Звіт

з лабораторної роботи №9

з курсу алгоритми і методи обчислень.

(БІНАРНІ ДЕРЕВА)

Студента 1-го курсу,

Парфенюка Тимофія Павловича

2019

Завдання:

Побудувати бінарне дерево пошуку із вхідного масиву даних, здійснити операції над деревом.

* Складіть програму, яка містить процедури та функції для обробки бінарних дерев: побудови бінарного дерева пошуку із масиву даних, обходу дерева, пошуку по дереву, вставки елемента в дерево, видалення елемента з дерева, видалення всього дерева. При цьому передбачте можливість введення вхідного масиву з клавіатури або файлу.
* Після побудови дерева та після кожної операції над ним виведіть на екран значення усіх вершин дерева у порядку прямого, зворотного та симетричного обходів.
* Доповніть програму процедурами та функціями, необхідними для виконання вашого індивідуального завдання. Результат виконання виведіть на екран.

**Код:**

using System;

namespace Tree

{

public enum BinSide

{

Left,

Right

}

public class BinaryTree: IDisposable

{

public long? Data { get; private set; }

public BinaryTree Left { get; set; }

public BinaryTree Right { get; set; }

public BinaryTree Parent { get; set; }

public void Insert(long data)

{

if (Data == null || Data == data)

{

Data = data;

return;

}

if (Data > data)

{

if (Left == null) Left = new BinaryTree();

Insert(data, Left, this);

}

else

{

if (Right == null) Right = new BinaryTree();

Insert(data, Right, this);

}

}

private void Insert(long data, BinaryTree node, BinaryTree parent)

{

if (node.Data == null || node.Data == data)

{

node.Data = data;

node.Parent = parent;

return;

}

if (node.Data > data)

{

if (node.Left == null) node.Left = new BinaryTree();

Insert(data, node.Left, node);

}

else

{

if (node.Right == null) node.Right = new BinaryTree();

Insert(data, node.Right, node);

}

}

private void Insert(BinaryTree data, BinaryTree node, BinaryTree parent)//Для методу Remove

{

if (node.Data == null || node.Data == data.Data)

{

node.Data = data.Data;

node.Left = data.Left;

node.Right = data.Right;

node.Parent = parent;

return;

}

if (node.Data > data.Data)

{

if (node.Left == null) node.Left = new BinaryTree();

Insert(data, node.Left, node);

}

else

{

if (node.Right == null) node.Right = new BinaryTree();

Insert(data, node.Right, node);

}

}

private BinSide? MeForParent(BinaryTree node)

{

if (node.Parent == null) return null;

if (node.Parent.Left == node) return BinSide.Left;

if (node.Parent.Right == node) return BinSide.Right;

return null;

}

private void Remove(BinaryTree node)

{

if (node == null) return;

var me = MeForParent(node);

//Якщо немає нащадків - видаляємо

if (node.Left == null && node.Right == null)

{

if (me == BinSide.Left)

{

node.Parent.Left = null;

}

else

{

node.Parent.Right = null;

}

return;

}

//Якщо нема лівого нащадка, то правий ставиться на місце цього

if (node.Left == null)

{

if (me == BinSide.Left)

{

node.Parent.Left = node.Right;

}

else

{

node.Parent.Right = node.Right;

}

node.Right.Parent = node.Parent;

return;

}

//Якщо нема правого нащадка, то лівий ставиться на місце цього

if (node.Right == null)

{

if (me == BinSide.Left)

{

node.Parent.Left = node.Left;

}

else

{

node.Parent.Right = node.Left;

}

node.Left.Parent = node.Parent;

return;

}

// правий ставимо на місце цього

//а лівий вставляємо в правий

if (me == BinSide.Left)

{

node.Parent.Left = node.Right;

}

if (me == BinSide.Right)

{

node.Parent.Right = node.Right;

}

if (me == null)

{

var bufLeft = node.Left;

var bufRightLeft = node.Right.Left;

var bufRightRight = node.Right.Right;

node.Data = node.Right.Data;

node.Right = bufRightRight;

node.Left = bufRightLeft;

Insert(bufLeft, node, node);

}

else

{

node.Right.Parent = node.Parent;

Insert(node.Left, node.Right, node.Right);

}

}

public void Remove(long data)//Метод шукає елемент, і передає у метод вище

{

var removeNode = Find(data);

if (removeNode != null)

{

Remove(removeNode);

}

}

public BinaryTree Find(long data)

{

if (Data == data) return this;

if (Data > data)

{

return Find(data, Left);

}

return Find(data, Right);

}

private BinaryTree Find(long data, BinaryTree node)

{

if (node == null) return null;

if (node.Data == data) return node;

if (node.Data > data)

{

return Find(data, node.Left);

}

return Find(data, node.Right);

}

public long CountElements()

{

return CountElements(this);

}

private long CountElements(BinaryTree node)

