Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»  
  
  
  
  
  
  
**ОТЧЁТ**

по лабораторной работе № 4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнил:

Студент групп 21ВВ1.3

Самохвалов Я.Д.

Приняли:

к.т.н., доцент Юрова О.А.

д.т.н., профессор, зав. каф. ВТ Митрохин М. А.

Пенза 2022

**Название:** Бинарное дерево поиска

**Цель работы:** выполнить лабораторные указания 1-4 используя бинарное дерево поиска.

**Лабораторные задания:**

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.

**Листинг:**

// Поиск и подсчёт числа вхождений в бинароном дереве.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <time.h>

#include <stdlib.h>

#include <Windows.h>

#include <conio.h>

#define lower(a, b) ((a) < (b))

#define higher(a, b) ((a) > (b))

int \_stateMenu;

int l;

void Menu()

{

printf("Enter action:\n1 - Print tree (no repetitions)\n2 - Find position of number in tree\n3 - Find number of repeats in tree\n4 - Delete node in the tree\n5 - Save tree in file (no repetitions)\n6 - Save tree in file (with repetitions)\n0 - Exit\n");

\_stateMenu = \_getch() - 48;

}

//структура одного элемента

typedef struct tree

{

int a; // данные

struct tree\* left; // левый сын

struct tree\* right; // правый сын

struct tree\* parent; // родитель

} Node;

typedef struct equqlity

{

int val, t;

} eq;

Node\* NewNode(int value, Node\* parent)

{

Node\* tmp = (Node\*)malloc(sizeof(Node));

tmp->left = tmp->right = NULL;

tmp->a = value;

tmp->parent = parent;

return tmp;

}

int InsertNode(Node\*\* head, int value)

{

Node\* tmp = NULL;

Node\* ins = NULL;

if (\*head == NULL)

{

\*head = NewNode(value, NULL);

return 1;

}

tmp = \*head;

while (tmp)

{

if (higher(value, tmp->a)) {

if (tmp->right) {

tmp = tmp->right;

continue;

}

else {

tmp->right = NewNode(value, tmp);

return 1;

}

}

else if (lower(value, tmp->a)) {

if (tmp->left) {

tmp = tmp->left;

continue;

}

else {

tmp->left = NewNode(value, tmp);

return 1;

}

}

else {

return 2;

}

}

}

void PrintTree(Node\* root, int l)

{

if (root == NULL)

{

return;

}

PrintTree(root->right, l + 1);

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

printf("%d\n", root->a);

PrintTree(root->left, l + 1);

}

void PrintTreeInFile(Node\* root, FILE\* f)

{

if (root) {

fprintf(f, "%d ", root->a);

PrintTreeInFile(root->left, f);

PrintTreeInFile(root->right, f);

}

}

void PrintTreeF(Node\* root, int l, int\* znach, int k)

{

if (root == NULL) {

return;

}

bool q = false;

PrintTreeF(root->right, l + 1, znach, k);

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

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

{

if (root->a == znach[i])

{

printf("%d\n", root->a);

q = true;

break;

}

}

if (!q) printf(" \n", root->a);

PrintTreeF(root->left, l + 1, znach, k);

}

int\* SearchTree(Node\* root, int number, int level, int\* znach, int k)

{

if (number == root->a)

{

znach = (int\*)realloc(znach, (k + 1) \* sizeof(int));

znach[k] = number;

return znach;

}

if (root->left == NULL && root->right == NULL) return znach = { 0 };

if (number < root->a)

{

znach = (int\*)realloc(znach, (k + 1) \* sizeof(int));

znach[k] = root->a;

return SearchTree(root->left, number, level + 1, znach, k + 1);

}

if (number > root->a)

{

znach = (int\*)realloc(znach, (k + 1) \* sizeof(int));

znach[k] = root->a;

return SearchTree(root->right, number, level + 1, znach, k + 1);

}

}

bool SearchTreeBool(Node\* root, int number)

{

if (number == root->a) return true;

if (root->left == NULL && root->right == NULL) return false;

if (number < root->a) return SearchTreeBool(root->left, number);

if (number > root->a) return SearchTreeBool(root->right, number);

