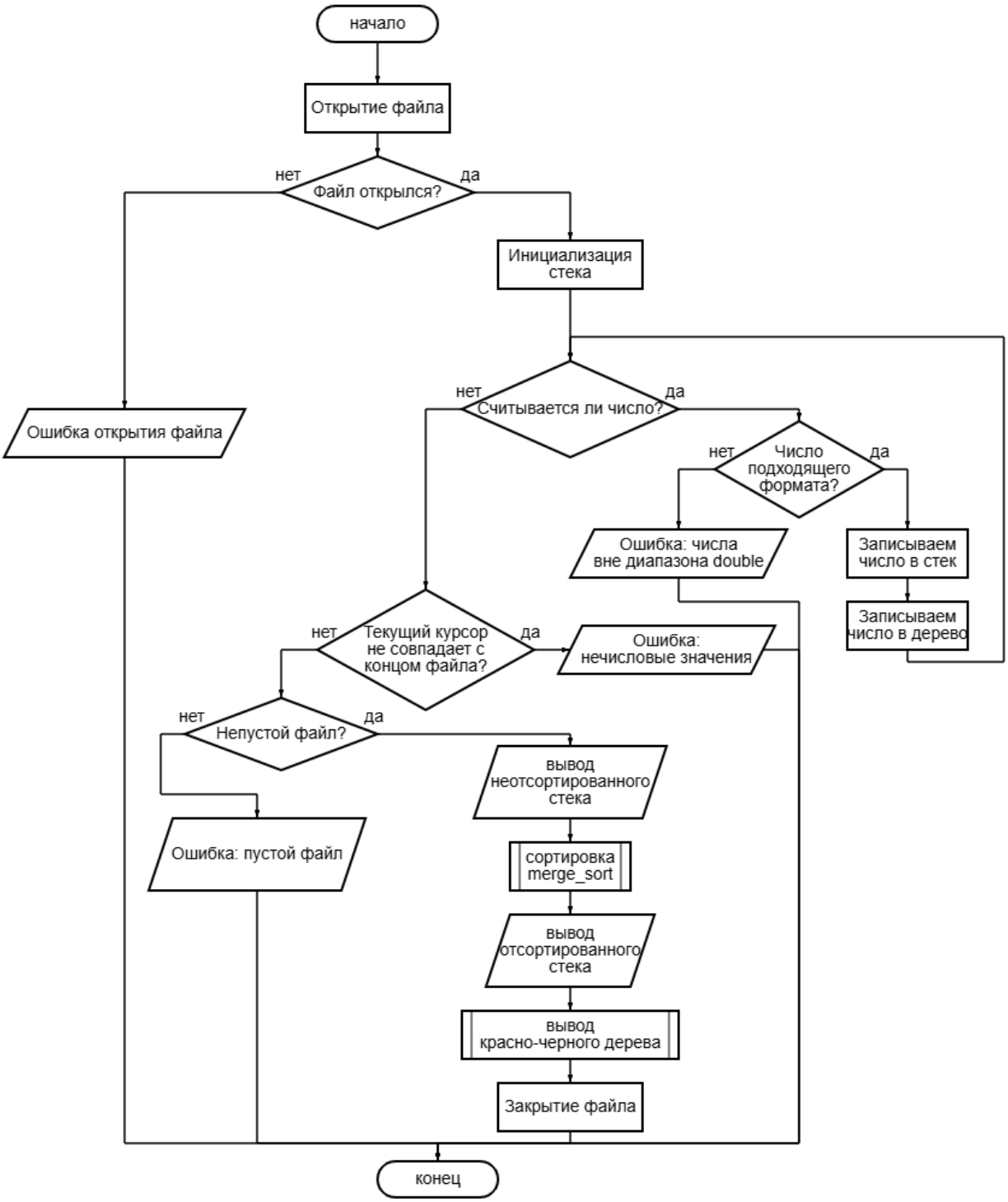

Рисунок 3 – Блок-схема основной части программы

Блок-схемы алгоритма для красно-черного дерева

