

Aufgabe 1-1

Aufgabe

Ähnlich wie der bereits bekannte "Größte Gemeinsame Teiler" (*Greatest Common Divisor, GCD*) findet auch das **Kleinste Gemeinsame Vielfache** (*Least Common Multiple, LCM*) Anwendung in elementaren mathematischen Anwendungen, etwa bei der Multiplikation von heterogenen Brüchen. Dies hat Anwendungen von der Anpassung von Kochrezepten an andere Portionsmengen bis hin zur Berechnung von Planetenbahnen.

Der *LCM* ist wie folgt definiert:
$$lcm(n_1, n_2) = \frac{n_1 \cdot n_2}{gcd(n_1, n_2)}$$

Implementieren Sie eine Funktion, die den *LCM* von zwei gegebenen Werten berechnet. Es darf dabei auf die *GCD*-Implementation aus der Vorlesung zurück gegriffen werden.

Hinweis: Das Verwenden von "import" ist nicht gestattet.

Beispiele

System.out.println(lcm(140, 72));	2520
-----------------------------------	------

Vorbelegung

<pre>int lcm(int a, int b) { return null; }</pre>

Testfälle

System.out.println(lcm(140, 72));	2520
System.out.println(studentAnswer.indexOf("import"));	-1
System.out.println(lcm(37, 973));	36001
System.out.println(lcm(2335, 258));	602430
System.out.println(lcm(13663, 15449));	211079687

Musterantwort

<pre>int gcd(int a, int b) {</pre>

```
    if (a < b) {
        int tmp = a;
        a = b;
        b = tmp;
    }
    int r = a % b;
    if (r == 0) {
        return b;
    }
    else {
        return gcd(r, b);
    }
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}
```

Aufgabe 2-1

Aufgabe

Implementieren Sie eine Klasse "Queue", die wie eine Schlange (wie in der Vorlesung beschrieben) funktioniert. Die Klasse soll mindestens die Methoden `emptyqueue()`, `head()`, `enqueue(x)` und `dequeue()` besitzen. Die Schlange muss nie mehr als 100 Elemente beinhalten können.

Hinweis 1: Im Java-Test wird erwartet, dass der Stack "Strings" speichert. Da Python dynamisch getypt ist, ist dies hier nicht der Fall.

Hinweis 2: Es ist nicht erlaubt, "import" zu verwenden!

Beispiele

<code>Queue q = new Queue(); System.out.println(q.emptyqueue());</code>	true
<code>Queue q = new Queue(); q.enqueue("Kekse"); q.enqueue("Kuchen"); System.out.println(q.head());</code>	Kekse

Vorbelegung

```
public class Queue {  
    public boolean emptyqueue() {  
        return false;  
    }  
  
    public String head() {  
        return "";  
    }  
  
    public void enqueue(String s) {  
    }  
  
    public String dequeue() {  
        return "";  
    }  
}
```

Testfälle

System.out.println(studentAnswer.indexOf("import "));	-1
Queue q = new Queue(); System.out.println(q.emptyqueue());	true
Queue q = new Queue(); q.enqueue("Kekse"); q.enqueue("Kuchen"); System.out.println(q.head());	true
Queue q = new Queue(); q.enqueue("Kekse"); q.enqueue("Kuchen"); q.dequeue(); System.out.println(q.head());	Kuchen
Queue q = new Queue(); for(int i = 1; i <= 100; ++i) { q.enqueue("" + i); } for(int i = 1; i <= 40; ++i) { q.dequeue(); } System.out.println(q.head());	41
Queue q = new Queue(); q.enqueue("Kakao"); System.out.println(q.emptyqueue());	false
Queue q = new Queue(); q.enqueue("Torte"); q.dequeue(); System.out.println(q.emptyqueue());	true
Queue q = new Queue(); q.enqueue("Torten"); System.out.println(q.dequeue()); System.out.println(q.emptyqueue());	Torten true
Queue q = new Queue(); Queue r = new Queue(); q.enqueue("Früchtetee"); System.out.println(r.emptyqueue()); r.enqueue(q.dequeue()); System.out.println(r.emptyqueue());	true false
Queue q = new Queue(); q.enqueue("6QLGYHCXUVDK56PCKH81KW54MRBFZNLAMTMNKF58"); System.out.println(q.head()); q.enqueue(q.dequeue()+"OTP564GCIB3IZS3N26E5"); System.out.println(q.head());	6QLGYHCXUVDK56PCKH81KW54MRBFZNLAMTMNKF58 6QLGYHCXUVDK56PCKH81KW54MRBFZNLAMTMNKF58OTP564GCIB3IZS3N26E5
Queue q = new Queue(); for(int i = 1; i <= 50; ++i) {	26

<pre> q.enqueue("" + i); } for(int i = 1; i <= 25; ++i) { q.dequeue(); } for(int i = 1; i <= 75; ++i) { q.enqueue("" + i); } System.out.println(q.head()); </pre>	
---	--

