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

РТУ МИРЭА

Отчёт по выполнению практического задания № 7.1
Тема:
«Балансировка дерева поиска»

Выполнил студент: Данов Арсений Иванович

Группа: ИКБО-10-23

Москва 2024

Вариант:

Номер: *6*

Тип значения узла: вещественное

Тип дерева: АВЛ-дерево

1.1 Формулировка задачи

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.

Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню.

Провести полное тестирование программы на дереве размером n=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Сделать выводы о проделанной работе, основанные на полученных результатах.

Оформить отчет с подробным описанием созданного дерева, принципов программной реализации алгоритмов работы с деревом, описанием текста исходного кода и проведенного тестирования программы.

1.2 Ход решения

AVL-дерево – это сбалансированное двоичное дерево поиска. Так как по определению оно обязано быть сбалансированным, после каждого добавления нового элемента AVL-дерево необходимо проверять на расбалансированность и, в случае её обнаружения, балансировать.

Для определения разбалансированности вводится параметр фактора баланса, который для каждого узла равен разницы высот правого и левого поддеревьев. Дерево является сбалансированным, если для всех его элементов справедливо правило: баланс фактор по модулю не больше единицы. Если же баланс фактор становится равен 2 или -2 — дерево нуждается в балансировке.

Для балансировки используются повороты дерева — они делятся на левые и правые, большие и маленькие. Причём большой поворот является комбинацией двух маленьких.

Также после балансировки изменяются значения высот поддеревьев, поэтому важно исправить их на корректные.

После вставки, балансировки и исправления высот поддеревьев к дереву возможно применять разнообразные операции:

Прямой обход — вывод значения узла, а потом рекурсивный вывод левого и правого поддеревьев.

Обратный обход — рекурсивный вывод левого и правого поддеревьев, а потом вывод значения узла.

Симметричный обход — рекурсивный вывод левого поддерева, вывод собственного значения узла, рекурсивный вывод правого поддерева. Именно при симметричном обходе на выводе получается отсортированная последовательность элементов дерева поиска.

Обход в ширину — последовательный вывод всех элементов, находящихся на одном уровне, после чего вывод следующего уровня. Отличается отсутствием рекурсивных вызовов методов.

Нахождение суммы значений листьев — рекурсивный обход всех элементов со сложением результатов для левого и правого поддеревьев, причем собственное значение возвращают только листья, остальные оставляют его без изменений.

Нахождение среднего арифметического значений всех узлов — рекурсивный обход всех элементов со сложением результатов для левого и правого поддеревьев, причем все узлы возвращают собственное значение. Для нахождения количества также производится рекурсивный обход, в ходе которого каждый узел увеличивает счетчик на единицу.

1.3 Исходный код программ

На рисунках 1-10 представлен исходный коды программы, файлы заголовков и реализации, а также функция main.

```cpp
#include "funcs.h"

using namespace std;

int main()
{
    avl *tree = nullptr;
    double x;
    while (true)
    {
        int choise;
        cout << endl
             << "Что вы хотите сделать?" << endl
             << "1 - Вывод содержимого" << endl
             << "2 - Добавление записи" << endl
             << "3 - Прямой обход" << endl
             << "4 - Обратный обход" << endl
             << "5 - Симметричный обход" << endl
             << "6 - Обход в ширину" << endl
             << "7 - Сумма значений листьев" << endl
             << "8 - Среднее арифметическое узлов" << endl
             << "9 - Длина пути от корня до значения" << endl
             << "10 - Высота дерева" << endl
             << "0 - Выход" << endl
             << "Номер действия: ";
        cin >> choise;
        cout << endl;

        if (choise == 2)
        {
            cout << "Введите данные новой записи: ";
            cin >> x;
            tree = insert(tree, x);
            continue;
        }

        if (!tree)
        {
            cout << "Дерево пока что пустое, нечего выводить\n";
            continue;
        }

        if (choise == 1)
        {
            cout << "AVL дерево:\n";
            tree->show_tree(tree);
            cout << endl;
        }
        else if (choise == 3)
        {
            tree->preorder();
            cout << endl;
        }
```