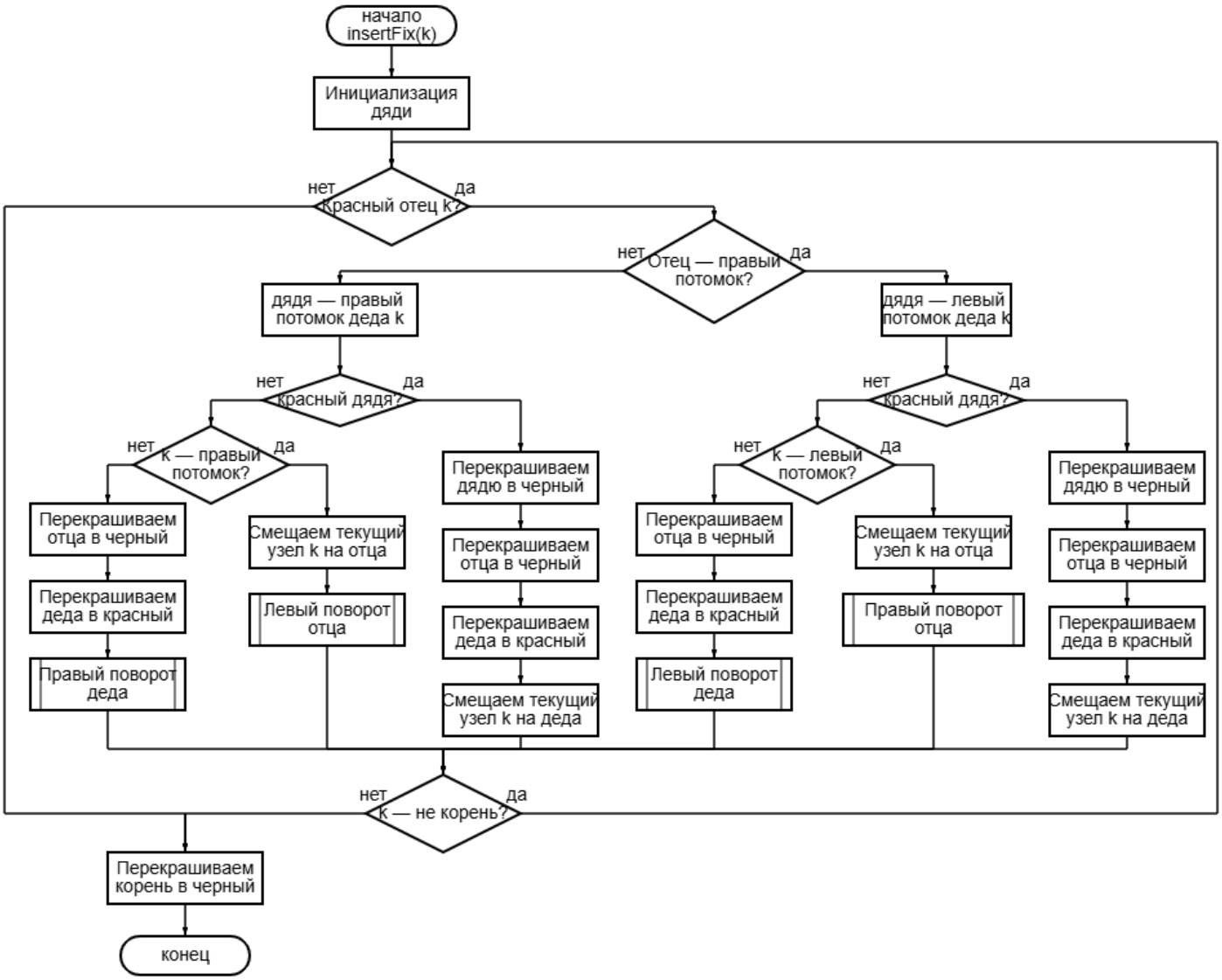

Рисунок 4 – Функция для согласования свойств красно-черного дерева при добавлении элемента

