ФАКУЛЬТЕТ «Информатика и системы управления»  
КАФЕДРА «Информационная безопасность»

ОТЧЕТ  
по домашнему заданию №2-3  
по учебной дисциплине «Алгоритмические языки»  
на тему: «Двоичное дерево поиска»

Вариант 23

Выполнил:   
Студент 1 курса, гр. ИУ8-24  
Спиридонов Олег

**Цель работы:**

Изучить принципы работы двоичного дерева поиска и самостоятельно создать его упрощённую версию.

**Условие задачи:**

Необходимо реализовать класс BSTree (двоичное дерево поиска). За основу для реализации элементов дерева написать структуру Node. Структура должна содержать указатели на левый и правый элемент дерева, поле данных типа int, а также опционально указатель на предыдущий элемент.

Класс должен содержать указатель на корень с модификатором доступа private, остальные методы и поля могут быть с модификатором доступа public (при защите необходимо будет обосновать). Класс должен содержать два конструктора:

BSTree(); // конструктор по умолчанию.

BSTree(initializer\_list<int> list); // конструктор с параметром.

Класс должен содержать следующие функции:

bool add\_element(int value); // функция добавления

bool delete\_element(int value); // функция удаления

bool find\_element(int value); // функция поиска элемента

void print(); // функция вывода дерева в консоль

bool save\_to\_file(const std::string& path); // функция сохранения в файл

bool load\_from\_file(const std::string& path); // функция загрузки из файла

~BSTree(); //деструктор.

**Выполнение работы:**

Файл **input.txt:**

4 2 1 3 6 5 7

Файл **Binary tree.cpp:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Node {

int data;

Node\* left; // smaller

Node\* right; // bigger

Node\* previous = nullptr;

};

class BSTree {

private:

Node\* root = nullptr;

public:

BSTree();

BSTree(initializer\_list<int> list);

~BSTree();

bool add\_element(int value);

bool delete\_element(int value);

bool find\_element(int value);

void print();

bool save\_to\_file(const string& path);

bool load\_from\_file(const string& path);

};

BSTree::BSTree() {

}

BSTree::BSTree(initializer\_list<int> list) {

for (auto i: list) {

add\_element(i);

}

}

BSTree::~BSTree() {

Node\* current = root;

while (current != nullptr) {

while (not(current->left == nullptr and current->right == nullptr)) {

while (current->left != nullptr) {

current = current->left;

}

while (current->right != nullptr) {

current = current->right;

}

}

Node\* to\_be\_deleted = current;

current = current->previous;

if (current != nullptr and current->right == to\_be\_deleted) {

current->right = nullptr;

}

else if (current != nullptr and current->left == to\_be\_deleted) {

current->left = nullptr;

}

delete to\_be\_deleted;

}

}

bool BSTree::add\_element(int data) {

Node\* new\_node = new Node{data, nullptr, nullptr, nullptr};

if (root == nullptr) {

root = new\_node;

return true;

}

else {

Node\* current = root;

while (not((new\_node->data > current->data and current->right == nullptr) or (new\_node->data < current->data and current->left == nullptr))) {

if (new\_node->data > current->data) {

current = current->right;

}

else if (new\_node->data < current->data) {

current = current->left;

}

else {

return false;

}

}

if (new\_node->data > current->data and current->right == nullptr) {

current->right = new\_node;

}

else {

current->left = new\_node;

}

new\_node->previous = current;

return true;

}

}

bool BSTree::delete\_element(int data) {

Node\* current = root;

while (not(current == nullptr)) {

if (data > current->data) {

current = current->right;

}

else if (data < current->data) {

current = current->left;

}

else {

current->right->previous = current->previous;

if (current->previous != nullptr) {

if (current == current->previous->right) {

current->previous->right = current->right;

}

else {

current->previous->left = current->right;

}

}

Node\* left\_edge = current->right;

while (left\_edge->left != nullptr) {

left\_edge = left\_edge->left;

}

current->left->previous = left\_edge->left;

left\_edge->left = current->left;

delete current;

return true;

}

}

return false;

}

bool BSTree::find\_element(int data) {

Node\* current = root;

while (not(current == nullptr)) {

if (data > current->data) {

current = current->right;

}

else if (data < current->data) {

current = current->left;

}

else {

return true;

}

}

return false;

}

string print\_recursion(Node\* current){

if (current == nullptr) {

return " ";

}

if (current->left == nullptr and current->right == nullptr) {

return to\_string(current->data);

}

return to\_string(current->data) + '(' + print\_recursion(current->left) + ';' + print\_recursion(current->right) + ')';

}

string print\_to\_file\_recursion(Node\* current) {

if (current == nullptr) {

return " ";

}

if (current->left == nullptr and current->right == nullptr) {

return to\_string(current->data);

}

return to\_string(current->data) + ' ' + print\_to\_file\_recursion(current->left) + ' ' + print\_to\_file\_recursion(current->right);

}

void BSTree::print() {

cout << print\_recursion(root) << endl;

}

bool BSTree::save\_to\_file(const string& path) {

ofstream file\_output("output.txt");

if (file\_output.is\_open()) {

file\_output << print\_to\_file\_recursion(root) << endl;

file\_output.close();

return true;

}

else {

cout << "Output file didn't open." << endl;

return false;

}

}

bool BSTree::load\_from\_file(const string& path) {

ifstream file\_input("input.txt");

if (file\_input.is\_open()) {

string data;

while (file\_input >> data) {

this->add\_element(stoi(data));

}

file\_input.close();

return true;

}

else {

cout << "Input file didn't open." << endl;

return false;

}

}

int main() {

ofstream file\_output("output.txt");

file\_output.close();

ifstream file\_input("input.txt");

BSTree drevo;

drevo.load\_from\_file("input.txt");

/\*initializer\_list<int> list = {4, 2, 6, 1, 3, 5, 7};

BSTree drevo(list);\*/

drevo.print();

drevo.save\_to\_file("output.txt");

}

**Результат работы программы:**

4(2(1;3);6(5;7))

**Вывод:**

В ходе выполнения домашнего задания были изучены принципы работы двоичного дерева поиска и была реализована его упрощённая версия.