**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по курсовой работе №3**

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2373 |  | Чесноков М. А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Изучение свойств и организации сбалансированных деревьев; получение практических навыков в работе с АВЛ-деревьями; определение преимуществ и недостатков подобных структур данных; проведение сравнительной характеристики скорости вставки, удаления и поиска элемента в АВЛ-деревьях.

**Основные теоретические положения.**

Сбалансированное по высоте [двоичное дерево поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0): для каждой его вершины высота её двух поддеревьев различается не более чем на 1. АВЛ-дерево — это прежде всего двоичное дерево поиска, ключи которого удовлетворяют стандартному свойству: ключ любого узла дерева не меньше любого ключа в левом поддереве данного узла и не больше любого ключа в правом поддереве этого узла. Это значит, что для поиска нужного ключа в АВЛ-дереве можно использовать стандартный алгоритм. Для простоты дальнейшего изложения будем считать, что все ключи в дереве целочисленны и не повторяются. Особенностью АВЛ-дерева является то, что оно является сбалансированным в следующем смысле: для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу.

**Постановка задачи**

 Программа должна поддерживать весь описанный функционал:

1.   Формирование АВЛ-дерева из N элементов:

a) пользователь вводит количество элементов N АВЛ-дерева, которое автоматически заполняется случайными числами (–99 до 99);

б) пользователь вводит в консоль элементы массива, N определяется автоматически по количеству введенных элементов;

Определение скорости формирования АВЛ-дерева.

2. Вывод в консоль AVL\_tree АВЛ-дерева.

Изображение выглядит как Шрифт, рукописный текст, диаграмма, линия

Автоматически созданное описание

3.   Определение скорости вставки, удаления и получения элемента  дерева.

4.   Определение скорости проверки на сбалансированность.

5.   Генерация заданий к практической работе по АВЛ-деревьям. Необходимо сгенерировать задания в файл output\_task в количестве вариантов, которые введет пользователь. В файл output\_key необходимо вывести короткие ответы к заданиям. В файл output\_ans необходимо вывести развернутые ответы к заданиям. Должны быть представлены задания на создание АВЛ-дерева из массива значений, удаление и вставка элементов в АВЛ-дерево.

**Выполнение работы.**

Код программы представлен в приложении А.

**Блок описания кода и использованных алгоритмов:**

Создаем структуру дерева, я также его корня.

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

1. Создаем функцию добавления:
2. Запишем функцию удаления элемента:

Изображение выглядит как снимок экрана, программное обеспечение, текст, Мультимедийное программное обеспечение

Автоматически созданное описание

1. Теперь функцию для получения элемента:

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

1. Генерация заданий:

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

**Выводы.**

Мы научились организации сбалансированных деревьев; получение практических навыков в работе с АВЛ-деревьями; определение преимуществ и недостатков подобных структур данных; проведение сравнительной характеристики скорости вставки, удаления и поиска элемента в АВЛ-деревьях.

**Блок скриншотов работы программы**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Приложение А

рабочий код

#include <iostream>

#include <string>

#include <chrono>

#include <fstream>

#include <Windows.h>

using namespace std;

using namespace chrono;

struct Tree {

int data;

int balance;

Tree\* left, \* right;

};

struct Trunk

{

Trunk\* prev;

string str;

Trunk(Trunk\* prev, string str)

{

this->prev = prev;

this->str = str;

}

};

struct Timer {

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

duration<\_\_int64, ratio<1, 1000000000>> duration;

} timer;

int Height(Tree\* (&root)) {

if (root == NULL) return 0;

int hLeft = Height(root->left);

int hRight = Height(root->right);

if (hLeft > hRight) return hLeft + 1;

return hRight + 1;

}

void SetBalance(Tree\* (&root)) {

if (root != NULL) {

root->balance = Height(root->right) - Height(root->left);

