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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья поиска

Студент гр. 7383 _____ Тян Е.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург
2018

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: научиться применять бинарные деревья для решения задач поиска.

Формулировка задачи: нужно реализовать структуру – АВЛ-дерево. Требуется:

а) по заданному файлу F(типа File of *Elem*), все элементы, которого различны построить БДП определенного типа;

б) для построенного БДП проверить, входит ли в него элемент *e* типа *Elem*, и если входит, то удалить этот элемент *e* из дерева поиска.

Входные данные: перечисление узлов.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе используются главная функция (int main()) и дополнительные функции (int imax(int a,int b), void delete_avltree(T*tree), T *avltree_create(int key), int avltree_height(T *tree), T *avltree_right_rotate(T *tree), T*avltree_left_rotate(T*tree), int avltree_balance(T *tree), T* balance(T* tree), T* avltree_add(T *&tree, int key,bool &flag), void outBT(T* b,int i,bool left, T* ptr), T* find_min(T*tree), T* remove_min(T* tree), T* dele(T *&tree, int key)).

Параметры передаваемые в функцию int imax(int a,int b):

- a – высота левого поддерева;
- b – высота правого поддерева.

Параметры передаваемые в функцию void delete_avltree(T*tree):

- tree – АВЛ-дерево.

Параметры передаваемые в функцию T *avltree_create(int key):

- key – значение, которое нужно записать в узел.

Параметры передаваемые в функцию int avltree_height(T *tree):

- tree – АВЛ-дерево.

Параметры передаваемые в функцию T *avltree_right_rotate(T *tree):

- tree – АВЛ-дерево.

Параметры передаваемые в функцию T*avltree_left_rotate(T*tree):

- tree – АВЛ-дерево.

Параметры передаваемые в функцию int avltree_balance(T *tree):

- tree – АВЛ-дерево.

Параметры передаваемые в функцию T* balance(T* tree):

- tree – АВЛ-дерево.

Параметры передаваемые в функцию T* avltree_add(T *&tree, int key,bool &flag):

- tree – АВЛ-дерево;
- key – значение, которое нужно вставить в дерево.
- flag – идентификатор, показывающий, есть ли такое значение уже в

дереве.

Параметры передаваемые в функцию void outBT(T* b,int i,bool left, T* ptr):

- b – узел дерева;
- i – счетчик для корректного вывода отступов;
- left – флаг, показывающий был при родитель левым поддеревом;
- ptr – узел, хранящий корень.

Параметры передаваемые в функцию T* find_min(T*tree):

- tree – АВЛ-дерево.

Параметры передаваемые в функцию T* remove_min(T* tree):

- tree – АВЛ-дерево.

Параметры передаваемые в функцию T* dele(T *&tree, int key):

- tree – АВЛ-дерево;
- key – значение, которое нужно удалить из дерева.

В функции main() выводится меню на консоль, где можно выбрать число, соответствующее выполняемой операции. Считывается целое число и, при помощи switch(), выбирается необходимая опция. При выборе «1» пользователь вводит перечисление узлов. При выборе «2» программа считывает перечисление узлов из файла, имя которого пользователь может задать сам. При выборе «3» программа завершает работу. При выборе другого значения программа выводит сообщение: «Проверьте введенные данные и повторите попытку» и ожидает дальнейших указаний. Реализации программы, при считывании с консоли и из файла не отличаются, за исключением части считывания данных с консоли и считывания данных из файла. Далее программа ведет себя одинаково. В обоих случаях при считывании программа добавляет в АВЛ-дерево узел с только что считанным значением. Вызывается функция T* avltree_add(T *&tree, int key,bool &flag), которая, если дерева не существует или это пустой узел, вызывает функцию T *avltree_create(int key), которая создает узел и возвращает его адрес, если хотя бы один узел существует и значение, которое подается еще не находится

в дереве, то происходит проверка: если значение-ключ больше значения в корне узла, то нужно вставить значение в правое поддерево, в противном случае – в левое поддерево. В конце обязательно меняется высота каждого узла, т.е. вызывается функция int imax(int a, int b), которая присваивает высоте узла наибольшую высоту одного из поддеревьев, увеличенную на 1, и происходит балансировка дерева с обращением к функции T* balance(T* tree), которая в зависимости от того на сколько отличаются высоты правого и левого поддеревьев каждого узла выполняет малые или большие вращения. Если высота правого поддерева на 2 больше высоты левого поддерева и высота правого поддерева уровнем ниже меньше высоты левого поддерева, как это приведено на рис. 1, то выполняется большое левое вращение,

