**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по курсовой работе №3**

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 2373 |  | Никитина М. Ю. |
| 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 <fstream>

#include <Windows.h>

#include <chrono>

#include <string>

using namespace std;

using namespace chrono;

struct Node { //создание узлов

int key;

int balance;

Node\* left;

Node\* right;

};

int Height(Node\* (&root))

{

if (root == NULL) {

return 0;

}

else {

unsigned char hleft = Height(root->left);

unsigned char hright = Height(root->right);

root->balance = (hleft > hright ? hleft : hright) + 1;

}

}

struct Timer {

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

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

} timer;

void SetBalance(Node\* (&root)) { //балансировка дерева

if (root != NULL) {

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

SetBalance(root->left);

SetBalance(root->right);

}

}

void TurnLeft(Node\* (&root)) { //малый поворот налево

Node\* RightSubTree, \* RightSubTreeLeftSubTree;

RightSubTree = root->right;

RightSubTreeLeftSubTree = RightSubTree->left;

RightSubTree->left = root;

root->right = RightSubTreeLeftSubTree;

root = RightSubTree;

SetBalance(root->left);

SetBalance(root);

}

void TurnRight(Node\* (&root)) { //малый поворот направо

Node\* LeftSubTree, \* LeftSubTreeRightSubTree;

LeftSubTree = root->left;

LeftSubTreeRightSubTree = LeftSubTree->right;

LeftSubTree->right = root;

root->left = LeftSubTreeRightSubTree;

root = LeftSubTree;

SetBalance(root->right);

SetBalance(root);

}

void Insert(Node\* (&root), int d) { //вставка ключа key в дерево с корнем root

if (root == NULL) {

root = new Node;

root->key = d;

root->balance = 0;

root->left = NULL;

root->right = NULL;

}

else {

if (d > root->key) {

Insert(root->right, d);

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

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

TurnRight(root->right);

}

TurnLeft(root);

}

}

else {

if (d < root->key) {

Insert(root->left, d);

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

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

TurnLeft(root->left);

}

TurnRight(root);

}

}

}

SetBalance(root);

}

}

struct Bole

{

Bole\* prev;

string str;

Bole(Bole\* prev, string str)

{

this->prev = prev;

this->str = str;

}

};

void OutputBole(Bole\* (&bole)) {

if (bole == NULL) return;

OutputBole(bole->prev);

cout << bole->str;

}

void Output(Node\* root, Bole\* prev, bool good) { //вывод дерева

if (root == NULL) return;

string prev\_str = " ";

Bole\* bole = new Bole(prev, prev\_str);

Output(root->right, bole, true);

if (!prev) {

bole->str = "--->";

}

else if (good) {

bole->str = ".-->";

prev\_str = " |";

}

else {

bole->str = "`-->";

prev->str = prev\_str;

}

OutputBole(bole);

cout << root->key << endl;

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

bole->str = " |";

Output(root->left, bole, false);

}

void Create1(Node\* (&root), int Amount) { //создание дерева 1-м способом

int count = 0, value;

while (count <= Amount) {

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

Insert(root, value);

count++;

}

}

void Create2(Node\* (&root)) { //создание дерева 2-м способом

string str;

int value;

cout << "Введите элементы дерева: (в конце поставьте 666)\n";

while (true) {

cin >> str;

if (str == "666") {

break;

}

value = stoi(str);

Insert(root, value);

}

}

void Clear(Node\* (&root)) { //очищение дерева

if (root != NULL) {

Clear(root->right);

Clear(root->left);

delete root;

root = NULL;

}

}

int MaxValueNode(Node\* (&origin)) {

if (origin->right == NULL) {

return origin->key;

}

else {

return MaxValueNode(origin->right);

}

}

void Delete(Node\* (&root), int value) {

if (value > root->key) {

Delete(root->right, value);

}

else if (value < root->key) {

Delete(root->left, value);

}

else {

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

int maxvalue = MaxValueNode(root->left);

root->key = maxvalue;

Delete(root->left, maxvalue);

}

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

Node\* Delete = root;

root = root->left;

delete Delete;

}

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

Node\* Delete = root;

root = root->right;

delete Delete;

}

else {

delete root;

root = NULL;

}

}

SetBalance(root);

}

void Search(Node\* (&root), int value) {

if (root == NULL) {

return;

}

if (value > root->key) {

Search(root->right, value);

}

else if (value < root->key) {

Search(root->left, value);

}

else {

cout << root->key << " - значение.\n";

}

}

void Tasks(int number) {

ofstream file;

file.open("output\_task");

