МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья поиска, АВЛ деревья

Студент гр. 7383 ___ Сычевский Р.А.

Преподаватель ___ Размочаева Н.В.

Санкт-Петербург

2018

Содержание

1. ЦЕЛЬ РАБОТЫ

Цель работы: Познакомиться со структурой и реализацией БДП на языке программирования C++.

Формулировка задания (Вариант 18): для входных данных:

а) построить АВЛ дерево;

б) Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран;

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе используется главная функция (main()) и функции pr_menu(), show_tr(), PR_b_tree(), f_height(), check_tree(), left_turn(), right_turn(), right_right_turn(), left_left_turn(), fix_tree(), add_to_tree(), del_b_tree(), check_str(), out_elem(), work_with_console(), work_with_file()).

Сначала функция main() вызывает функцию pr_menu(), которая используется для вывода меню. Пользователь выбирает один из пунктов меню и вводит число, которое передается в switch(). В зависимости от выбранного пункта менювыбирается необходимая опция:

1 – ввод бинарного дерева с клавиатуры;

2 – считывание бинарного дерева из файла;

0 – выход из программы;

При вводе единицы, программа вызывает функцию work_with_console(), которая считывает строку из консоли, после чего проверяет её на корректность и заполняет АВЛ дерево при помощи функции add_to_tree().

При вводе двойки, программа вызывает функцию work_with_file(), которая считывает строку из файла, проверяет её на корректность и заполняет АВЛ дерево при помощи функции add_to_tree().

Функция add_to_tree() находит место для очередного элемента и добавляет его в дерево.

После этого дерево проверяется на корректность и исправляется, если это необходимо.

В конце верное дерево выводится на экран и узлы дерева перечисляются в порядке возрастания и записываются в файл “output.txt”.

3. ТЕСТИРОВАНИЕ

3.1 ПРОЦЕСС ТЕСТИРОВАНИЯ

Программа собрана в операционной системе Ubuntu 18.04.1 LTS bionic компилятором g++ (Ubuntu 7.3.0-16ubuntu3) 7.3.0. В других ОС и компиляторах тестирование не проводилось.

3.2 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Тестовые случаи представлены в Приложении А.

Результаты тестирования показали, что программа работает верно, значит поставленная задача выполнена.

4. ВЫВОД

В ходе выполнения данной работы были получены навыки работы с АВЛ деревом. Было построено БДП на базе указателей. БДП рекурсивной структурой и для неё легко определяются рекурсивные функции.

ПРИЛОЖЕНИЕ А: ТЕСТОВЫЕ СЛУЧАИ

№	Ввод	Вывод
1	34 6547 3	.-->6547 -->34 `-->3
2	4 -6 8	.-->8 -->4 `-->-6
3	1 2 3 4 5 6 7 8 9 0 10 11 12 13	.-->13 .-->12 .-->11 | `-->10 | `-->9 -->8 | .-->7 | | `-->6 `-->5 | .-->4 | .-->3 `-->2 `-->1 `-->0
4	45747- 47	неверное выражение
5	-	неверное выражение
6		неверное выражение

ПРИЛОЖЕНИЕ Б: КОД ПРОГРАММЫ

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include <cctype>
using namespace std;

void pr_menu(){
    cout << "Выберите действие:" << endl;
    cout << "1 - ввод с клавиатуры" << endl;
    cout << "2 - ввод из файла" << endl;
    cout << "0 - выход" << endl;
}

struct elem{
    int val;
    int height;
    elem* left;
    elem* right;
};

struct trunk{
    trunk* prev;
    string str;
    trunk(trunk* prev, string str){
        this->prev = prev;
        this->str = str;
    }
};

void show_tr(trunk* p, int &count){
    if(p == NULL)
        return;
    show_tr(p->prev, count);
    count++;
    cout << p->str;
}

void PR_b_tree(elem* tree, trunk* prev, bool is_right){
    if(tree == NULL)
        return;
    string prev_str = "    ";
    trunk* tmp = new trunk(prev, prev_str);
    PR_b_tree(tree->right, tmp, 1);
    if(!prev)
        tmp->str = "-->";
    else if(is_right){
        tmp->str = ".-->";
        prev_str = "   |";
    } else {
```

