Übungsaufgaben Informatik 4 Stünder KW18, Binäre Bäume

Wiederholung Abstrakte Datentypen und Algorithmen

Im Lehrplan finden sich vier ADTs, die explizit im Unterricht behandelt werden sollen:

Array, Verkettete Liste, Stack, Queue, Binärer Baum.

Ein schöner Übersichtsartikel findet sich hier: https://u-helmich.de/inf/BlueJ/kurs121/folge14/folge14-2.html

Simulation zum Thema „Linked List“: https://www.cs.usfca.edu/~galles/visualization/QueueLL.html

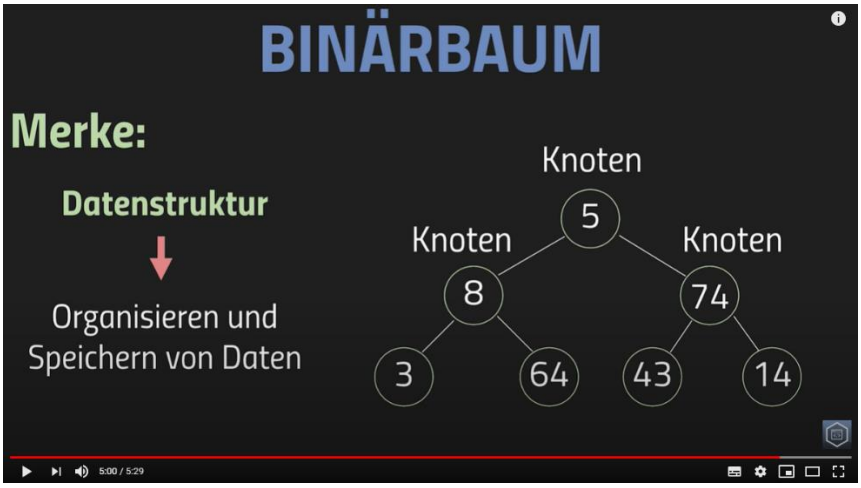
Simulation zum Thema „Queue“: https://www.cs.usfca.edu/~galles/visualization/QueueLL.html

Simulation zum Thema „Stack“: https://www.cs.usfca.edu/~galles/visualization/StackLL.html

Die ersten vier sind recht einfach, wir haben sie auch schon in der Abi-Wiederholung behandelt, der letzte ist der komplizierteste und daher vertiefen wir ihn jetzt:

Binärer Baum

https://www.youtube.com/watch?v=dqKoYntQHzc



Übung 1: Aufbauen der Datenstruktur

a) Baue einen geordneten Binärbaum aus dem Array [46, 25, 53, 24, 27, 52, 71]. Ist der Baum vollständig (ja/nein)?

b) Baue einen geordneten Binärbaum aus dem Array [12, 14, 6, 4, 20, 10, 7]. Ist der Baum vollstänig (ja/nein)?

c) einer der beiden Bäume ist vollständig, einer nicht. Strukturiere den unvollständigen Baum bitte so um, sodass er vollständig wird. Wieso lassen sich beide Bäume eigentlich vervollständigen? Stichwort: Anzahl an Knoten.

Hinweis: Wenn's mit dem Baumaufbau / Umstrukturieren garnicht klappt, bitte hier versuchen:
https://www.cs.usfca.edu/~galles/visualization/BST.html

Traversierung von Binärbäumen

„Traversierung" bedeutet, den Baum systematisch zu Durchlaufen. Zum Beispiel, um einen bestimmten Wert zu finden. Die Traversierung eines Baumes funktioniert algorithmisch entweder iterativ oder rekursiv – das heisst entweder mittels Programmschleifen oder mit sich-selbst-aufrufenden Funktionen. Letztere, also rekursive Algorithmen, sind ebenfalls abirelevant! Deshalb: Bevor wir die Traversierung üben, hier nochmal eine Wiederholung der Rekursion:

Exkurs: Rekursion

https://www.youtube.com/watch?v=weTpjhDnLnc

Schöne Visualisierung einer Fakultäts-Rekursion mit Stack!!

