|  |
| --- |
| SEW |
| Generic Binary Tree |
| Treees are so beautiful! |

|  |
| --- |
| Felix  27.11.2021  3h |

# Inhaltsverzeichnis

Inhalt

[Inhaltsverzeichnis 1](#_Toc88063768)

[Angabe 1 <<…>> 2](#_Toc88063769)

[Theorie 1 <<…>> 2](#_Toc88063770)

[Klassendiagramm 1 <<…>> 2](#_Toc88063771)

[Quellcode 1 <<…>> 2](#_Toc88063772)

[Testfälle 1 <<…>> 4](#_Toc88063773)

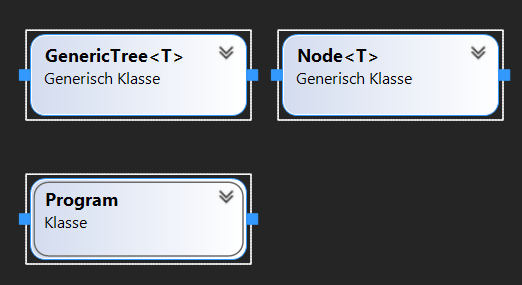
# Angabe 1 <<…>>

Programmiere einen Binary Tree mit Integer. Ich habe diesen gleich Generic programmiert, weil ich es so als einfacher empfinde.

# Theorie 1 <<…>>

Jeder Binary Tree hat eine Root-Node, die über eine Node Left und Right verfügt. Größere Werte bzw. weiter hinten liegende Zeichenketten werden immer im Right Node gespeichert. Andernfalls Left. In-, Pre- und PostOrder sind Ausgabemethoden, die entweder die Mitte in der Mitte, als Erstes oder Letztes ausgeben.

# Klassendiagramm 1 <<…>>



# Quellcode 1 <<…>>

using System;

using System.Collections.Generic;

using System.Text;

namespace GenericBinaryTree

{

public class GenericTree<T> where T : IComparable<T>

{

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

public Node<T> Find(T item)

{

Node<T> tmp = Root;

return FindRec(item, tmp);

}

public Node<T> FindRec(T item, Node<T> tmp)

{

if(tmp == null || tmp.Data.ToString() == item.ToString())

return tmp;

if (tmp.Data.CompareTo(item) < 0)

return FindRec(item, tmp.Right);

return FindRec(item, tmp.Left);

}

public int GetDepth()

{

Node<T> tmp = Root;

return GetDepth(tmp);

}

private int GetDepth(Node<T> tmp)

{

if (tmp == null)

return -1;

else

{

int lDepth = GetDepth(tmp.Left);

int rDepth = GetDepth(tmp.Right);

if(lDepth > rDepth)

return lDepth + 1;

else return rDepth + 1;

}

}

public void Insert(T[] arr)

{

foreach (var item in arr)

Insert(item);

}

public void Insert(T item)

{

Insert(new Node<T>(item));

}

public void Insert(Node<T> newNode)

{

if(Root == null)

Root = newNode;

else

{

Node<T> current = Root;

Node<T> parent;

while (true)

{

parent = current;

if(newNode.Data.CompareTo(current.Data) < 0)

{

current = current.Left;

if(current == null)

{

parent.Left = newNode;

return;

}

}

else

{

current = current.Right;

if(current == null)

{

parent.Right = newNode;

return;

}

}

}

}

}

public void PreOrder()

{

if (Root == null)

throw new TreeIsEmpty();

PreOrder(Root);

}

private void PreOrder(Node<T> tmp)

{

if (tmp != null)

{

Console.Write(tmp.Data.ToString() + ", ");

PreOrder(tmp.Left);

PreOrder(tmp.Right);

}

}

public void InOrder()

{

if (Root == null)

throw new TreeIsEmpty();

InOrder(Root);

}

private void InOrder(Node<T> tmp)

{

if (tmp != null)

{

InOrder(tmp.Left);

Console.Write(tmp.Data.ToString() + ", ");

InOrder(tmp.Right);

}

}

public void PostOrder()

{

if (Root == null)

throw new TreeIsEmpty();

PostOrder(Root);

}

private void PostOrder(Node<T> tmp)

{

if (tmp != null)

{

PostOrder(tmp.Left);

PostOrder(tmp.Right);

Console.Write(tmp.Data.ToString() + ", ");

}

}

}

public class TreeIsEmpty:Exception

{

public TreeIsEmpty() { }

public TreeIsEmpty(string msg):base(msg) { }

}

}

# Testfälle 1 <<…>>

using Microsoft.VisualStudio.TestTools.UnitTesting;

using GenericBinaryTree;

namespace UT\_GenericBinaryTree

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestFind()

{

GenericTree<int> genericTree = new GenericTree<int>();

genericTree.Insert(new int[] { 7, 1, 4, 2, 8, 3, 9 });

Assert.AreEqual(genericTree.Find(8), genericTree.Root.Right);

}

[TestMethod]

public void TestInsert()

{

GenericTree<int> genericTree = new GenericTree<int>();

genericTree.Insert(new int[] { 7, 1, 4, 2, 8, 3, 9 });

genericTree.Insert(6);

genericTree.Insert(new Node<int>(5));

Assert.AreEqual(genericTree.Root.Left.Data, 1);

Assert.AreEqual(genericTree.Root.Left.Right.Left.Data, 2);

Assert.AreEqual(genericTree.Root.Left.Right.Left.Right.Data, 3);

Assert.AreEqual(genericTree.Root.Left.Right.Data, 4);

Assert.AreEqual(genericTree.Root.Left.Right.Right.Left.Data, 5);

Assert.AreEqual(genericTree.Root.Left.Right.Right.Data, 6);

Assert.AreEqual(genericTree.Root.Data, 7);

Assert.AreEqual(genericTree.Root.Right.Data, 8);

Assert.AreEqual(genericTree.Root.Right.Right.Data, 9);

}

[TestMethod]

public void TestGetDepth()

{

GenericTree<int> genericTree = new GenericTree<int>();

genericTree.Insert(4);

genericTree.Insert(1);

genericTree.Insert(6);

genericTree.Insert(7);

genericTree.Insert(8);

Assert.AreEqual(genericTree.GetDepth(), 3);

}

[TestMethod]

[ExpectedException(typeof(TreeIsEmpty))]

public void TestTreeIsEmpty()

{

GenericTree<int> genericTree = new GenericTree<int>();

genericTree.InOrder();

}

}

}

Ein Bild, das Text, Monitor, Bildschirm, Screenshot enthält.

Automatisch generierte Beschreibung