Звіт до лабораторної роботи 2

Алгоритми та структури даних

ІПЗ-11/1

Студент: Зганяйко Володимир Віталійович

Викладач: Бичков Олексій Сергійович

1. Умова завдання лабораторної роботи.

Використовуючи структуру даних: бінарне дерево, пошуку, червоно-чорне дерево та АВЛ дерево реалізувати такі алгоритми:

* Додавання вузла;
* Видалення вузла;
* Лівий, правий повороти;
* Перевірка на перебування у структурі;
* Перефарбування братніх вузлів у протилежний колір.

1. Аналіз задачі.

Для зручності роботи з алгоритмами, будуть використані структури дерев, що складаються з цілих чисел. Для кожної структури буде створено окремо клас дерева та окремо клас вузла дерева, класи дерев будуть у новому файлі, виклик методів дерев буде здійснюватись через клас Program

Для зручної роботи програми буде створене меню, за яким користувач зможе обрати з якою структурою даних він хоче працювати. Крім того, буде створено окреме меню для кожної структури, де користувач зможе обрати методи з якими він хоче працювати.

Для перевірки ефективності кожного алгоритму та для визначення, яка структура даних є кращою, буде створений об’єкт класу Stopwatch, методи якого будуть визначати час затрачений на виконання методу.

1. Структура основних вхідних та вихідних даних.

Основними вхідними даними, які будуть введенні з клавіатури: пункт меню вибору структури даних, пункт меню вибору методу, значення елементу для додавання, для видалення та для пошуку.

Вихідні дані, які будуть отримані у результаті виконання програми: повідомлення чи був елемент доданий чи видалений чи знайдений, а також елементи дерева у порядку зростання чи спадання.

1. Алгоритм розв’язання задачі.

**Для бінарного дерева пошуку**

**Алгоритм додавання вузла:**

public void Add(int data)

{

if (Head == null)

{

Head = new BinaryTreeNode(data);

}

else

{

AddTo(Head, data);

Count++;

}

}

public void AddTo(BinaryTreeNode node, int data)

{

if (data.CompareTo(node.Data) < 0)

{

if (node.Left == null)

{

node.Left = new BinaryTreeNode(data);

}

else

{

AddTo(node.Left, data);

}

}

else

{

if (node.Right == null)

{

node.Right = new BinaryTreeNode(data);

}

else

{

AddTo(node.Right, data);

}

}

}

**Алгоритм видалення вузла:**

public bool Remove(int data)

{

BinaryTreeNode current;

BinaryTreeNode parent;

current = FindWithParent(data, out parent);

if (current == null)

{

return false;

}

Count--;

if (current.Right == null)

{

if (parent == null)

{

Head = current.Left;

}

else

{

int result = parent.CompareTo(current.Data);

if (result > 0)

{

parent.Left = current.Left;

}

else if (result < 0)

{

parent.Right = current.Left;

}

}

}

else if (current.Right.Left == null)

{

current.Right.Left = current.Left;

if (parent == null)

{

Head = current.Right;

}

else

{

int result = parent.CompareTo(current.Data);

if (result > 0)

{

parent.Left = current.Right;

}

else if (result < 0)

{

parent.Right = current.Right;

}

}

}

else

{

BinaryTreeNode leftmost = current.Right.Left;

BinaryTreeNode leftmostParent = current.Right;

while (leftmost.Left != null)

{

leftmostParent = leftmost;

leftmost = leftmost.Left;

}

leftmostParent.Left = leftmost.Right;

leftmost.Left = current.Left;

leftmost.Right = current.Right;

if (parent == null)

{

Head = leftmost;

}

else

{

int result = parent.CompareTo(current.Data);

if (result > 0)

{

parent.Left = leftmost;

}

else if (result < 0)

{

parent.Right = leftmost;

}

}

}

return true;

}

**Алгоритм пошуку елемента:**

public BinaryTreeNode FindWithParent(int data, out BinaryTreeNode parent)

{

parent = null;

BinaryTreeNode current = Head;

while (current != null)

{

int result = current.CompareTo(data);

if (result < 0)

{

parent = current;

current = current.Right;

}

else if (result > 0)

{

parent = current;current = current.Left;

}

else

{

break;

}

}

return current;

}

**Алгоритм виведення елементів у порядку спадання (здійснюється через цикл foreach):**

public IEnumerator GetEnumerator()

{

return PrintFromHighToLow();

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public IEnumerator PrintFromHighToLow()

{

if (Head != null)

{

Stack<BinaryTreeNode> stack = new Stack<BinaryTreeNode>();

BinaryTreeNode current = Head;

bool flag = true;

stack.Push(current);

while (stack.Count > 0)