https://www.cs.usfca.edu/~galles/visualization/RecFact.html

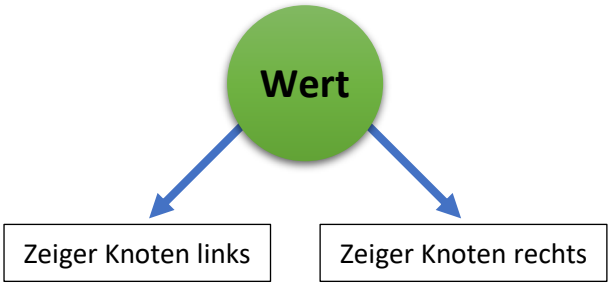
```java
/** Uebungsbeispiel 1: Gib an, was der Aufruf  rekBsp(7)  erbringt.*/
public void rekBsp(int n) {
    if (n <= 0) {return;}   // Abbruchfall
    System.out.println(n);
    rekBsp(n-3);
}
```

```java
/** Uebungsbeispiel 2: Gib an, was der Aufruf rekBsp2(5)  liefert. */
public int rekBsp2(int n) {
    if (n <= 0) {return 0;}   // Abbruchfall
    int z = rekBsp2(n-1);
    return (z+n);
}
```

Exkurs Ende

Es gibt drei unterschiedliche rekursive Methoden, um einen Binärbaum zu traversieren. Alle drei sind grundsätzlich nahezu identisch, unterscheiden sich lediglich in der Reihenfolge ihrer Einzelmethoden.

Sie leiten sich ab vom Aufbau eines Knotens:



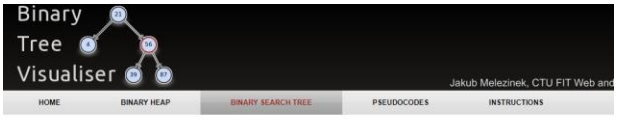
Die drei Einzelmethoden:

1. Lese den Wert des Knotens aus
2. ↙ Gehe zu Knoten links(*Rekursion*)
3. ↘ Gehe zu Knoten rechts(*Rekursion*)

Die drei Methoden nennt man je nach Abfolge dieser drei Unterfunktionen:

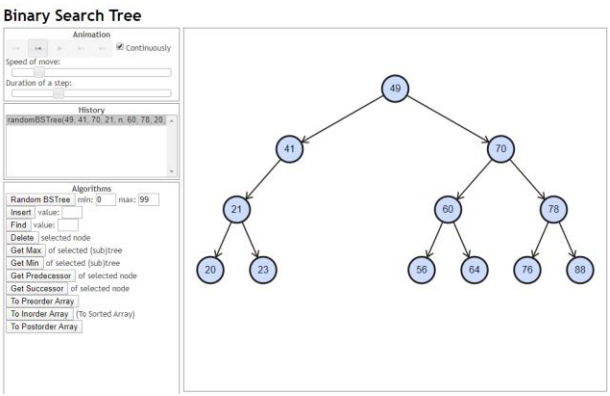
Pre-Order	In-Order	Post-Order
Lese den Wert des Knotens aus	↙ Gehe zu Knoten links(*Rekursion*)	↙ Gehe zu Knoten links(*Rekursion*)
↙ Gehe zu Knoten links(*Rekursion*)	Lese den Wert des Knotens aus	↘ Gehe zu Knoten rechts(*Rekursion*)
↘ Gehe zu Knoten rechts(*Rekursion*)	↘ Gehe zu Knoten rechts(*Rekursion*)	Lese den Wert des Knotens aus

Um die drei Methoden zu verstehen, bitte diese Simulation durcharbeiten:



http://btv.melezinek.cz/binary-search-tree.html

Bitte schau dir die Simulationen an:



„To Preorder Array“, „to Inorder Array“, „to Prostorder Array“

d) Übung: Nimm den Baum aus a) und generiere jeweils die Ausgabe-Arrays der drei rekursiven Algorithmen:

Implementierung in Java (auf der nächsten Seite)

