Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №4  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Бинарное дерево поиска.»

Выполнили студенты группы 22ВВВ2:  
Хоссейни Нежад С. А. С. М.

Приняли:  
Акифьев И. В.  
Митрохин М. А.

Пенза 2023

**Название**

Бинарное дерево поиска.

**Лабораторное задание**

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.

**Ход работы**

* Задание 1:

Написал функцию для поиска элемента

void find(struct tnode\* tree, int element)

{

if (tree != NULL)

{

if(element < tree->data)

find(tree->left, element);

if (tree->data == element)

printf("Element %d found\n", element);

if (element > tree->data)

find(tree->right, element);

}

}

* Задание 2:

Написал функцию для подсчета числа вхождений элемента

void occurrence(struct tnode\* tree, int element)

{

if (tree != NULL)

{

if (element < tree->data)

occurrence(tree->left, element);

count++;

if (element > tree->data)

occurrence(tree->right, element);

}

}

* Задание 3:

Изменил функцию добавления элементов для исключения добавления одинаковых символов.

struct tnode\* add\_unequal\_element(int x, tnode\* tree)

{

total(tree, x);

if (count > 0)

{

printf("Such an element already exists");

return(tree);

}

if (tree == NULL)

{

tree = (struct tnode\*)malloc(sizeof(struct tnode));

tree->data = x;

tree->left = NULL;

tree->right = NULL;

}

else if (x < tree->data)

tree->left = add\_unequal\_element(x, tree->left);

else

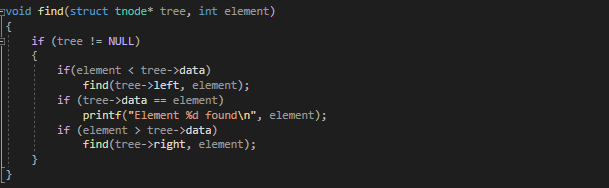
tree->right = add\_unequal\_element(x, tree->right);

return(tree);

}

Задание 4:

Оценил сложность процедуры поиска по значению в бинарном дереве.



Сложность алгоритма поиска в среднем и в лучшем случае составляет **O(log n)**. Это связано с тем, что с каждым шагом алгоритма поиска количество оставшихся для проверки узлов дерева уменьшается. В худшем случае, когда каждый узел имеет только одного потомка, сложность поиска становится **O(n)**.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

struct tnode\* add\_element(int x, tnode\* tree);

struct tnode\* add\_unequal\_element(int x, tnode\* tree);

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

void view\_tree(struct tnode\* tree);

void find(struct tnode\* tree, int element);

void total(struct tnode\* tree, int element);

void total\_amount(struct tnode\* tree);

void occurrence(struct tnode\* tree, int element);

int count = 0;

int main()

{

int start = 1;

int data = 0, element = 9;

char ch;

int ext = 0;

struct tnode\* root = NULL;

do

{

system("cls");

printf("1. Add element\n2. Adding unequal elements\n3. Print tree\n4. Find element\n5. Occurrence of the element\n6. View tree\n7. Amount of the element\n");

printf("Select item: ");

scanf("%c", &ch);

system("cls");

if (ch != '\n')

{

scanf("%\*[^\n]");

getchar();

}

switch (ch)

{

case '1':

printf("Enter number: ");

scanf("%d", &data);

root = add\_element(data, root);

break;

case '2':

printf("Enter number: ");

scanf("%d", &data);

root = add\_unequal\_element(data, root);

printf("\nPress any key to exit");

if (count > 0)

{

count = 0;

getchar();

getchar();

