**Звіт до**

**лабораторної роботи №3**

**З АСД**

Павлюченко Артем

ІПЗ-11/1

# **Умови завдання:**

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

* Вставка вузла в дерево
* Видалення вузла із дерева
* Лівий, правий повороти
* Перевірка на перебування у структурі
* Перефарбовування братніх вузлів у протилежний колір

# **Аналіз задачі:**

Усі алгоритми працюватимуть з цілими числами, аби краще проаналізувати результати. Для усіх структур ми створюємо окремі класи їх реалізації, викликом методів займається клас Program.

Робота з бінарним деревом пошуку: за допомогою оператора switch користувач обирає роботу з деревом, створює його, та за допомогою новоствореного меню вибирає ті дії і алгоритми, які хоче провести над деревом.

Робота з червоно-чорним деревом: за допомогою оператора switch користувач обирає роботу з деревом, створює його, та за допомогою новоствореного меню вибирає ті дії і алгоритми, які хоче провести над деревом.

Робота з АВЛ деревом: за допомогою оператора switch користувач обирає роботу з деревом, створює його, та за допомогою новоствореного меню вибирає ті дії і алгоритми, які хоче провести над деревом. (у всіх випадках меню майже не відрізнятиметься)

Для повноцінної перевірки працездатності моїх алгоритмів я перевіряв структури різних розмірів та отримав досить інформативні результати.

# **Структура основних вхідних та вихідних даних:**

Під час виконання кожного методу з вставки або видалення вхідними даними була безпосередньо структура і елемент, а вихідними даними структура із потрібними користувачеві змінами. Під час методів поворотів і вхідними і вихідними даними виступала певна структура.

# **Алгоритм розв'язання задачі:**

**Вставка:**

Для бінарного дерева пошуку:

public void Add(T data)

{

if (head == null)

{

head = new BinaryTreeNode<T>(data);

}

else

{

AddTo(head, data);

}

count++;

}

private void AddTo(BinaryTreeNode<T> node, T data)

{

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

{

if (node.Left == null)

{

node.Left = new BinaryTreeNode<T>(data);

}

else

{

AddTo(node.Left, data);

}

}

else

{

if (node.Right == null)

{

node.Right = new BinaryTreeNode<T>(data);

}

else

{

AddTo(node.Right, data);

}

}

}

Для АВЛ дерева:

public void Add(T value)

{

if (Head == null)

{

Head = new AVLTreeNode<T>(value, null, this);

}

else

{

AddTo(Head, value);

}

Count++;

}

private void AddTo(AVLTreeNode<T> node, T value)

{

if (value.CompareTo(node.Value) < 0)

{

if (node.Left == null)

{

node.Left = new AVLTreeNode<T>(value, node, this);

}

else

{

AddTo(node.Left, value);

}

}

else

{

if (node.Right == null)

{

node.Right = new AVLTreeNode<T>(value, node, this);

}

else

{

AddTo(node.Right, value);

}

}

node.Balance();

}

Для червоно-чорного дерева:

public void Insert(int item)

{

Node newItem = new Node(item);

if (root == null)

{

root = newItem;

root.colour = Color.Black;

return;

}

Node Y = null;

Node X = root;

while (X != null)

{

Y = X;

if (newItem.data < X.data)

{

X = X.left;

}

else

{

X = X.right;

}

}

newItem.parent = Y;

if (Y == null)

{

root = newItem;

}

else if (newItem.data < Y.data)

{

Y.left = newItem;

}

else

{

Y.right = newItem;

}

newItem.left = null;

newItem.right = null;

newItem.colour = Color.Red;

InsertFixUp(newItem);

}

**Видалення:**

Для бінарного дерева пошуку:

public bool Remove(T data)

{

BinaryTreeNode<T> current;

BinaryTreeNode<T> 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.Right;

}

}

}

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.Left;

}

else if (result < 0)

{

parent.Right = current.Right;

}

}

}

else

{

BinaryTreeNode<T> leftmost = current.Right.Left;

BinaryTreeNode<T> 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;

}

private BinaryTreeNode<T> FindWithParent(T data, out BinaryTreeNode<T> parent)

{

BinaryTreeNode<T> current = head;

parent = null;

while (current != null)

{

int result = current.CompareTo(data);

if (result > 0)

