Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе № 4

по дисциплине: "Логика и ОА в ИЗ"

на тему: "Бинарное дерево поиска"

Выполнили:

Трундов Н.А.

Евдокимов Р.E.

Приняли:

Акифьев И.В.

Юрова О.В.

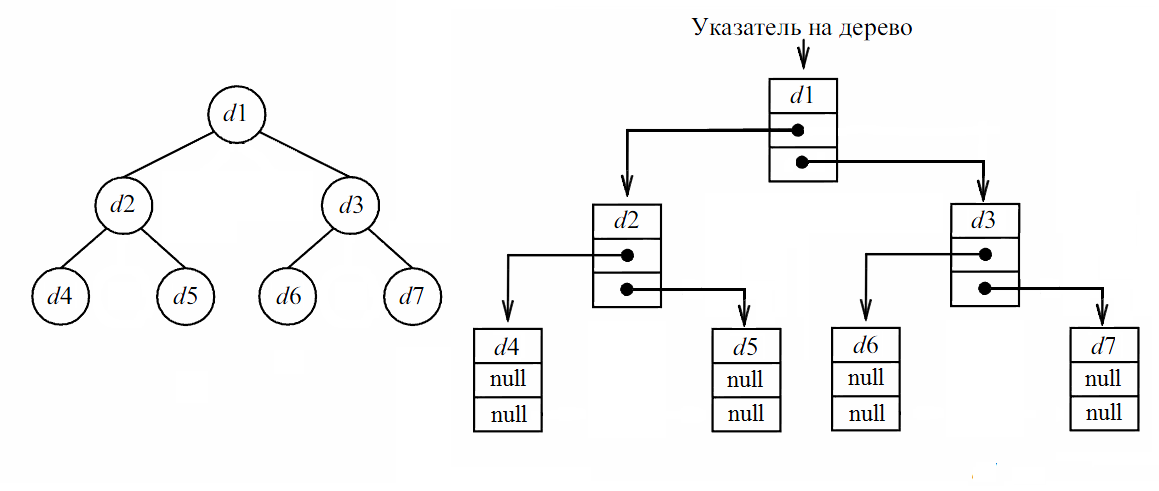
Пенза, 2023

**Бинарное дерево поиска.**

**Цель: Реализовать бинарное дерево поиска.**

**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:

– 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;

}

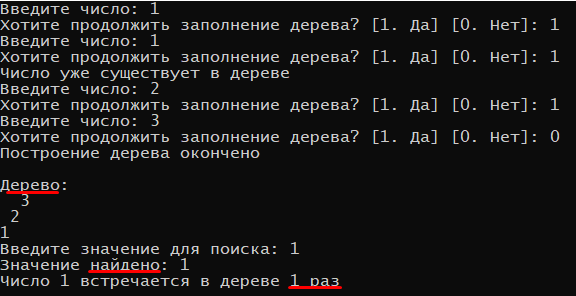
**Задание**

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

Реализовали алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве (Функция есть в Приложении А “Листинг”).

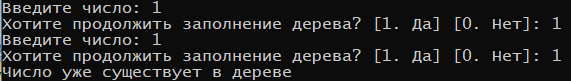
1. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

Реализовали функцию подсчёта числа вхождений заданного элемента в дерево (Функция есть в Приложении А “Листинг”).



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

Изменили функцию добавления элементов для исключения добавления одинаковых символов. (Функция есть в Приложении А “Листинг”).



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

Сложность процедуры поиска по значению в бинарном дереве = log(n), так как в самой процедуре поиска используются сравнения данных, находящихся в разных элементах структуры данных.

**Вывод**

Реализовали структуру данных – дерево, реализовали поиск данных внутри структуры, реализовали исключение добавления одинаковых символов в структуру данных. Оценили сложность процедуры поиска данных.

**Приложение А “Листинг”**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

// Структура для узла дерева

struct Node {

int data; // Значение узла

Node\* left; // Указатель на левого потомка

Node\* right; // Указатель на правого потомка

// Конструктор для узла, инициализирующий данные и указатели на nullptr

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

// Функция для добавления узла в дерево (без дубликатов)

Node\* InsertWithoutDuplicates(Node\* root, int data) {

if (root == nullptr) {

return new Node(data); // Если дерево пустое, создаем новый узел

}

if (data == root->data) {

printf("Число уже существует в дереве\n");

return root; // Если элемент уже существует в дереве, не вставляем его повторно

}

if (data < root->data) {

root->left = InsertWithoutDuplicates(root->left, data); // Рекурсивно добавляем в левое поддерево

}

else {

root->right = InsertWithoutDuplicates(root->right, data); // Рекурсивно добавляем в правое поддерево

}

return root;

}

// Функция для поиска значения в дереве

Node\* Search(Node\* root, int key) {

if (root == nullptr || root->data == key) {

return root; // Если узел пустой или содержит искомое значение, возвращаем узел

}

if (root->data < key) {

return Search(root->right, key); // Если искомое значение больше, ищем в правом поддереве

}

return Search(root->left, key); // В противном случае ищем в левом поддереве

}

// Функция для вывода дерева на экран (90 градусов поворот)

void PrintTree(Node\* root, int level) {

if (root == nullptr) {

return; // Если узел пустой, выходим из функции

}

PrintTree(root->right, level + 1); // Рекурсивно выводим правое поддерево

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

printf(" "); // Выводим отступы в зависимости от уровня узла

}

printf("%d\n", root->data); // Выводим данные узла

PrintTree(root->left, level + 1); // Рекурсивно выводим левое поддерево

}

// Функция для подсчета вхождений элемента в дерево

int CountOccurrences(Node\* root, int key) {

if (root == nullptr) {

return 0; // Если узел пустой, возвращаем 0

}

int count = 0;

if (root->data == key) {

count++; // Если узел содержит искомое значение, увеличиваем счетчик

}

count += CountOccurrences(root->left, key); // Рекурсивно ищем в левом поддереве

count += CountOccurrences(root->right, key); // Рекурсивно ищем в правом поддереве

return count;

}

int main() {

setlocale(LC\_ALL, "RUS");

Node\* root = nullptr;

int data;

int start = 1;

int stk = 1;

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

while (start) {

printf("Введите число: ");

scanf("%d", &data);

printf("Хотите продолжить заполнение дерева? [1. Да] [0. Нет]: ");

scanf("%d", &stk);

if (stk == 0) {

root = InsertWithoutDuplicates(root, data);

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

start = 0;

}

else if (stk == 1) {

root = InsertWithoutDuplicates(root, data);

}

else {

printf("Ошибка ввода\n\n");

}

}

printf("Дерево:\n");

PrintTree(root, 0);

printf("Введите значение для поиска: ");

scanf("%d", &data);

Node\* result = Search(root, data);

if (result != nullptr) {

printf("Значение найдено: %d\n", result->data);

}

else {

printf("Значение не найдено\n");

}

int occurrences = CountOccurrences(root, data);

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

return 0;

}