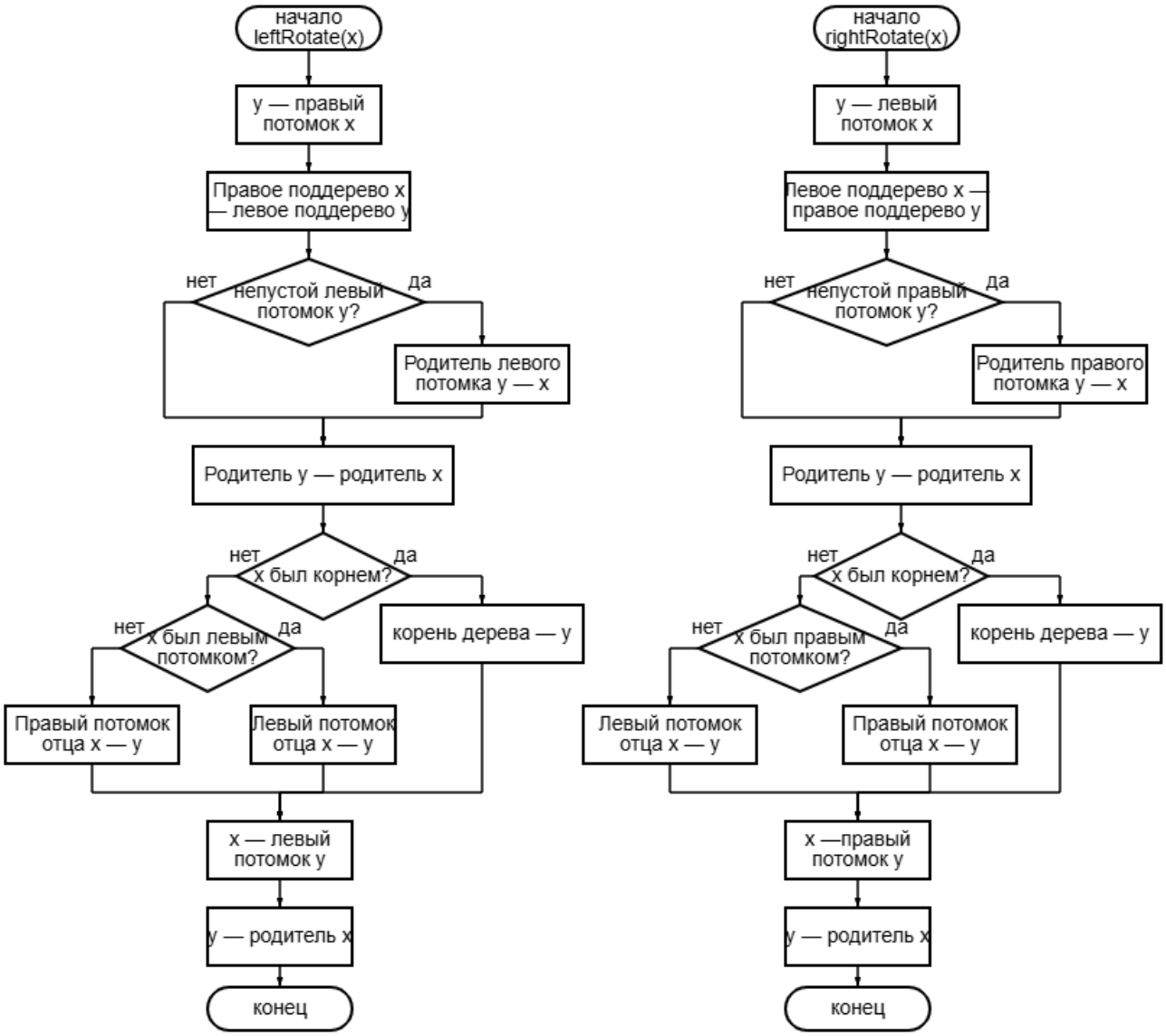

Рисунок 5 – Функции правого и левого поворота красно-черного дерева

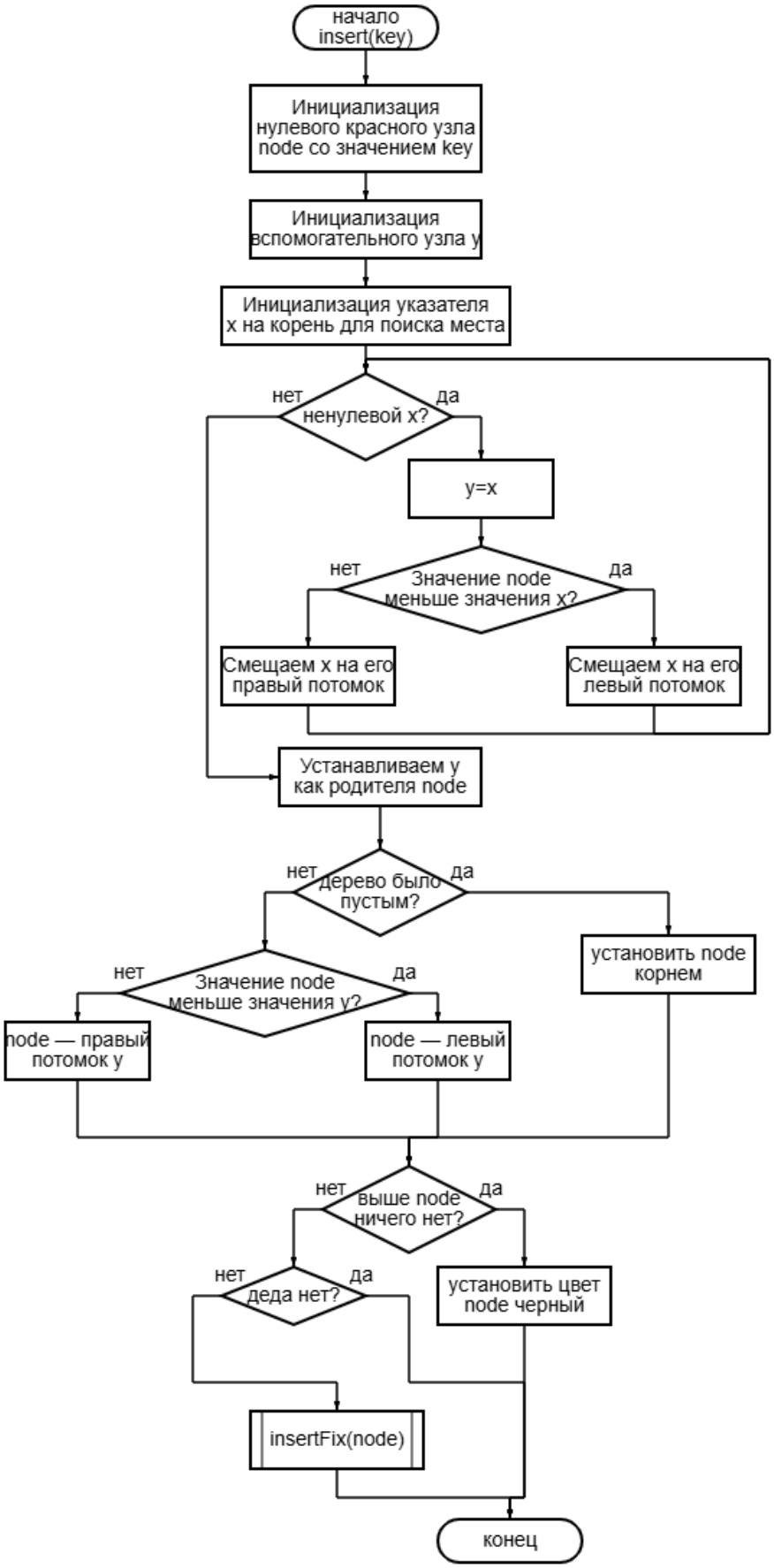

Рисунок 6 – Функция добавления элемента в дерево

КОД

Ниже приведен код, написанный для сортировки слиянием чисел, подаваемых через файл в кольцевую очередь, по указанной выше блок-схеме. Программа реализована на языке C++.

```
#include <iostream>
#include <fstream>
#include <limits>
#include <stdexcept>
#include "red&black.cpp"

using namespace std;
const int N=50;
// Структура
struct node {
      double data;
      node *next;
};
class stack {
private:
      node *top;
      int size_;
public:
      stack(): top(nullptr), size_(0) {}
      // Добавление элемента
      void push (double x) {
            node *Newnode = new node;
            Newnode->data=x;
            Newnode->next=top;
            top = Newnode;
            size_++;
      }
      // Удаление элемента
      double pop() {
            double x;
            node *q=top;
            if(top==nullptr){//пустой стек
                  throw runtime_error("Stack is empty\n");
            }
            x = top->data;//запоминаем удаленное значение
            top = top->next;//смещение указателя
            delete q;//освобождаем память
            size_--;
            return x;
      }
      // Возвращение значения по индексу
