AVL-Tree
Borys Bondos

1. ***Task***

The aim of the exercise was to create AVL (Height-Balanced) Tree which is able to store data of arbitrary type (templates). Every node has got unique key and info which is not unique. Position of each node is determined by the value of key. Balance factor is defined as difference between left subtree and right subtree of a given node. To obtain balance factor we have to call method int diff(node*Ptr) on given node.

2. ***Design of the class***

In private section of the class I created structure which keeps all data of specified node. Each node has got place for key, info and stores pointers to the left subtree and the right subtree. I defined also pointer root which is the root of the whole data structure.

The definition of the class Dictionary contains several member functions that are private members of the class. These functions are used to implement the public member functions of the class and the user does not need to know about their existence.

In public section of the class I created all methods necessary to use the program such as adding methods, removing methods, printing methods, constructors and destructor. I defined also iterator with it' s methods to give access to read private data such as information about the root. ***3. Description of the methods*** *Public part:*

Dictionary(); //constructor

~Dictionary(); //destructor

Dictionary(const Dictionary &source); //copy constructor void

add_node(const Key &k, const Info &i); //add node void print_in_order();

//print all nodes with info about balance factor In order

void print_reverse_order();//print all nodes with info about balance factor in reverse order

void print(); //print whole tree void print_leaves();//print

nodes which do not have children void print_by_info(Info

info); //print all nodes with given info void remove(Key key);

//remove node with given key void remove_tree();//remove

whole tree

Dictionary &operator = (const Dictionary &source); //overloaded operator

Dictionary &operator + (const Dictionary &n); //overloaded operator

Dictionary &operator - (const Dictionary &n); //overloaded operator ***Private***

part:

node* add_node_private(Key k, Info i, node *Ptr); //method for void add_node(const Key &k, const Info &i); int height_private(node *Ptr); //returns the height starting from given

point int diff(node *Ptr); //returns the balance factor (left side – right side) void

print_in_order_private(node *Ptr); // prints in order void

print_reverse_order_private(node *Ptr); //prints reverse order void print_private(node

root, int space); //prints whole tree node remove_private(Key k, node *Ptr); //removes

node by key node* find_smallest(node* Ptr); //finds smallest node starting from Ptr void

remove_tree_private(node *&Ptr); //removes tree node* copy_tree_private(node *lhs,

node * const source); //copy the tree source to the lhs void plus_operator_helper(node *

const n); //helper method for overloaded + void minus_operator_helper(node * const

n);//helper method for overloaded - node *balance(node *temp);//balancing the tree to

keep AVL node* rr_rotation(node *parent);//right-right rotation node *ll_rotation(node

*parent);//left-left rotation node *lr_rotation(node *parent);//left-right rotation node

*rl_rotation(node *parent);//right-left rotation void print_leaves_private(node *Ptr);//

prints the leaves void print_by_info_private(node* Ptr, Info condition);//prints nodes with

given info

Iterator methods:

Iterator begin()const - Returns root

Methods of class iterator:

Iterator() - constructor

Iterator(node *source) - copy constructor const

Iterator &operator = (const Iterator &copyIter) bool

operator == (const Iterator &compare)const

Iterator operator ++ () //moves iterator to right child

Iterator operator ++ (int)

Iterator operator -- () //moves iterator to left child

Iterator operator -- (int) key getKey() info

getInfo() bool isnull() void print_iterator()

bool operator != (const Iterator &compare)const

4. ***Testing***

Testing scenarios cover proper and improper usage of every method. Description of every test scenario is shown during execution of the program.