# КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ТАРАСА ШЕВЧЕНКА



## ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**Кафедра прикладних інформаційних систем**

**Звіт до лабораторної роботи №7**

# з курсу

**«Алгоритми і структури даних»**

*Студента 1 курсу*

*групи ПП-11 спеціальності 122 «Комп'ютерні науки» ОП «Прикладне програмування»*

%username%

*Викладач:*

д.е.н., к.т.н., проф. Плескач В.Л.

к.ф.-м.н., доц. Шолохов О.В.

## Київ – 202

**1.Назва роботи**

Дерева та сортування за допомогою дерев

1. **Тема роботи**

Дерева та сортування за допомогою дерев

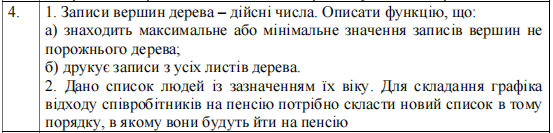
1. **Мета роботи**

Дослідити та оволодіти навичками написання програм, що

використовують при роботі із деревами та при сортуванні за допомогою дерев

1. **Умова завдання**

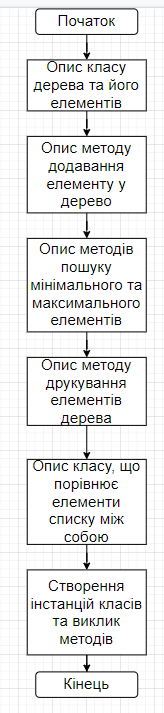
(4 варіант)



1. **Рішення**

Описуємо клас елементу та загальний клас дерева, методи додавання елементу у дерево, пошук мінімального та максимального елементів у дереві, вивід усіх елементів на екран. Описуємо клас, що буде сортувати список та створюємо екземпляри дерева та списку. Додаємо у вищевказані дерево та список елементи, знаходимо мінімум та максимум, виводимо на екран дерево, після чого сортуємо та виводимо на екран список.

Блоксхема:



Код:

using System;

using System.Collections.Generic;

namespace Laba7 {

class Program {

class Node<T> where T : IComparable {

public Node<T> Left { get; set; }

public Node<T> Right { get; set; }

public T Element { get; set; }

public Node(T element) {

Element = element;

}

}

class Tree<T> where T : IComparable {

protected Node<T> \_root;

public void AddElement(T newElement)

{

if (\_root == null) {

\_root = new Node<T>(newElement);

return;

}

AddElementRecursion(\_root, newElement);

}

private static void AddElementRecursion(Node<T> currentNode, T newElement) {

if (currentNode.Element.CompareTo(newElement) < 0) {

if (currentNode.Right == null)

currentNode.Right = new Node<T>(newElement);

else

AddElementRecursion(currentNode.Right, newElement);

}

else {

if (currentNode.Left == null)

currentNode.Left = new Node<T>(newElement);

else

AddElementRecursion(currentNode.Left, newElement);

}

}

public T GetMinElement() {

return GetMinElementRecursion(\_root);

}

private static T GetMinElementRecursion(Node<T> currentNode) {

if (currentNode.Left == null)

return currentNode.Element;

else

return GetMinElementRecursion(currentNode.Left);

}

public T GetMaxElement() {

return GetMaxElementRecursion(\_root);

}

private static T GetMaxElementRecursion(Node<T> currentNode) {

if (currentNode.Right == null)

return currentNode.Element;

else

return GetMaxElementRecursion(currentNode.Right);

}

}

class TreeDoubleType : Tree<double> {

public void Print() {

double sum = 0;

PrintRecursion(\_root, ref sum);

Console.WriteLine(\_root);

}

private static void PrintRecursion(Node<double> currentNode, ref double sum) {

Console.WriteLine(currentNode.Element);

if (currentNode.Left != null)

PrintRecursion(currentNode.Left, ref sum);

if (currentNode.Right != null)

PrintRecursion(currentNode.Right, ref sum);

}

}

public class retireeComparer : IComparer<int> {

public int Compare(int o1, int o2) {

int a = o1;

int b = o2;

if (a < b) {

return 1;

}

else if (a > b) {

return -1;

}

return 0;

}

}

static void Main(string[] args) {

double[] doubleArray = new[] { 1, 5, -1.4, 7, 3, 0, 4, -1, 6, -10, 8.4, 5, -7, -6, 5, 3, 7, -2 };

int[] agesArray = new[] {22, 34, 67, 21, 69, 35, 71 };

TreeDoubleType tree = new TreeDoubleType();

foreach (double d in doubleArray)

tree.AddElement(d);

double min = tree.GetMinElement();

tree.Print();

double max = tree.GetMaxElement();

Console.WriteLine("The smallest element is " + min);

Console.WriteLine("The biggest element is " + max);

List<int> ages = new List<int>();

foreach (int q in agesArray)

ages.Add(q);

retireeComparer compare = new retireeComparer();

ages.Sort(compare);

foreach (int age in ages)

Console.WriteLine(age);

}

}

}

Результат роботи програми:

[screenshot]

1. **Контрольні запитання**
2. Визначення бінарнго дерева

Структура даних у вигляді дерева, в якому кожна вершина має не більше двох дітей

1. Операції над вузлами дерев

Додавання, видалення, отримання (зчитування)

1. Подібність дерев

1. Властивості дерев

Кожна вершина є коренем наступного піддерева;

З кореня до довільної вершини існує лише один шлях

Якщо вершина *x* є предком *y* та не існує вершин поміж ними (тобто *x* та *y* з'єднані одним ребром), а також існують предки для *x* (тобто *x* не є коренем), то вершина *x* називається батьком до *y,* відповідно *y* є дитиною *x.*

Вершини зі спільними батьками називаються братами

Дерева з відносним порядком на піддеревах називаються пласкими/впорядкованими

Множина дерев, що не перетинаються - ліс

1. Упорядкування дерев

Дерева з відносним порядком на піддеревах називаються пласкими/впорядкованими. Зазвичай дерева впорядкувуються під час додавання нових елементів (якщо новий елемент менше поточного, то він додається в ліву вершину, якщо більше - в праву і т.д.)

1. Методи обходу

Прямий (корінь -> ліве піддерево -> праве піддерево), зворотній (ліве піддерево -> праве піддерево -> корінь), серединний (ліве піддерево -> корінь -> праве піддерево)

1. **Висновки**

В результаті виконання даної лабораторної роботи я оволодів практичними навичками використання дерев, навчився складати програми для виконання операцій з деревами. Реалізував створення бінарного дерева, додавання нових елементів до нього, вивід на екран усіх елементів, пошук мінімального/максимального елементу. Вважаю дану лабораторну роботу виконаною в повному обсязі.