Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему « **Бинарное дерево поиска**»

**Выполнили:**

Студенты группы 24ВВВ1

Куничкина В.А.

Суркова Д.А.

**Приняли:**

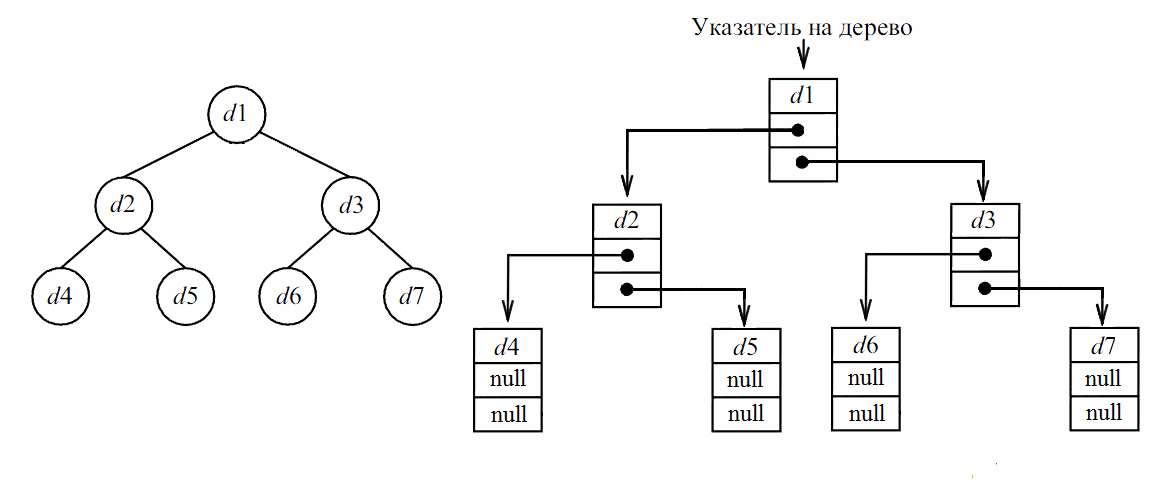
к.т.н. Деев М.В.

к.т.н. доцент Юрова О.В.

Пенза 2025

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

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:

– значение левого потомка меньше значения родителя;

– 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;

}

**Задание**

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

2. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

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

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

**Практическая часть:**

**Листинг:**

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <windows.h>

#include <limits.h>

#include <string.h>

#include <errno.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root = NULL;

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\n");

exit(0);

}

r->left = NULL;

r->right = NULL;

r->data = data;

if (root == NULL) return r;

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

else root->left = r;

return r;

}

if (data > r->data)

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

else

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

return r;

}

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);

}

struct Node\* searchTree(struct Node\* r, int value)

{

if (r == NULL || r->data == value)

return r;

if (value < r->data)

return searchTree(r->left, value);

else

return searchTree(r->right, value);

}

int countOccurrences(struct Node\* r, int value)

{

if (r == NULL) return 0;

int count = 0;

if (r->data == value) count = 1;

return count + countOccurrences(r->left, value) + countOccurrences(r->right, value);

}

void freeTree(struct Node\* r)

{

if (r == NULL) return;

freeTree(r->left);

freeTree(r->right);

free(r);

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int D, start = 1;

int searchValue;

char input[100];

root = NULL;

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

while (start) {

printf("Введите число: ");

if (fgets(input, sizeof(input), stdin) == NULL) {

printf("Ошибка ввода! Программа завершена.\n");

freeTree(root);

getchar();

return 0;

}

char\* endptr;

errno = 0;

long num = strtol(input, &endptr, 10);

if (endptr == input || \*endptr != '\n' && \*endptr != '\0') {

printf("Ошибка ввода! Введите целое число. Программа завершена.\n");

freeTree(root);

getchar();

return 0;

}

if (errno == ERANGE || num < INT\_MIN || num > INT\_MAX) {

printf("Ошибка ввода! Число должно быть в диапазоне %d до %d. Программа завершена.\n", INT\_MIN, INT\_MAX);

freeTree(root);

getchar();

return 0;

}

D = (int)num;

if (D == -1) {

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

start = 0;

}

else {

root = CreateTree(root, root, D);

}

}

printf("Дерево:\n");

print\_tree(root, 0);

printf("\n");

printf("Введите число для поиска: ");

if (fgets(input, sizeof(input), stdin) == NULL) {

printf("Ошибка ввода! Программа завершена.\n");

freeTree(root);

getchar();

return 0;

}

char\* endptr;

errno = 0;

long searchNum = strtol(input, &endptr, 10);

if (endptr == input || \*endptr != '\n' && \*endptr != '\0' || errno == ERANGE || searchNum < INT\_MIN || searchNum > INT\_MAX) {

printf("Ошибка ввода! Программа завершена.\n");

freeTree(root);

getchar();

return 0;

}

searchValue = (int)searchNum;

struct Node\* found = searchTree(root, searchValue);

if (found != NULL)

printf("Число %d найдено в дереве.\n", searchValue);

else

printf("Число %d не найдено в дереве.\n", searchValue);

printf("Введите число для подсчёта вхождений: ");

if (fgets(input, sizeof(input), stdin) == NULL) {

printf("Ошибка ввода! Программа завершена.\n");

freeTree(root);

getchar();

return 0;

}

int occurrences = countOccurrences(root, searchValue);

printf("Число %d встречается в дереве %d раз(а).\n", searchValue, occurrences);

freeTree(root);

printf("Нажмите Enter для выхода...");

getchar();

return 0;

}

**Вывод**

В ходе выполнения лабораторной работы были изучены бинарные деревья и работа с ними в языке Си