class IntTNode

{

private int data;

private IntTNode left;

private IntTNode right;

public int Data

{

get { return data; }

set { data = value; }

}

public IntTNode Left

{

get { return left; }

set { left = value; }

}

public IntTNode Right

{

get { return right; }

set { right = value; }

}

public IntTNode(int x = 0)

{

data = x;

left = null;

right = null;

}

public bool InsertNode(int x)

{

if (data == x) //Trùng giá trị

return false;

if (data > x) //Giá trị cần thêm nhỏ hơn -> chèn về trái

{

if (left == null)

left = new IntTNode(x);

else

return left.InsertNode(x);

}

else //Giá trị cần thêm lớn hơn -> chèn về phải

{

if (right == null)

right = new IntTNode(x);

else

return right.InsertNode(x);

}

return true;

}

public void NLR()

{

Console.Write(Data + "; ");

if (left != null)

left.NLR();

if (right != null)

right.NLR();

}

}

class MyBinaryTree

{

private IntTNode root;

public IntTNode Root

{

get { return root; }

set { root = value; }

}

public MyBinaryTree()

{

Root = null;

}

public bool Insert(int x)

{

if (Root == null)

{

Root = new IntTNode(x);

return true;

}

return Root.InsertNode(x);

}

public void Input()

{

do

{

int x;

Console.Write("Nhap vao gia tri (trung ket thuc): ");

int.TryParse(Console.ReadLine(), out x);

if (Insert(x)==true)

Console.WriteLine("Da them gia tri vao cay");

else

{

Console.WriteLine("Gia tri bi trung, ket thuc");

return;

}

} while (true);

}

public void PreOrder()

{

if (Root == null)

return;

Root.NLR();

Console.WriteLine();

}

// Bổ sung các phương thức mới vào đây

public IntTNode FindX(int x)

{

// Thực hiện tìm kiếm node có giá trị x

IntTNode current = Root;

while (current != null)

{

if (current.Data == x)

{

return current;

}

else if (current.Data > x)

{

current = current.Left;

}

else

{

current = current.Right;

}

}

return null; // Không tìm thấy node có giá trị x

}

public IntTNode FindMin()

{

// Thực hiện tìm kiếm node có giá trị nhỏ nhất

IntTNode current = Root;

while (current.Left != null)

{

current = current.Left;

}

return current;

}

public IntTNode FindMax()

{

// Thực hiện tìm kiếm node có giá trị lớn nhất

IntTNode current = Root;

while (current.Right != null)

{

current = current.Right;

}

return current;

}

public void RemoveX(int x)

{

// Thực hiện xóa node có giá trị x

Root = RemoveRec(Root, x);

}

private IntTNode RemoveRec(IntTNode root, int x)

{

if (root == null) return root;

if (x < root.Data)

root.Left = RemoveRec(root.Left, x);

else if (x > root.Data)

root.Right = RemoveRec(root.Right, x);

else

{

if (root.Left == null)

return root.Right;

else if (root.Right == null)

return root.Left;

root.Data = MinValue(root.Right);

root.Right = RemoveRec(root.Right, root.Data);

}

return root;

}

private int MinValue(IntTNode root)

{

int minv = root.Data;

while (root.Left != null)

{

minv = root.Left.Data;

root = root.Left;

}

return minv;

}

}