{

if (flag)

{

while (current.Right != null)

{

stack.Push(current);

current = current.Right;

}

}

yield return current.Data;

if (current.Left != null)

{

current = current.Left;

flag = true;

}

else

{

current = stack.Pop();

flag = false;

}

}

}

}

**Для АВЛ дерева**

**Алгоритм додавання вузла:**

public void Add(int data)

{

if (Head == null)

{

Head = new AVL\_TreeNode(data, null, this);

}

else

{

AddTo(Head, data);

}

Count++;

}

public void AddTo(AVL\_TreeNode node, int data)

{

if (data.CompareTo(node.Data) < 0)

{

if (node.left == null)

{

node.left = new AVL\_TreeNode(data, node, this);

}

else

{

AddTo(node.left, data);

}

}

else

{

if (node.right == null)

{

node.right = new AVL\_TreeNode(data, node, this);

}

else

{

AddTo(node.right, data);

}

}

}

**Алгоритм видалення вузла:**

public bool Remove(int data)

{

AVL\_TreeNode current;

current = Find(data);

if (current == null)

{

return false;

}

AVL\_TreeNode treeToBalance = current.Parent;

Count--;

if (current.right == null)

{

if (current.Parent == null)

{

Head = current.left;

if (Head != null)

{

Head.Parent = null;

}

}

else

{

int result = current.Parent.CompareTo(current.Data);

if (result > 0)

{

current.Parent.left = current.left;

}

else if (result < 0)

{

current.Parent.right = current.left;

}

}

}

else if (current.right.left == null)

{

current.right.left = current.left;

if (current.Parent == null)

{

Head = current.right;

if (Head != null)

{

Head.Parent = null;

}

}

else

{

int result = current.Parent.CompareTo(current.Data);

if (result > 0)

{

current.Parent.left = current.right;

}

else if (result < 0)

{

current.Parent.right = current.right;

}

}

}

else

{

AVL\_TreeNode leftmost = current.right.left;

while (leftmost.left != null)

{

leftmost = leftmost.left;

}

leftmost.Parent.left = leftmost.right;

leftmost.left = current.left;

leftmost.right = current.right;

if (current.Parent == null)

{

Head = leftmost;

if (Head != null)

{

Head.Parent = null;

}

}

else

{

int result = current.Parent.CompareTo(current.Data);

if (result > 0)

{

current.Parent.left = leftmost;

}

else if (result < 0)

{

current.Parent.right = leftmost;

}

}

}

if (treeToBalance != null)

{

treeToBalance.Balance();

}

else

{

if (Head != null)

{

Head.Balance();

}

}

return true;

}

**Алгоритм пошуку елемента:**

public bool Contains(int data)

{

return Find(data) != null;

}

public AVL\_TreeNode Find(int data)

{

AVL\_TreeNode current = Head;

while (current != null)

{

int result = current.CompareTo(data);

if (result > 0)

{

current = current.left;

}

else if (result < 0)

{

current = current.right;

}

else

{

break;

}

}

return current;

}

**Алгоритм виведення елементів у порядку спадання:**

public IEnumerator InOrderTraversal()

{

if (Head != null)

{

Stack<AVL\_TreeNode> stack = new Stack<AVL\_TreeNode>();

AVL\_TreeNode current = Head;

bool goLeftNext = true;

stack.Push(current);

while (stack.Count > 0)

{

if (goLeftNext)

{

while (current.left != null)

{

stack.Push(current);

current = current.left;

}

}

yield return current.Data;

if (current.right != null)

{

current = current.right;

goLeftNext = true;

}

else

{

current = stack.Pop();

goLeftNext = false;

}

}

}

}

public IEnumerator GetEnumerator()

{

return InOrderTraversal();

}

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

**Додаткові методи:**

**Балансування:**

public void Balance()

{

if (State == TreeState.RightHeavy)

{

if (right != null && right.BalanceFactor < 0)

{

LeftRightRotation();

}

else

{

LeftRotation();

}

}

else if (State == TreeState.LeftHeavy)

{

if (left != null && left.BalanceFactor > 0)

{

RightLeftRotation();

}

else

{

RightRotation();

}

}

}

**Лівий та правий повороти:**

public void LeftRotation()

{

AVL\_TreeNode newRoot = right;

ReplaceRoot(newRoot);

right = newRoot.left;

newRoot.left = this;

}

public void RightRotation()

{

AVL\_TreeNode newRoot = left;

ReplaceRoot(newRoot);

left = newRoot.right;

newRoot.right = this;

}

public void LeftRightRotation()

{

right.RightRotation();

LeftRotation();

}

public void RightLeftRotation()

{

left.LeftRotation();

RightRotation();