}

}

void LeftRotate(Tree\* (&root)) {

Tree\* rightSubTree, \* rightLeftSubTree;

rightSubTree = root->right;

rightLeftSubTree = rightSubTree->left;

rightSubTree->left = root;

root->right = rightLeftSubTree;

root = rightSubTree;

}

void RightRotate(Tree\* (&root)) {

Tree\* leftSubTree, \* leftRightSubTree;

leftSubTree = root->left;

leftRightSubTree = leftSubTree->right;

leftSubTree->right = root;

root->left = leftRightSubTree;

root = leftSubTree;

}

void BigLeftRotate(Tree\* (&root)) {

RightRotate(root->right);

LeftRotate(root);

}

void BigRightRotate(Tree\* (&root)) {

LeftRotate(root->left);

RightRotate(root);

}

void Balancing(Tree\* (&root)) {

if (root == NULL) return;

if (Height(root->left) - Height(root->right) > 1) {

if (Height(root->left->left) < Height(root->left->right)) BigRightRotate(root);

else RightRotate(root);

}

else if (Height(root->right) - Height(root->left) > 1) {

if (Height(root->right->right) < Height(root->right->left)) BigLeftRotate(root);

else LeftRotate(root);

}

}

void Insert(Tree\* (&root), int data) {

if (root == NULL) {

root = new Tree;

root->data = data;

root->left = root->right = NULL;

}

else {

if (root->data == data) return;

if (data < root->data) Insert(root->left, data);

else if (data > root->data) Insert(root->right, data);

}

Balancing(root);

SetBalance(root);

}

void CreateRandTree(Tree\* (&tree), int amount) {

int data;

for (int i = 0; i < amount; i++) {

data = 99 - (rand() % 199);

Insert(tree, data);

}

}

void CreateInputTree(Tree\* (&tree)) {

string inp;

int data;

while (true) {

cout << "Введите элемент дерева (000 - закончить ввод): ";

cin >> inp;

if (inp == "000") break;

data = stoi(inp);

Insert(tree, data);

}

}

void ShowTrunks(Trunk\* (&trunk)) {

if (trunk == NULL) return;

ShowTrunks(trunk->prev);

cout << trunk->str;

}

void ShowTree(Tree\* (&root), Trunk\* prev, bool isRight) {

if (root == NULL) return;

string prev\_str = " ";

Trunk\* trunk = new Trunk(prev, prev\_str);

ShowTree(root->right, trunk, true);

if (!prev) {

trunk->str = "--->";

}

else if (isRight) {

trunk->str = ".-->";

prev\_str = " |";

}

else {

trunk->str = "`-->";

prev->str = prev\_str;

}

ShowTrunks(trunk);

cout << root->data << endl;

if (prev) prev->str = prev\_str;

trunk->str = " |";

ShowTree(root->left, trunk, false);

}

bool CheckForRoot(Tree\* (&root), int data) {

if (root == NULL) return false;

if (data == root->data) return true;

if (data < root->data) return CheckForRoot(root->left, data);

else return CheckForRoot(root->right, data);

}

int GetMaxData(Tree\* (&root)) {

if (root->right == NULL) return root->data;

else return GetMaxData(root->right);

}

void DelRoot(Tree\* (&root), int data) {

if (data < root->data) DelRoot(root->left, data);

else if (data > root->data) DelRoot(root->right, data);

else {

if (root->left == NULL && root->right == NULL) {

delete root;

root = NULL;

}

else if (root->right == NULL) {

Tree\* delRoot = root;

root = root->left;

delete delRoot;

}

else if (root->left == NULL) {

Tree\* delRoot = root;

root = root->right;

delete delRoot;

}

else {

int newData = GetMaxData(root->left);

root->data = newData;

DelRoot(root->left, newData);

}

}

Balancing(root);

SetBalance(root);