}

break;

case '3':

printf("Tree: \n");

print\_tree(root, 6);

printf("\nPress any key to exit");

getchar();

break;

case '4':

printf("Enter element: ");

scanf("%d", &element);

find(root, element);

total(root, element);

if (count == 0)

printf("Element not found\n");

count = 0;

printf("Press any key to exit");

getchar();

getchar();

break;

case '5':

printf("Enter element: ");

scanf("%d", &element);

occurrence(root, element);

printf("Occurrence: %d", count);

count = 0;

printf("\nPress any key to exit");

getchar();

getchar();

break;

case '6':

printf("Tree: \n");

view\_tree(root);

printf("\nPress any key to exit");

getchar();

break;

case '7':

count = 0;

total\_amount(root);

printf("Amount: %d", count++);

count = 0;

printf("\nPress any key to exit");

getchar();

break;

default:

break;

}

} while (ext == 0);

return 0;

}

struct tnode

{

int data;

struct tnode\* left;

struct tnode\* right;

};

struct tnode\* add\_element(int x, tnode\* tree)

{

if (tree == NULL)

{

tree = (struct tnode\*)malloc(sizeof(struct tnode));

tree->data = x;

tree->left = NULL;

tree->right = NULL;

}

else if (x < tree->data)

tree->left = add\_element(x, tree->left);

else

tree->right = add\_element(x, tree->right);

return(tree);

}

struct tnode\* add\_unequal\_element(int x, tnode\* tree)

{

total(tree, x);

if (count > 0)

{

printf("Such an element already exists");

return(tree);

}

if (tree == NULL)

{

tree = (struct tnode\*)malloc(sizeof(struct tnode));

tree->data = x;

tree->left = NULL;

tree->right = NULL;

}

else if (x < tree->data)

tree->left = add\_unequal\_element(x, tree->left);

else

tree->right = add\_unequal\_element(x, tree->right);

return(tree);

}

void print\_tree(struct tnode\* 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);

}

void view\_tree(struct tnode\* tree)

{

if (tree != NULL) {

printf("%d ", tree->data);

count++;

view\_tree(tree->left);

view\_tree(tree->right);

}

}

void find(struct tnode\* tree, int element)

{

if (tree != NULL)

{

if(element < tree->data)

find(tree->left, element);

if (tree->data == element)

printf("Element %d found\n", element);

if (element > tree->data)

find(tree->right, element);

}

}

void total(struct tnode\* tree, int element)

{

if (tree != NULL)

{

if (element <= tree->data)

total(tree->left, element);

if (tree->data == element)

count++;

if (element >= tree->data)

total(tree->right, element);

}

}

void total\_amount(struct tnode\* tree)

{

if (tree != NULL) {

count++;

total\_amount(tree->left);

total\_amount(tree->right);

}

}

void occurrence(struct tnode\* tree, int element)

{

if (tree != NULL)

{

if (element < tree->data)

occurrence(tree->left, element);

count++;

if (element > tree->data)

occurrence(tree->right, element);

}

}

**Результат работы программы**

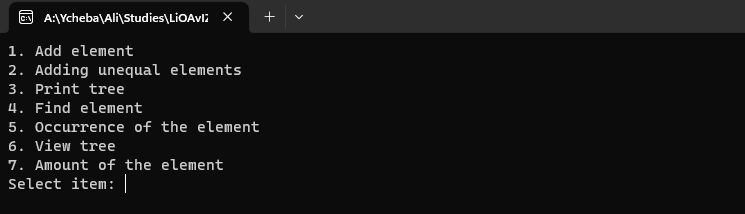


Рисунок 1 – меню



Рисунок 2 - добавление элемента

После добавил несколько элементов

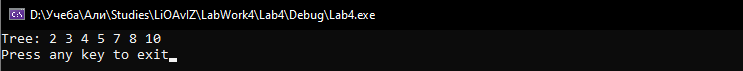


Рисунок 3 - просмотр дерева

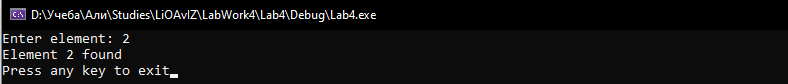


Рисунок 4 - поиск элемента



Рисунок 5 - проверка вхождений элемента

**Вывод**

В результате выполнения лабораторной работы я изучил бинарное дерево.