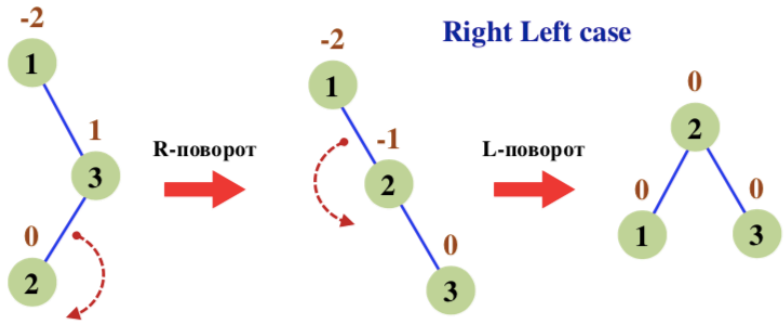

Рисунок 1 — Большое левое вращение

которое содержит малые правое и левое вращения. Если же высота правого

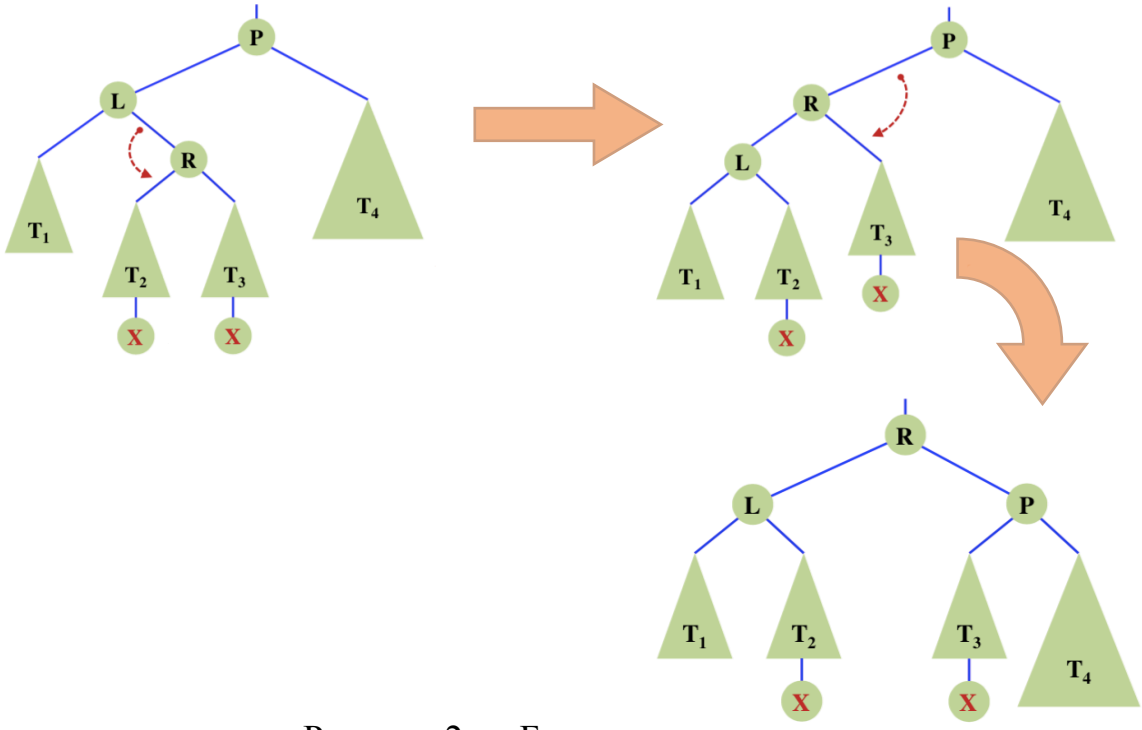

Рисунок 2 — Большое правое вращение

поддерева на 2 меньше высоты левого поддерева и высота правого поддерева уровнем ниже больше высоты левого поддерева уровнем ниже, то выполняется большое правое вращение, иллюстрация к которому приведена на рис. 2, которое содержит малые левое и правое вращения. Если высота правого поддерева на 2 больше высоты левого поддерева и высота правого поддерева уровнем ниже больше высоте левого поддерева, как это приведено на рис. 3, то выполняется малое левое вращение. Если же высота правого