}

**Заміна вузла:**

public void ReplaceRoot(AVL\_TreeNode newRoot)

{

if (this.Parent != null)

{

if (this.Parent.left == this)

{

this.Parent.left = newRoot;

}

else if (this.Parent.right == this)

{

this.Parent.right = newRoot;

}

}

else

{

Tree.Head = newRoot;

}

newRoot.Parent = this.Parent;

this.Parent = newRoot;

}

**Для Червоно-чорного дерева**

**Алгоритм додавання вузла:**

public void Insert(int data)

{

Node newItem = new Node(data);

if (Root == null)

{

Root = newItem;

Root.Colour = Colour.Black;

return;

}

Node? node = null;

Node? Current = Root;

while (Current != null)

{

node = Current;

if (newItem.Data < Current.Data)

{

Current = Current.Left;

}

else

{

Current = Current.Right;

}

}

newItem.Parent = node;

if (node == null)

{

Root = newItem;

}

else if (newItem.Data < node.Data)

{

node.Left = newItem;

}

else

{

node.Right = newItem;

}

newItem.Left = null;

newItem.Right = null;

newItem.Colour = Colour.Red;

InsertFixUp(newItem);

}

**Алгоритм видалення дерева:**

public void Delete(int key)

{

Node item = FindElement(key);

Node node1 = null;

Node node2 = null;

if (item == null)

{

Console.WriteLine("Такого елементу немає в дереві.");

return;

}

if (item.Left == null || item.Right == null)

{

node2 = item;

}

else

{

node2 = TrueSuccessor(item);

}

if (node2.Left != null)

{

node1 = node2.Left;

}

else

{

node1 = node2.Right;

}

if (node1 != null)

{

node1.Parent = node2;

}

if (node2.Parent == null)

{

Root = node1;

}

else if (node2 == node2.Parent.Left)

{

node2.Parent.Left = node1;

}

else

{

node2.Parent.Left = node1;

}

if (node2 != item)

{

item.Data = node2.Data;

}

if (node2.Colour == Colour.Black)

{

DeleteFixUp(node1);

}

}

**Алгоритм пошуку елемента:**

public Node? FindElement(int key)

{

bool flag = false;

Node? temp = Root;

Node? item = null;

while (!flag && temp != null)

{

if (key > temp.Data)

{

temp = temp.Right;

}

else if (key < temp.Data)

{

temp = temp.Left;

}

else

{

flag = true;

item = temp;

}

}

if (flag)

{

Console.WriteLine("Елемент знайдено.");

return item;

}

else

{

Console.WriteLine("Елемент відсутній.");

return null;

}

}

**Алгоритм виведення елементів у порядку спадання:**

public void DisplayTree()

{

if (Root == null)

{

Console.WriteLine("Елементів в дереві немає!");

}

else

{

RecursiveDisplay(Root);

}

}

public void RecursiveDisplay(Node? current)

{

if (current != null)

{

RecursiveDisplay(current.Left);

if (current.Colour == Colour.Red) Console.ForegroundColor = ConsoleColor.Red;

else Console.ForegroundColor = ConsoleColor.Gray;

Console.Write(current.Data + " ");

RecursiveDisplay(current.Right);

}

}

**Додаткові методи:**

**Балансування після вставки:**

public void InsertFixUp(Node? item)

{

while (item != Root && item.Parent.Colour == Colour.Red)

{

if (item.Parent == item.Parent.Parent.Left)

{

Node? node = item.Parent.Parent.Right;

if (node != null && node.Colour == Colour.Red)

{

item.Parent.Colour = Colour.Black;

node.Colour = Colour.Black;

item.Parent.Parent.Colour = Colour.Red;

item = item.Parent.Parent;

}

else

{

if (item == item.Parent.Right)

{

item = item.Parent;

LeftRotate(item);

}

item.Parent.Colour = Colour.Black;

item.Parent.Parent.Colour = Colour.Red;

RightRotate(item.Parent.Parent);

}

}

else

{

Node? node = null;

node = item.Parent.Parent.Left;

if (node != null && node.Colour == Colour.Black)

{

item.Parent.Colour = Colour.Red;

node.Colour = Colour.Red;

item.Parent.Parent.Colour = Colour.Black;

item = item.Parent.Parent;

}

else

{

if (item == item.Parent.Left)

{

item = item.Parent;

RightRotate(item);

}

item.Parent.Colour = Colour.Black;

item.Parent.Parent.Colour = Colour.Red;

LeftRotate(item.Parent.Parent);

}

}

Root.Colour = Colour.Black;

}

}

**Балансування після видалення:**

public void DeleteFixUp(Node? item)

{

while (item != null && item != Root && item.Colour == Colour.Black)

