МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Отчёт по лабораторной работе № 6 по курсу «Алгоритмы и структуры данных»**

«Деревья. Сбалансированные по высоте деревья (АВЛ-деревья). 2-3

деревья. Б-деревья. Красно-черные деревья. Практическое применение.»

ВЫПОЛНИЛ студент группы 21-ИТ-1

Макеёнок Д.И.

ПРОВЕРИЛ преподаватель

Виноградова А.Д.

Полоцк, 2022 г.

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

**Теоретические сведения:**

1. Определение понятия АВЛ-дерево.

АВЛ-дерево – сбалансированное двоичное дерево поиска с k=1. Для его узлов определен коэффициент сбалансированности (balance factor).

1. Как осуществляется операция балансировки дерева?

Она осуществляется путем вращения (поворота) узлов – изменения связей в поддереве. Вращения не меняют свойств бинарного дерева поиска, и выполняются за константное время.

1. Типы вращений при балансировке.
   1. малое правое вращение;
   2. большое правое вращение;
   3. малое левое вращение;
   4. большое левое вращение;
2. Определение понятия 2-3 дерево.

2-3-дерево — структура данных, являющаяся B-деревом Степени 1, страницы которого могут содержать только 2-вершины (вершины с одним полем и 2 детьми) и 3-вершины (вершины с 2 полями и 3 детьми).

1. Свойства 2-3 дерева.

* Все нелистовые вершины содержат одно поле и 2 поддерева или 2 поля и 3 поддерева.
* Все листовые вершины находятся на одном уровне (на нижнем уровне) и содержат 1 или 2 поля.
* Все данные отсортированы (по принципу двоичного дерева поиска).
* Нелистовые вершины содержат одно или два поля, указывающие на диапазон значений в их поддеревьях.

1. Опишите алгоритм вставки в 2-3дерево элемента с ключом.
   1. Если дерево пусто, то создать новую вершину, вставить ключ и вернуть в качестве корня эту вершину, иначе
   2. Если вершина является листом, то вставляем ключ в эту вершину и если получили 3 ключа в вершине, то разделяем её, иначе
   3. Сравниваем ключ key с первым ключом в вершине, и если key меньше данного ключа, то идем в первое поддерево и переходим к пункту b, иначе
   4. Смотрим, если вершина содержит только 1 ключ (является 2вершиной), то идем в правое поддерево и переходим к пункту b, иначе
   5. Сравниваем ключ key со вторым ключом в вершине, и если key меньше второго ключа, то идем в среднее поддерево и переходим к пункту b, иначе
   6. Идем в правое поддерево и переходим к пункту b. Для примера вставим keys = {1, 2, 3, 4, 5, 6, 7}: При вставке key=1 мы имеем пустое дерево, а после получаем единственную вершину с единственным ключа key=1
2. Определение понятия Б-дерево.

B-дерево (читается как Би-дерево) — это особый тип сбалансированного дерева поиска, в котором каждый узел может содержать более одного ключа и иметь более двух дочерних элементов. Из-за этого свойства B-дерево называют сильно ветвящимся.

1. Где используются Б-деревья?

* В базах данных и файловых системах.
* Для хранения блоков данных (вторичные носители).
* Для многоуровневой индексации.

1. Условия для балансировки красно-черных деревьев.

* корень дерева окрашен в черный цвет;
* у красной вершины дети черные (если они есть);
* всякий путь от корня дерева к внешней вершине (листу) содержит одно и то же число черных вершин.

**Код программы:**

#include <iostream>

#include <Windows.h>

// стркуктура узла

struct node {

int data;

unsigned char height;

node\* left;

node\* right;

node(int n\_data) {

data = n\_data;

left = right = 0;

height = 1;

}

};

// 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 ? height\_left : height\_right) + 1;

}

// 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;

}

// Меняе 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;

}

// 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) {

in\_postorder(p\_node->left);

in\_postorder(p\_node->right);

std::cout << p\_node->data << "\t";

}

}

node\* findmin(node\* p) // поиск узла с минимальным ключом в дереве p

{

return p->left ? findmin(p->left) : p;

}

node\* removemin(node\* p) // удаление узла с минимальным ключом из дерева p

{

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 << "Нет узла";

return(p\_node);

}

else {

while (p\_node->left != NULL) {

p\_node = p\_node->left;

}

return p\_node;

}

}

// Идем вправо до упора что-бы найти максимальное

node\* find\_max(node\* p\_node) {

if (!p\_node) {

std::cout << "Нет узла";

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);

}

int main() {

// menu

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int task = 0;

int x = 0;

std::cout << "Введите 1 узел: ";

std::cin >> x;

node\* root = new node(x);

while (task != -1) {

std::cout << "1-Добавить узел\n2-Удалить узел\n3-Вывести\n4-Найти мин. и макс.\n";

std::cin >> task;

switch (task)

{

case 1:

std::cout << "\nДобавляемый узел -> ";

std::cin >> x;

insert(root, x);

break;

case 2:

std::cin >> x;

remove\_node(root, x);

std::cout << "\n";

break;

case 3:

std::cout << "\nIN\_ORDER ->\n";

in\_order(root);

std::cout << "\nIN\_POSTORDER ->\n";

in\_postorder(root);

std::cout << "\nIN\_PREORDER ->\n";

in\_preorder(root);

std::cout << "\n";

break;

case 4:

std::cout << "MIN -> " << find\_min(root)->data;

std::cout << "\n";

std::cout << "MAX -> " << find\_max(root)->data;

std::cout << "\n";

break;

default:

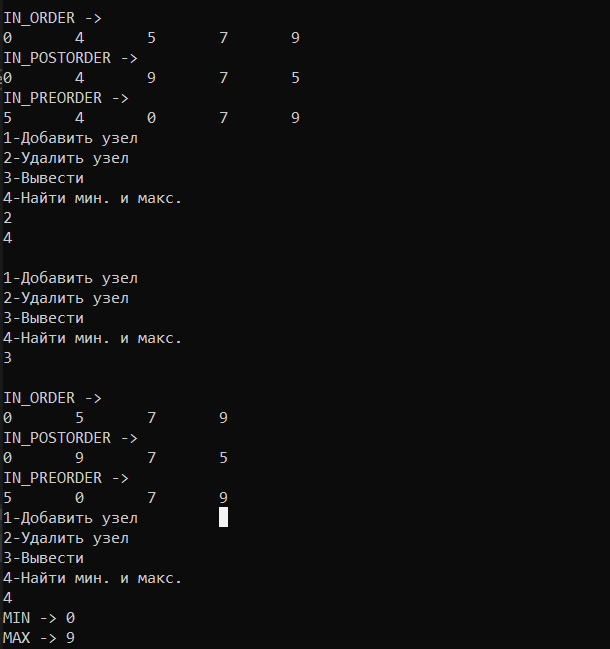
break;

}

}

}

**Результат работы программы:**



**Вывод:** Я ознакомился с понятиями «Деревья», «АВЛ-деревья», «Б-деревья», «Красно-черные деревья», изучил основные алгоритмы их обработки, научился применять полученные знания на практике.