      double top_value() { return top->data;}
      // Размер стека
    int size() const { return size_; }
    // Проверка на пустоту
    bool empty() const { return size_ == 0; }
    void print() const {
        node* cur = top;//указатель на вершину
        while (cur != nullptr) {
            cout<<cur->data<<" ";
            cur = cur->next;
        }
        cout<<endl;
```

```
    }
};
void merge(stack &s, stack &left, stack &right) {
      stack result;
      //Добавление меньших чисел
      while (!left.empty() && !right.empty()) {
            // Сравнение первых элементов из частей и добавление меньшего
        if (left.top_value()<right.top_value()) {
            result.push(left.pop());
        }
            else {
            result.push(right.pop());
        }
    }
    // Добавление оставшихся элементов
    while (!left.empty()) result.push(left.pop());
    while (!right.empty()) result.push(right.pop());
    // Разворачиваем стек
    while (!result.empty()) s.push(result.pop());
}
// Рекурсивная сортировка слиянием
void merge_sort(stack &s) {
    if(s.size() <= 1) return;
    // Разделение очереди на две части
    stack left, right;
    int half = s.size() / 2;
    // Заполнение первой части
    for (int i=0; i<half; i++) {
        left.push(s.pop());
    }
    // Заполнение второй части
    while (!s.empty()) {
        right.push(s.pop());
    }
    // Сортировка каждой части