}

Node\* SearchTreeInt(Node\* root, int number)

{

while (root) {

if (root->a > number) {

root = root->left;

continue;

}

else if (root->a < number) {

root = root->right;

continue;

}

else {

return root;

}

}

return NULL;

}

int FindNumOcc(Node\* root, int\* p, int l, int number)

{

int k = 0;

if (SearchTreeBool(root, number))

{

k++;

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

{

if (p[i] == number) k++;

}

return k;

}

else return 0;

}

Node\* getMaxNode(Node\* root) {

while (root->right) {

root = root->right;

}

return root;

}

void removeNodeByPtr(Node\* target) {

if (target->left && target->right) {

Node\* localMax = getMaxNode(target->left);

target->a = localMax->a;

removeNodeByPtr(localMax);

return;

}

else if (target->left) {

if (target == target->parent->left) {

target->parent->left = target->left;

}

else {

target->parent->right = target->left;

}

}

else if (target->right) {

if (target == target->parent->right) {

target->parent->right = target->right;

}

else {

target->parent->left = target->right;

}

}

else {

if (target == target->parent->left) {

target->parent->left = NULL;

}

else {

target->parent->right = NULL;

}

}

free(target);

}

void DeleteValue(Node\* root, int value) {

Node\* target = SearchTreeInt(root, value);

removeNodeByPtr(target);

}

int main()

{

setlocale(0, "rus");

srand(time(NULL));

int n;

bool k;

size\_t m;

int\* zn = { 0 };

char FILE\_NAME[50];

printf("Input file with tree: ");

scanf("%s", FILE\_NAME);

int\* pov = { 0 };

l = 0;

Node\* root = NULL;

FILE\* f = fopen(FILE\_NAME, "r");

if (!f) {

fprintf(stderr, "Can't open input file!\n");

return 1;

}

while (fscanf(f, "%d ", &n) == 1) {

int q = InsertNode(&root, n);

if (q == 2) {

pov = (int\*)realloc(pov, (l + 1) \* sizeof(int));

pov[l] = n;

l++;

}

}

if (ferror(f) | fclose(f))

fprintf(stderr, "Error reading input file!\n");

system("pause");

system("cls");

Menu();

while (\_stateMenu != 0)

{

switch (\_stateMenu)

{

case 1:

system("cls");

if (root == NULL)

{

printf("Empty tree\n");

system("pause");

system("cls");

Menu();

break;

}

printf("Tree: \n");

PrintTree(root, 0);

system("pause");

system("cls");

Menu();

break;

case 2:

system("cls");

if (root == NULL)

{

printf("Empty tree\n");

system("pause");

system("cls");

Menu();

break;

}

int r;

printf("Enter number for search: ");

scanf("%d", &r);

zn = SearchTree(root, r, 0, zn, 0);

m = 0;

if (zn == 0)

{

printf("Not found\n");

system("pause");

system("cls");

Menu();

break;

}

while (zn[m] != r)

{

m++;

}

PrintTreeF(root, 0, zn, m + 1);

system("pause");

system("cls");

Menu();

break;

case 3:

system("cls");

if (root == NULL)

{

printf("Empty tree\n");

system("pause");

system("cls");

Menu();

break;

}

int j;

printf("Enter number for find repeats: ");

scanf("%d", &j);

printf("Number: %d\n", FindNumOcc(root, pov, l, j));

system("pause");

system("cls");

Menu();

break;

case 4:

system("cls");

if (root == NULL)

{

printf("Empty tree\n");

system("pause");

system("cls");

Menu();

break;

}

int d;

printf("Value of node in tree to delete: ");

scanf("%d", &d);

if (root->a == d) printf("Tou can't delete the root of the tree!\n");

else

{

DeleteValue(root, d);

int\* tmp = (int\*)malloc(sizeof(int) \* l);

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

{

tmp[i] = pov[i];

}

pov = { 0 };

int k = l;

l = 0;

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

{

if (tmp[i] != d)

{

pov = (int\*)realloc(pov, (l + 1) \* sizeof(int));

pov[l] = tmp[i];

l++;

