**Лабораторная робота №5**

Деревья. Бинарные деревья поиска

**Цель работы:** Изучение основных алгоритмов работы с деревьями, получение практических навыков разработки и использования этих структур и алгоритмов для решения задач. Приобретение навыков обработки бинарного дерева поиска.

*Задания*

Задание 1:

В заданном бинарном дереве найти первое вхождение заданного элемента и напечатать пройденные при поиске узлы дерева: при прямом обходе дерева;

Задание 2:

Из входной последовательности вещественных чисел построить бинарное дерево поиска. Вывести на экран значения в тех узлах, которые имеют уровень, заданный пользователем.

**Ход работы**

Задание 1:

1. Текст программы

#include <iostream>

#include <vector>

#include <string>

#include "Tree.hpp"

template <typename T>

T prompt(const char label[]);

void preOrderSearch(dg::Tree::Node\* node, int item, std::vector<int>& way);

void preOrder(dg::Tree::Node\* node, std::vector<int>& way);

void getTree(dg::Tree& tree);

void printArr(std::vector<int> arr);

void printHello();

int main() {

printHello();

dg::Tree tree;

getTree(tree);

std::vector<int> items;

preOrder(tree.getRoot(), items);

std::cout << "Items of the tree (pre-order):\n";

printArr(items);

std::cout << std::endl;

while (true) {

int num = prompt<int>("Enter the number to search: ");

std::vector<int> wayToNum;

preOrderSearch(tree.getRoot(), num, wayToNum);

printArr(wayToNum);

std::cout << "\n- - - - - -\n\n";

}

system("pause");

return 0;

}

void preOrderSearch(dg::Tree::Node\* node, int item, std::vector<int>& way) {

if (!node || (!way.empty() && way.back() == item)) return;

way.push\_back(node->value);

preOrderSearch(node->left, item, way);

preOrderSearch(node->right, item, way);

}

void preOrder(dg::Tree::Node\* node, std::vector<int>& way) {

if (!node) return;

way.push\_back(node->value);

preOrder(node->left, way);

preOrder(node->right, way);

}

void getTree(dg::Tree& tree) {

auto root = tree.getRoot();

auto\* v1 = new dg::Tree::Node(1);

auto\* v2 = new dg::Tree::Node(2);

auto\* v3 = new dg::Tree::Node(3);

auto\* v4 = new dg::Tree::Node(4);

auto\* v5 = new dg::Tree::Node(5);

auto\* v6 = new dg::Tree::Node(6);

auto\* v7 = new dg::Tree::Node(7);

auto\* v8 = new dg::Tree::Node(8);

root->value = 10;

root->left = v2;

root->right = v5;

v2->left = v3;

v2->right = v8;

v5->left = v6;

v5->right = v7;

v7->left = v1;

v7->right = v4;

}

void printArr(std::vector<int> arr) {

for (auto &item : arr) {

std::cout << item << " ";

}

std::cout << std::endl;

}

void printHello() {

std::cout << "\* \* \* Algorithms: Trees (part 1) \* \* \*\n\n";

}

template <typename T>

T prompt(const char label[]) {

std::cout << label;

while (true) {

T val;

std::cin >> val;

if (std::cin.fail()) {

std::cin.clear();

std::cin.ignore(32767, '\n');

std::cout << "Wrong. Try again: ";

}

else {

std::cin.ignore(32767, '\n');

return val;

}

}

}

***Tree.hpp:***

#pragma once

namespace dg {

class Tree {

public:

struct Node {

Node() : value(0) {}

Node(int value) : value(value) {}

int value;

Node\* left = nullptr;

Node\* right = nullptr;

};

Tree() {};

Node\* getRoot() {

return m\_root;

}

void setRoot(Node\* node) {

m\_root = node;

}

~Tree() {};

