Міністерство освіти і науки України

Національний технічний університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра «Системного програмування і спеціалізованих комп’ютерних систем»

Лабораторна робота №4

з дисципліни «Операційні системи»   
на тему: «Файлові системи»

Виконав:

студент IV курсу,

група КВ-41

Яковенко Максим

Перевірив:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ-2017

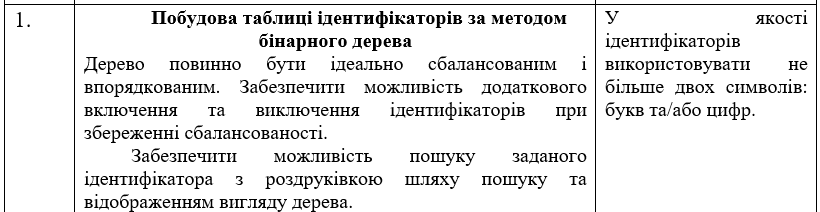
**Завдання:**

1. Написати програму, що моделює роботу складових заданої файлової системи згідно варіанта (перелік варіантів представлений нижче у табл.4.3).

Вхідні дані студент задає самостійно з урахуванням особливостей індивідуального варіанта завдання.

1. Зробити візуалізацію роботи програми і кінцевих результатів на різних наборах вхідних даних.

Варіант 1 (16mod15)



**Лістинг**

**avltree.hpp**

#pragma once

#include <iostream>

#include <memory>

#include <algorithm>

//avl tree

template<typename T>

class avltree{

private:

struct \_node;

//interface functions

public:

avltree<T> () : nil (new \_node){

root = nil;

root->left = nil;

root->right = nil;

root->parent = nil;

root->balance\_factor = 0;

}

~avltree<T> (){

ClearPath (root);

delete nil;

}

bool Insert (T key){

\_node\* node = new \_node{ 0,key, nil,nil,nil };

\_node\* y\_tmp\_node = nil;

\_node\* x\_tmp\_node = root;

while (x\_tmp\_node != nil){

y\_tmp\_node = x\_tmp\_node;

if (node->key < x\_tmp\_node->key){

x\_tmp\_node = x\_tmp\_node->left;

} else{

x\_tmp\_node = x\_tmp\_node->right;

}

}

node->parent = y\_tmp\_node;

if (y\_tmp\_node == nil){

root = node;

} else{

if (node->key < y\_tmp\_node->key){

y\_tmp\_node->left = node;

} else{

y\_tmp\_node->right = node;

}

Rebalance (y\_tmp\_node);

}

return true;

}

bool Delete (T key){

\_node\* node\_to\_delete = FindNodeByKey (root, key);

\_node\* tmp=nullptr;

if (node\_to\_delete != nil){

if (node\_to\_delete->left == nil)

Transplant (node\_to\_delete, node\_to\_delete->right);

else{

if(node\_to\_delete->right == nil)

Transplant (node\_to\_delete, node\_to\_delete->left);

else{

tmp = TreeMinimum (node\_to\_delete->right);

if (tmp->parent != node\_to\_delete){

Transplant (tmp, tmp->right);

tmp->right = node\_to\_delete->right;

tmp->right->parent = tmp;

}

Transplant (node\_to\_delete, tmp);

tmp->left = node\_to\_delete->left;

tmp->left->parent = tmp;

}

}

Rebalance (tmp);

return true;

}

return false;

}

bool Find (T key){

return (FindNodeByKey (root, key, true) != nil);

}

void PrintTree (){

PrintNode (root);

}

//private methods

private:

void Rebalance (\_node\* node){

SetBalanceFactor (node);

if (node->balance\_factor == -2){

if (Height(node->left->left) >= Height (node->left->right))

node = RotateRight (node);

else

node = RotateLeftRight (node);

} else{

if (node->balance\_factor == 2){

if (Height (node->right->right) >= Height (node->right->left))

node = RotateLeft (node);

else

node = RotateRightLeft (node);

}

}

if (node->parent != nil){

Rebalance (node->parent);

} else{

root = node;

}

}

void PrintNode (\_node\* node = root, unsigned int node\_level=0){

if (node != nil){

auto level\_buf = node\_level;

while (level\_buf--){

std::cout << "||";

}

std::cout << node->key<<" || Node height: "<<Height(node)<< " || Balance Factor: " << node->balance\_factor<< std::endl;

PrintNode (node->left, node\_level + 1);

PrintNode (node->right, node\_level + 1);

}

}

void ClearPath (\_node\* node){

if (node->left != nil){

ClearPath (node->left);

}

if (node->right != nil){

ClearPath (node->right);

}

delete node;

}

\_node\* RotateLeft (\_node\* node){

\_node\* tmp = node->right;

tmp->parent = node->parent;

node->right = tmp->left;

if (node->right != nil){

node->right->parent = node;

}

tmp->left = node;

node->parent = tmp;

if (tmp->parent != nil){

if (tmp->parent->right == node)

tmp->parent->right = tmp;

else

tmp->parent->left = tmp;

}

SetBalanceFactor (node);

SetBalanceFactor (tmp);

return tmp;

}

\_node\* RotateRight (\_node\* node){

\_node\* tmp = node->left;

tmp->parent = node->parent;

node->left = tmp->right;

if (node->left != nil){

node->left->parent = node;

}

tmp->right = node;

node->parent = tmp;

if (tmp->parent != nil){

if (tmp->parent->right == node)

tmp->parent->right = tmp;

else

tmp->parent->left = tmp;