}

}

}

system("pause");

system("cls");

Menu();

break;

case 5:

system("cls");

if (root == NULL)

{

printf("Empty tree\n");

system("pause");

system("cls");

Menu();

break;

}

printf("File name: ");

scanf("%s", FILE\_NAME);

f = fopen(FILE\_NAME, "w");

PrintTreeInFile(root, f);

system("pause");

system("cls");

Menu();

break;

case 6:

system("cls");

if (root == NULL)

{

printf("Empty tree\n");

system("pause");

system("cls");

Menu();

break;

}

printf("File name: ");

scanf("%s", FILE\_NAME);

f = fopen(FILE\_NAME, "w");

PrintTreeInFile(root, f);

if (l > 0)

{

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

{

fprintf(f, "%d ", pov[i]);

}

}

fclose(f);

system("pause");

system("cls");

Menu();

break;

case 0:

exit(EXIT\_SUCCESS);

default:

system("cls");

printf("Wrong number\n");

system("pause");

system("cls");

Menu();

break;

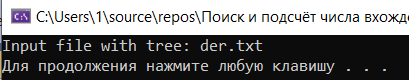
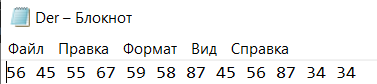
}

}

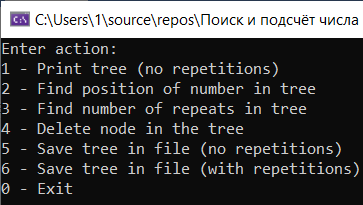
}

**Выводы результатов:**

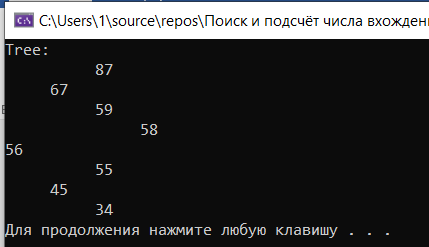
Ввод имени файла, в котором располагается дерево в виде массива.

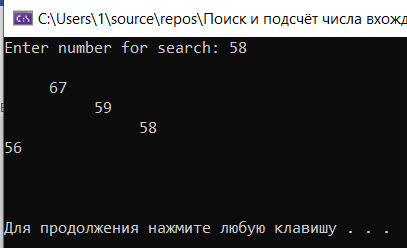
Вывод меню:



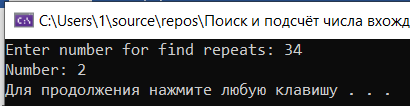
1 – Вывод дерева.



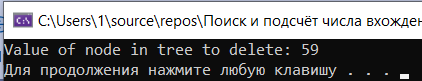
2 – Поиск числа в дереве. Выводом является ветка дерева.



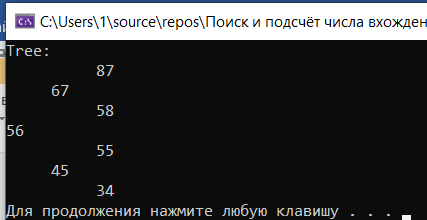
3 – Нахождение количества повторений.



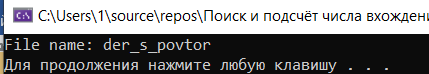
4 – Удаление узла в дереве.

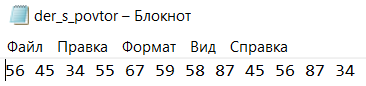


Вывод нового дерева.

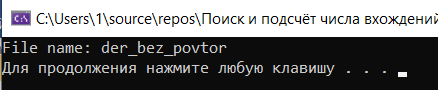


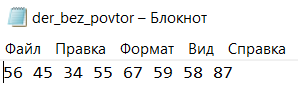
5 – Сохранение дерева в виде массива без повторений значений





6 – Сохранение дерева в виде массива с повторениями.





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

Сложность поиска по значению в бинарном дереве зависит от самого дерева. В худшем случае O(n), в лучшем случае – O(log(n)).

**Вывод:** мы создали алгоритм бинарного дерева поиска, и он корректно работает.