}

void GetRoot(Tree\* (&root), int data) {

if (root == NULL) return;

if (data < root->data) GetRoot(root->left, data);

else if (data > root->data) GetRoot(root->right, data);

else {

timer.end = steady\_clock::now();

timer.duration = timer.end - timer.start;

cout << "Значение: " << root->data << "\tАдрес в памяти: " << root << endl;

}

}

void ClearTree(Tree\* (&root)) {

if (root != NULL) {

ClearTree(root->left);

ClearTree(root->right);

delete root;

root = NULL;

}

}

void TrunksToFile(Trunk\* (&trunk), ofstream& file) {

if (trunk == NULL) return;

TrunksToFile(trunk->prev, file);

file << trunk->str;

}

void TreeToFile(Tree\* (&root), Trunk\* prev, bool isRight, ofstream& file) {

if (root == NULL) return;

string prev\_str = " ";

Trunk\* trunk = new Trunk(prev, prev\_str);

TreeToFile(root->right, trunk, true, file);

if (!prev) {

trunk->str = "--->";

}

else if (isRight) {

trunk->str = ".-->";

prev\_str = " |";

}

else {

trunk->str = "`-->";

prev->str = prev\_str;

}

TrunksToFile(trunk, file);

file << root->data << endl;

if (prev) prev->str = prev\_str;

trunk->str = " |";

TreeToFile(root->left, trunk, false, file);

}

void CreateTasks(int tasks) {

ofstream file;

file.open("output\_task");

if (!file.is\_open()) cout << "Ошибка открытия файла\n";

else {

for (int i = 0; i < tasks; i++) {

file << (10 + (rand() % 41)) << " " << (99 - (rand() % 199)) << " " << (99 - (rand() % 199)) << "\n";

}

}

file.close();

}

void DoTasks(int tasks) {

ifstream fileTask;

ofstream fileKey;

ofstream fileAns;

fileTask.open("output\_task");

fileKey.open("output\_key");

fileAns.open("output\_ans");

if (!fileTask.is\_open() && !fileKey.is\_open() && !fileAns.is\_open()) cout << "Ошибка открытия файла. Задания не выполнены.\n";

else {

int amount, del, ins;

Tree\* tree = NULL;

SetConsoleCP(1251);

for (int i = 0; i < tasks; i++) {

fileTask >> amount >> del >> ins;

fileKey << "\tЗадание " << (i + 1) << endl;

fileAns << "\tЗадание " << (i + 1) << endl;

fileAns << "Создано дерево размером " << amount << ".\n";

CreateRandTree(tree, amount);

TreeToFile(tree, NULL, false, fileAns);

fileAns << "\nУдаление числа " << del << ".\n";

if (CheckForRoot(tree, del)) {

DelRoot(tree, del);

TreeToFile(tree, NULL, false, fileAns);

}

else fileAns << "Число для удаления не найдено.";

fileAns << "\nДобавление числа " << ins << ".\n";

if (!CheckForRoot(tree, ins)) {

Insert(tree, ins);

TreeToFile(tree, NULL, false, fileAns);

}

else fileAns << "Число для добавления уже есть.";

TreeToFile(tree, NULL, false, fileKey);

fileAns << "\n\n";

fileKey << "\n\n";

ClearTree(tree);

}

SetConsoleCP(866);

cout << "Задания успешно выполнены.\nКратуие ответы в файле output\_key, развёрнутые ответы в файле output\_ans.\n";

}

}

void Tasks() {

int tasks;

cout << "В каждом задании будет создано дерево размером от 10 до 50 элементов в диапазоне от -99 до 99.\n"

<< "Будет создано рандомное число, которое удалится из дерева, если будет в нём.\n"

<< "Будет создано рандомное число, которое добавится в дерево, если не будет в нём.\n"

<< "Введите, сколько заданий сгенерировать: ";

cin >> tasks;

CreateTasks(tasks);

DoTasks(tasks);

}

void main() {

srand(time(NULL));

setlocale(LC\_ALL, "rus");