Рисунок 1 — Исходный код программы

```
54          else if (choise == 4)
55          {
56              tree->postorder();
57              cout << endl;
58          }
59          else if (choise == 5)
60          {
61              tree->inorder();
62              cout << endl;
63          }
64          else if (choise == 6)
65          {
66              tree->breadth_first();
67              cout << endl;
68          }
69          else if (choise == 7)
70          {
71              cout << "Сумма значений листьев: " << tree->leaves_sum();
72              cout << endl;
73          }
74          else if (choise == 8)
75          {
76              cout << "Среднее арифметическое узлов: " << tree->avg();
77              cout << endl;
78          }
79          else if (choise == 9)
80          {
81              cout << "Введите искомое значение: ";
82              cin >> x;
83              cout << "Длина пути от корня до значения: " << tree->find_length(x);
84              cout << endl;
85          }
86          else if (choise == 10)
87          {
88              cout << "Высота дерева: " << tree->find_height();
89              cout << endl;
90          }
91          else if (choise == 0)
92          {
93              break;
94          }
95          else {
96              cout << "Неизвестная команда.";
97              cout << endl;
98          }
99      }
100     return 0;
101 }
```

Рисунок 2 — Исходный код программы

```
#ifndef __FUNCS__H
#define __FUNCS__H

#include "avl.h"

int height(avl *p);

int bfactor(avl *p);

void fixheight(avl *p);

avl *rotateright(avl *p);

avl *rotateleft(avl *q);

avl *balance(avl *p);

avl *insert(avl *p, double k);

#endif
```

Рисунок 3 — Исходный код программы

```
#ifndef __AVL__H
#define __AVL__H

#include <iostream>
#include <queue>

struct avl
{
    double value;
    avl *left;
    avl *right;
    int height;

    avl(double value);
    void preorder();
    void inorder();
    void postorder();
    void breadth_first();
    double leaves_sum();
    double avg();
    int find_length(double value);
    int find_height();
    void show_tree(avl* root);

private:
    int count();
    double sum();
};

#endif
```

Рисунок 4 — Исходный код программы

```c
#include "funcs.h"

int height(avl *p)
{
    return p ? p->height : 0;
}

int bfactor(avl *p)
{
    return height(p->right) - height(p->left);
}

void fixheight(avl *p)
{
    int hl = height(p->left);
    int hr = height(p->right);
    p->height = (hl > hr ? hl : hr) + 1;
}

avl *rotateright(avl *p) // правый поворот вокруг p
{
    avl *q = p->left;
    p->left = q->right;
    q->right = p;
    fixheight(p);
    fixheight(q);
    return q;
}

avl *rotateleft(avl *q) // левый поворот вокруг q
{
    avl *p = q->right;
    q->right = p->left;
    p->left = q;
    fixheight(q);
    fixheight(p);
    return p;
}
```

Рисунок 5 — Исходный код программы

```
40  avl *balance(avl *p) // балансировка узла p
41  {
42      fixheight(p);
43      if (bfactor(p) == 2)
44      {
45          if (bfactor(p->right) < 0)
46              p->right = rotateright(p->right);
47          return rotateleft(p);
48      }
49      if (bfactor(p) == -2)
50      {
51          if (bfactor(p->left) > 0)
52              p->left = rotateleft(p->left);
53          return rotateright(p);
54      }
55      return p; // балансировка не нужна
56  }
57
58  avl *insert(avl *p, double k) // вставка ключа k в дерево с корнем p
59  {
60      if (!p)
61          return new avl(k);
62      if (k < p->value)
63          p->left = insert(p->left, k);
64      else
65          p->right = insert(p->right, k);
66      return balance(p);
67  }
```

Рисунок 6 — Исходный код программы

```cpp
#include "avl.h"

using namespace std;

avl::avl(double value)
{
    this->value = value;
    this->left = nullptr;
    this->right = nullptr;
    this->height = 1;
}

void avl::preorder()
{
    cout << this->value << " ";
    if (this->left)
    {
        this->left->preorder();
    }
    if (this->right)
    {
        this->right->preorder();
    }
}

void avl::inorder()
{

    if (this->left)
    {
        this->left->inorder();
    }
    cout << this->value << " ";
    if (this->right)
    {
        this->right->inorder();
    }
}

void avl::postorder()
{

    if (this->left)
    {
        this->left->postorder();
    }
    if (this->right)
    {
        this->right->postorder();
    }
    cout << this->value << " ";
}
```