```
        tmp->str = "`-->";
        prev->str = prev_str;
    }
    int count = 0;
    show_tr(tmp, count);
    cout << tree->val << endl;

    if(prev)
        prev->str = prev_str;
    tmp->str = "   |";
    PR_b_tree(tree->left, tmp, 0);
}

int f_height(elem* root){
    int a = 0;
    int b = 0;
    if(root->left !=NULL)
        a = f_height(root->left);
    if(root->right != NULL)
        b = f_height(root->right);
    if(a<b)
        a = b;
    root->height = a+1;
    return root->height;
}

int check_tree(elem* root){
    int tmp = 0;
    int left = 0;
    int right = 0;
    if(root->left != NULL)
        left = root->left->height;
    if(root->right != NULL)
        right = root->right->height;
    if(abs(left - right) > 1)
        return root->val;
    else{
        if(root->left != NULL)
            tmp += check_tree(root->left);
        if(root->right != NULL)
            tmp += check_tree(root->right);
    }
    return tmp;
}

void left_turn(elem* tmp, elem* root){
    tmp->right = root->left;
    root->left = tmp;
}

void right_turn(elem* tmp, elem* root){
    tmp->left = root->right;
    root->right = tmp;
}

void right_right_turn(elem* root, elem* tmp){
```

```
    tmp->left->right = root->left;
    root->left = tmp->left;
    tmp->left = root->right;
    root->right = tmp;
}

void left_left_turn(elem* root, elem* tmp){
    tmp->right->left = root->right;
    root->right = tmp->right;
    tmp->right = root->left;
    root->left = tmp;
}

elem* fix_tree(elem* root, int tmp){
    int left = 0;
    int right = 0;
    if(root->left != NULL)
        left = root->left->height;
    if(root->right != NULL)
        right = root->right->height;

    if(root->val == tmp){
        elem* tmp_elem = new elem;
        tmp_elem = root;
        if(right > left){
            root = tmp_elem->right;
            left_turn(tmp_elem, root);
            f_height(root);
            tmp = check_tree(root);
            if(tmp == root->val){
                tmp_elem = root;
                root = tmp_elem->left->right;
                right_right_turn(root, tmp_elem);
            }
        } else {
            root = tmp_elem->left;
            right_turn(tmp_elem, root);
            f_height(root);
            tmp = check_tree(root);
            if(tmp == root->val){
                tmp_elem = root;
                root = tmp_elem->right->left;
                left_left_turn(root, tmp_elem);
            }
        }
    } else {
        if(tmp > root->val)
            root->right = fix_tree(root->right, tmp);
        else
            root->left = fix_tree(root->left, tmp);
    }
    return root;
}

void add_to_tree(elem* root, int tmp_val){
    if(root->val < tmp_val){
```

```
        if(root->right != NULL)
            add_to_tree(root->right, tmp_val);
        else{
            root->right = new elem;
            root->right->left = NULL;
            root->right->right = NULL;
            root->right->val = tmp_val;
        }
    } else if(root->val > tmp_val){
        if(root->left != NULL)
            add_to_tree(root->left, tmp_val);
        else{
            root->left = new elem;
            root->left->left = NULL;
            root->left->right = NULL;
            root->left->val = tmp_val;
        }
    }
}

void del_b_tree(elem* root){
    if(root->left != NULL){
        del_b_tree(root->left);
        delete(root->left);
    }
    if(root->right != NULL){
        del_b_tree(root->right);
        delete(root->right);
    }
}

int check_str(string str){
    for(int i = 0; i < str.length(); i++){
        if(!isdigit(str[i]))
            if((str[i] == '-') || (str[i]== ' '))
                return -1;
            else
                return 0;
    }
    return 1;
}

void out_elem(ofstream& f2, elem* root){
    if(root->left != NULL)
        out_elem(f2, root->left);
    f2 << root->val << " ";
    if(root->right != NULL)
        out_elem(f2, root->right);
}

int work_with_console(){
    cout << "Введите элменты дерева через пробел" << endl;
    int tmp = 0;
    char str[256];
    string str1;
    elem* root = new elem;
```