}

SetBalanceFactor (node);

SetBalanceFactor (tmp);

return tmp;

}

\_node\* RotateLeftRight (\_node\* node){

node->left = RotateLeft (node->left);

return RotateRight (node);

}

\_node\* RotateRightLeft (\_node\* node){

node->right = RotateRight (node->right);

return RotateLeft (node);

}

\_node\* FindNodeByKey (\_node\* node, T key, bool print\_path = false){

while (node != nil){

if (print\_path){

std::cout << node->key;

}

if (node->key == key){

std::cout << std::endl;

return node;

}

if (print\_path)

std::cout << "-->";

node = (key<node->key) ? node->left : node->right;

}

if (print\_path)

std::cout << "nil" << std::endl;

return node;

}

void SetBalanceFactor (\_node\* node){

node->balance\_factor = Height (node->right) - Height (node->left);

}

int32\_t Height (\_node\* node){

return (node != nil) ? (1+std::max(Height(node->left), Height(node->right))) : (-1);

}

\_node\* TreeMinimum (\_node\* node){

while (node->left != nil){

node = node->left;

}

return node;

}

void Transplant (\_node\* u\_node, \_node\* v\_node){

if (u\_node->parent == nil){

root = v\_node;

} else{

if (u\_node == u\_node->parent->left){

u\_node->parent->left = v\_node;

} else{

u\_node->parent->right = v\_node;

}

}

v\_node->parent = u\_node->parent;

}

//private data

private:

struct \_node{

int32\_t balance\_factor;

T key;

\_node\* left;

\_node\* right;

\_node\* parent;

};

\_node\* root;

\_node\* nil;

};

**Тестування**

**Тест 1**

avltree<std::string> test\_tree;

test\_tree.Insert ("a1");

test\_tree.Insert ("a2");

test\_tree.Insert ("a3");

test\_tree.Insert ("a4");

test\_tree.Insert ("a5");

test\_tree.Insert ("a6");

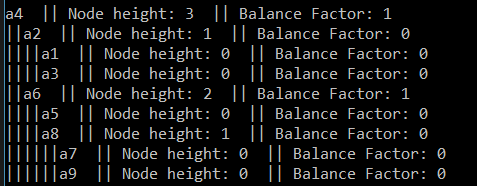
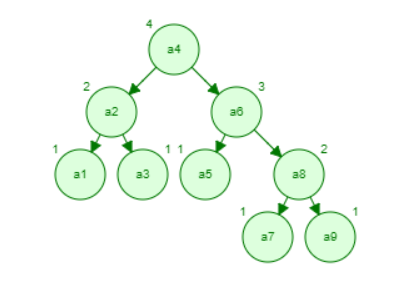
test\_tree.Insert ("a7");

test\_tree.Insert ("a8");

test\_tree.Insert ("a9");

test\_tree.PrintTree ();

Вигляд avl-дерева в програмі: Теоретичний вигляд avl-дерева:

**Тест 2**

avltree<std::string> test\_tree;

test\_tree.Insert ("9a");

test\_tree.Insert ("8a");

test\_tree.Insert ("7a");

test\_tree.Insert ("6a");

test\_tree.Insert ("5a");

test\_tree.Insert ("4a");

test\_tree.Insert ("3a");

test\_tree.Insert ("2a");

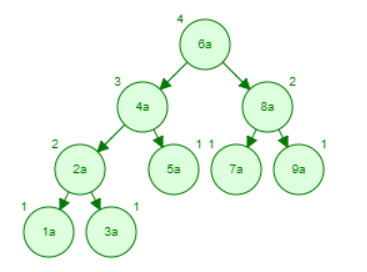
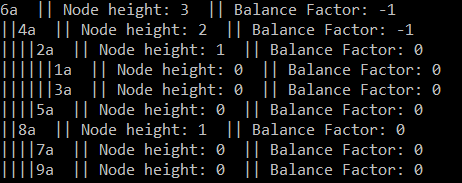
test\_tree.Insert ("1a");

test\_tree.PrintTree ();

Теоретичний шлях до вузла 2а: 6а->4a->2a

Шлях в програмі:



Вигляд avl-дерева в програмі: Теоретичний вигляд avl-дерева:

**Тест 3**

avltree<std::string> test\_tree;

test\_tree.Insert ("9a");

test\_tree.Insert ("8a");

test\_tree.Insert ("7a");

test\_tree.Insert ("6a");

test\_tree.Insert ("5a");

test\_tree.Insert ("4a");

test\_tree.Insert ("3a");

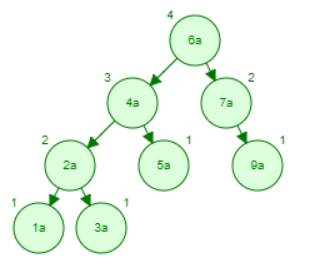
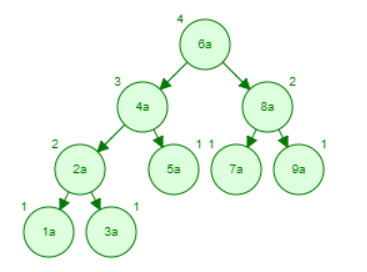
test\_tree.Insert ("2a");

test\_tree.Insert ("1a");

test\_tree.Delete ("3a");

test\_tree.PrintTree ();

Вигляд дерева до видалення вузла 8а: Після видалення вузла 8а:



Вигляд дерева в програмі:

