**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Кафедра «Вычислительная техника»**

**ОТЧЁТ**

**по дисциплине «Л и ОА в ИЗ»**

**по лабораторной работе №4 «бинарное дерево поиска»**

Выполнил студент гр. 22ВВВ3:

Гречихин П. П.

Приняли:

к.т.н., доцент Юрова О.В.

к.э.н., доцент Акифьев И.В.

Пенза

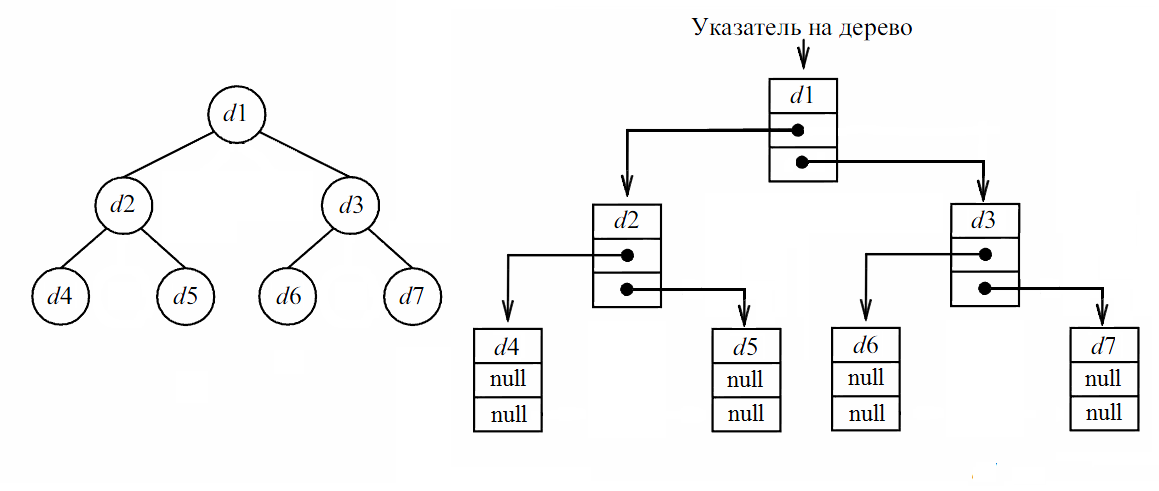
2023

**Цель:**

В данной лабораторной работе мы изучаем принципы работы бинарного дерева на языке программирования C++.

**Общие сведения:**

Бинарные деревья – это деревья, у каждого узла которого возможно наличие только двух сыновей. Двоичные деревья являются упорядоченными.



Двоичное дерево можно представить в виде нелинейного связанного списка.

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами:

1. Значение левого потомка меньше значения родителя;

2. Значение правого потомка больше значения родителя.

Такие структуры используются для сохранения данных в отсортированном виде.

В простейшем случае для создания элемента списка используется структура, в которой объединяются полезная информация и ссылка на следующий элемент списка:

struct Node {

int data;

struct Node \*left;

struct Node \*right;

};

В качестве информации в дереве хранятся целые числа.

Обращение к дереву и его элементам осуществляется посредством указателей:

struct Node \*root;

Так как деревья по своей сути являются рекурсивными структурами данных, то и большинство функций, работающих с деревьями, рекурсивны.

Функция создания дерева выделяет память под каждый новый элемент и добавляет его в дерево:

struct Node \*CreateTree(struct Node \*root, struct Node \*r, int data)

{

if (r == NULL)

{

r = (struct Node \*)malloc(sizeof(struct Node));

if (r == NULL)

{

printf("Ошибка выделения памяти");

exit(0);

}

r->left = NULL;

r->right = NULL;

r->data = data;

if (root == NULL) return r;

if (data > root->data) root->left = r;

else root->right = r;

return r;

}

if (data > r->data)

CreateTree(r, r->left, data);

else

CreateTree(r, r->right, data);

return root;

}

Функция вывода дерева на экран (дерево выводится повёрнутым на 90 градусов, корень находится слева):

void print\_tree(struct Node \*r, int l)

{

if (r == NULL)

{

return;

}

print\_tree(r->right, l + 1);

for(int i = 0; i < l; i++)

{

printf(" ");

}

printf("%d\n", r->data);

print\_tree(r->left, l+1);

}

Программа, использующая приведенные функции:

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1;

root = NULL;

printf("-1 - окончание построения дерева\n");

while (start)

{

printf("Введите число: ");

scanf\_s("%d", &D);

if (D == -1)

{

printf("Построение дерева окончено\n\n");

start = 0;

}

else

root = CreateTree(root, root, D);

}

print\_tree(root,0);

scanf\_s("%d", &D);

return 0;

}

**Практическая часть:**

**Задание 1:**

Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.

**Задание 2:**

Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

**Задание 3:**

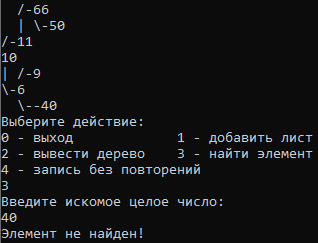
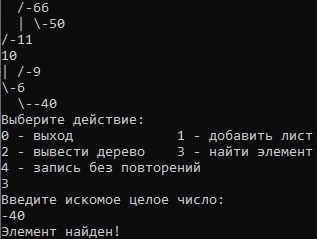
\*Изменить функцию добавления элементов для исключения добавления одинаковых символов.