```
root->left = NULL;
root->right = NULL;
getline(cin, str1);
getline(cin, str1);
if(!check_str(str1)){
    cout << "неверное выражение" << endl;
    return 0;
}
for(int i = 0; i < str1.length(); i++)
    if(str1[i] == '-'){
        if(i == str1.length()-1){
            cout << "неверное выражение" << endl;
            return 0;
        }
        if(!isdigit(str1[i+1])){
            cout << "неверное выражение" << endl;
            return 0;
        }
    }
int flag = 0;
int current_s = 0;
int current_c = 0;
while(!flag){
    if(str1[current_s] == ' ')
        current_s++;
    else
        flag = 1;
    if(current_s == str1.length()){
        cout << "неверное  выражение" << endl;
        return 0;
    }
}
while(flag){
    if(isdigit(str1[current_s]) || (str1[current_s] == '-')){
        str[current_c] = str1[current_s];
        current_s++;
        current_c++;
    } else{
        flag = 0;
    }
}
tmp = atoi(str);
for(int i = 0; i < current_c; i++)
    str[i] = 0;
current_c = 0;
root->val = tmp;
f_height(root);
int end_flag = 0;
while(!end_flag){

    while(!flag){
        if(current_s == str1.length()){
            end_flag = 1;
            break;
        }
        if(str1[current_s] == ' ')
```

```
                current_s++;
            else
                flag = 1;
        }
        if(end_flag)
            break;
        while(flag){
            if(current_s == str1.length()){
                end_flag = 1;
                break;
            }
            if(isdigit(str1[current_s]) || (str1[current_s] == '-')){
                str[current_c] = str1[current_s];
                current_s++;
                current_c++;
            } else{
                flag = 0;
            }
        }
        tmp = atoi(str);
        for(int i = 0; i < current_c; i++)
            str[i] = 0;
        current_c = 0;

        add_to_tree(root, tmp);
        f_height(root);
        tmp = check_tree(root);
        while(tmp){
            root = fix_tree(root, tmp);
            f_height(root);
            tmp = check_tree(root);
        }
    }
    cout << "Бинарное дерево:" << endl;
    PR_b_tree(root, NULL, 0);
    ofstream f2;
    f2.open("output.txt");
    out_elem(f2, root);
    f2.close();
    del_b_tree(root);
    delete(root);
    return 0;
}

int work_with_file(){
    cout << "Введите имя файла, в котором записаны элементы дерева" << endl;
    string file_name;
    cin >> file_name;
    ifstream f;
    f.open(file_name.c_str());
    if (!f){
        cout << "Файл не открыт!" << endl;
        return 0;
    }
    int tmp = 0;
    char str[256];
```

```
    elem* root = new elem;
    f >> str;
    if(check_str(str))
        tmp = atoi(str);
    else{
        cout << "неверное выражение" << endl;
        return 0;
    }
    cout << tmp << endl;
    root->val = tmp;
    root->left = NULL;
    root->right = NULL;
    f_height(root);
    while(!f.eof()){
        f >> str;
        if(check_str(str))
            tmp = atoi(str);
        else{
            cout << "неверное выражение" << endl;
            return 0;
        }
        add_to_tree(root, tmp);
        f_height(root);
        tmp = check_tree(root);
        while(tmp){
            root = fix_tree(root, tmp);
            f_height(root);
            tmp = check_tree(root);
        }
    }
    cout << "Бинарное дерево:" << endl;
    PR_b_tree(root, NULL, 0);
    ofstream f2;
    f2.open("output.txt");
    out_elem(f2, root);
    f2.close();
    f.close();
    del_b_tree(root);
    delete(root);
    return 0;
}

int main(){
    pr_menu();
    char way;
    cin >> way;
    while(way != '0'){
        switch (way){
            case '1':
                work_with_console();
                cout << endl;
                pr_menu();
                cin >> way;
                break;
            case '2':
                work_with_file();
```

```
                cout << endl;
                pr_menu();
                cin >> way;
                break;
            default:
                cout << "Неверно введены данные!" << endl;
                pr_menu();
                cin >> way;
        }
    }
    return 0;
}
```