{

parent = current;

current = current.Left;

}

else if (result < 0)

{

parent = current;

current = current.Right;

}

else

{

break;

}

}

return current;

}

Для АВЛ дерева:

private AVLTreeNode<T> Find(T value)

{

AVLTreeNode<T> current = Head;

while (current != null)

{

int result = current.CompareTo(value);

if (result > 0)

{

current = current.Left;

}

else if (result < 0)

{

current = current.Right;

}

else

{

break;

}

}

return current;

}

public bool Remove(T value)

{

AVLTreeNode<T> current;

current = Find(value);

Console.WriteLine(current.Value);

if (current == null)

{

return false;

}

AVLTreeNode<T> 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.Value);

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.Value);

if (result > 0)

{

current.Parent.Left = current.Right;

}

else if (result < 0)

{

current.Parent.Right = current.Right;

}

}

}

else

{

AVLTreeNode<T> 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.Value);

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;

}

Для червоно-чорного дерева:

public void Delete(int key)

{

Node item = Find(key);

Node X = null;

Node Y = null;

if (item == null)

{

Console.WriteLine("Nothing to delete!");

return;

}

if (item.left == null || item.right == null)

{

Y = item;

}

else

{

Y = TreeSuccessor(item);

}

if (Y.left != null)

{

X = Y.left;

}

else

{

X = Y.right;

}

if (X != null)

{

X.parent = Y;

}

if (Y.parent == null)

{

root = X;

}

else if (Y == Y.parent.left)

{

Y.parent.left = X;

}

else

{

Y.parent.left = X;

}

if (Y != item)

{

item.data = Y.data;

}

if (Y.colour == Color.Black)

{

DeleteFixUp(X);

}

}

Rotate:

private void LeftRightRotation()

{

Right.RightRotation();

LeftRotation();

}

private void RightLeftRotation()

{

Left.LeftRotation();

RightRotation();

}

private void LeftRotation()

{

AVLTreeNode<TNode> newRoot = Right;

ReplaceRoot(newRoot);

Right = newRoot.Left;

newRoot.Left = this;

}

private void RightRotation()

{

AVLTreeNode<TNode> newRoot = Left;

ReplaceRoot(newRoot);

Left = newRoot.Right;

newRoot.Right = this;

}

private void ReplaceRoot(AVLTreeNode<TNode> 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;

}

# **Код програми:**

Увесь код програми знаходиться в репозиторії GitHub:

https://github.com/Pavich123/Programming-Algorithms

# **Набір тестів:**

1. Дерево з 10 вузлів:

Вхідне дерево: 0 1 8 16 17 17 25 27 29 42

**Видалення вузла 16:**

Вихідне дерево: 0 1 8 17 17 25 27 29 42

1. Дерево з 100 вузлів:

Вхідне дерево: 9 14 15 15 20 20 27 29 32 33 40 45 46 48 62 64 67 72 79 83 83 86 90 94 94 101 104 106 111 113 118 119 122 131 135 136 144 144 162 166 169 172 173 193 194 194 195 203 212 229 234 235 236 238 238 242 249 262 276 278 279 287 288 295 301 305 308 320 321 321 348 352 354 356 361 362 362 369 370 372 380 381 385 396 404 406 414 417 418 425 442 446 456 471 472 476 483 489 496 499

**Видалення вузла 442:**

Вихідне дерево: 9 14 15 15 20 20 27 29 32 33 40 45 46 48 62 64 67 72 79 83 83 86 90 94 94 101 104 106 111 113 118 119 122 131 135 136 144 144 162 166 169 172 173 193 194 194 195 203 212 229 234 235 236 238 238 242 249 262 276 278 279 287 288 295 301 305 308 320 321 321 348 352 354 356 361 362 362 369 370 372 380 381 385 396 404 406 414 417 418 425 456 471 472 476 483 489 496 499

1. Дерево з 1000 вузлів:

Вхідне дерево: 24 29 36 47 54 56 60 64 68 69 81 81 89 105 106 112 126 139 153 156 162 163 165 169 178 183 188 197 202 203 204 225 243 245 256 323 326 342 345 346 347 354 376 384 390 392 398 432 433 451 470 499 500 518 522 525 535 541 546 559 564 582 590 603 617 629 639 647 650 656 659 663 687 692 713 726 744 751 764 768 ……………… 9614 9617 9631 9635 9652 9658 9659 9661 9669 9681 9688 9710 9717 9736 9743 9743 9758 9759 9764 9770 9780 9793 9809 9810 9820 9847 9860 9868 9869 9888 9926 9949 9953 9958 9983 9997

