АВЛ-деревья

Разработчики:
Бодун Алёна
Буссукайа Дарина
Матвиенко Варвара
Новикова Елизавета
Новикова София

Описание структуры

АВЛ-дерево (англ. AVL-Tree) — сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ-деревья названы по первым буквам фамилий их изобретателей, **Г. М. Адельсона-Вельского и Е. М. Ландиса**, которые впервые предложили использовать АВЛ-деревья в 1962 году.

```
typedef struct node {
    int key;
    int height; //высота поддерева
    struct node *left;
    struct node *right;
    struct node *parent;
} avl_node;
```

Поиск

```
avl_node* poisk(avl_node* tree, int key) { //поиск элемента
    if (tree == NULL) return NULL;

    if (key < tree->key) {
        return poisk(tree->left, key);
    } else if (key > tree->key) {
        return poisk(tree->right, key);
    } else {
        return tree;
    }
}
```

1.0 Формирование дерева - подготовительные функции

```
int max(int a, int b) {
    return (a > b) ? a : b;
}

int get_height(avl_node *node) { //высота узла
    if (node == NULL) return 0;
    return node->height;
}

void change_height(avl_node *tree) { //изменение высоты
    if (tree == NULL) return;
    tree->height = 1 + max(get_height(tree->left), get_height(tree->right));
}

int difference(avl_node *tree) { //разница между правым и левым поддеревьями
    if (tree == NULL) return 0;
    return get_height(tree->right) - get_height(tree->left);
}
```

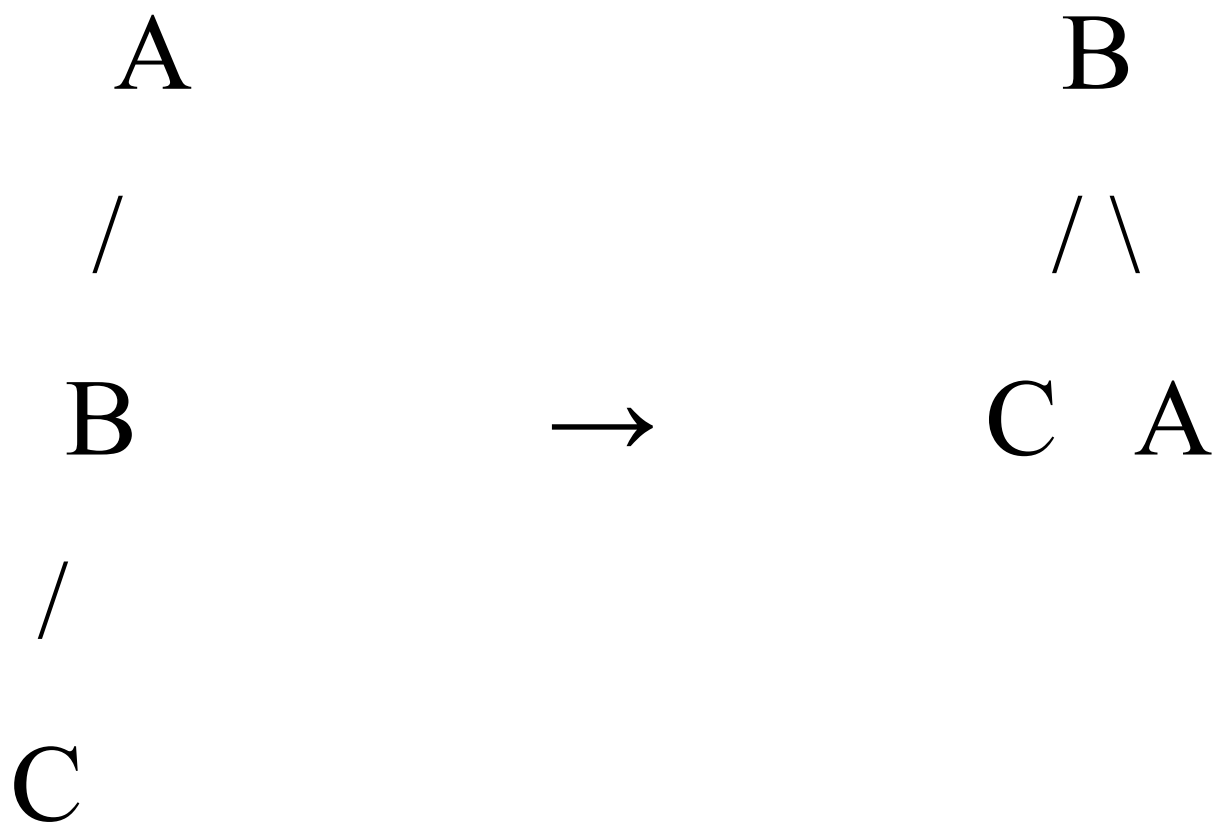
1.1 Формирование дерева - вставка

```
avl_node* avl_add(avl_node* tree, int key) { //добавление элемента
    if (tree == NULL) { //если это первый элемент
        avl_node* new_node = (avl_node*)malloc(sizeof(avl_node));
        new_node->key = key;
        new_node->height = 1;
        new_node->left = NULL;
        new_node->right = NULL;
        return new_node;
    }

    if (key == tree->key) { //если элемент повторяется
        printf("Элемент %d уже существует в дереве\n", key);
        return tree;
    }

    if (key < tree->key) {
        tree->left = avl_add(tree->left, key);
    } else {
        tree->right = avl_add(tree->right, key);
    }

    change_height(tree);
    balance(&tree);

    return tree;
}
```