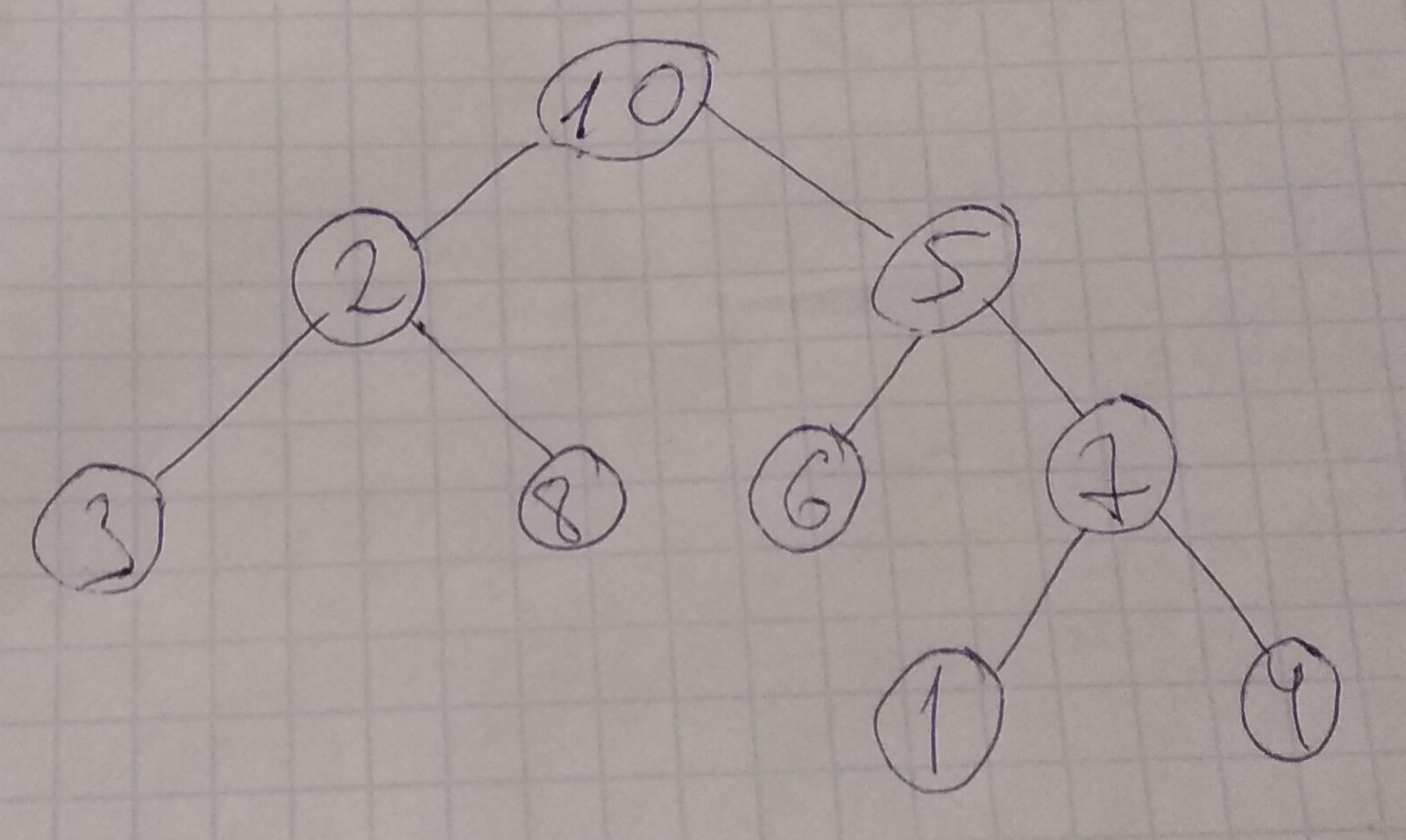
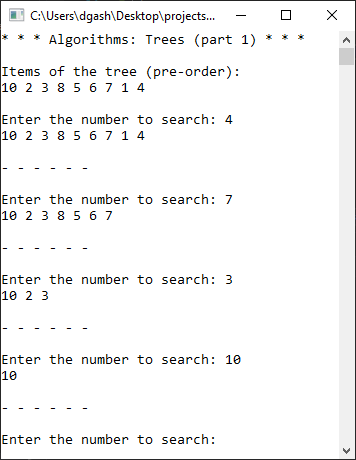
private:

Node\* m\_root = nullptr;

};

}; // namespace dg

1. Результат выполнения программы



Задание 2:

1. Текст программы

#include <iostream>

#include <vector>

#include <string>

#include "Tree.hpp"

template <typename T>

T prompt(const char label[]);

void inOrder(dg::Tree::Node\* node, std::vector<int>& way);

void printArr(std::vector<int> arr);

void printHello();

void run();

std::vector<int> getNumberToSort();

void addNumbersToSearchTree(dg::Tree& tree, int num);

void printItemsOnLevel(dg::Tree::Node\* node, int level, int curLevel = 0);

int main() {

printHello();

run();

system("pause");

return 0;

}

void run() {

dg::Tree tree;

auto numbers = getNumberToSort();

for (auto& num : numbers) {

addNumbersToSearchTree(tree, num);

}

std::vector<int> sortedNumbers;

inOrder(tree.getRoot(), sortedNumbers);

std::cout << "Sorted items:\n";

printArr(sortedNumbers);

while (true) {

std::cout << std::endl;

int level = prompt<int>("Enter a level: ") - 1;

std::cout << "Elements on level " << level + 1 << ": ";

printItemsOnLevel(tree.getRoot(), level);

std::cout << std::endl;

}

}

std::vector<int> getNumberToSort() {

auto size = prompt<int>("Enter the number of a numbers to sort: ");

std::vector<int> numbers(size);

std::cout << "Enter numbers to sort:\n";

for (auto& num : numbers) {

std::cin >> num;

}

return numbers;

}

void addNumbersToSearchTree(dg::Tree& tree, int num) {

auto newNode = new dg::Tree::Node(num);

dg::Tree::Node\* next = tree.getRoot();

if (!next) {

tree.setRoot(newNode);

return;

}

dg::Tree::Node\* prev = next;

while (next) {

prev = next;

next = (num < next->value) ? next->left : next->right;

}

if (num < prev->value) {

prev->left = newNode;

}

else {

prev->right = newNode;

}

}

void printItemsOnLevel(dg::Tree::Node\* node, int level, int curLevel) {

if (!node) return;

bool hasNext = curLevel < level;

if (hasNext) printItemsOnLevel(node->left, level, curLevel + 1);

if (!hasNext) std::cout << node->value << " ";

if (hasNext) printItemsOnLevel(node->right, level, curLevel + 1);

}

void inOrder(dg::Tree::Node\* node, std::vector<int>& way) {

if (!node) return;

inOrder(node->left, way);

way.push\_back(node->value);

inOrder(node->right, way);

}

void printArr(std::vector<int> arr) {

for (auto& item : arr) {

std::cout << item << " ";

}

std::cout << std::endl;