**Видалення вузлів 24, 29, 433, 9983:**

Вихідне дерево: 36 47 54 56 60 64 68 69 81 81 89 105 106 112 126 139 153 156 162 163 165 169 178 183 188 197 202 203 204 225 243 245 256 323 326 342 345 346 347 354 376 384 390 392 398 432 499 500 518 522 525 535 541 546 559 564 582 590 603 617 629 639 647 650 656 659 663 687 692 713 726 744 751 764 768 768 770 774 782 789 789 803 803 803 809 823 833 842 847 862 864 866 910 912 914 917 926 933 933 ………………………………….9764 9770 9780 9793 9809 9810 9820 9847 9860 9868 9869 9888 9926 9949 9953 9958 9997

1. Дерево з 100000 вузлів:

Вхідне дерево: 5 10 31 54 67 74 85 86 89 97 101 108 114 116 116 116 118 123 148 166 166 166 172 184 187 200 212 215 245 252 261 279 284 300 307 339 349 362 363 371 381 383 398 402 406 408 434 434 445 447 457 465 468 470 473 475 485 492 520 523 …………… 999445 999476 999515 999515 999519 999528 999563 999576 999590 999593 999596 999600 999604 999607 999609 999610 999619 999634 999657 999671 999679 999689 999692 999696 999697 999705 999724 999738 999748 999748 999752 999760 999763 999765 999767 999776 999781 999794 999804 999830 999851 999852 999874 999903 999906 999917 999921 999933 999938 999945 999957 999963 999964

**Видалення вузлів 5, 10, 31, 999933, 999964:**

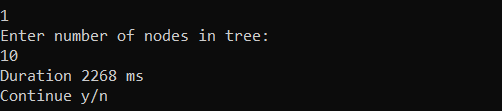
Вихідне дерево: 54 67 74 85 86 89 97 101 108 114 116 116 116 118 123 148 166 166 166 172 184 187 200 212 215 245 252 261 279 284 300 307 339 349 362 363 371 381 383 398 402 406 408 434 434 445 447 457 465 468 470 473 475 485 492 520 523 …………… 999445 999476 999515 999515 999519 999528 999563 999576 999590 999593 999596 999600 999604 999607 999609 999610 999619 999634 999657 999671 999679 999689 999692 999696 999697 999705 999724 999738 999748 999748 999752 999760 999763 999765 999767 999776 999781 999794 999804 999830 999851 999852 999874 999903 999906 999917 999921 999938 999945 999957 999963

# **Результати тестування:**

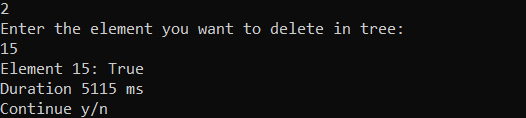
Тест №1:

**Binary Tree Search:**

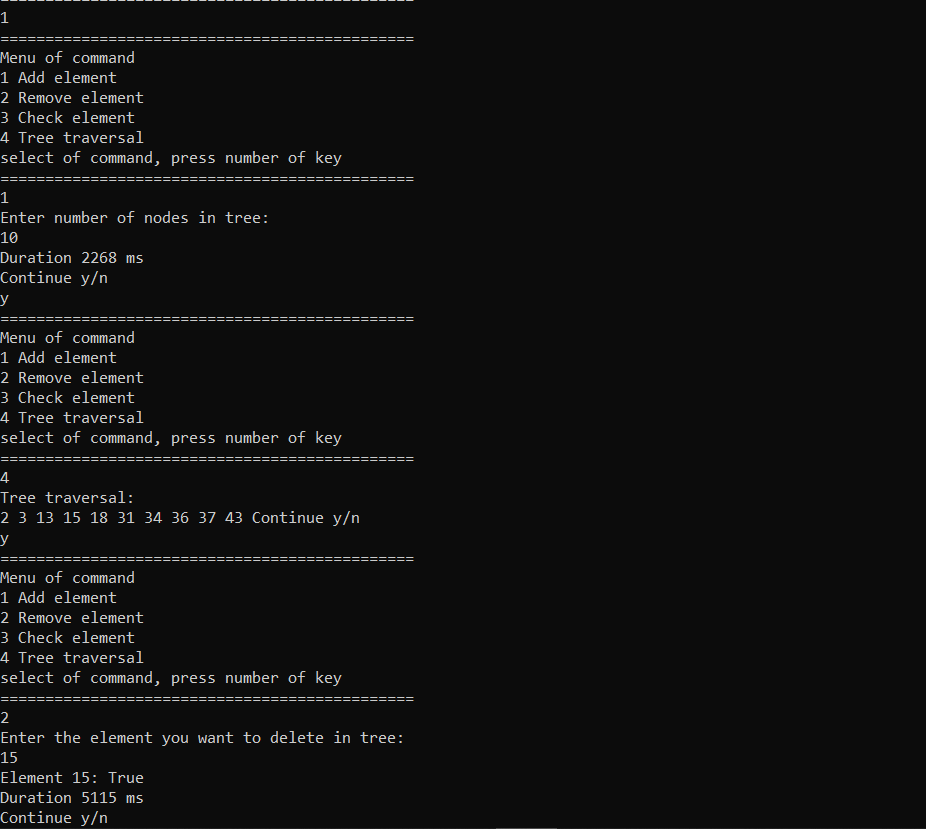
**Add:**

****

**Remove:**

****

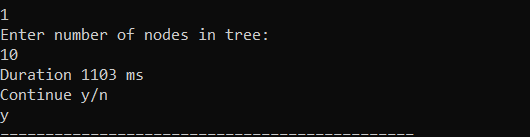
**All process:**

****

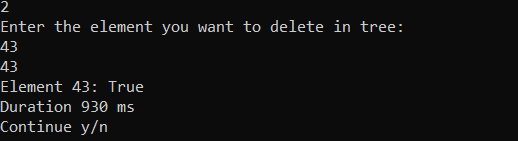
Тест №2:

**AVL Tree:**

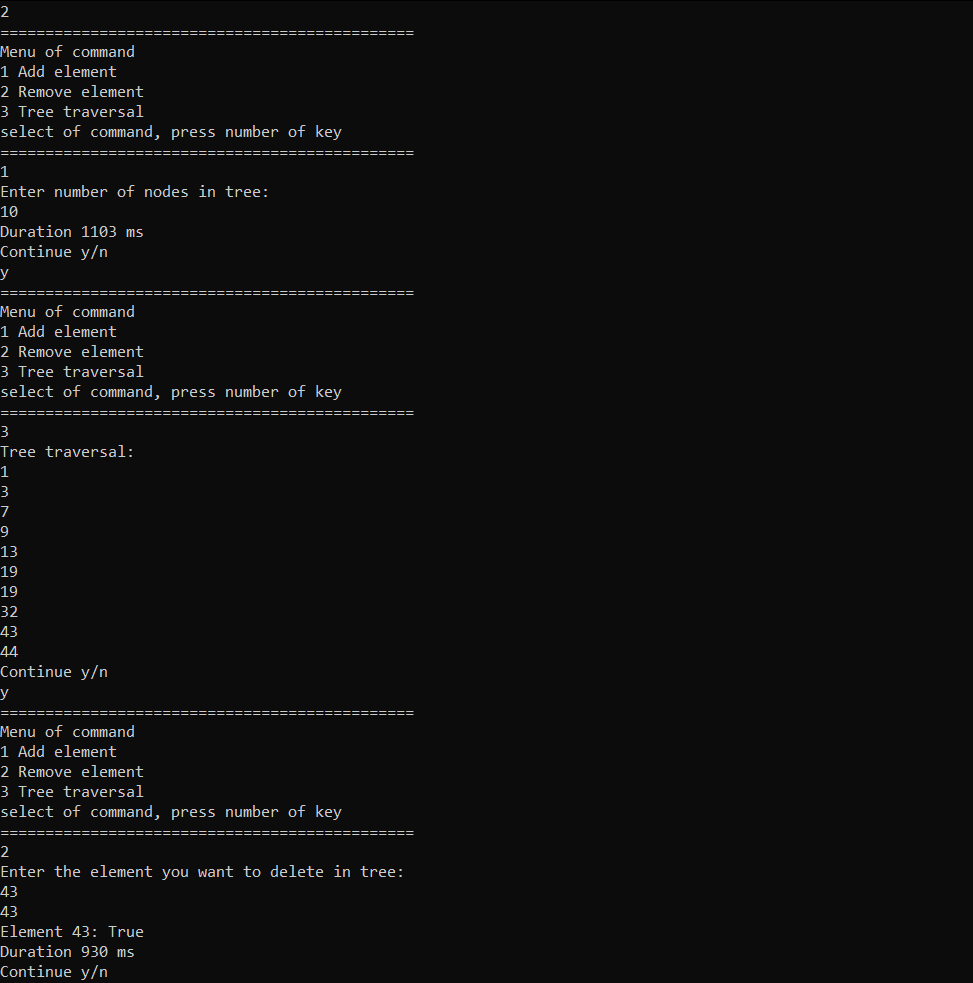
**Add:**

****

**Remove:**

****

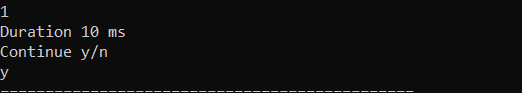
**All process:**

****

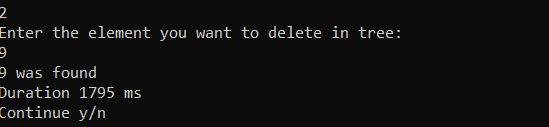
Тест №3:

**Red Black Tree:**

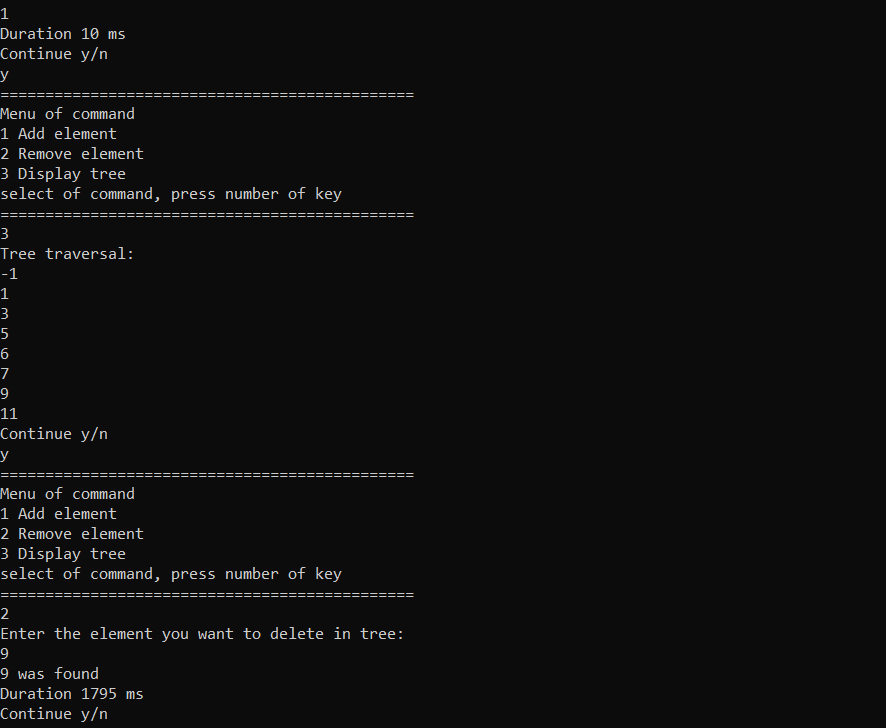
**Add:**

****

**Remove:**

****

**All process:**

****

У фінальному варіанті програми не виявлено жодних помилок під час тестувань, отже усі алгоритми працюють правильно, видають очікувані результати з приводу часу та ефективності.

Під час усіх перевірок, найшвидшим за результатами тестувань стало Червоно-Чорне дерево. Але саме під час кодування його алгоритмів у мене виникли труднощі. Через те що під час перефарбовування братніх вузлів у інший колір, в мене нащадки червоних вузлів перефарбовувалися у червоний колір, що не є властивістю такого дерева. Цю проблему не було вирішено, тож у програмі цей алгоритм не наведено.

Усі інші алгоритми реалізовано правильно.