AVL-деревья: Самобалансирующиеся бинарные деревья поиска

AVL-дерево — это сбалансированное бинарное дерево поиска. Названо по фамилиям изобретателей.

Использует повороты для поддержания высоты и оптимальной производительности.

```
class Node {
public:
    int data, height;
    Node *left, *right;

    Node(int value) {
        data = value;
        left = right = nullptr;
        height = 1;
    }
};

int getHeight(Node *root);

int getBalance(Node *root);       You, 2 weeks ago • ad

Node* rightRotation(Node *root);

Node* leftRotation(Node *root);

bool search(Node *root, int key);

Node* insert(Node *root, int key);

Node* deleteNode(Node *root, int key);

Node* update(Node *root, int oldValue, int newValue );

void preOrder(Node *root);
void inOrder(Node *root);
void postOrder(Node *root);
```

Общая структура AVL-дерева

Элементы ноды

Ключ, левый и правый потомок, фактор баланса (высота).

Фактор баланса

Разница высот левого и правого поддерева: -1, 0 или 1.

Поддержание структуры

Балансировка предотвращает рост высоты сверх логарифмического уровня.

Повороты для балансировки

Одиночные повороты

- Левый поворот (LL) при разнице >= 2 в одном направлении
- Правый поворот (RR) при разнице >= 2 в одном направлении

Двойные повороты

- Лево-правый (LR) при сменяющимся направлении
- Право-левый (RL) при сменяющимся направлении

```
Node* leftRotation(Node *root) {
    Node *child = root->right;
    Node *childLeft = child->left;

    child->left = root;
    root->right = childLeft;

    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
    child->height = 1 + max(getHeight(child->left), getHeight(child->right));

    return child;
}
```

```
Node* rightRotation(Node *root) {
    Node *child = root->left;
    Node *childRight = child->right;

    child->right = root;
    root->left = childRight;

    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
    child->height = 1 + max(getHeight(child->left), getHeight(child->right));

    return child;
}
```

Вставка в AVL-дерево

Шаг 1

Вставка как в обычном бинарном дереве поиска.

Шаг 2

Обновление факторов баланса после вставки.

Шаг 3

Выполнение необходимых поворотов для балансировки.

```
Node* insert(Node *root, int key) {
    if (!root) return new Node(key);

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

    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
```

```
int balance = getBalance(root);

if (balance > 1 && key < root->left->data){
    return rightRotation(root);
}
else if (balance < -1 && root->right->data < key) {
    return leftRotation(root);
}
else if (balance > 1 && key > root->left->data) {
    root->left = leftRotation(root->left);
    return rightRotation(root);
}
else if (balance < -1 && root->right->data > key) {
    root->right = rightRotation(root->right);
    return leftRotation(root);
}
else {
    return root;
}
```

Удаление из AVL-дерева

Шаг 1

Удаление аналогично обычному бинарному дереву поиска.

Шаг 2

Обновление факторов баланса после удаления.

Шаг 3

Выполнение необходимых поворотов для восстановления баланса.

```
    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
    int balance = getBalance(root);
    if (balance > 1) {
        if (getBalance(root->left) >= 0) {
            return rightRotation(root);
        } else {
            root->left = leftRotation(root->left);
            return rightRotation(root);
        }
    }
    else if (balance < -1) {
        if (getBalance(root->right) <= 0) {
            return leftRotation(root);
        } else {
            root->right = rightRotation(root->right);
            return leftRotation(root);
        }
    }
    else {return root;}
}
```

```
Node* deleteNode(Node *root, int key) {
    if (!root) return NULL;

    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 && !root->right) {
            delete root;
            return NULL;
        }
        else if (root->left && !root->right) {
            Node *temp = root->left;
            delete root;
            return temp;
        }
        else if (!root->left && root->right) {
            Node *temp = root->right;
            delete root;
            return temp;
        }
        else {
            Node *current = root->right;
            while (current->left) {
                current=current->left;
            }
            root->data = current->data;
            root->right = deleteNode(root->right, current->data);
        }
    }
```

Преимущества AVL-деревьев

Самобалансировка

Обеспечивает O(log n) для поиска, вставки и удаления.

Высокая производительность

Быстрее, чем у обычных бинарных деревьев поиска.

Предсказуемость

Стабильная производительность даже в худшем случае.

Недостатки AVL-деревьев

Сложность реализации

Требует более сложного программирования, чем простые деревья.

Накладные расходы

Поддержка факторов баланса увеличивает затраты по времени и памяти.

Ресурсоемкие повороты

Иногда повороты способно вызывать задержки при обновлении баланса.

Применение AVL-деревьев

Базы данных

Эффективные индексы для быстрого доступа к данным.

Компиляторы

Поддержка таблиц символов и быстрая навигация.

Другие задачи

Где необходимы быстрые операции вставки, поиска и удаления.