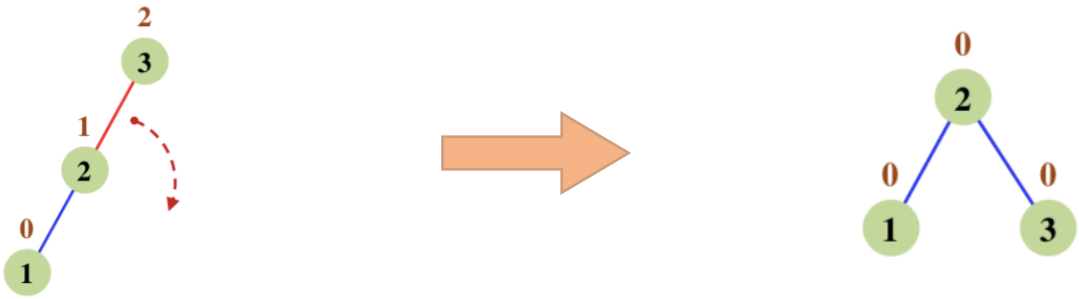

Рисунок 3 — Малое левое вращение

поддерева на 2 меньше высоты левого поддерева и высота правого поддерева уровнем ниже меньше высоты левого поддерева уровнем ниже, то выполняется малое правое вращение, иллюстрация к которому приведена на рис. 4.

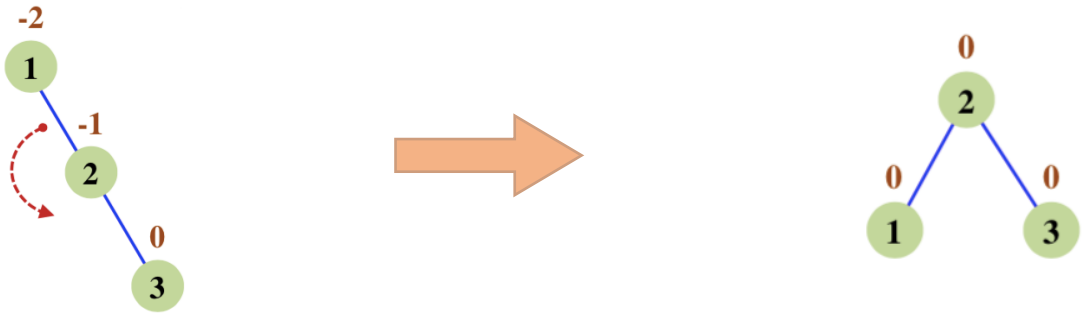

Рисунок 4 — Малое правое вращение

Если ставка прошла успешно, то выводится дерево, и программа продолжает работу пока не закончатся узлы дерева. Далее выводится окончательный результат построенного дерева, с помощью функции void outBT(T* b,int i,bool left, T* ptr).

Далее программа спрашивает, сколько узлов нужно удалит и просит и ввести по очереди, через запятую. После, программа начинает удалить

каждый узел, вызывая функцию T* dele(T *&tree, int key), которая проходится по дереву влево или вправо в зависимости от того, когда меньше или больше «ключ» соотвественно. Если такой ключ в дереве найден, то происходит его удаление и замена удаленного узла на самый малый узел в правом поддереве, удаленного узла, используя функции T* find_min(T*tree) и T* remove_min(T* tree), далее опять происходит балансировка дерева, полученного в результате удаления узла. После, выводится дерево с удаленным узлом.

Рассмотрим пример работы программы при записи из файла и укажем файл «test2.txt». Последующие шаги работы программы приведены на рис. 5, рис. 6.

Файл «test2.txt» содержит строку: 5

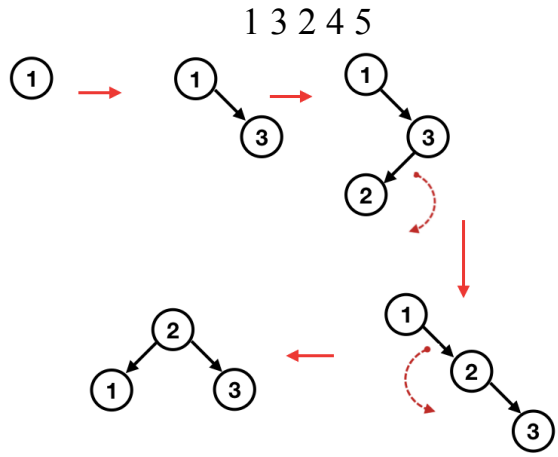

Рисунок 5 — Иллюстрация шагов 1, 2 и 3

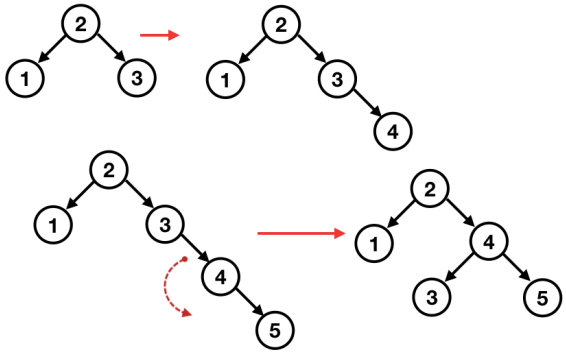

Рисунок 6 — Иллюстрация шагов 4 и 5

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе G++ в OS Linux Ubuntu 12.04. Программа может быть скомпилирована с помощью команды:

g++ <имя файла>.cpp

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно увидеть, что тестовые случаи не выявили некорректной работы программы, что говорит о том, что по результатам тестирования было показано: поставленная задача была выполнена.

