Class Node:

public class Node

{

public Node LeftNode { get; set; }

public Node RightNode { get; set; }

public int Data { get; set; }

}

Class BinaryTree:

public class BinaryTree

{

public Node Root { get; set; }

public bool Add(int value)

{

Node before = null, after = this.Root;

while (after != null)

{

before = after;

if (value < after.Data)

after = after.LeftNode;

else if (value > after.Data)

after = after.RightNode;

else

{

return false;

}

}

Node newNode = new Node();

newNode.Data = value;

if (this.Root == null)

this.Root = newNode;

else

{

if (value < before.Data)

before.LeftNode = newNode;

else

before.RightNode = newNode;

}

return true;

}

public Node Find(int value)

{

return this.Find(value, this.Root);

}

public void Remove(int value)

{

this.Root = Remove(this.Root, value);

}

private Node Remove(Node parent, int key)

{

if (parent == null) return parent;

if (key < parent.Data) parent.LeftNode = Remove(parent.LeftNode, key);

else if (key > parent.Data)

parent.RightNode = Remove(parent.RightNode, key);

else

{

if (parent.LeftNode == null)

return parent.RightNode;

else if (parent.RightNode == null)

return parent.LeftNode;

parent.Data = MinValue(parent.RightNode);

parent.RightNode = Remove(parent.RightNode, parent.Data);

}

return parent;

}

private int MinValue(Node node)

{

int minv = node.Data;

while (node.LeftNode != null)

{

minv = node.LeftNode.Data;

node = node.LeftNode;

}

return minv;

}

private Node Find(int value, Node parent)

{

if (parent != null)

{

if (value == parent.Data) return parent;

if (value < parent.Data)

return Find(value, parent.LeftNode);

else

return Find(value, parent.RightNode);

}

return null;

}

public int GetTreeDepth()

{

return this.GetTreeDepth(this.Root);

}

private int GetTreeDepth(Node parent)

{

if (parent == null)

{

return 0;

}

else

{

return Math.Max(GetTreeDepth(parent.LeftNode), GetTreeDepth(parent.RightNode)) + 1;

}

}

public void TraversePreOrder(Node parent)

{

if (parent != null)

{

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

TraversePreOrder(parent.LeftNode);

TraversePreOrder(parent.RightNode);

}

}

public void TraverseInOrder(Node parent)

{

if (parent != null)

{

TraverseInOrder(parent.LeftNode);

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

TraverseInOrder(parent.RightNode);

}

}

public void TraversePostOrder(Node parent)

{

if (parent != null)

{

TraversePostOrder(parent.LeftNode);

TraversePostOrder(parent.RightNode);

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

}

}

}

Main:

static void Main(string[] args)

{

Console.OutputEncoding = Encoding.UTF8;

BinaryTree binaryTree = new BinaryTree();

binaryTree.Add(1);

binaryTree.Add(2);

binaryTree.Add(7);

binaryTree.Add(3);

binaryTree.Add(10);

binaryTree.Add(5);

binaryTree.Add(8);

Node node = binaryTree.Find(5);

if (node != null)

{

Console.WriteLine("Đã tìm thấy node có giá trị bằng 5!");

}

else

{

Console.WriteLine("Không tìm thấy node!");

}

int depth = binaryTree.GetTreeDepth();

Console.WriteLine("PreOrder Traversal:");

binaryTree.TraversePreOrder(binaryTree.Root);

Console.WriteLine();

Console.WriteLine("InOrder Traversal:");

binaryTree.TraverseInOrder(binaryTree.Root);

Console.WriteLine();

Console.WriteLine("PostOrder Traversal:");

binaryTree.TraversePostOrder(binaryTree.Root);

Console.WriteLine();

binaryTree.Remove(7);

binaryTree.Remove(8);

Console.WriteLine("PreOrder Traversal After Removing Operation:");

binaryTree.TraversePreOrder(binaryTree.Root);

Console.WriteLine();

Console.ReadLine();

}