**Задание 4:**

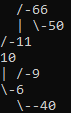
\*Оценить сложность процедуры поиска по значению в бинарном дереве.

**Расчёты:**

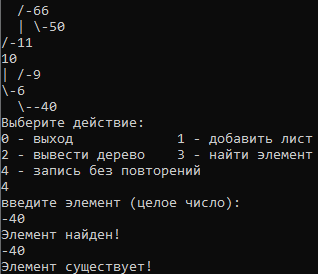
**Задание 1:**

****

**Задание 2:**

****

**Задание 3:**

****

**Задание 4:**

int search(Tree\* tr, int num)

{

bool key = 0;

if (!tr) return num;

if (num == tr->data)

{

tr = NULL;

key = 1;

}

else if (num < tr->data and tr->left != NULL) num = search(tr->left, num);

else if (num > tr->data and tr->right != NULL) num = search(tr->right, num);

else { num = 0; key = 1; }

if (num and key) { cout << "Элемент найден!" << endl; key = 0; }

else if (num == 0 and key) { cout << "Элемент не найден!" << endl; key = 0; }

return num;

}

O(logN)

**Заключение:**

В данной лабораторной работе были изучены навыки работы с абстрактной структурой – дерево.

**Листинг:**

**tree.h**

#include <Windows.h>

#include <iostream>

#include <iomanip>

#include <string>

#include <vector>

#define LN { std::cout << \_\_LINE\_\_ << std::endl; }

#define DEB(x) { std::cout << #x << "=(" << (x) << ") "; }

using namespace std;

static std::string ch\_hor = "-", ch\_ver = "|", ch\_ddia = "/", ch\_rddia = "\\", ch\_udia = "\\", ch\_ver\_hor = "|-", ch\_udia\_hor = "\\-", ch\_ddia\_hor = "/-", ch\_ver\_spa = "| ";

//static std::string ch\_hor = "\u2500" /\*─\*/, ch\_ver = "\u2502" /\*│\*/, ch\_ddia = "\u250C" /\*┌\*/, ch\_rddia = "\u2510" /\*┐\*/, ch\_udia = "\u2514" /\*└\*/, ch\_ver\_hor = "\u251C\u2500" /\*├─\*/, ch\_udia\_hor = "\u2514\u2500" /\*└─\*/, ch\_ddia\_hor = "\u250C\u2500" /\*┌─\*/, ch\_ver\_spa = "\u2502 " /\*│ \*/;

typedef struct tree

{

int data;

tree\* left, \* right;

}Tree;

void AddSheet(Tree\*, int num, Tree\* tmp = NULL);

int search(Tree\*, int);

void wr\_block(Tree\*, int);

void PrintTree(Tree const\* node, std::string const& rpref = "", std::string const& cpref = "", std::string const& lpref = "");

**main.cpp**

#include "tree.h"

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Tree\* t = new Tree;

int var;

t->left = NULL;

t->right = NULL;

cout << "Введите корень (целое число):" << endl;

cin >> t->data;

do {

cout << "Выберите действие:\n0 - выход 1 - добавить лист\n2 - вывести дерево 3 - найти элемент" << endl

<< "4 - запись без повторений 5 - узнать колличество элементов" << endl;

cin >> var;

switch (var)

{

case 0:

goto L1;

case 1:

cout << "Введите элемент (целое число):" << endl;

cin >> var;

AddSheet(t, var);

break;

case 2:

PrintTree(t);

break;

case 3:

cout << "Введите искомое целое число:" << endl;

cin >> var;

search(t, var);

break;

case 4:

cout << "введите элемент (целое число):" << endl;

cin >> var;

wr\_block(t, var);

break;

}

} while (1);

L1:

return 0;

}

**tree.cpp**

#include "tree.h"

void wr\_block(Tree\* t, int num)

{

int i = search(t, num);

cout << i << endl;

if (i == 0) AddSheet(t, num);

else cout << "Элемент существует!" << endl;

}

void AddSheet(Tree\* tr, int num, Tree\* tmp)

{

if (!tr)

{

return;

}

if (!tmp)

{

tmp = new Tree;

tmp->data = num;

tmp->left = NULL;

tmp->right = NULL;

}

if (num < tr->data)

{

if (!tr->left)

{

tr->left = tmp;

tr = NULL;

return;

}

else AddSheet(tr->left, num, tmp = tmp);

}

else

{

if (!tr->right)

{

tr->right = tmp;

tr = NULL;

return;

}

else AddSheet(tr->right, num, tmp = tmp);

}

}

void PrintTree(Tree const\* node, std::string const& rpref, std::string const& cpref, std::string const& lpref) {

if (!node) return;

if (node->right)

PrintTree(node->right, rpref + " ", rpref + ch\_ddia\_hor, rpref + ch\_ver\_spa);

std::cout << cpref << std::to\_string(node->data) << std::endl;

if (node->left)

PrintTree(node->left, lpref + ch\_ver\_spa, lpref + ch\_udia\_hor, lpref + " ");

}

int search(Tree\* tr, int num)

{

bool key = 0;

if (!tr) return num;

if (num == tr->data)

{

tr = NULL;

key = 1;

}

else if (num < tr->data and tr->left != NULL) num = search(tr->left, num);

else if (num > tr->data and tr->right != NULL) num = search(tr->right, num);

else { num = 0; key = 1; }

if (num and key) { cout << "Элемент найден!" << endl; key = 0; }

else if (num == 0 and key) { cout << "Элемент не найден!" << endl; key = 0; }

return num;

}