Рисунок 7 — Исходный код программы

```
54  void avl::breadth_first()
55  {
56      queue<avl *> q;
57      q.push(this);
58      while (!q.empty())
59      {
60          avl *cur = q.front();
61          q.pop();
62          cout << cur->value << " ";
63          if (cur->left)
64          {
65              q.push(cur->left);
66          }
67          if (cur->right)
68          {
69              q.push(cur->right);
70          }
71      }
72  }
73
74  double avl::leaves_sum()
75  {
76      if (!this->left && !this->right)
77      {
78          return this->value;
79      }
80      double sum = 0;
81      if (this->left)
82      {
83          sum += this->left->leaves_sum();
84      }
85      if (this->right)
86      {
87          sum += this->right->leaves_sum();
88      }
89      return sum;
90  }
91
92  double avl::avg()
93  {
94      return this->sum() / this->count();
95  }
```

Рисунок 8 — Исходный код программы

```
97  double avl::sum()
98  {
99      int sum = this->value;
100     if (this->left)
101     {
102         sum += this->left->sum();
103     }
104     if (this->right)
105     {
106         sum += this->right->sum();
107     }
108     return sum;
109 }
110
111 int avl::count()
112 {
113     int sum = 1;
114     if (this->left)
115     {
116         sum += this->left->count();
117     }
118     if (this->right)
119     {
120         sum += this->right->count();
121     }
122     return sum;
123 }
124
125 int avl::find_length(double value) {
126     if (this->value == value) {
127         return 0;
128     }
129     int mn = 1000;
130     int x;
131     if (this->left) {
132         x = this->left->find_length(value);
133         mn = x < mn ? x : mn;
134     }
135     if (this->right) {
136         x = this->right->find_length(value);
137         mn = x < mn ? x : mn;
138     }
139     return mn + 1;
140 }
```

Рисунок 9 — Исходный код программы

```
142  int avl::find_height() {
143      if (!this->left && !this->right) {
144          return 0;
145      }
146      int mx = -1;
147      int x;
148      if (this->left) {
149          x = this->left->find_height();
150          mx = x > mx ? x : mx;
151      }
152      if (this->right) {
153          x = this->right->find_height();
154          mx = x > mx ? x : mx;
155      }
156      return mx + 1;
157  }
158
159
160  void avl::show_tree(avl* root) {
161      cout << string((root->find_length(this->value)) * 4, ' ') << this->value << '\n';
162      if (this->left) {
163          this->left->show_tree(root);
164      }
165      if (this->right) {
166          this->right->show_tree(root);
167      }
168  }
```

Рисунок 10 — Исходный код программы

1.4 Результаты тестирования

На рисунках 11-19 представлены результаты тестирования программы — все возможные обходы, а также дейтсвия над узлами.

```
1 - Вывод содержимого
2 - Добавление записи
3 - Прямой обход
4 - Обратный обход
5 - Симметричный обход
6 - Обход в ширину
7 - Сумма значений листьев
8 - Среднее арифметическое узлов
9 - Длина пути от корня до значения
10 - Высота дерева
0 - Выход
Номер действия: 1

AVL дерево:
42.13
    23.743
        5.1
        34.15
            30.152
            36.15
    64.1
        51.7
        78.52
            71
            83.1
```

Рисунок 11 — Результаты тестирования программы

```
Что вы хотите сделать?
1 - Вывод содержимого
2 - Добавление записи
3 - Прямой обход
4 - Обратный обход
5 - Симметричный обход
6 - Обход в ширину
7 - Сумма значений листьев
8 - Среднее арифметическое узлов
9 - Длина пути от корня до значения
10 - Высота дерева
0 - Выход
Номер действия: 3

42.13 23.743 5.1 34.15 30.152 36.15 64.1 51.7 78.52 71 83.1
```

Рисунок 12 — Результаты тестирования программы

```
Что вы хотите сделать?
1 - Вывод содержимого
2 - Добавление записи
3 - Прямой обход
4 - Обратный обход
5 - Симметричный обход
6 - Обход в ширину
7 - Сумма значений листьев
8 - Среднее арифметическое узлов
9 - Длина пути от корня до значения
10 - Высота дерева
0 - Выход
Номер действия: 4

5.1 30.152 36.15 34.15 23.743 51.7 71 83.1 78.52 64.1 42.13
```

