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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: АВЛ деревья, вставка и исключение. Текущий контроль

Студент гр. 7383 _____ Сычевский Р.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Сычевский Р.А.

Группа 7383

Тема работы: АВЛ деревья, вставка и исключение. Текущий контроль

Содержание пояснительной записки:

- Содержание
- Введение
- Описание использованных функций
- Примеры работы программы
- Исследование реализованных алгоритмов
- Заключение

Предполагаемый объём пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Сычевский Р.А.

Преподаватель _____ Размочаева Н.В.

АННОТАЦИЯ

В ходе работы была реализована программа на языке программирования C++, генерирующая варианты заданий по теме «АВЛ деревья» и ответы для этих вариантов.

SUMMARY

During the work the program was implemented in C++, that generates variants of tasks on the topic «AVL trees» and answers for variants.

ВВЕДЕНИЕ

Целью работы является практическое освоение стандартных алгоритмов работы с АВЛ деревьями.

Задачей является создать программу, генерирующую задания по теме «АВЛ деревья» вставка и исключение.

ФОРМУЛИРОВКА ЗАДАЧИ

Написать программу, на языке C++, которая генерирует заданное количество вариантов заданной сложности с заданиями на построение АВЛ деревьев, и ответы к этим заданиям.

РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе была разработана программа, которая спрашивает у пользователя необходимое количество вариантов и сложность для них, после чего генерирует необходимое количество случайных чисел и строит из них АВЛ деревья, после чего записывает в файл с заданиями задания, а в файл с ответами – ответы.

Структура elem

В данной структуре содержатся данные об узле дерева, такие как указатели на сыновей, вес узла и высота данного поддерева.

Функция work_with_console

Данная функция спрашивает у пользователя количество вариантов и сложность деревьев, после чего, используя остальные функции, необходимое количество раз генерирует случайные числа для деревьев и строит их.

Функция add_to_tree

Используется для добавления нового элемента в дерево.

Функция f_height

Записывает в каждый узел его высоту.

Функция check_tree

Проверяет построенное АВЛ дерево на корректность.

Функция fix_tree

Используя функции left_turn, right_turn, right_right_turn, left_left_turn исправляет неправильное АВЛ дерево.

Функция del_b_tree

Удаляет дерево.

Функция out_elem

Записывает правильное дерево в файл с ответами.

Код программы представлен в приложении А

ТЕСТИРОВАНИЕ

Тестирование программы проводилось в OS Linux Mint 18.01. Созданные во время работы программы файлы размещены в той же папке, что и запускаемый файл программы.

Запустили программу:

Введите количество вариантов

6

Введите количество элементов в дереве

8

Файл с заданиями:

1)1804289383 846930886 1681692777 1714636915 1957747793 424238335 719885386 1649760492

2)596516649 1189641421 1025202362 1350490027 783368690 1102520059 2044897763 1967513926

3)1365180540 1540383426 304089172 1303455736 35005211 521595368 294702567 1726956429

4)336465782 861021530 278722862 233665123 2145174067 468703135 1101513929 1801979802

5)1315634022 635723058 1369133069 1125898167 1059961393 2089018456 628175011 1656478042

6)1131176229 1653377373 859484421 1914544919 608413784 756898537 1734575198 1973594324

Файл с ответами:

1)(1681692777(719885386(424238335)(846930886#(1649760492)))(1804289383(1714636915)(1957747793)))

2)(1025202362(596516649#(783368690))(1189641421(1102520059)(1967513926(1350490027)(2044897763))))

3)(1303455736(304089172(35005211#(294702567))(521595368))(1540383426(1365180540)(1726956429)))

4)(336465782(278722862(233665123))(861021530(468703135)(1801979802(1101513929)(2145174067))))

5)(1315634022(1059961393(635723058(628175011))(1125898167))(1656478042(1369133069)(2089018456)))

6)(1131176229(756898537(608413784)(859484421))(1734575198(1653377373)(1914544919#(1973594324))))

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была реализована программа на языке C++ для генерации заданного количества заданий заданной сложности и ответов к ним по теме «АВЛ деревья».