Tree\* tree = NULL;

char menu, create;

int amount, data;

while (true) {

cout << "\tМеню\n"

<< "1 - Создать новое дерево (существующее дерево будет удалено).\n"

<< "2 - Вывести дерево.\n"

<< "3 - Вставить элемент в дерево.\n"

<< "4 - Удалить элемент дерева.\n"

<< "5 - Получить элемент дерева.\n"

<< "6 - Сгенерировать практические задания.\n"

<< "0 - Выйти из программы\n"

<< "Ваш выбор: ";

cin >> menu;

switch (menu) {

case '1':

system("cls");

if (tree) ClearTree(tree);

cout << "1 - Заполнить дерево случайными числами от -99 до 99.\n"

<< "2 - Ввести элементы дерева вручную.\n"

<< "Ваш выбор: ";

cin >> create;

switch (create) {

case '1':

cout << "Введите количество элементов дерева: ";

cin >> amount;

timer.start = steady\_clock::now();

CreateRandTree(tree, amount);

timer.end = steady\_clock::now();

timer.duration = timer.end - timer.start;

cout << "Создано дерево:\n";

ShowTree(tree, NULL, false);

cout << "Время выполнения: " << timer.duration.count() << " наносек.\n";

break;

case '2':

timer.start = steady\_clock::now();

CreateInputTree(tree);

timer.end = steady\_clock::now();

timer.duration = timer.end - timer.start;

cout << "Создано дерево:\n";

ShowTree(tree, NULL, false);

cout << "Время выполнения: " << timer.duration.count() << " наносек.\n";

break;

default:

cout << "Некорректный выбор. Вы будете возвращены в меню.\n";

}

system("pause");

system("cls");

break;

case '2':

system("cls");

if (tree) ShowTree(tree, NULL, false);

else cout << "Дерево не создано.\n";

system("pause");

system("cls");

break;

case '3':

system("cls");

if (tree) {

ShowTree(tree, NULL, false);

cout << "Введите элемент, который хотите добавить: ";

cin >> data;

if (!CheckForRoot(tree, data)) {

timer.start = steady\_clock::now();

Insert(tree, data);

timer.end = steady\_clock::now();

timer.duration = timer.end - timer.start;

cout << "Элемент добавлен.\n";

ShowTree(tree, NULL, false);

cout << "Время выполнения: " << timer.duration.count() << " наносек.\n";

}

else cout << "Данный элемент уже есть в дереве\n";

}

else cout << "Дерево не создано.\n";

system("pause");

system("cls");

break;

case '4':

system("cls");

if (tree) {

ShowTree(tree, NULL, false);

cout << "Введите элемент, который хотите удалить: ";

cin >> data;

if (CheckForRoot(tree, data)) {

timer.start = steady\_clock::now();

DelRoot(tree, data);

timer.end = steady\_clock::now();

timer.duration = timer.end - timer.start;

cout << "Элемент удалён.\n";

ShowTree(tree, NULL, false);

cout << "Время выполнения: " << timer.duration.count() << " наносек.\n";

}

else cout << "Данный элемент отсутствует в дереве.\n";

}

else cout << "Дерево не создано.\n";

system("pause");

system("cls");

break;

case '5':

system("cls");

if (tree) {

if (tree) {

ShowTree(tree, NULL, false);

cout << "Введите элемент, который хотите получить: ";

cin >> data;

if (CheckForRoot(tree, data)) {

timer.start = steady\_clock::now();

GetRoot(tree, data);

cout << "Время выполнения: " << timer.duration.count() << " наносек.\n";

}

else cout << "Данный элемент отсутствует в дереве.\n";

}

}

else cout << "Дерево не создано.\n";

system("pause");

system("cls");

break;

case '6':

system("cls");

Tasks();

system("pause");

system("cls");

break;

case '0':

return;

default:

cout << "Пункт выбран неправильно!\n";

system("pause");

system("cls");

break;

}

}

}