{

if (item == item.Parent.Left)

{

Node? node = item.Parent.Right;

if (node.Colour == Colour.Red)

{

node.Colour = Colour.Black;

item.Parent.Colour = Colour.Red;

LeftRotate(item.Parent);

node = item.Parent.Right;

}

if (node.Left.Colour == Colour.Black && node.Right.Colour == Colour.Black)

{

node.Colour = Colour.Red;

item = item.Parent;

}

else if (node.Right.Colour == Colour.Black)

{

node.Left.Colour = Colour.Black;

node.Colour = Colour.Red;

RightRotate(node);

node = item.Parent.Right;

}

node.Colour = item.Parent.Colour;

item.Parent.Colour = Colour.Black;

node.Right.Colour = Colour.Black;

LeftRotate(item.Parent);

item = Root;

}

else

{

Node? node = item.Parent.Left;

if (node.Colour == Colour.Red)

{

node.Colour = Colour.Black;

item.Parent.Colour = Colour.Red;

RightRotate(item.Parent);

node = item.Parent.Left;

}

if (node.Right.Colour == Colour.Black && node.Left.Colour == Colour.Black)

{

node.Colour = Colour.Black;

item = item.Parent;

}

else if (node.Left.Colour == Colour.Black)

{

node.Right.Colour = Colour.Black;

node.Colour = Colour.Red;

LeftRotate(node);

node = item.Parent.Left;

}

node.Colour = item.Parent.Colour;

item.Parent.Colour = Colour.Black;

node.Left.Colour = Colour.Black;

RightRotate(item.Parent);

item = Root;

}

}

if (item != null)

{

item.Colour = Colour.Black;

}

}

**Пошук мінімального:**

public Node? TrueSuccessor(Node? item)

{

if (item.Left != null)

{

return Minimum(item);

}

else

{

Node? node = item.Parent;

while (node != null && item == node.Right)

{

item = node;

node = node.Parent;

}

return node;

}

}

public Node? Minimum(Node? item)

{

while (item.Left.Left != null)

{

item = item.Left;

}

if (item.Left.Right != null)

{

item = item.Left.Right;

}

return item;

}

**Лівий та правий повороти:**

public void LeftRotate(Node? Current)

{

Node? node = Current.Right;

Current.Right = node.Left;

if (node.Left != null)

{

node.Left.Parent = Current;

}

if (node != null)

{

node.Parent = Current.Parent;

}

if (Current.Parent == null)

{

Root = node;

}

if (Current == Current.Parent.Left)

{

Current.Parent.Left = node;

}

else

{

Current.Parent.Right = node;

}

node.Left = Current;

if (Current != null)

{

Current.Parent = node;

}

}

public void RightRotate(Node? Current)

{

Node? node = Current.Left;

Current.Left = node.Right;

if (node.Right != null)

{

node.Right.Parent = Current;

}

if (node != null)

{

node.Parent = Current.Parent;

}

if (Current.Parent == null)

{

Root = node;

}

if (Current == Current.Parent.Right)

{

Current.Parent.Right = node;

}

if (Current == Current.Parent.Left)

{

Current.Parent.Left = node;

}

node.Right = Current;

if (Current != null)

{

Current.Parent = node;

}

}

1. Текст програми.

Текст програми ви можете знайти за посиланням.

https://github.com/Vovan4ok/ASD

1. Набір тестів.

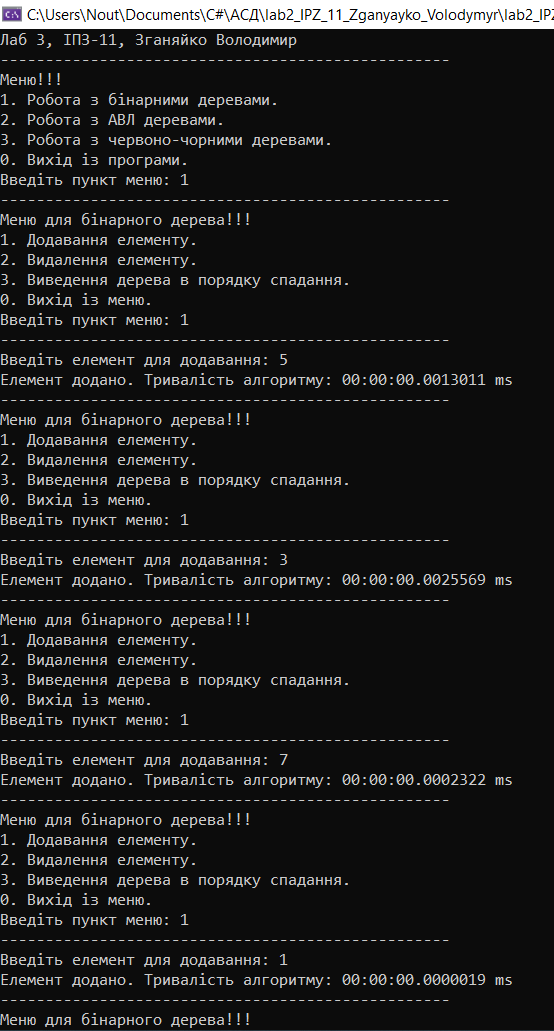
Набір тестів підібраний таким чином, щоб була абсолютно однакова перевірка для кожного дерева, тобто будуть виконані однакові дії:

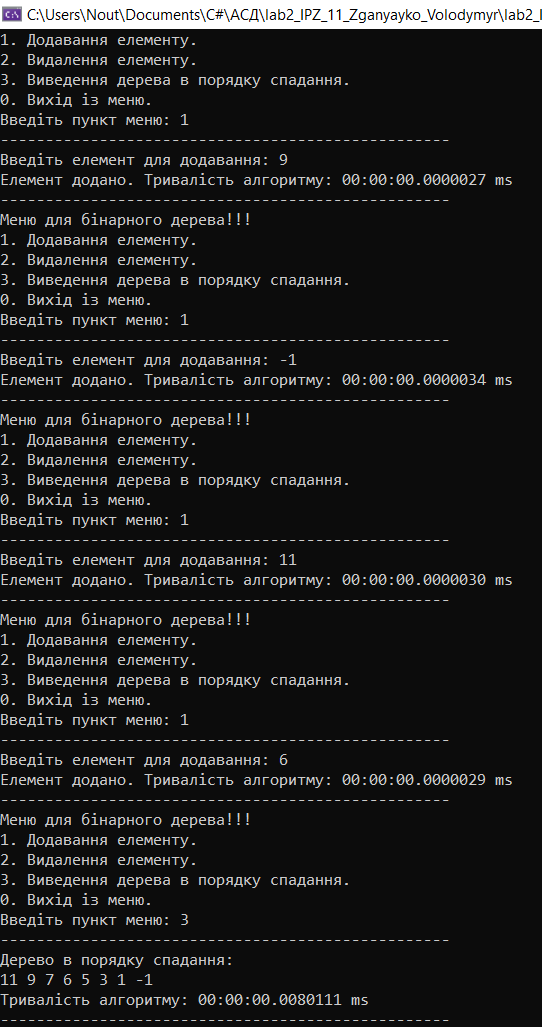
Спочатку буде додано вузли в такій послідовності:

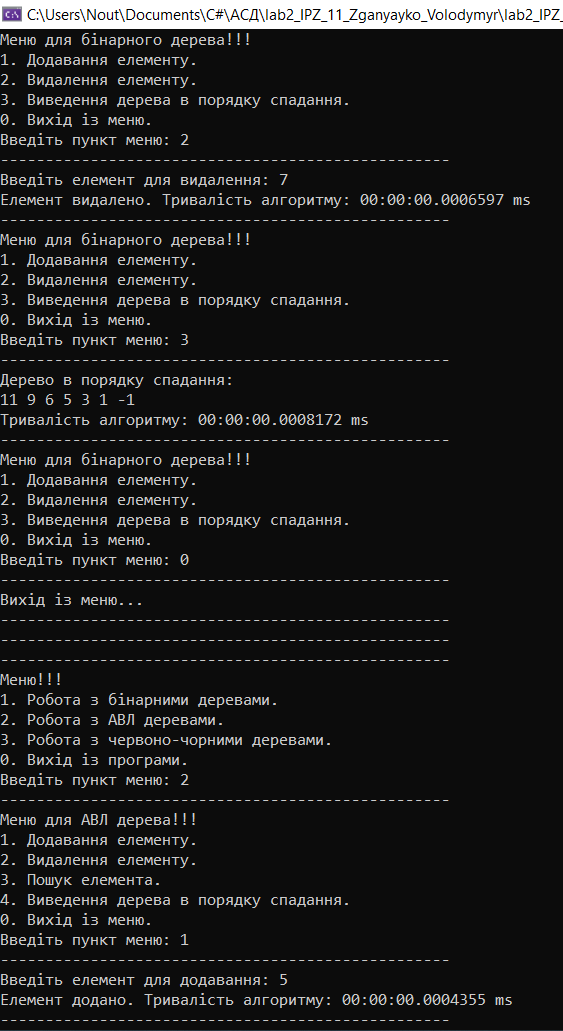
5, 3, 7, 1, 9, -1, 11, 6.

