# ציור עץ

הפעולה מקבלת עץ מסוג char, string, int

public static void VisualDrawTree<T>(T t)

{

object o = t;

if (t.GetType() == typeof(BinNode<char>))

VisualBinTree<char>.DrawTree((BinNode<char>)o);

else if (t.GetType() == typeof(BinNode<int>))

VisualBinTree<int>.DrawTree((BinNode<int>)o);

else if (t.GetType() == typeof(BinNode<string>))

VisualBinTree<string>.DrawTree((BinNode<string>)o);

}

עבור עץ שהוא מסוג אוביקט ונרצה לצייר אותו, לקרא לפעולה שמחזירה עץ של מחרוזות

יש לדאוג שהפעולה ToString ממומשת באויקט

public static BinNode<string> BuildTreeString<T>(BinNode<T> t)

{

if (t == null)

return null;

BinNode<string> b = new BinNode<string>(

BuildTreeString(t.GetLeft()),

t.GetValue().ToString(),

BuildTreeString(t.GetRight()));

return b;

}

# בניית עץ ממערך

class TNVH<T>

{

public int left;

public int right;

public BinNode<T> node;

public TNVH(int left, int right, T value)

{

this.left = left;

this.right = right;

this.node = new BinNode<T>(value);

}

}

public static BinNode<T> CreateBinTreeFromArr<T>(TNVH<T>[] a)

{

int root = 1;

// find the root

bool[] bArr = new bool[a.Length + 1];

for (int i = 0; i < a.Length; i++)

{

bArr[a[i].left] = true;

bArr[a[i].right] = true;

}

while (bArr[root])

root++;

for (int i = 0; i < a.Length; i++)

{

if (a[i].left != 0)

a[i].node.SetLeft(a[a[i].left - 1].node);

if (a[i].right != 0)

a[i].node.SetRight(a[a[i].right - 1].node);

}

return a[root - 1].node;

}

public static BinNode<char> Tree1()

{

TNVH<char>[] vh = {

new TNVH<char>(0,0,'e'),

new TNVH<char>(1,0,'d'),

new TNVH<char>(0,0,'f'),

new TNVH<char>(3,2,'b'),

new TNVH<char>(4,7,'a'),

new TNVH<char>(0,0,'g'),

new TNVH<char>(0,6,'c'),

};

return TreeUtils.CreateBinTreeFromArr(vh);// root = 5

}

public static BinNode<int> Tree2()

{

TNVH<int>[] vh = {

new TNVH<int>(2,3,40), // 1

new TNVH<int>(4,5,12), // 2

new TNVH<int>(0,6,70), // 3

new TNVH<int>(7,8,25), // 4

new TNVH<int>(9,0,40), // 5

new TNVH<int>(10,0,41), // 6

new TNVH<int>(0,0,11), // 7

new TNVH<int>(0,0,81), // 8

new TNVH<int>(0,11,29), // 9

new TNVH<int>(12,13,27),// 10

new TNVH<int>(0,0,34), // 11

new TNVH<int>(14,15,68),// 12

new TNVH<int>(0,0,88), // 13

new TNVH<int>(16,0,55), // 14

new TNVH<int>(0,0,28), // 15

new TNVH<int>(0,0,49), // 16

};

return TreeUtils.CreateBinTreeFromArr(vh); // root = 1

}

public static BinNode<int> Tree2\_1()

{

TNVH<int>[] vh = {

new TNVH<int>(2,3,18), // 1

new TNVH<int>(4,5,31), // 2

new TNVH<int>(6,0,9), // 3

new TNVH<int>(0,0,7), // 4

new TNVH<int>(7,0,8), // 5

new TNVH<int>(0,8,14), // 6

new TNVH<int>(9,10,44), // 7

new TNVH<int>(11,12,10), // 8

new TNVH<int>(13,0,22), // 9

new TNVH<int>(0,0,1),// 10

new TNVH<int>(14,15,21), // 11

new TNVH<int>(0,0,2),// 12

new TNVH<int>(0,0,42), // 13

new TNVH<int>(0,0,27), // 14

new TNVH<int>(16,0,20), // 15

new TNVH<int>(0,0,4), // 16

};

return TreeUtils.CreateBinTreeFromArr(vh); // root = 1

}

public static BinNode<Pair> Tree5()

{

TNVH<Pair>[] vh = {

new TNVH<Pair>(2,3,new Pair(3,6)), // 1

new TNVH<Pair>(4,5,new Pair(11,2)), // 2

new TNVH<Pair>(6,7,new Pair(6,8)), // 3

new TNVH<Pair>(8,9,new Pair(5,61)), // 4

new TNVH<Pair>(0,10,new Pair(13,34)), // 5

new TNVH<Pair>(0,0,new Pair(5,9)), // 6

new TNVH<Pair>(0,0,new Pair(38,21)), // 7

new TNVH<Pair>(0,0,new Pair(37,45)), // 8

new TNVH<Pair>(0,0,new Pair(34,7)), // 9

new TNVH<Pair>(11,0,new Pair(13,14)), // 10

new TNVH<Pair>(0,0,new Pair(23,64)), // 11

};

return TreeUtils.CreateBinTreeFromArr(vh); // root = 1

}

public static BinNode<int> Tree6()

{

TNVH<int>[] vh = {

new TNVH<int>(2,3,50), // 1

new TNVH<int>(4,0,48), // 2

new TNVH<int>(5,6,70), // 3

new TNVH<int>(7,8,30), // 4

new TNVH<int>(0,9,65), // 5

new TNVH<int>(0,10,90), // 6

new TNVH<int>(11,12,20), // 7

new TNVH<int>(13,14,32), // 8

new TNVH<int>(15,16,67), // 9

new TNVH<int>(17,18,98), // 10

new TNVH<int>(0,0,15), // 11

new TNVH<int>(0,0,25), // 12

new TNVH<int>(0,0,31), // 13

new TNVH<int>(0,0,35), // 14

new TNVH<int>(0,0,66), // 15

new TNVH<int>(0,0,69), // 16

new TNVH<int>(0,0,94), // 17

new TNVH<int>(0,0,99), // 18

};

return TreeUtils.CreateBinTreeFromArr(vh); // root = 1

}

public static BinNode<int> Tree7()

{

TNVH<int>[] vh = {

new TNVH<int>(2,3,15), // 1

new TNVH<int>(4,5,6), // 2

new TNVH<int>(6,7,18), // 3

new TNVH<int>(8,9,3), // 4

new TNVH<int>(0,10,7), // 5

new TNVH<int>(0,0,17), // 6

new TNVH<int>(0,0,20), // 7

new TNVH<int>(0,0,2), // 8

new TNVH<int>(0,0,4), // 9

new TNVH<int>(11,0,13), // 10

new TNVH<int>(0,0,9), // 11

};

return TreeUtils.CreateBinTreeFromArr(vh); // root = 1

}