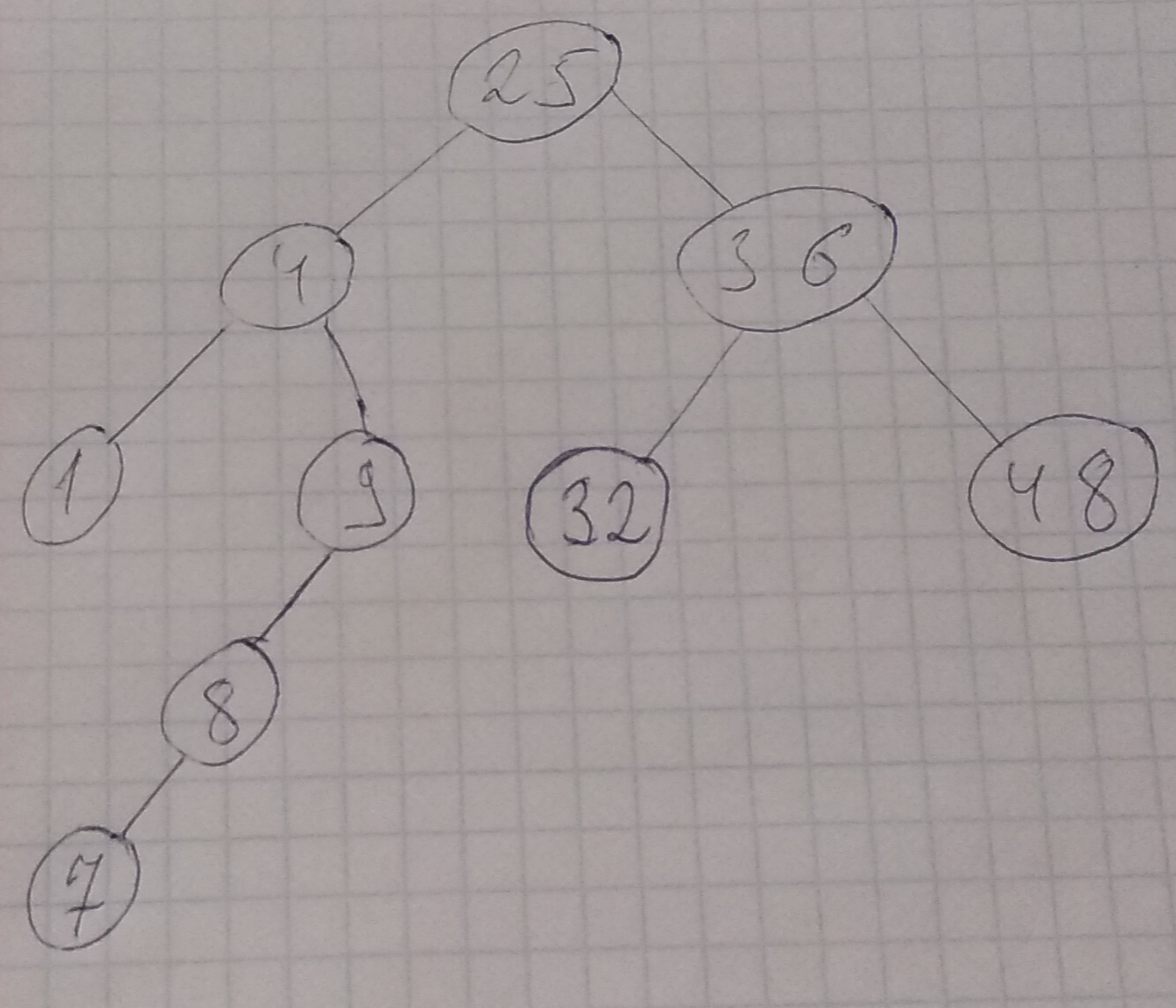
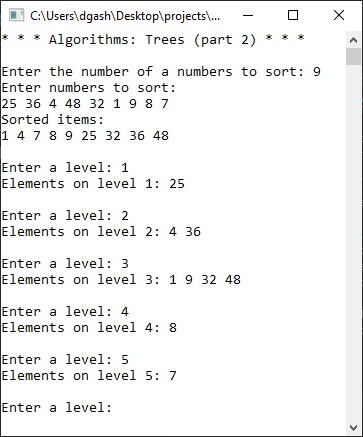
}

void printHello() {

std::cout << "\* \* \* Algorithms: Trees (part 2) \* \* \*\n\n";

}

1. Результат выполнения программы



**Итог:** на этой лабораторной работе я изучил основные алгоритмы работы с деревьями, получил практические навыки разработки и использования этих структур и алгоритмов для решения задач; приобрел навыки обработки бинарного дерева поиска.