Рисунок 13 — Результаты тестирования программы

```
Что вы хотите сделать?
1 - Вывод содержимого
2 - Добавление записи
3 - Прямой обход
4 - Обратный обход
5 - Симметричный обход
6 - Обход в ширину
7 - Сумма значений листьев
8 - Среднее арифметическое узлов
9 - Длина пути от корня до значения
10 - Высота дерева
0 - Выход
Номер действия: 5

5.1 23.743 30.152 34.15 36.15 42.13 51.7 64.1 71 78.52 83.1
```

Рисунок 14 — Результаты тестирования программы

```
Что вы хотите сделать?
1 - Вывод содержимого
2 - Добавление записи
3 - Прямой обход
4 - Обратный обход
5 - Симметричный обход
6 - Обход в ширину
7 - Сумма значений листьев
8 - Среднее арифметическое узлов
9 - Длина пути от корня до значения
10 - Высота дерева
0 - Выход
Номер действия: 6

42.13 23.743 64.1 5.1 34.15 51.7 78.52 30.152 36.15 71 83.1
```

Рисунок 15 — Результаты тестирования программы

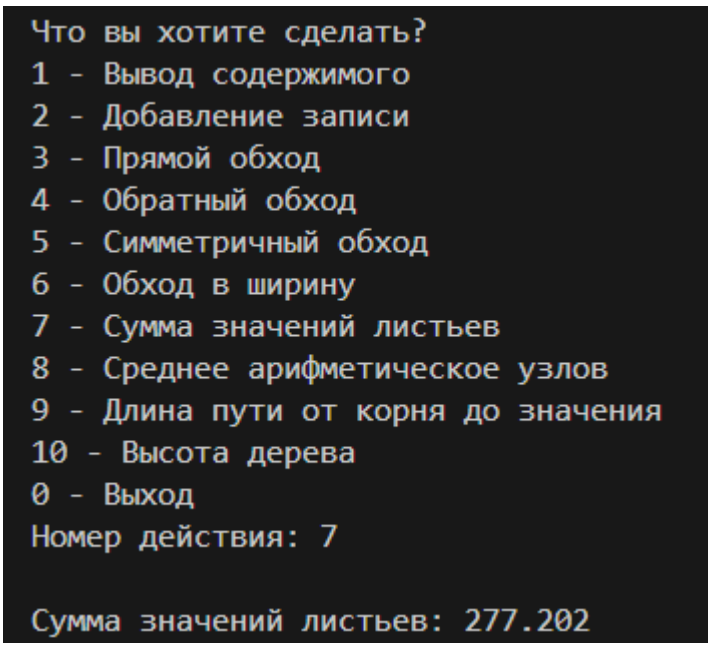

Рисунок 16 — Результаты тестирования программы

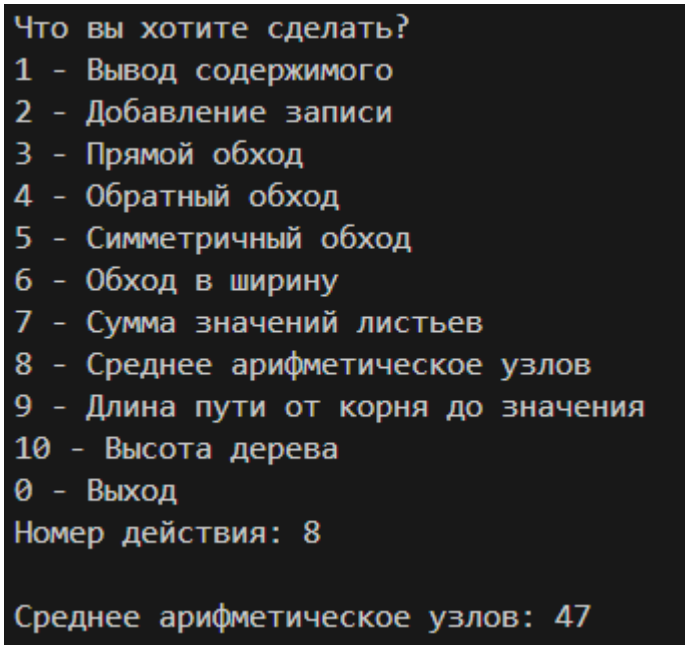

Рисунок 17 — Результаты тестирования программы

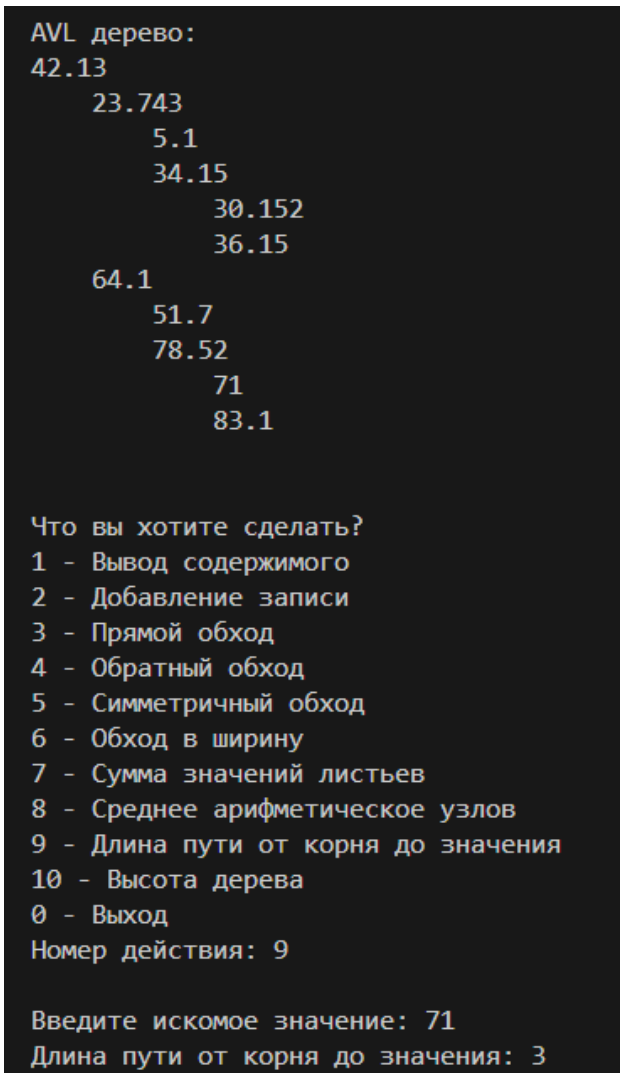

Рисунок 18 — Результаты тестирования программы

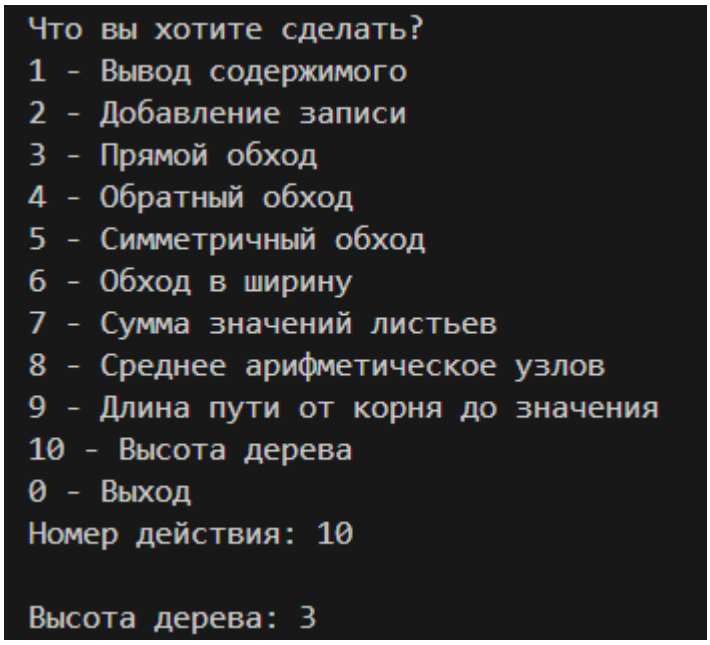

Рисунок 19 — Результаты тестирования программы

2.1 Выводы

Бинарные деревья поиска — удобные структуры данных, совмещающие удобство восприятия человеком, а также позволяют для разных целей совершать разные обходы.