Quelltext stammt ursprünglich von dieser Webseite:

https://www.sanfoundry.com/java-program-implement-binary-tree/

Auf der folgenden Seite sind die Klassen für den Knoten und die Klasse für den Baum. Es wird häufig mit dem Überladen von Funktionen gearbeitet. (**Erläutere, was das ist**).

```java
class BTNode
{
    BTNode left, right;
    int data;

    public BTNode()
    {
        left = null;
        right = null;
        data = 0;
    }
    public BTNode(int n)
    {
        left = null;
        right = null;
        data = n;
    }
    public void setLeft(BTNode n)
    {
        left = n;
    }
    public void setRight(BTNode n)
    {
        right = n;
    }
    public BTNode getLeft()
    {
        return left;
    }
    public BTNode getRight()
    {
        return right;
    }
    public void setData(int d)
    {
        data = d;
    }
    public int getData()
    {
        return data;
    }
}
```

```java
class BT{
    private BTNode root;
    public BT(){
        root = null;
    }
    public boolean isEmpty(){
        return root == null;
    }
    public void insert(int data){
        root = insert(root, data);
    }
    private BTNode insert(BTNode node, int data){
        if (node == null) {
            node = new BTNode(data);
        }else{
            if (node.getRight() == null) {
                node.right = insert(node.right, data);
            }else{
                node.left = insert(node.left, data);
            }
        }
        return node;
    }
    public int countNodes(){
        return countNodes(root);
    }
    private int countNodes(BTNode r){
        if (r == null){
            return 0;
        }else{
            int l = 1;
            l += countNodes(r.getLeft());
            l += countNodes(r.getRight());
            return l;
        }
    }
    public boolean search(int val){
        return search(root, val);
    }
    private boolean search(BTNode r, int val){
        if (r.getData() == val)
            return true;
        if (r.getLeft() != null)
            if (search(r.getLeft(), val))
                return true;
        if (r.getRight() != null)
            if (search(r.getRight(), val))
                return true;
        return false;
    }
    public void inorder(){
        inorder(root);
    }
    private void inorder(BTNode r){
        if (r != null){
            inorder(r.getLeft());
            System.out.print(r.getData() +" ");
            inorder(r.getRight());
        }
    }
    public void preorder(){
        preorder(root);
    }
    private void preorder(BTNode r){
        if (r != null){
            System.out.print(r.getData() +" ");
            preorder(r.getLeft());
            preorder(r.getRight());
        }
    }
    public void postorder(){
        postorder(root);
    }
    private void postorder(BTNode r){
        if (r != null){
            postorder(r.getLeft());
            postorder(r.getRight());
            System.out.print(r.getData() +" ");
        }
    }
}
```

Main-Klasse:

```
import java.util.Scanner;

public class BinaryTree
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        BT bt = new BT();
        /*  Perform tree operations  */
        System.out.println("Binary Tree Test\n");
        char ch;
        do
        {
            System.out.println("\nBinary Tree Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. search");
            System.out.println("3. count nodes");
            System.out.println("4. check empty");

            int choice = scan.nextInt();
            switch (choice)
            {
                case 1 :
                    System.out.println("Enter integer element to insert");
                    bt.insert( scan.nextInt() );
                    break;
                case 2 :
                    System.out.println("Enter integer element to search");
                    System.out.println("Search result : "+ bt.search( scan.nextInt() ));
                    break;
                case 3 :
                    System.out.println("Nodes = "+ bt.countNodes());
                    break;
                case 4 :
                    System.out.println("Empty status = "+ bt.isEmpty());
                    break;
                default :
                    System.out.println("Wrong Entry \n ");
                    break;
            }
            /*  Display tree  */
            System.out.print("\nPost order : ");
            bt.postorder();
            System.out.print("\nPre order : ");
            bt.preorder();
            System.out.print("\nIn order : ");
            bt.inorder();

            System.out.println("\n\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
    }
}
```

e) Erläutere die drei farblich gekennzeichneten Methoden inorder(), preorder() und postorder(), bzw. deren überladenen Methoden.