Формирование дерева - балансировка
Правое вращение

```
// Функции балансировки
void balance_right (avl_node ** tree){ //поворот вправо для исправления LL случая
    avl_node * pointer = (*tree)->left;
    (*tree)->left = pointer->right;
    pointer->right = (*tree);
    (*tree) = pointer;
    change_height((*tree)->right); //корректировка данных о высоте нового и старого корня
    change_height(*tree);
}
```

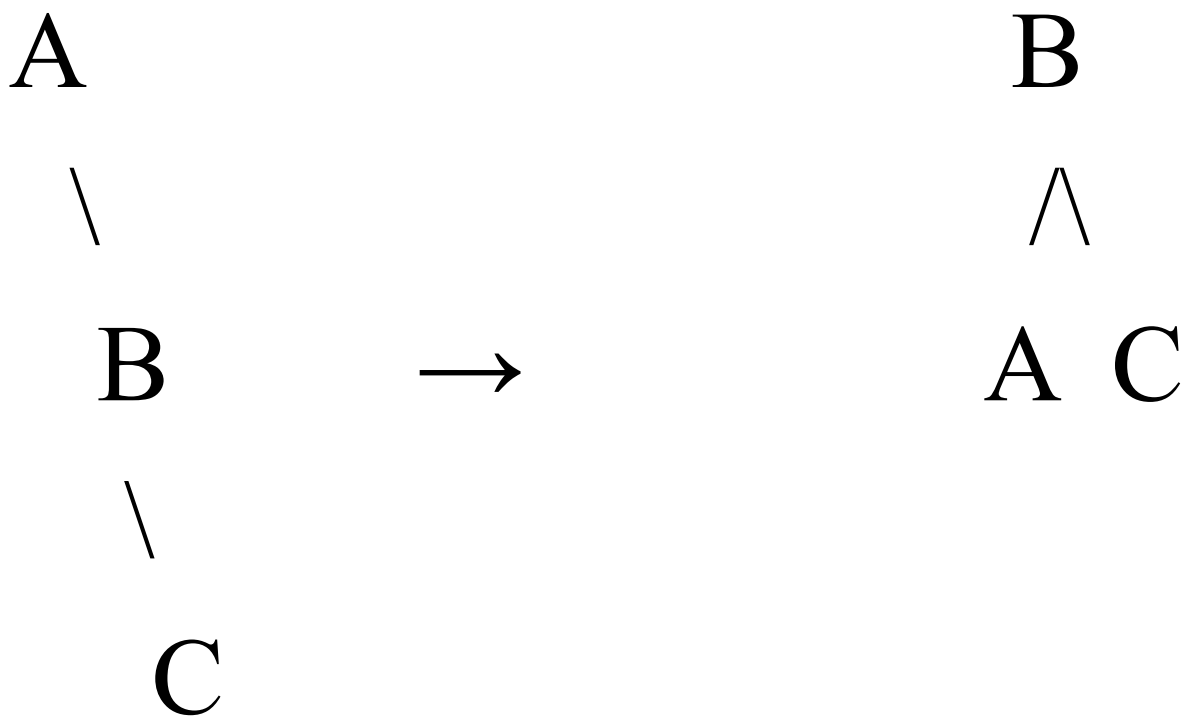
Пример



Формирование дерева - балансировка
Левое вращение

```
void balance_left (avl_node ** tree){ //поворот влево для исправления RR случая
    avl_node * pointer = (*tree)->right;
    (*tree)->right = pointer->left;
    pointer->left = (*tree);
    (*tree) = pointer;
    change_height((*tree)->left); //корректировка данных о высоте нового и старого корня
    change_height(*tree);
}
```

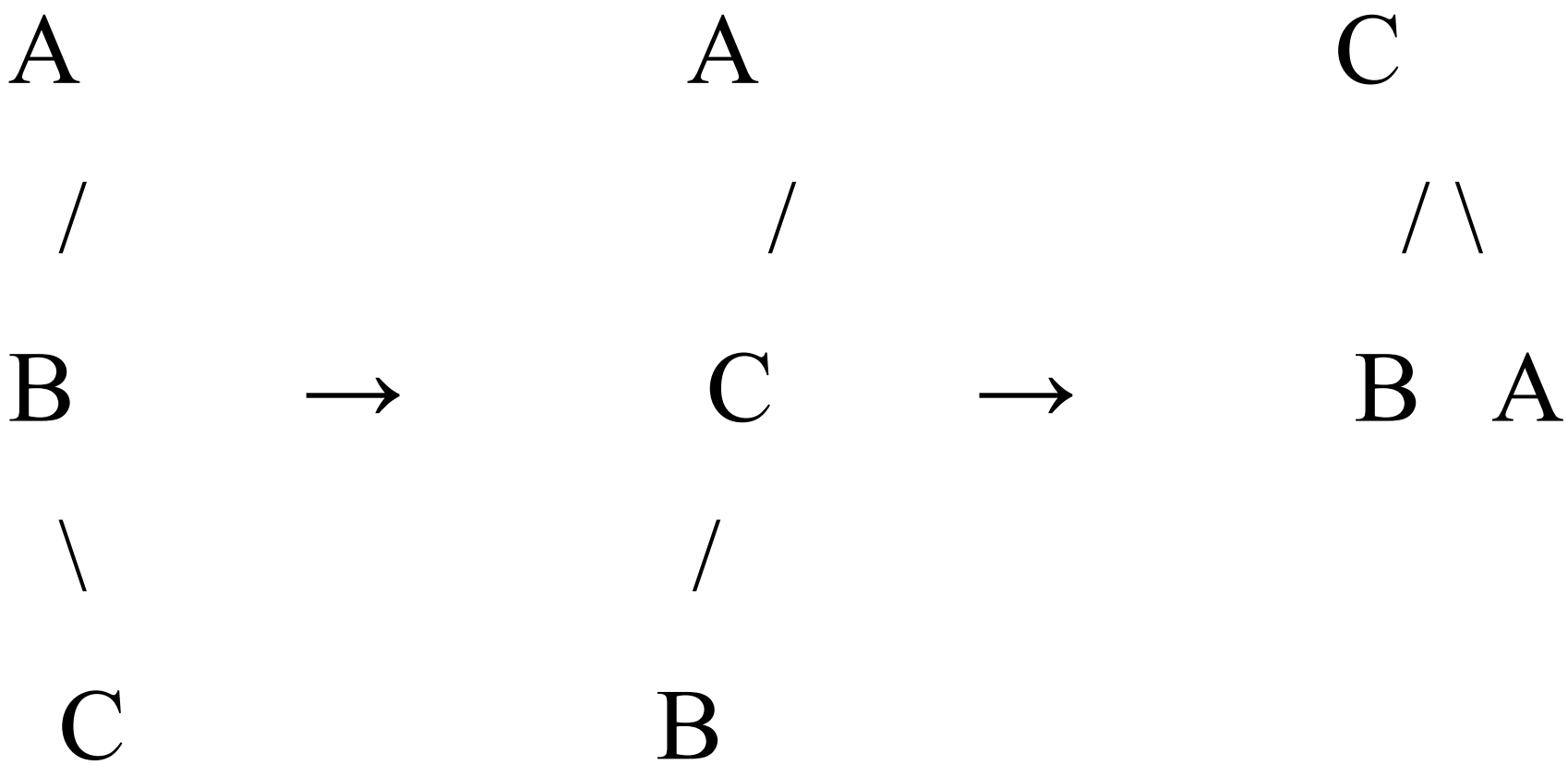
Пример:



Формирование дерева - балансировка Лево-правое вращение

```
void balance_lr (avl_node ** tree){ //комбинация поворотов для
исправления LR случая
    balance_left(&(*tree)->left);
    balance_right(tree);
}
```