    merge_sort(left);
    merge_sort(right);
    // Слияние двух отсортированных частей
    merge(s, left, right);
}

int main() {
      string file_path="data.txt";
    // Открываем файл для чтения
    ifstream file(file_path);
    // Проверка на ошибку при открытии
    if (!file.is_open()) {
        cerr << "Error opening file" << endl;
        return 1;
    }
      stack s;
      long double num;
    RedBlackTree tree;
    while (file >> num) {
      // Проверка на корректность данных
      if(num<numeric_limits<double>::max() and
num>numeric_limits<double>::lowest()){
            s.push(num);
            tree.insert(num);
            continue;
            }
            cout<<"Error: numbers out of range";
```

```
            return 1;
    }

      // Проверка на ввод всех символов
    file.clear(); // Сброс возможных ошибок
    int cur=file.tellg(); // Сохраняем позицию курсора
    file.seekg(0, ios::end); // Перемещаем курсор в конец файла
    if(cur!=file.tellg()) { //Сравниваем сохраненный курсор с концом файла
      cout<<"Error: non-numeric data";
      return 1;
      }
    // Проверка на наличие данных
    if(s.size()==0){
      cout<<"Error: empty file";
      return 1;
      }
      cout<<"Unsorted stack:\n";
      s.print();
    merge_sort(s);
      cout<<"Sorted stack:\n";
      s.print();
      tree.printTree();
return 0;
}
```