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

else {

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

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

}

}

file.close();

}

bool Checking(Node\* (&origin), int value) {

if (origin == NULL) {

return false;

}

if (value == origin->key) {

return true;

}

if (value < origin->key) {

return Checking(origin->left, value);

}

else {

return Checking(origin->right, value);

}

}

void BoleInFile(Bole\* (&bole), ofstream& file) {

if (bole == NULL) {

return;

}

BoleInFile(bole->prev, file);

file << bole->str;

}

void NodeInFile(Node\* (&origin), Bole\* prev, bool good, ofstream& file) {

if (origin == NULL) {

return;

}

string prev\_str = " ";

Bole\* bole = new Bole(prev, prev\_str);

NodeInFile(origin->right, bole, true, file);

if (!prev) {

bole->str = "--->";

}

else if (good) {

bole->str = ".-->";

prev\_str = " |";

}

else {

bole->str = "`-->";

prev->str = prev\_str;

}

BoleInFile(bole, file);

file << origin->key << endl;

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

bole->str = " |";

NodeInFile(origin->left, bole, false, file);

}

void SolvedTasks(int number) {

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 value, del, ins;

Node\* root = NULL;

SetConsoleCP(1251);

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

fileTask >> value >> del >> ins;

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

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

Create1(root, value);

SetBalance(root);

NodeInFile(root, NULL, false, fileAns);

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

if (Checking(root, del)) {

Delete(root, del);

SetBalance(root);

NodeInFile(root, NULL, false, fileAns);

}

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

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

if (!Checking(root, ins)) {

Insert(root, ins);

SetBalance(root);

NodeInFile(root, NULL, false, fileAns);

}

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

NodeInFile(root, NULL, false, fileKey);

fileAns << "\n\n";

fileKey << "\n\n";

Clear(root);

}

SetConsoleCP(866);

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

}

}

void main() {

srand(time(NULL));

setlocale(0, "");

int choice, number;

int N;

Node\* root;

root = NULL;

while (true) {

cout << "Меню:\n"

<< "1 - Создать AVL-дерево\n"

<< "2 - Вывод AVL-дерева\n"

<< "3 - Вставить элемент в AVL-дерево\n"

<< "4 - Удалить элемент из AVL-дерева\n"

<< "5 - Вывести на консоль элемент AVL-дерева\n"

<< "6 - Вычисление времени проверки на сбалансированность\n"

<< "7 - Генерация заданий\n"

<< "0 - Выход\n";

cin >> choice;

switch (choice) {

case 0:

return;

case 1:

Clear(root);

int choice1;

system("cls");

cout << "Каким способом вы хотите создать дерево?\n"

<< "1 - Наполнить рандомными числами от -99 до 99\n"

<< "2 - Самому ввести числа\n";

cin >> choice1;

switch (choice1) {

case 1:

cout << "Введите число элементов дерева:\n";

cin >> N;

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

Create1(root, N);

SetBalance(root);

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

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

Output(root, NULL, false);

cout << "Время, затраченное на создание дерева: " << timer.duration.count() << " наносекунд.\n";

break;

case 2:

Create2(root);

SetBalance(root);

Output(root, NULL, false);

break;

}

break;

case 2:

system("cls");

Output(root, NULL, false);

break;

case 3:

system("cls");

Output(root, NULL, false);

cout << "Введите число, которое хотите добавить:\n";

cin >> number;

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

Insert(root, number);

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

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

Output(root, NULL, false);

cout << "Время, затраченное на вставку элемента: " << timer.duration.count() << " наносекунд.\n";

break;

case 4:

system("cls");

Output(root, NULL, false);

cout << "Введите число, которое хотите удалить:\n";

cin >> number;

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

Delete(root, number);

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

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

Output(root, NULL, false);

cout << "Время, затраченное на удаление элемента: " << timer.duration.count() << " наносекунд.\n";

break;

case 5:

system("cls");

Output(root, NULL, false);

cout << "Введите число, которое хотите найти:\n";

cin >> number;

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

Search(root, number);

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

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

Output(root, NULL, false);

cout << "Время, затраченное на поиск элемента: " << timer.duration.count() << " наносекунд.\n";

break;

case 6:

system("cls");

Output(root, NULL, false);

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

SetBalance(root);

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

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

cout << "Время проверки на сбалансированность: " << timer.duration.count() << " наносекунд.\n";

break;

case 7:

system("cls");

cout << "Введите кол-во заданий, которое хотите создать:\n";

cin >> number;

Tasks(number);

SolvedTasks(number);

break;

}

}

}