Текущий контроль осуществлен с помощью записи вариантов заданий в файл, а ответов в другой файл.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ 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;
}
void pr_menu2(){
    cout << "Выберите действие:" << endl;
    cout << "1 - добавить элемент" << endl;
    cout << "2 - удалить элемент" << endl;
    cout << "3 - шаг назад" << endl;
    cout << "0 - закончить работу с этим деревом" << endl;
    cout << 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 error = 0;
    int tmp = 0;
    int left = 0;
    int right = 0;
    if(root->left != NULL){
        error = check_tree(root->left);
        left = root->left->height;
    }
    if(error > 0)
        return error;
    if(root->right != NULL){
        error = check_tree(root->right);
        right = root->right->height;
    }
```

```
    if(error > 0)
        return error;
    if(abs(left - right) > 1)
        return root->val;
    else
        return 0;
}
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]))
            return 0;
    }
    return 1;
}
void out_elem(ofstream& f2, elem* root){
    f2 << '(' << root->val;
    if(root->left != NULL)
        out_elem(f2, root->left);
    else if(root->right != NULL)
        f2 << '#';
    if(root->right != NULL)
        out_elem(f2, root->right);
    f2 << ')';
}
int work_with_console(){
    int tmp = 0;
    int variants = 0;
    int complexity = 0;
    char str[256];
    string str1;
    int flag = 0;
    int current_s = 0;
    int current_c = 0;
    int is_right = 0;

    while(!is_right){
        is_right = 1;
        cout << "Введите количество вариантов" << endl;
        getline(cin, str1);
        while(!flag){
            if(str1[current_s] == ' ')
                current_s++;
```

```
    else
        flag = 1;
    if(current_s >= str1.length()){
        is_right = 0;
        break;
    }
}
while(flag){
    if(current_s < str1.length())
        if(isdigit(str1[current_s])){
            str[current_c] = str1[current_s];
            current_s++;
            current_c++;
        } else{
            flag = 0;
        }
    else{
        break;
    }
}
while(!flag){
    if(current_s == str1.length())
        break;
    if(str1[current_s] == ' ')
        current_s++;
    else{
        is_right = 0;
        break;
    }
}
if(!is_right){
    cout << "неверное выражение" << endl;
    for(int i = 0; i < current_c; i++)
```

```
            str[i] = 0;
        flag = 0;
        current_s = 0;
        current_c = 0;
        continue;
    }
    variants = atoi(str);
    for(int i = 0; i < current_c; i++)
        str[i] = 0;
    flag = 0;
    current_s = 0;
    current_c = 0;
}
is_right = 0;
while(!is_right){
    is_right = 1;
    cout << "Введите количество элементов в дереве" << endl;
    getline(cin, str1);
    while(!flag){
        if(str1[current_s] == ' ')
            current_s++;
        else
            flag = 1;
        if(current_s >= str1.length()){
            is_right = 0;
            break;
        }
    }
    while(flag){
        if(current_s < str1.length())
            if(isdigit(str1[current_s])){
                str[current_c] = str1[current_s];
                current_s++;
```

```
            current_c++;
        } else{
            flag = 0;
        }
    else{
        break;
    }
}
while(!flag){
    if(current_s == str1.length())
        break;
    if(str1[current_s] == ' ')
        current_s++;
    else{
        is_right = 0;
        break;
    }
}
if(!is_right){
    cout << "неверное выражение" << endl;
    for(int i = 0; i < current_c; i++)
        str[i] = 0;
    flag = 0;
    current_s = 0;
    current_c = 0;
    continue;
}
complexity = atoi(str);
for(int i = 0; i < current_c; i++)
    str[i] = 0;
flag = 0;
current_s = 0;
current_c = 0;
```

```
}
ofstream f1;
f1.open("output_task.txt");
ofstream f2;
f2.open("output_ans.txt");
int max = variants;
int tmp_comp = 0;
while(variants){
    if(complexity == 0)
        break;
    tmp_comp = complexity;
    elem* root = new elem;
    root->left = NULL;
    root->right = NULL;

    tmp = rand();
    f1 << max-variants+1 << ')' << tmp << "  ";
    root->val = tmp;
    f_height(root);
    tmp_comp--;
    while(tmp_comp){
        tmp = rand();
        f1 << tmp << "  ";
        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);
        }
        tmp_comp--;
    }
```

```
        cout << "Бинарное дерево:" << endl;
        PR_b_tree(root, NULL, 0);
        f1 << "\\\\" << '\n';
        f2 << max-variants+1 << ')';
        out_elem(f2, root);
        f2 << "\\\\" << '\n';
        del_b_tree(root);
        delete(root);
        variants--;
    }
    f1.close();
    f2.close();
    return 0;
}
int main(){
    work_with_console();
    return 0;
}
```