Код программы red&black для реализации красно-черного дерева

```
#include <iostream>
using namespace std;
struct Node {
    double data;
    Node *parent;
    Node *left;
    Node *right;
    int color;};
typedef Node *NodePtr;
class RedBlackTree {
private:
    NodePtr root;
    NodePtr TNULL;

    void insertFix(NodePtr k) {
        NodePtr u;//дядя
        while (k->parent->color == 1) {
            if (k->parent == k->parent->parent->right) {//если родитель—
правый потомок
                u = k->parent->parent->left;//тогда дядя левый потомок
                if (u->color == 1) {//перекрашиваем дядю и отца, если они
красные
                    u->color = 0;
                    k->parent->color = 0;
                    k->parent->parent->color = 1;
                    k = k->parent->parent;
                } else {//если дядя черный, делаем поворот
                    if (k == k->parent->left) {
                        k = k->parent;
                        rightRotate(k);                    }
                    k->parent->color = 0;
                    k->parent->parent->color = 1;
                    leftRotate(k->parent->parent);
                }
```

```
            } else {//то же самое, если родитель левый
                u = k->parent->parent->right;

                if (u->color == 1) {
                    u->color = 0;
                    k->parent->color = 0;
                    k->parent->parent->color = 1;
                    k = k->parent->parent;
                } else {
                    if (k == k->parent->right) {
                        k = k->parent;
                        leftRotate(k);
                    }
                    k->parent->color = 0;
                    k->parent->parent->color = 1;
                    rightRotate(k->parent->parent);
                }
            }
            if (k == root) {//дошли до корня
                break;
            }
        }
        root->color = 0;
    }
    void printHelper(NodePtr root, string indent, bool last) {
        if (root != TNULL) {
            cout << indent;
            if (last) {//правый потомок
                cout << "R____";
                indent += "    ";
            } else {//левый потомок
                cout << "L____";
                indent += "|   ";
            }
                  // Выбор цвета
            if (root->color) {
                cout << "\033[31m" << root->data << "\033[0m\n";  // Красный
            } else {
                cout << root->data << "\n";  // Чёрный (или обычный)
            }
            //вывод левого и правого поддерева
            printHelper(root->left, indent, false);
            printHelper(root->right, indent, true);
        }
    }
public:
    RedBlackTree() {
        TNULL = new Node;
        TNULL->color = 0;
        TNULL->left = nullptr;
        TNULL->right = nullptr;
        root = TNULL;
    }
    void leftRotate(NodePtr x) {
        NodePtr y = x->right;
        x->right = y->left;//левое поддерево y теперь правое поддерево x
        if (y->left != TNULL) {
            y->left->parent = x;
        }
        y->parent = x->parent;
        if (x->parent == nullptr) { // если x был корнем
            this->root = y;
```

```
    } else if (x == x->parent->left) {// если x был левым потомком
        x->parent->left = y;
    } else {// если x был правым потомком
        x->parent->right = y;
    }
    y->left = x; // x — левый потомок y
    x->parent = y; // y — родитель x
}
void rightRotate(NodePtr x) {
    NodePtr y = x->left;
    x->left = y->right; //правое поддерево y теперь леввое поддерево x
    if (y->right != TNULL) {
        y->right->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == nullptr) {// если x был корнем
        this->root = y;
    } else if (x == x->parent->right) {// если x был правым потомком
        x->parent->right = y;
    } else {// если x был левым потомком
        x->parent->left = y;
    }
    y->right = x;// x — левый потомок y
    x->parent = y;// y — родитель x
}
void insert(double key) {
    NodePtr node = new Node;
    node->parent = nullptr;
    node->data = key;
    node->left = TNULL;
    node->right = TNULL;
    node->color = 1;

    NodePtr y = nullptr; //вспомогательный узел
    NodePtr x = this->root;//указатель для поиска места
        //поиск места
    while (x != TNULL) {
        y = x;
        if (node->data < x->data) {
            x = x->left;
        } else {
            x = x->right;
        }
    }
    node->parent = y;//устанавливаем родителя
    if (y == nullptr) {//если дерево было пустым
        root = node;
    } else if (node->data < y->data) {//вставляем влево или право
        y->left = node;
    } else {
        y->right = node;
    }
    if (node->parent == nullptr) {//ничего больше нет
        node->color = 0;
        return;}
    if (node->parent->parent == nullptr) {//нет деда
        return;}
        //восстанавливаем свойства rbt
    insertFix(node);}
void printTree() {
    if (root) {
        printHelper(this->root, "", true);}}};
```

ТЕСТИРОВАНИЕ

Проведено тестирование программы с помощью различных входных данных. Был использован компилятор C++ 3.4.2 и среда разработки Embarcadero Dev-C++ 6.3.

1. Очередь с разнообразными числами (1e+2 -200 300 -9 4)

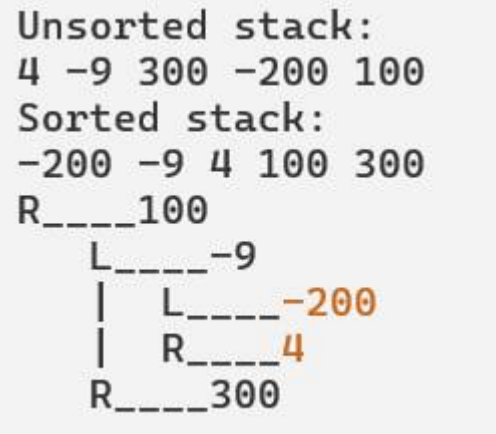

Рисунок 7 – Тест №1

2. Очередь с большим количеством чисел (-88 59 -13 -51 2 -15 8 49 -78 35 2 20 -4 52 85 6 49 1 4 28 -2 03 -2 7753 -3456 -1647 782)

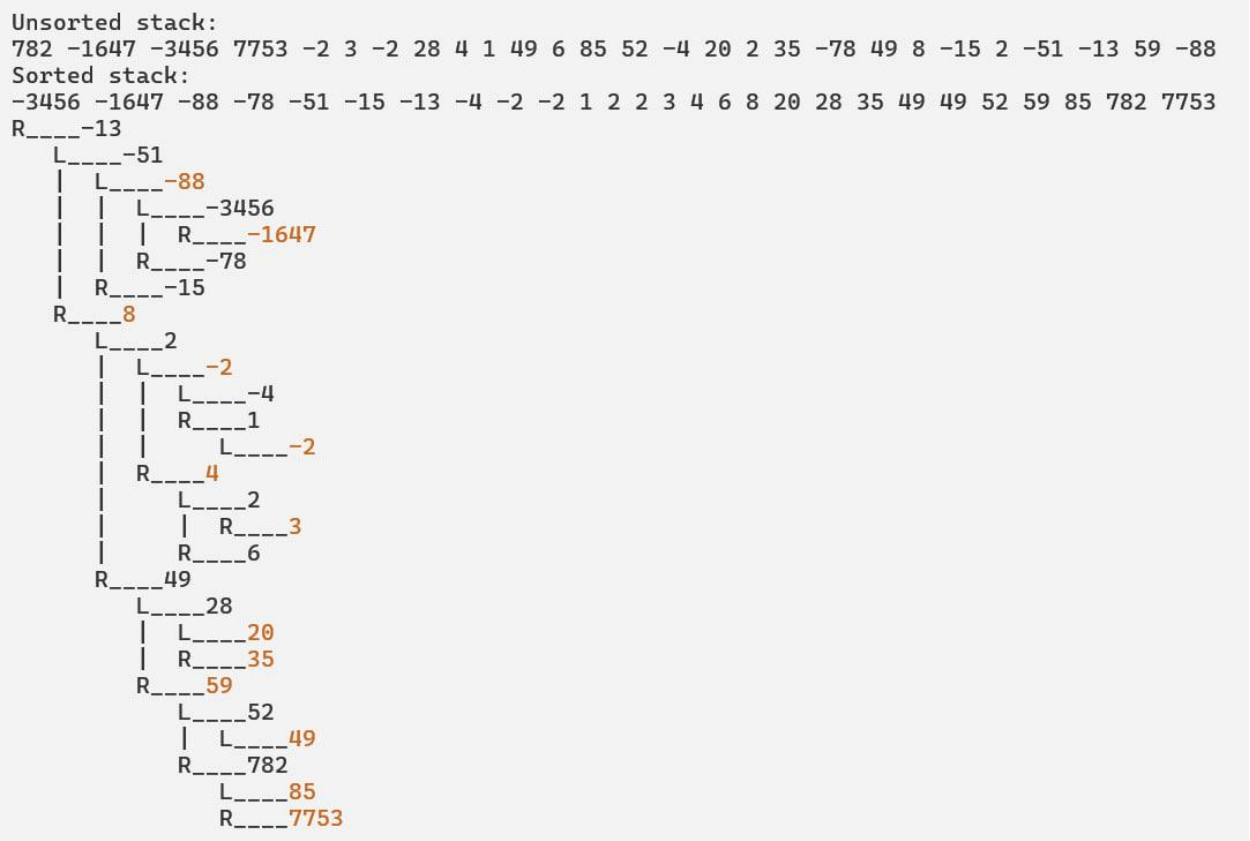

Рисунок 8 – Тест №2

ЗАКЛЮЧЕНИЕ

Был изучен структура стека и красно-черного дерева, написана блок-схема алгоритма программы, которая реализует сортировку слиянием чисел из файла, используя стек и выводя результат в виде красно-черного дерева. На языке C++ написан код программы с выводом неотсортированной и отсортированной очереди и красно-черного дерева, обработкой ошибок и интерфейсом для пользователя. Программа успешно протестирована на различных наборах данных.

Все задачи выполнены и цель достигнута.