{

long count = 1;

if (node.Right != null)

{

count += CountElements(node.Right);

}

if (node.Left != null)

{

count += CountElements(node.Left);

}

return count;

}

public BinaryTree MinElement(BinaryTree node)

{

if (node.Left != null) return MinElement(node.Left);

if (node.Left == null) return node;

return null;

}

public void Dispose()

{

Console.Beep();

}

}

public class BinaryTreePrint

{

public static void Print(BinaryTree node)

{

if (node != null)

{

if (node.Parent == null)

{

Console.WriteLine("ROOT:{0}", node.Data);

}

else

{

if (node.Parent.Left == node)

{

Console.WriteLine("Left for {1} --> {0}", node.Data, node.Parent.Data);

}

if (node.Parent.Right == node)

{

Console.WriteLine("Right for {1} --> {0}", node.Data, node.Parent.Data);

}

}

if (node.Left != null)

{

Print(node.Left);

}

if (node.Right != null)

{

Print(node.Right);

}

}

}

static public void CLR(BinaryTree node, ref string s, bool detailed)//Прямий обхід

{

if (node != null)

{

if (detailed)

s += " Value " + node.Data.ToString() + Environment.NewLine;

else

s += node.Data.ToString() + " ";

if (detailed) s += " Left" + Environment.NewLine;

CLR(node.Left, ref s, detailed);

if (detailed) s += " Right" + Environment.NewLine;

CLR(node.Right, ref s, detailed);

}

else if (detailed) s += " - null" + Environment.NewLine;

}

static public void LRC(BinaryTree node, ref string s, bool detailed)//Обернений

{

if (node != null)

{

if (detailed) s += " Left" + Environment.NewLine;

LRC(node.Left, ref s, detailed);

if (detailed) s += " Rigt" + Environment.NewLine;

LRC(node.Right, ref s, detailed);

if (detailed)

s += " Value " + node.Data.ToString() + Environment.NewLine;

else

s += node.Data.ToString() + " ";

}

else if (detailed) s += " - null" + Environment.NewLine;

}

static public void LCR(BinaryTree node, ref string s, bool detailed)//Симетричний

{

if (node != null)

{

if (detailed) s += " Left" + Environment.NewLine;

LCR(node.Left, ref s, detailed);

if (detailed)

s += " Value " + node.Data.ToString() + Environment.NewLine;

else

s += node.Data.ToString() + " ";

if (detailed) s += " Rigt" + Environment.NewLine;

LCR(node.Right, ref s, detailed);

}

else if (detailed) s += " - null" + Environment.NewLine;

}

}

public class Programm

{

static public void Main()

{

var tree = new BinaryTree();

int[] values = { 9, 44, 0, -7, 10, 6, -12, 45, 12, 11, 15 };

foreach (int value in values)

{

tree.Insert(value);

}

BinaryTreePrint.Print(tree);

Console.WriteLine();

string s = "";

BinaryTreePrint.CLR(tree, ref s, false);

Console.WriteLine("Прямий: " + s);

s = "";

BinaryTreePrint.LRC(tree, ref s, false);

Console.WriteLine("Обернений: "+s);

s = "";

BinaryTreePrint.LCR(tree, ref s, false);

Console.WriteLine("Симетричний: " + s);

Console.Write("Найменьший: ");

var minNode = new BinaryTree();

minNode = tree.MinElement(tree);

Console.Write(minNode.Data);

Console.WriteLine("");

Console.WriteLine();

Console.WriteLine("Додаємо 50");

tree.Insert(50);

Console.WriteLine("Видаляємо 44");

tree.Remove(44);

BinaryTreePrint.Print(tree);

tree.Dispose();

Console.ReadKey();

}

}

}

***Висновок:***

Дерево – динамічна структура даних, що представляє собою сукупність елементів і зв’язків, які утворюють ієрархічну структуру цих елементів.

Динамічна структура даних – це дані, що розміщуються у динамічній пам’яті, яка виділяється на етапі використання програми.

Степінь внутрішньої вершини – число її синів. Листки дерева мають степінь 0. Степінь дерева – максимальний серед степенів усіх його вершин.

Типи дерев :

