



Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ

«Фундаментальные Науки»

КАФЕДРА

ФН-12 «Математическое моделирование»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ НА ТЕМУ:

АВЛ-дерево

Студент:

Мацеевский И. М.

дата, подпись

Ф.И.О.

Преподаватель:

Волкова Л. Л.

дата, подпись

Ф.И.О.

Москва, 2023

Содержание

Введение	2
1 Аналитическая часть	3
2 Конструкторская часть	4
3 Технологическая часть	7
Заключение	17
Список используемых источников	18

Введение

Цель лабораторной работы: описать структуру данных АВЛ-дерева.

Для достижения поставленной цели требуется решить следующие **задачи**.

1. Описать алгоритмы добавления и удаления элемента, поиска элемента в дереве, балансировки дерева.
2. Разработать программу, предоставляющую пользователю выбор пункта меню, отображающую меню в цикле с постусловием, реализующую все описанные алгоритмы.

1 Аналитическая часть

АВЛ-дерево — это сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1. Преимущество таких деревьев в том, что они поддерживают логарифмическое время поиска, в отличие обычных деревьев, в которых в худшем случае поиск проходит за $O(N)$.

Для описания структуры АВЛ-дерева используется класс *AVLTree*. Для работы с этой структурой реализованы следующие методы: поиск высоты дерева, нахождение разницы между левым и правым поддеревом, поворот направо и налево для балансировки дерева при добавлении нового элемента, добавление элемента, нахождение узла с минимальным ключом, удаление узла, поиск узла, вывод меню на экран, вывод дерева на экран, вывод указателя на корень дерева.

2 Конструкторская часть

Реализованные методы и функции

1. *Height* возвращает 0, если дерево пусто, иначе его высоту.
2. *BalanceFactor* подсчитывает разницу между поддеревьями одного узла и возвращает ее.
3. *rotateRight* выполняет операцию правого вращения, для этого сначала сохраняется указатель на левого потомка(x) и правое поддерево ($T2$) данного узла y , затем происходит поворот: x становится узлом, y становится левым поддеревом, а $T2$ правым поддеревом, возвращается указатель на новый корень поддерева.
4. *rotateLeft* выполняет операцию левого вращения, для этого сначала сохраняется указатель на правого потомка(y) и левое поддерево ($T2$) данного узла x , затем происходит поворот: y становится узлом, x становится левым поддеревом, а $T2$ правым поддеревом, возвращается указатель на новый корень поддерева.
5. *balance* выполняет балансировку дерева, начиная с заданного узла проверяет, нарушается ли баланс в узле, если да, применяет соответствующие вращения для восстановления баланса.
 - (a) Если узел *node* равен *nullptr*, то возвращается *nullptr*. Дерево пусто и балансировка не требуется.
 - (b) Высота узла *node* обновляется, исходя из максимальных высот его левого и правого поддеревьев.
 - (c) Вычисляется фактор баланса для узла *node*, который представляет собой разницу между высотой правого и левого поддеревьев.
 - (d) Если фактор баланса больше 1 и фактор баланса левого поддерева $node->left$ неотрицателен, то происходит правое вращение (*rotateRight(node)*).
 - (e) Если фактор баланса меньше -1 и фактор баланса правого поддерева $node->right$ не положителен, то происходит левое вращение (*rotateLeft(node)*).
 - (f) Если фактор баланса больше 1 и фактор баланса левого поддерева $node->left$ отрицателен, то сначала выполняется левое вращение для $node->left$, затем правое вращение для *node* (*rotateLeft(node->left)* и *rotateRight(node)*).
 - (g) Если фактор баланса меньше -1 и фактор баланса правого поддерева $node->right$ положителен, то сначала выполняется правое вращение для $node->right$, затем левое вращение для *node* (*rotateRight(node->right)* и *rotateLeft(node)*).
 - (h) Если балансировка не требуется, возвращается исходный узел *node*.

6. *insert* вставляет ключ в дерево и производит балансировку:

- (a) Если узел *node* равен *nullptr*, то создается новый узел с ключом *key*, высотой 1 и без дочерних узлов.
- (b) Если *key* меньше значения узла *node* — $> data$, рекурсивно вызывается *insert* для левого поддеревья, и результат присваивается *node* — $> left$.
Если *key* больше значения узла *node* — $> data$, рекурсивно вызывается *insert* для правого поддеревья, и результат присваивается *node* — $> right$.
Если *key* равен значению узла, возвращается сам узел, поскольку в AVL-дереве не может быть двух узлов с одинаковыми ключами на одном уровне.
- (c) После вставки узла обновляется его высота. Высота узла устанавливается в 1 плюс максимальная высота его левого и правого поддеревьев.
- (d) Вызывается функция *balance(node)*, которая обеспечивает балансировку дерева. Функция *balance* проверяет фактор баланса узла и применяет соответствующие вращения для восстановления баланса. Если дерево становится несбалансированным после вставки, эта часть обеспечивает его балансировку.
- (e) Возвращается указатель на текущий узел. Если вставка производится в корень дерева, это может изменить указатель на корень.