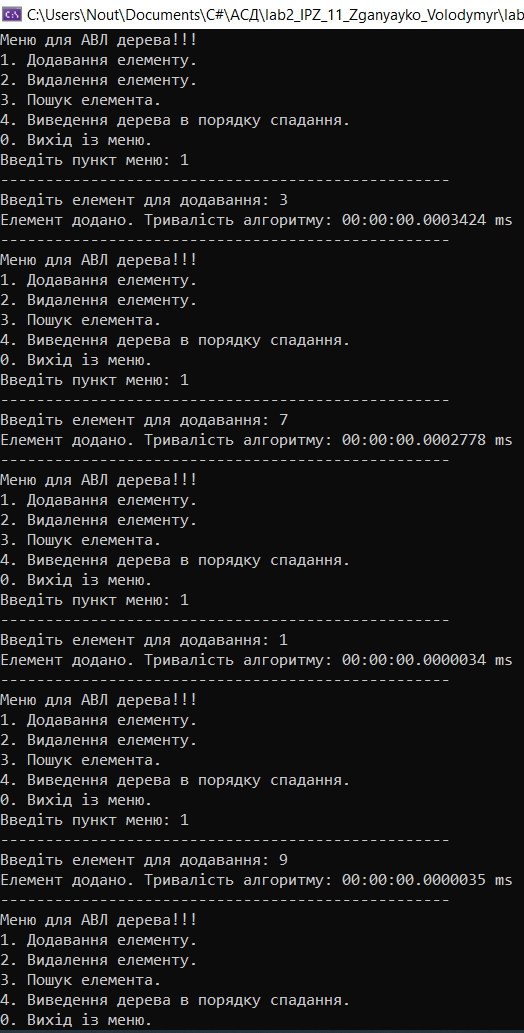
Потім робиться обхід дерева, після цього видаляється елемент 7, робиться виведення, робиться пошук елементу 6 та пошук елементу 8.

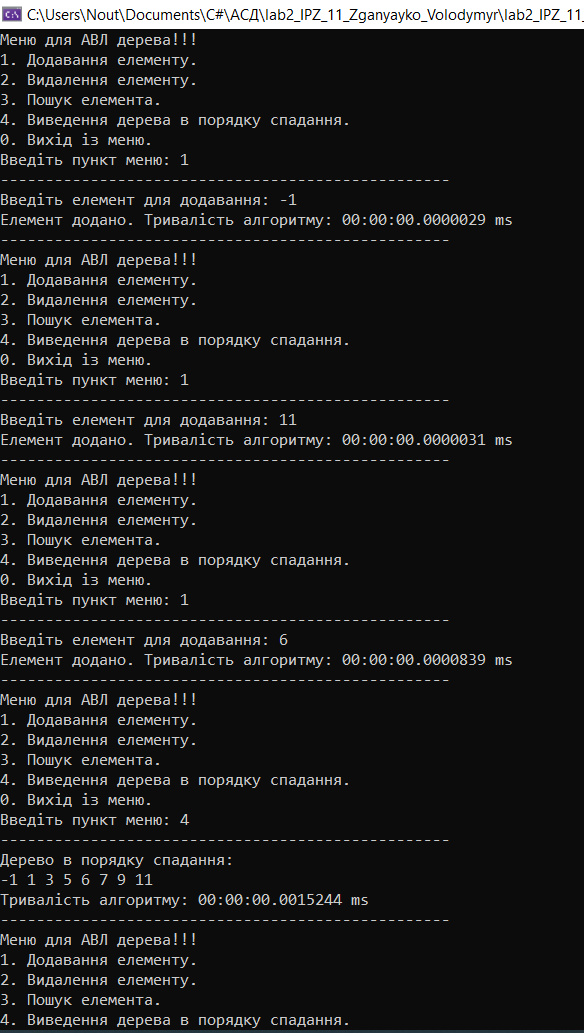
1. Результати тестування програми та аналіз отриманих помилок.