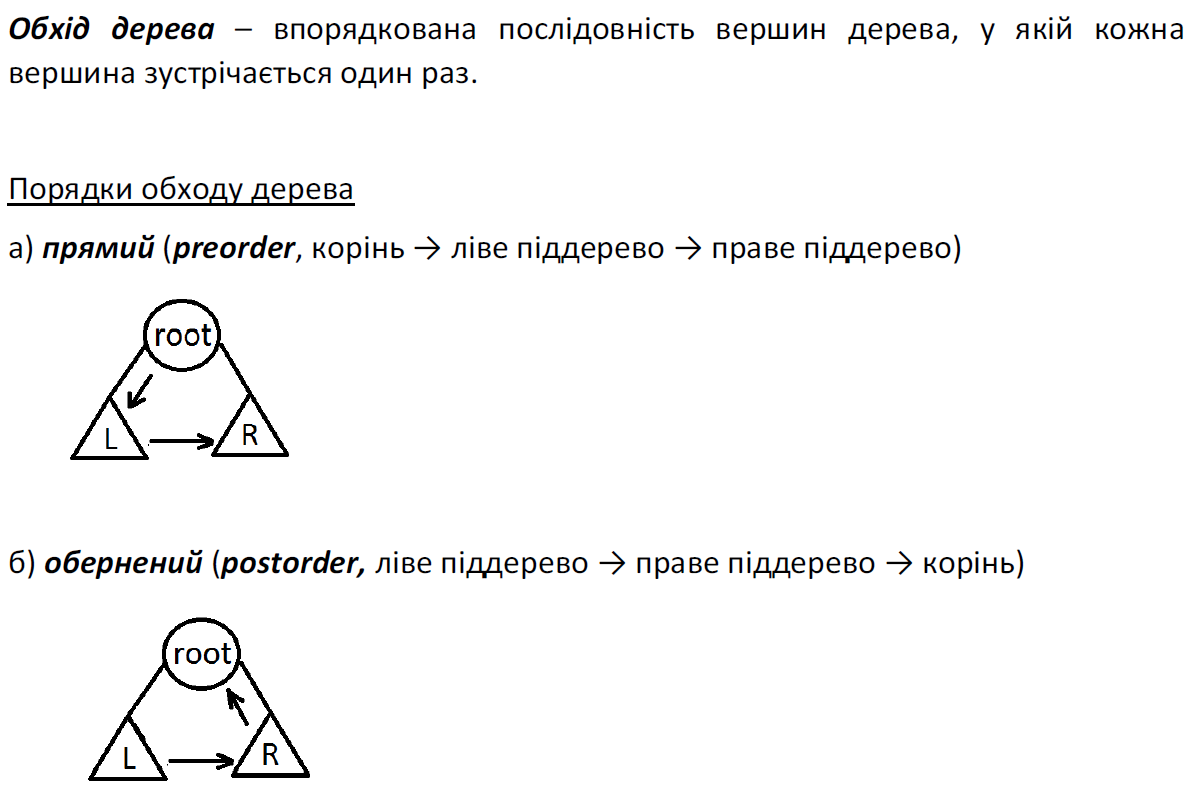
* бінарні (Binary Tree) – дерева другого степеню;
* сильно розгалужені (Multiway Tree) – дерева степеню більшого за 2.

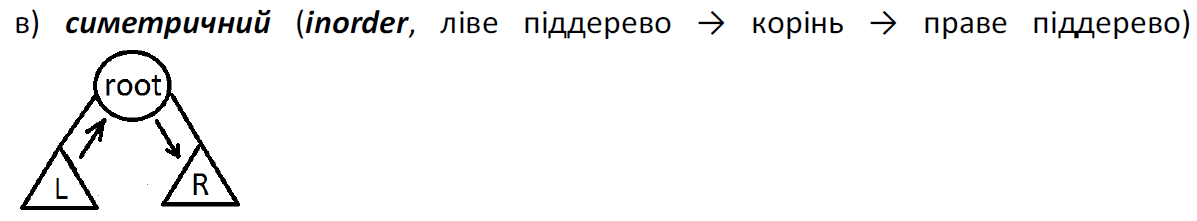
**Строге бінарне дерево** – вершини дерева мають степінь 0 (у листя) або 2 (у внутрішньої вершини).

**Нестроге бінарне дерево** - вершини дерева мають степінь 0 (у листя), 1 або 2 (у внутрішньої вершини).

**Повне бінарне дерево** – містить повністю заповнені рівні. Інакше – неповне.

**Ідеально збалансоване дерево** – кількість вершин у його лівому і правому піддеревах відрізняється не більше, ніж на 1.





У рекурсивному або невпорядкованому дереві має значення лише структура самого дерева без урахування порядку нащадків для кожного вузла. Дерево, в якому є заданий порядок (наприклад, кожному ребру, провідному до нащадка, присвоєні різні натуральні числа) називається деревом з іменованими ребрами або впорядкованим деревом зі структурою даних, заданої перед ім'ям. Впорядковані дерева є найбільш поширеними серед деревовидних структур. Двійкове дерево пошуку — одне з різновидів упорядкованого дерева. Двійкове дерево — структура даних у вигляді дерева, в якому кожна вершина має не більше двох дітей. Зазвичай такі діти називаються правим та лівим. На базі двійкових дерев будуються такі структури, як двійкові дерева пошуку та двійкові купи.

**Висота вузла** – кількість ребер у найдовшому простому шляху від вузла до листка. **Висота дерева** – висота його кореня. Висота порожнього дерева - 0.

***Вхідні та вихідні данні:***

