Министерство образования Республики Беларусь
Учреждение образования «Полоцкий государственный университет»

Кафедра технологий программирования

Алгоритмы и структуры данных

Отчет по лабораторной работе №6

Вариант 11

Выполнил — Ланцев Евгений Николаевич. 21-ИТ-1, ФИТ

Проверил — преподаватель Виноградова А.Д.

Полоцк
2022 г.

Лабораторная работа № 6

“Деревья. Сбалансированные по высоте деревья (АВЛ-деревья). 2-3 деревья. Б-деревья. Красно-черные деревья. Практическое применение.”

Цель работы: ознакомиться с понятиями «Деревья», «АВЛ-деревья», «Б-деревья», «Красно-черные деревья», изучить основные алгоритмы их обработки, научиться применять полученные знания на практике.

Вариант 11

```
jejikeh, 3 weeks ago | 1 author (jejikeh)
struct node {
    int data;
    unsigned char height;
    node* left;
    node* right;
    node(int n_data) {
        data = n_data;
        left = right = 0;
        height = 1;
    }
};
```

```
});

jejikeh, 3 weeks ago | 1 author (jejikeh)
class Node {
    constructor(data) {
        this.data = data;
        this.height = 1;
        this.left = 0;
        this.right = 0;
    }
}
```

Рисунок 1 - Класс узла бинарного дерева

```
// Возвращает height узла если он существует
char height(node *p_node){
    return p_node ? p_node->height : 0;
}

// Возвращает фактор баланса между левым и правым
int balance_factor(node *p_node){
    return height(p_node->right) - height(p_node->left);
}

// Исправляет высоту
void fix_height(node *p_node){
    unsigned char height_left = height(p_node->left);
    unsigned char height_right = height(p_node->right);
    p_node->height = (height_left > height_right ? heigh
}

// Меняе узлы местами -> малый поворот
node* rotate_right(node *p_node){
    node *temp_node = p_node->left;
    p_node->left = temp_node->right;
    temp_node->right = p_node;
    fix_height(p_node);
    fix_height(temp_node);
    return temp_node;
}
```

```
}

function Height(node) {
    if (node) return node.height;
    else return 0;
}

function BF(node) {
    return Height(node.right) - Height(node.left);
}

function OverHeight(node) {
    height_left = Height(node.left);
    height_right = Height(node.right);
    node.height = height_left > height_right ? height_le
}

function RightRotation(node) {
    y = node.left;
    y.left = x.right;
    y.right = node;
    OverHeight(node);
    OverHeight(y);
    return y;
}

function LeftRotation(node) {
```

```
// Меняя узлы местами → малый поворот
node* rotate_left(node *p_node){
    node *temp_node = p_node→right;
    p_node→right = temp_node→left;
    temp_node→left = p_node;
    fix_height(p_node);
    fix_height(temp_node);
    return temp_node;
}


// Большие повороты
node* balance(node *p_node){
    fix_height(p_node);
    if(balance_factor(p_node) == 2){
        if ( balance_factor(p_node→right) < 0){
            p_node→right = rotate_right(p_node→right);
        }
        return rotate_left(p_node);
    }
    if(balance_factor(p_node) == -2 ){
        if(balance_factor(p_node→left) > 0){
            p_node→left = rotate_left(p_node→left);
        }
        return rotate_right(p_node);
    }
    return p_node;
}
```

```
function LeftRotation(node) {
    y = node.right;
    y.right = x.left;
    y.left = node;
    OverHeight(node);
    OverHeight(y);
    return y;
}

function Balance(node) {
    OverHeight(node);
    if (BF(node) == 2) {
        if (BF(node.right) < 0) {
            node.right = RightRotation(node.right);
        }
        return LeftRotation(node);
    }
    if (BF(node) == -2) {
        if (BF(node.left) > 0) {
            node.left = LeftRotation(node.left);
        }
        return RightRotation(node);
    }
    return node;
}

function Insert(node, data) {
```

```
      // если два, то вызов этой же функции для элемента
      rl.question(
        `Выберете к какому элементу добавить узел, к ${currentNode.children[0].data}
        (index) => {
          if (index < 2) {
            this.add(data, currentNode.children[index]);
          } else {
            tasks();
          }
        }
      );
    }
  }
  display(currentNode) {
    console.log("-------");
    console.log("Корень : " + currentNode.data);