4. ВЫВОД

В ходе выполнения лабораторной работы было изучено применение бинарных деревьев для решения задач поиска, а именно реализация АВЛ-деревьев для решения задач поиска, на языке C++. Также было осуществлено удаление и вставка узлов в АВЛ-дерево.

Была реализована программа строящая АВЛ-дерево по заданным перечислениям узлов, удаляющая необходимые узлы и выводящая изображение АВЛ-дерева на экран.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

№	Ввод	Вывод
1	Input quantity of nodes: 7 Input nodes: 0 13 67 923 -3 29 5 How many nodes will be deleted? 3 Input nodes to be deleted: -3 67 29	Built AVL-tree: .---(923) .---(67) `---(29) ---(13) .---(5) `---(0) `---(-3) After deleting: .---(923) ---(13) .---(5) `—(0)
2	Input quantity of nodes: 3 Input nodes: 1 2 3 How many nodes will be deleted? 3 Input nodes to be deleted: 2, 3, 1	Built AVL-tree: .---(3) ---(2) `---(1) After deleting:
3	Input quantity of nodes: 3 Input nodes: 1 2 1 2 3 How many nodes will be deleted? 0	This symbol exists. Try again. This symbol exists. Try again. Built AVL-tree: .---(3) ---(2) `---(1) After deleting:

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

```
#include <iostream>
#include <cctype>
#include <cstring>
#include <fstream>

#define N 100

using namespace std;

typedef struct Node {
    int key;
    Node* left;
    Node* right;
    int height;
}T;

int imax(int a,int b){
    return (a-b>0) ? a : b;
}

void delete_avltree(T*tree){
    if(tree==NULL)
        return;
    delete_avltree(tree->left);
    delete_avltree(tree->right);
    delete(tree);
}

T *avltree_create(int key){
    T *node=new T;
    if (node != NULL) {
        node->key = key;
        node->left = NULL;
        node->right = NULL;
        node->height = 0;
    }
    return node;
}

int avltree_height(T *tree){
    return (tree != NULL) ? tree->height : -1;
}

T *avltree_right_rotate(T *tree){
    T *left;
    left = tree->left;
    tree->left = left->right;
    left->right = tree;
    tree->height = imax(avltree_height(tree->left),avltree_height(tree->right)) + 1;
    left->height = imax(avltree_height(left->left),tree->height) + 1;
    return left;
```

```
}

T*avltree_left_rotate(T*tree){
    T *right;
    right = tree->right;
    tree->right = right->left;
    right->left = tree;
    tree->height = imax(avltree_height(tree->left),avltree_height(tree->right)) + 1;
    right->height = imax(avltree_height(right->right),tree->height) + 1;
    return right;
}

int avltree_balance(T *tree){
    return ((avltree_height(tree->right) - avltree_height(tree->left)));
}

T* balance(T* tree){
   if(avltree_balance(tree) == 2){
        if(avltree_balance(tree->right)<0)
            tree->right=avltree_right_rotate(tree->right);
        return avltree_left_rotate(tree);
    }
    if(avltree_balance(tree)==-2){
        if(avltree_balance(tree->left)>0)
            tree->left=avltree_left_rotate(tree->left);
        return avltree_right_rotate(tree);
    }
    return tree;
}

T* avltree_add(T *&tree, int key,bool &flag){
    if (tree == NULL){
        return avltree_create(key);
    }
    if(key==tree->key){
        flag=false;
    }
    if (key < tree->key) {
        tree->left = avltree_add(tree->left,key,flag);
    }else if (key > tree->key) {
        tree->right = avltree_add(tree->right,key,flag);
    }
    tree->height = imax(avltree_height(tree->left),avltree_height(tree->right)) + 1;
    return balance(tree);
}

void outBT(T* b,int i,bool left, T* ptr){
    if (b!=NULL){
        outBT(b->right,i+4,true,b);
        if(ptr==NULL){
            for(int k=0;k<i;k++)
                cout<<" ";
            cout<<"---";
            cout << "("<<b->key<<")"<<endl;
        }else if(left){
```

```
            for(int k=0;k<i;k++)
                cout<<" ";
            cout<<".---";
            cout << "("<<b->key<<")"<<endl;
        }else{
            for(int k=0;k<i;k++)
                cout<<" ";
            cout<<"`---";
            cout << "("<<b->key<<")"<<endl;
        }
        outBT(b->left,i+4,false,b);
    }else
        return;
}

T* find_min(T*tree){
    return tree->left ? find_min(tree->left) : tree;
}

T* remove_min(T* tree){
    if(tree->left==NULL)
        return tree->right;
    tree->left=remove_min(tree->left);
    return balance(tree);
}

T* dele(T *&tree, int key){
    if (tree == NULL){
        return NULL;
    }
    if(key==tree->key){
        T*q=tree->left;
        T*r=tree->right;
        delete tree;
        if(!r){
            return q;
        }
        T* min;
        min=find_min(tree->right);
        min->right=remove_min(r);
        min->left=q;
        return balance(min);
    }
    if (key < tree->key) {
        tree->left=dele(tree->left,key);
    }else if (key > tree->key) {
        tree->right=tree->right=dele(tree->right,key);
    }
    tree->height = imax(avltree_height(tree->left),avltree_height(tree->right)) + 1;
    return balance(tree);
}

int main(){
    int var=0;
    while(var != 3){
```

```
        cout << "Выберите дальнейшие действия и введите цифру:"<<endl;
        cout << "1. Ввести перечисления узлов вручную."<<endl;
                    cout << "2. Считать перечисления узлов из выбранного файла <имя
файла>.txt."<<endl;
        cout << "3. Завершить работу."<<endl;
        cin >> var;
        switch(var){
           case 1:{
              int q,num;
              cout<<"Input quantity of nodes: ";
              cin>>q;
              cout<<"Input nodes: ";
              T* root=NULL;
              T*b=NULL;
              for(int i=0;i<q;i++){
                 cin>>num;
                 bool flag=true;
                 root=avltree_add(root,num,flag);
                 if(flag==false){
                    cout<<"This symbol exists. Try again."<<endl;
                    i--;
                 }
                 if(flag==true){
                    b=NULL;
                    cout<<"Was inputed: "<<num<<endl;
                    outBT(root,0,false,b);
                 }
              }
              b=NULL;
              cout<<"Built AVL-tree:"<<endl;
              outBT(root,0,false,b);
              q=0;
              cout<<"How many nodes will be deleted: ";
              cin>>q;
              if(q!=0){
                 getchar();
                 int a[q],val=0;
                 cout<<"Input values to be deleted: ";
                 char str[N];
                 fgets(str,N,stdin);
                 char * pch = strtok (str,",");
                 while (pch != NULL){
                    val=atoi(pch);
                    b=NULL;
                    cout<<"Value to be deleted: "<<val<<endl;
                    root=dele(root,val);
                    outBT(root,0,false,b);
                    val=0;
                    pch = strtok (NULL, ",");
                 }
              }
              delete_avltree(root);
              break;
           }
           case 2:{
```

```
            int num,q;
            getchar();
            char fname[N];
            printf("Choose file\n");
            fgets(fname,N,stdin);
            fname[strlen(fname)-1]='\0';
            fstream myfile(fname,ios_base::in);
            T*root=NULL;
            T* b;
            while (myfile >> num){
               bool flag=true;
               root=avltree_add(root,num,flag);
               if(flag==false){
                  cout<<"This symbol exists: "<<num<<endl;
               }else{
                  b=NULL;
                  cout<<"Was inputed: "<<num<<endl;
                  outBT(root,0,false,b);
               }
            }
            b=NULL;
            cout<<"Built AVL-tree:"<<endl;
            outBT(root,0,false,b);
            q=0;
            cout<<"How many nodes will be deleted: ";
            cin>>q;
            if(q!=0){
               getchar();
               int a[q],val=0;
               cout<<"Input values to be deleted: ";
               char str[N];
               fgets(str,N,stdin);
               char * pch = strtok (str,",");
               while (pch != NULL){
                  val=atoi(pch);
                  b=NULL;
                  cout<<"Value to be deleted: "<<val<<endl;
                  root=dele(root,val);
                  outBT(root,0,false,b);
                  val=0;
                  pch = strtok (NULL, ",");
               }
            }
            delete_avltree(root);
            break;
         }
         case 3:
            return 0;
         default:
            cerr << "Проверьте введенные данные и повторите попытку." << endl;
            break;
      }
   }
   return 0;
}
```