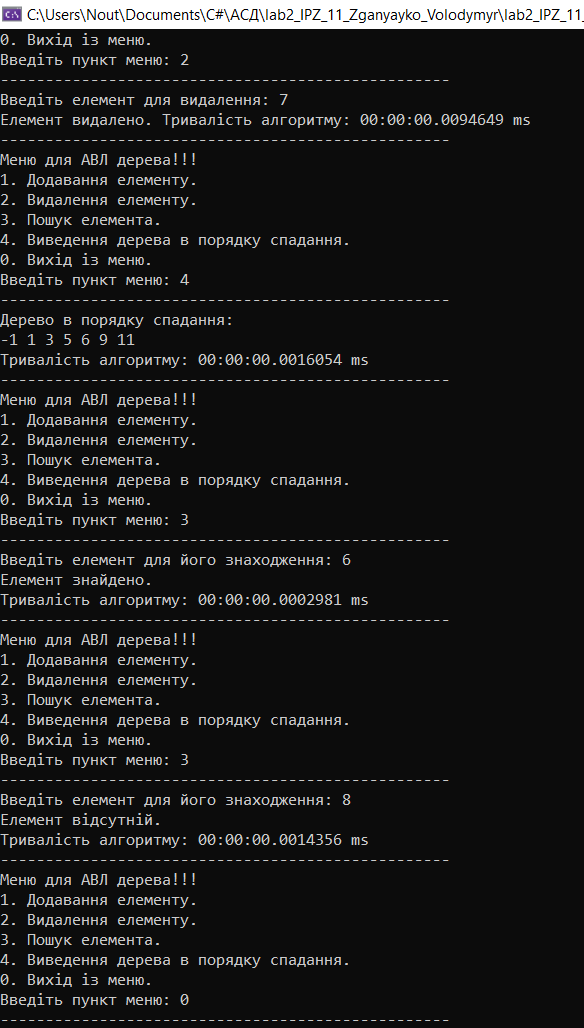


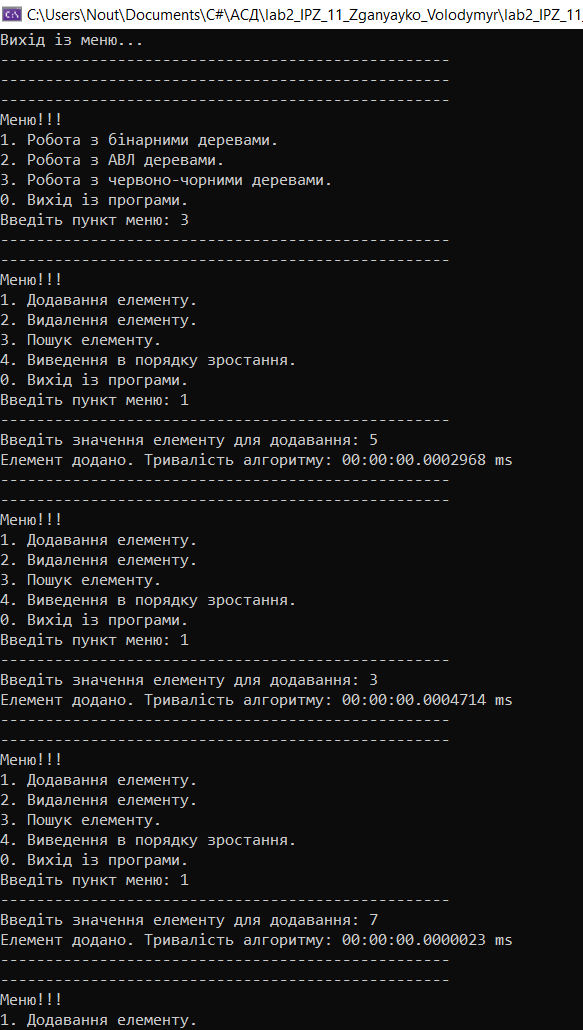


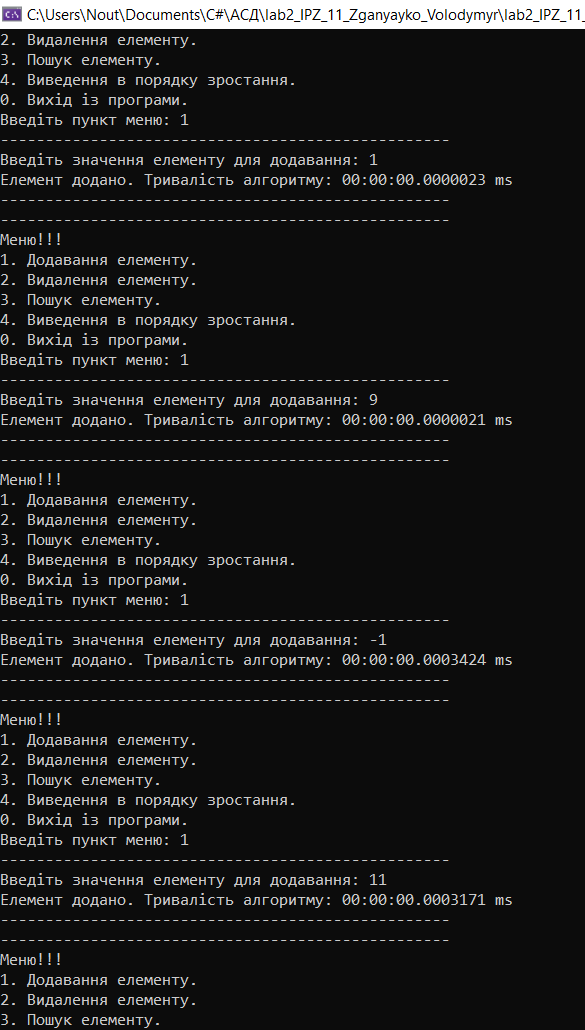


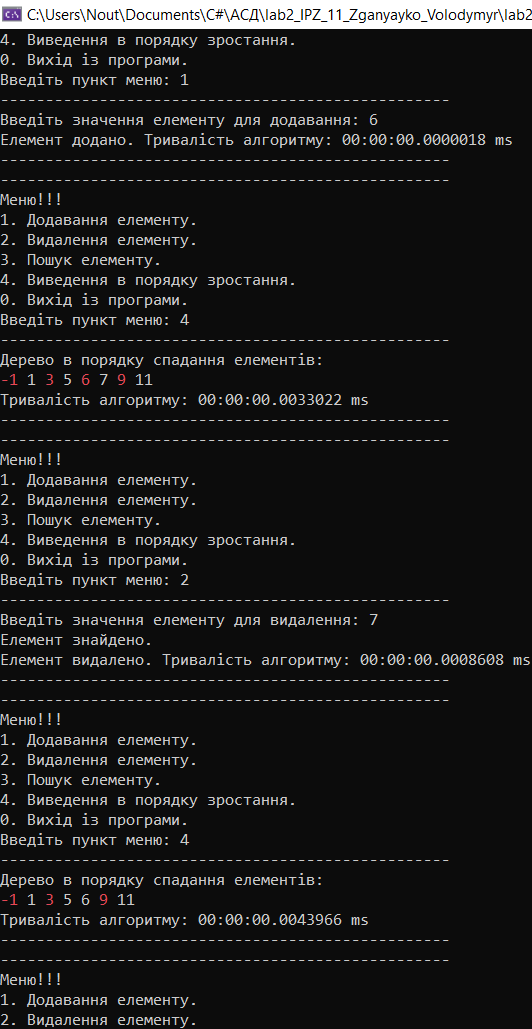


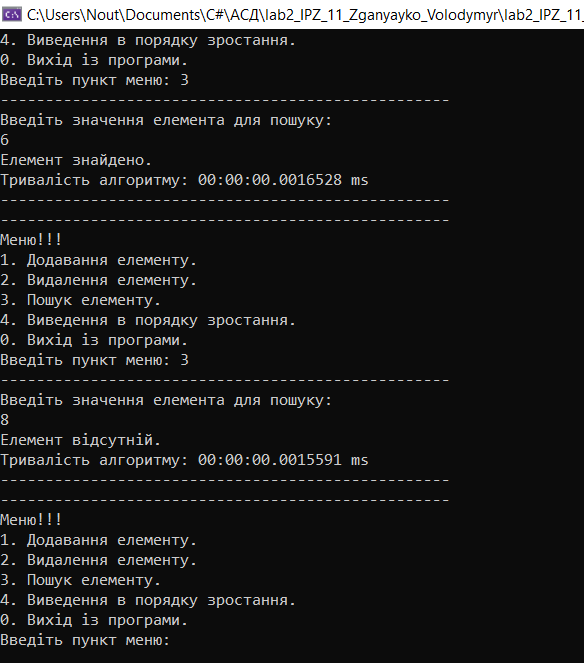












**Висновок та аналіз тестів:** Згідно з отриманими результатами можна зробити певні висновки: здебільшого алгоритми додавання вузла для АВЛ дерева та бінарного працювали з однаковою швидкістю, а от алгоритм додавання вузла в червоно-чорне дерево працював дещо довше, це можна пояснити тим, що після кожного додавання вузла у дане дерево, необхідно “балансувати дерево”, тобто роботи повороти та перефарбування вузлів. Щодо видалення вузла: найдовше алгоритм працював в АВЛ дереві, адже там так само як і в червоно-чорному дереві, необхідно робити балансування дерева, найшвидше алгоритм працює в бінарному дереві пошуку, адже там не потрібно робити ніяких додаткових балансувань. Пошук елементу в усіх типах дерев працює однаково та має однаковий алгоритм. Цікаво щодо виведення: для бінарного та АВЛ дерев було зроблено виведення через реалізацію інтерфейсу IEnumerable та через цикл foreach, а от для червоно-чорного через рекурсивний виклик. Дивлячись на час виконання алгоритмів, можна сказати, що рекурсія робиться дещо довше, проте вона набагато простіша у реалізації та і код простіший для розуміння через рекурсію.