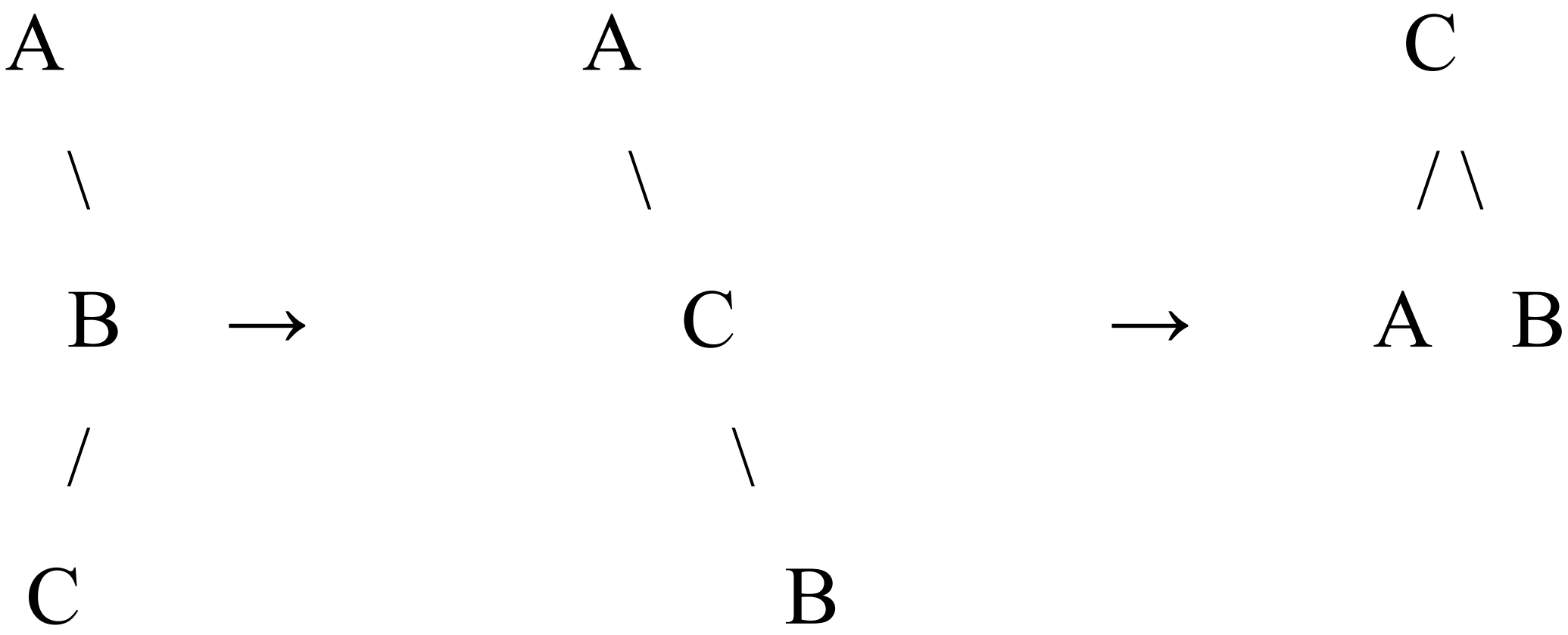
Пример:



Формирование дерева - балансировка
Право-левое вращение

```
void balance_rl (avl_node ** tree){ //комбинация поворотов для исправления RL случая
    balance_right(&(*tree)->right);
    balance_left(tree);
}
```

Пример:



Формирование дерева - балансировка

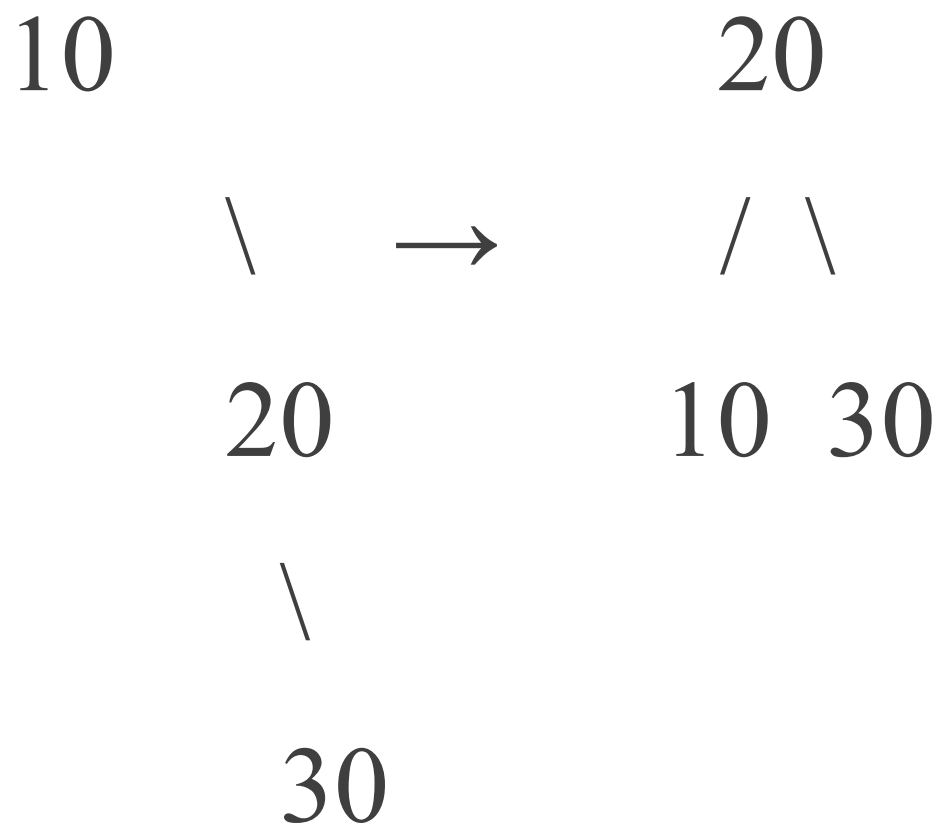
```
void balance(avl_node **tree){ //общая рекурсивная балансировка
    if (!(*tree)) return;
    balance(&(*tree)->right);
    balance(&(*tree)->left);
    change_height(*tree);
    if (difference(*tree) == -2){ //если слева длиннее
        if (difference((*tree)->left)<=0){
            balance_right(tree);
        }
        else{
            balance_lr(tree);
        }
    }
    else if (difference(*tree) == 2){ //если справа длиннее
        if (difference((*tree)->right)>=0){
            balance_left(tree);
        }
        else{
            balance_rl(tree);
        }
    }
}
```

1.2 Формирование дерева - удаление

```
avl_node* find_min(avl_node* tree) {
    while (tree->left != NULL) {
        tree = tree->left;
    }
    return tree;
}
```

```
avl_node* avl_delete(avl_node* tree, int key) {
    if (tree == NULL) {
        printf("Элемент %d не найден в дереве\n", key);
        return NULL;
    }
    if (key < tree->key) {
        tree->left = avl_delete(tree->left, key);
    } else if (key > tree->key) {
        tree->right = avl_delete(tree->right, key);
    } else {
        if ((tree->left == NULL) || (tree->right == NULL)) {
            avl_node* temp = tree->left ? tree->left : tree->right;

            if (temp == NULL) {
                temp = tree;
                tree = NULL;
            } else {
                *tree = *temp;
            }

            free(temp);
        } else {
            avl_node* temp = find_min(tree->right);
            tree->key = temp->key;
            tree->right = avl_delete(tree->right, temp->key);
        }
    }
    if (tree == NULL) {
        return tree;
    }
    change_height(tree);
    balance(&tree);

    return tree;
}
```

Пример вставки



Сложность операций

- **Поиск**
- **Вставка**
- **Удаление**

O(log n)

Преимущества и недостатки

Преимущества:

- **Быстрые операции**
- **Строгая балансировка**
- **Предсказуемость производительности**

Недостатки:

- **Сложность реализации**
- **Больше вращений**
- **Больше памяти**

Применение

- **Базы данных**
- **Компиляторы и интерпретаторы**
- **Системы реального времени**
- **Кэш-системы**

Пример использования - онлайн игры

Рейтинговая система должна поддерживать:

1. Добавление нового игрока (добавление нового элемента).

2. Обновление рейтинга после игры (удаление старого значения и вставка нового).

3. Поиск игрока по ID.

4. Определение ранга игрока (позиции в таблице лидеров).

5. Вывод топ-N игрока (поиск элемента, а потом его печать).

Структура AVL-дерева для рейтингов

```
typedef struct Player {
    int player_id;
    int rating;
    int height;
    struct AVLNode* left;
    struct AVLNode* right;
} Player;
```

Git-Hub

https://github.com/Fafichka/AVL_tree.git

Спасибо за внимание!