МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

АНАЛИЗ АВЛ-ДЕРЕВА

ОТЧЁТ

студента 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Карасева Вадима Дмитриевича

Проверено:
доцент, к. ф.-м. н. _____ М. И. Сафрончик

Саратов 2025

СОДЕРЖАНИЕ

1 Пример программы

```
#include <iostream>
#include <algorithm>
using namespace std;

struct Node {
    int key;
    Node* left;
    Node* right;
    int height;

    Node(int k) : key(k), left(nullptr), right(nullptr), height(1) {}
};

int getHeight(Node* node) {
    return node ? node->height : 0;
}

int getBalance(Node* node) {
    return node ? getHeight(node->left) - getHeight(node->right) : 0;
}

void updateHeight(Node* node) {
    node->height = max(getHeight(node->left), getHeight(node->right)) + 1;
}

Node* rotateRight(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;

    x->right = y;
    y->left = T2;

    updateHeight(y);
    updateHeight(x);

    return x;
}

Node* rotateLeft(Node* x) {
    Node* y = x->right;
```

```
41      Node* T2 = y->left;
42
43      y->left = x;
44      x->right = T2;
45
46      updateHeight(x);
47      updateHeight(y);
48
49      return y;
50  }
51
52  Node* balance(Node* node) {
53      updateHeight(node);
54      int balanceFactor = getBalance(node);
55
56      // Левый перекос
57      if (balanceFactor > 1) {
58          if (getBalance(node->left) < 0)
59              node->left = rotateLeft(node->left);  // LR
60          return rotateRight(node);  // LL
61      }
62
63      // Правый перекос
64      if (balanceFactor < -1) {
65          if (getBalance(node->right) > 0)
66              node->right = rotateRight(node->right);  // RL
67          return rotateLeft(node);  // RR
68      }
69
70      return node;
71  }
72
73  Node* insert(Node* node, int key) {
74      if (!node)
75          return new Node(key);
76      if (key < node->key)
77          node->left = insert(node->left, key);
78      else if (key > node->key)
79          node->right = insert(node->right, key);
80      else
81          return node; // дубликаты не вставляем
```

```
82
83      return balance(node);
84  }
85
86  Node* getMinValueNode(Node* node) {
87      Node* current = node;
88      while (current && current->left)
89          current = current->left;
90      return current;
91  }
92
93  Node* remove(Node* root, int key) {
94      if (!root)
95          return root;
96
97      if (key < root->key)
98          root->left = remove(root->left, key);
99      else if (key > root->key)
100         root->right = remove(root->right, key);
101     else {
102         // Один или ноль потомков
103         if (!root->left || !root->right) {
104             Node* temp = root->left ? root->left : root->right;
105             delete root;
106             return temp;
107         }
108
109         // Два потомка
110         Node* temp = getMinValueNode(root->right);
111         root->key = temp->key;
112         root->right = remove(root->right, temp->key);
113     }
114
115     return balance(root);
116 }
117
118 void inorder(Node* root) {
119     if (root) {
120         inorder(root->left);
121         cout << root->key << " ";
122         inorder(root->right);
```

```
123     }
124 }
125
126 int main() {
127     Node* root = nullptr;
128     int n, val;
129
130     cout << "Введите количество узлов для вставки: ";
131     cin >> n;
132
133     cout << "Введите " << n << " значений:\n";
134     for (int i = 0; i < n; ++i) {
135         cin >> val;
136         root = insert(root, val);
137     }
138
139     cout << "Симметричный обход после вставки: ";
140     inorder(root);
141     cout << endl;
142
143     cout << "Введите значение для удаления: ";
144     cin >> val;
145     root = remove(root, val);
146
147     cout << "Симметричный обход после удаления: ";
148     inorder(root);
149     cout << endl;
150
151     return 0;
152 }
```

2 Вставка в АВЛ-дереве

В худшем случае вставка требует времени O(logn), где n-количество элементов в дереве. Это происходит из-за необходимости сбалансировать дерево после каждой вставки, что занимает время, пропорциональное высоте дерева. Следовательно, время вставки для n элементов будет O(nlogn).

3 Удаление в АВЛ-дереве

Как и вставка, удаление также требует времени O(logn) в худшем случае. После удаления элемента дерево также требуется сбалансировать. Таким образом, время удаления для n элементов также будет O(nlogn).

4 Поиск в АВЛ-дереве

Время поиска в АВЛ-дереве также O(logn) в худшем случае. Поиск выполняется по пути от корня до листа в дереве, при этом каждый раз отбрасывается половина оставшихся узлов.

5 Обходы дерева

Префиксный (preorder), постфиксный (postorder) и инфиксный (inorder) обходы занимают O(n) времени, так как каждый узел дерева должен быть посещен ровно один раз. Таким образом, общая временная сложность операций вставки, удаления, поиска и обходов в АВЛ-дереве составляет O(nlogn) для n элементов.

6 Расход памяти

Узел дерева

Для каждого узла дерева выделяется фиксированное количество памяти, состоящее из: inf: O(1) памяти left и right: указатели на другие узлы дерева, каждый из которых занимает O(1) памяти height: переменная типа unsignedchar, занимающая O(1) памяти Таким образом, общее количество памяти, выделенное под каждый узел дерева, составляет O(1).

Входные данные

Расход памяти на входные данные (значения, которые вставляются в дерево) также не зависит от размера дерева и может быть считан O(n), где n — количество элементов.

Стек вызовов

Во время выполнения рекурсивных операций используется стек вызовов, чтобы хранить информацию о текущем состоянии выполнения каждой рекурсивной функции. Глубина стека вызовов зависит от высоты дерева и количества рекурсивных вызовов, что может быть O(logn) в худшем случае для операций вставки,удаления и поиска. Таким образом, общий асимптотический анализ сложности расхода памяти составляет O(n+logn), где n - количество элементов в дереве.

Итог

Вращение происходит за время O(1). Временная сложность всех функций равна O(logN), потому что дерево всегда сбалансированное.