7. *minValueNode* находит узел с минимальным ключом в дереве:

- (a) Указатель *current* инициализируется узлом *node*, переданным в функцию.
- (b) Перемещение по дереву влево (так как минимальные ключи слева) до тех пор, пока есть левые потомки.
- (c) Возвращается значение *current*.

8. *deleteNode* удаляет элемент и балансирует дерево.

9. *search* ищет заданный ключ в дереве.

- (a) Создается указатель *current* и инициализируется значением корневого узла (*root*).
- (b) Запускается цикл, который будет выполняться, пока *current* не станет равным *nullptr*.
- (c) Если значение ключа (*key*) равно значению текущего узла (*current* — $> data$), то функция возвращает *true*, то есть ключ найден в дереве.
Если значение ключа меньше значения текущего узла, то обновляется значение *current* на левого потомка текущего узла (*current* — $> left$).
Если значение ключа больше значения текущего узла, то обновляется значение *current* на правого потомка текущего узла (*current* — $> right$).

- (d) Если цикл завершается (*current* становится равным *nullptr*), это означает, что ключ не был найден в дереве, и функция возвращает *false*.

10. *displayMenu* выводит меню на экран.

11. *displayTree* выводит дерево на экран.

3 Технологическая часть

Для реализации выбран язык C++. На листинге 1 представлена реализация программы. (Реализация 1)

Листинг 1 – Исходный код

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

struct Node {
    int data;
    Node* left;
    Node* right;
    int height;
};

class AVLTree {
private:
    Node* root;

    int Height(Node* node) {
        if (node == nullptr)
            return 0;
        return node->height;
    }

    int BalanceFactor(Node* node) {
        if (node == nullptr)
            return 0;
        return Height(node->left) - Height(node->right);
    }

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

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

        y->height = max(Height(y->left), Height(y->right)) + 1;
        x->height = max(Height(x->left), Height(x->right)) + 1;
    }
};
```



```

        return x;
    }

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

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

    x->height = max(Height(x->left), Height(x->right)) + 1;
    y->height = max(Height(y->left), Height(y->right)) + 1;

    return y;
}

Node* balance(Node* node) {
    if (node == nullptr)
        return nullptr;

    // Обновляет вес
    node->height = max(Height(node->left), Height(node->right)) +
1;

    int balance = BalanceFactor(node);

    if (balance < -1 && BalanceFactor(node->right) <= 0)
        return rotateLeft(node);

    if (balance > 1 && BalanceFactor(node->left) >= 0)
        return rotateRight(node);

    if (balance < -1 && BalanceFactor(node->right) > 0) {
        node->right = rotateRight(node->right);
        return rotateLeft(node);
    }

    if (balance > 1 && BalanceFactor(node->left) < 0) {
        node->left = rotateLeft(node->left);
        return rotateRight(node);
    }
}

```

```

    }

    return node;
}

Node* insert(Node* node, int key) {
    if (node == nullptr)
        return new Node{key, nullptr, nullptr, 1};

    if (key < node->data)
        node->left = insert(node->left, key);
    else if (key > node->data)
        node->right = insert(node->right, key);
    else
        return node;

    node->height = 1 + max(Height(node->left), Height(node->right))
;

    return balance(node);
}

Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left != nullptr)
        current = current->left;
    return current;
}

Node* deleteNode(Node* root, int key) {
    if (root == nullptr)
        return root;

    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        if (root->left == nullptr || root->right == nullptr) {
            Node* temp = root->left ? root->left : root->right;

```

```

        if (temp == nullptr) {
            temp = root;
            root = nullptr;
        } else
            *root = *temp;

        delete temp;
    } else {
        Node* temp = minValueNode(root->right);

        root->data = temp->data;

        root->right = deleteNode(root->right, temp->data);
    }
}

if (root == nullptr)
    return root;
root->height = 1 + max(Height(root->left), Height(root->right))
;

return balance(root);
}

public:
    AVLTree() : root(nullptr) {}

    void insert(int key) {
        root = insert(root, key);
    }

    void remove(int key) {
        root = deleteNode(root, key);
    }

    bool search(int key) {
        Node* current = root;
        while (current != nullptr) {
            if (key == current->data)
                return true;
            else if (key < current->data)

```