    for (let i = 0; i < currentNode.children.length; i++) {
      console.log("Ребенок : " + currentNode.children[i].data);
    }
    for (let i = 0; i < currentNode.children.length; i++) {
      this.display(currentNode.children[i]);
    }
  }
```

```cpp
// Добавление узла
node* insert(node *p_node,int data){
    if(!p_node) return new node(data);
    if(data < p_node->data){
        p_node->left = insert(p_node->left,data);
    }else {
        p_node->right = insert(p_node->right,data);
    }
    return balance(p_node);
}

void in_order(node *p_node){
    if(p_node){
        in_order(p_node->left);
        std::cout << p_node->data << "\t";
        in_order(p_node->right);
    }
}
void in_preorder(node *p_node){
    if(p_node){
        std::cout << p_node->data << "\t";
        in_preorder(p_node->left);
        in_preorder(p_node->right);
    }
}

void in_postorder(node *p_node){
    if(p_node){
```

```cpp
void in_postorder(node *p_node){
    if(p_node){
        in_postorder(p_node->left);
        in_postorder(p_node->right);
        std::cout << p_node->data << "\t";
    }
}

node* findmin(node* p) // поиск узла с минимальным ключо
{
    return p->left?findmin(p->left):p;
}

node* removemin(node* p) // удаление узла с минимальным
{
    if( p->left==0 )
        return p->right;
    p->left = removemin(p->left);
    return balance(p);
}


// Идем влево до упора что-бы найти минимальное
node* find_min(node* p_node){
    if(!p_node){
        std::cout << "No node";
        return(p_node);
    }else {
        while(p_node->left != NULL){
```

```js
function Insert(node, data) {
    if (!node) return new Node(data);
    if (data < node.data) {
        node.left = Insert(node.left, data);
    } else {
        node.right = Insert(node.right, data);
    }
    return Balance(node);
}

function SearchMin(node) {
    return node.left ? SearchMin(node.left) : node;
}

function InOrder(node) {
    if (node) {
        InOrder(node.left);
        console.log(node.data);
        InOrder(node.right);
    }
}

function InPreOrder(node) {
    if (node) {
        console.log(node.data);
        InPreOrder(node.right);
        InPreOrder(node.left);
    }
```

```js
function InPostOrder(node) {
    if (node.data) {
        InPostOrder(node.left);
        InPostOrder(node.right);
        console.log(node.data);
    }
}

function DeleteMin(node) {
    if (node.left == 0) return node.right;
    node.left = DeleteMin(node.left);
    return Balance(node);       jejikeh, 3 weeks ago •
}

function Delete(node, data) {
    if (!node) return 0;
    if (data < node.data) {
        node.left = Delete(node.left, data);
    } else if (data > node.data) {
        node.right = Delete(node.right, data);
    } else {
        y = node.left;
        z = node.right;
        delete node;
        if (!z) return y;
        min = SearchMin(z);
        min.right = DeleteMin(z);
        min.left = y;
        return Balance(min);
```

```
node* find_max(node* p_node){
    if(!p_node){
        std::cout << "No node";
        return(p_node);
    }else {
        while(p_node->right != NULL){
            p_node = p_node->right;
        }
        return p_node;
    }
}

node* remove_node(node* p_node, int data){
    if(!p_node) return 0;
    if(data < p_node->data){
        p_node->left = remove_node(p_node->left,data);
    }else if( data > p_node->data){
        p_node->right = remove_node(p_node->right,data);
    }else {
        node* q = p_node->left;
        node* r = p_node->right;
        delete p_node;
        if(!r) return q;
        node* min = findmin(r);
        min->right = removemin(r);
        min->left = q;
        return balance(min);
    }
    return balance(p_node);
```

Рисунок 2 - Базовые методы для работы и балансировка бинарного дерева

```
4-Find min and max
Task -> 1

Add node -> 24
1-Add node
2-Delete node
3-Print list
4-Find min and max
Task -> 3

IN_ORDER ->
3
IN_POSTORDER ->
3
IN_PREORDER ->
3
1-Add node
2-Delete node
3-Print list
4-Find min and max
Task -> 5
1-Add node
2-Delete node
3-Print list
4-Find min and max
Task -> 4
MIN -> 3
MAX -> 3
1-Add node
2-Delete node
3-Print list
4-Find min and max
Task -> 
```

Рисунок 3 - Результат работы программы