```

        current = current->left;
    else
        current = current->right;
}
return false;
}

void displayMenu() {
    cout << "\Меню АВЛдерева-:\n";
    cout << "1 - Показать дерево\n";
    cout << "2 - Вставка ключа\n";
    cout << "3 - Удаление ключа\n";
    cout << "4 - Поиск ключа\n";
    cout << "5 - Завершение\n";
}

void displayTree(Node* root, int space) {
    const int INDENT = 5;
    if (root == nullptr)
        return;

    space += INDENT;

    displayTree(root->right, space);

    cout << endl;
    for (int i = INDENT; i < space; i++)
        cout << " ";

    cout << root->data << "\n";

    displayTree(root->left, space);
}

Node* Root() {
    return root;
}

};

int main() {
    AVLTree avl;

```

```

int choice, key;

do {
    avl.displayMenu();
    cout << "Выберите действие: ";
    cin >> choice;

    switch (choice) {
        case 1:
            avl.displayTree(avl.Root(), 0);
            break;
        case 2:
            cout << "Введите ключ, который хотите добавить: ";
            cin >> key;
            avl.insert(key);
            break;
        case 3:
            cout << "Введите ключ, который хотите удалить: ";
            cin >> key;
            avl.remove(key);
            break;
        case 4:
            cout << "Введите ключ, который хотите найти: ";
            cin >> key;
            if (avl.search(key)) {
                cout << "Ключ найден\n";
            } else {
                cout << "Ключ не найден\n";
            }
            break;
        case 5:
            cout << "Завершение программы\n";
            break;
        default:
            cout << "Неверный выбор из меню\n";
    }
} while (choice != 5);

return 0;
}

```

Примеры работы

На рисунках 1—4 представлены примеры работы работы АВЛ-дерева.

1. Входные данные: произвольные ключи. Результат приведён на рис.1 — рис.4.

```
Меню АВЛ-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 2
Введите ключ, который хотите добавить: 4

Меню АВЛ-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 2
Введите ключ, который хотите добавить: 2

Меню АВЛ-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 2
Введите ключ, который хотите добавить: 8

Меню АВЛ-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 2
Введите ключ, который хотите добавить: 1
```

Рис. 1 – Пример работы 1

```
Меню AVL-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 2
Введите ключ, который хотите добавить: 3

Меню AVL-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 2
Введите ключ, который хотите добавить: 5

Меню AVL-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 2
Введите ключ, который хотите добавить: 9

Меню AVL-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 2
Введите ключ, который хотите добавить: 0
```

Рис. 2 – Пример работы 2

```
Меню AVL-дерева:  
1 – Показать дерево  
2 – Вставка ключа  
3 – Удаление ключа  
4 – Поиск ключа  
5 – Завершение  
Выберите действие: 2  
Введите ключ, который хотите добавить: 5  
  
Меню AVL-дерева:  
1 – Показать дерево  
2 – Вставка ключа  
3 – Удаление ключа  
4 – Поиск ключа  
5 – Завершение  
Выберите действие: 2  
Введите ключ, который хотите добавить: 7  
  
Меню AVL-дерева:  
1 – Показать дерево  
2 – Вставка ключа  
3 – Удаление ключа  
4 – Поиск ключа  
5 – Завершение  
Выберите действие: 1
```

Рис. 3 – Пример работы 3


```

      9
    8
      7
    5
  4
    3
  2
    1
    0

Меню AVL-дерева:
1 – Показать дерево
2 – Вставка ключа
3 – Удаление ключа
4 – Поиск ключа
5 – Завершение
Выберите действие: 5
Завершение программы
Program ended with exit code: 0
```

Рис. 4 – Пример работы 4

Заключение

Цель достигнута, описана структура данных АВЛ-дерева. В результате выполнения лабораторной работы были выполнены все задачи.

1. Описаны алгоритмы добавления и удаления элемента, поиска элемента в дереве, балансировки дерева.
2. Разработана программа, предоставляющая пользователю выбор пункта меню, отображающая меню в цикле с постусловием, реализующая все описанные алгоритмы.
3. Приведены примеры работы программы.

Список литературы

1. Матвеева. Т.К., Балансировка при включении в AVL дерево: Методическая разработка 2013. – 13 с.