Musterantwort

```

public class Queue {
    private int start = 0;
    private int end = 0;
    private String[] buffer = new String[100];

    public boolean emptyqueue() {
        return start == end;
    }

    public String head() {
        return buffer[start];
    }

    public void enqueue(String s) {
        buffer[end] = s;
        end++;
        if(end == 100) {
            end = 0;
        }
    }

    public String dequeue() {
        String s = buffer[start];
        start++;
        if(start == 100) {
            start = 0;
        }
        return s;
    }
}

```

Aufgabe 2-2

Aufgabe

Die [Umgekehrte Polnische Notation / Postfix-Notation](#) ist eine spezielle Schreibweise für Mathematische Ausdrücke, bei der zuerst die Zahlen und danach die Operatoren angegeben werden. Der Ausdruck $(1 * 2) + (3 * 4)$ wäre dementsprechend in dieser Notation **1 2 * 3 4 * +**.

Ein Rechner für diese Notation lässt sich gut mit Hilfe eines Stacks abbilden. Dabei wird die Notation von links nach rechts durchgegangen. Falls eine Zahl bearbeitet wird, so wird diese auf den Stack gelegt. Wenn ein Operator kommt, dann werden die obersten beiden Zahlen vom Stack genommen, der Operator ausgeführt und das Ergebnis wieder auf den Stack gelegt. Nach der Rechnung ist das Ergebnis als einzige Zahl auf dem Stack.

In dieser Aufgabe sollen sie einen solchen Rechner in einer Methode implementieren. Dieser soll die Operatoren plus (+), minus (-), mal (*) und geteilt (/ abgerundet) verarbeiten können. **Sie erhalten dafür einen Stack mit folgenden bekannten Methoden, den Sie bei Ihrer Implementation verwenden sollen:**

```
public class IntegerStack {  
    public boolean emptystack();  
    public int head();  
    public void push(int i);  
    public int pop();  
}
```

Die Methode erhält als Eingabe eine Liste mit Strings, bei der die Zahlen und Operatoren bereits getrennt sind (z.B.: `String[] input = {"1", "2", "*", "3", "4", "*", "+"};`). Strings können Sie mit Hilfe von ["Integer.parseInt\(myString\);"](#) umwandeln. **Zudem erhalten Sie einen Stack, den Sie verwenden sollen.** Zurückgeben soll die Methode das richtige Ergebnis als Integer.

Sie können davon ausgehen, dass die Eingabe eine korrekte Formel ist.

Beispiele

<pre>IntegerStack s = new IntegerStack(); String[] input = {"2", "1", "-"}; int result = Calculator(input, s); System.out.println(result);</pre>	1
<pre>IntegerStack s = new IntegerStack(); String[] input = {"1", "2", "*", "3", "4", "*", "+"}; int result = Calculator(input, s); System.out.println(result);</pre>	14
<pre>IntegerStack s = new IntegerStack(); String[] input = {"4", "2", "/", "4", "4", "4", "/", "2", "*", "/", "-"}; int result = Calculator(input, s);</pre>	0

System.out.println(result);	
IntegerStack s = new IntegerStack(); String[] input = {"278241", "489", "/", "6456", "*, "46546", "+"}; int result = Calculator(input, s); System.out.println(result);	3720010

Vorbelegung

public int Calculator(String[] input, IntegerStack s) { return 0; }

Testfälle

IntegerStack s = new IntegerStack(); String[] input = {"2", "1", "-"}; int result = Calculator(input, s); System.out.println(result);	1
IntegerStack s = new IntegerStack(); String[] input = {"1", "2", "*", "3", "4", "*", "+"}; int result = Calculator(input, s); System.out.println(result);	14
IntegerStack s = new IntegerStack(); String[] input = {"4", "2", "/", "4", "4", "4", "/", "2", "*", "/", "-"}; int result = Calculator(input, s); System.out.println(result);	0
IntegerStack s = new IntegerStack(); String[] input = {"278241", "489", "/", "6456", "*, "46546", "+"}; int result = Calculator(input, s); System.out.println(result);	3720010
IntegerStack s = new IntegerStack(); String[] input = {"1", "2", "*", "3", "4", "*", "+"}; Calculator(input, s); System.out.println(s.compareHistory(new String[] {"[1]", "[1, 2]", "[1]", "[]", "[2]", "[2, 3]", "[2, 3, 4]", "[2, 3]", "[2]", "[2, 12]", "[2]", "[]", "[14]", "[]"}));	true
IntegerStack s = new IntegerStack(); String[] input = {"4", "2", "/", "4", "4", "4", "/", "2", "*", "/", "-"}; Calculator(input, s);	true

<pre>System.out.println(s.compareHistory(new String[] {"[4]", "[4, 2]", "[4]", "[]", "[2]", "[2, 4]", "[2, 4, 4]", "[2, 4, 4, 4]", "[2, 4, 4]", "[2, 4]", "[2, 4, 1]", "[2, 4, 1, 2]", "[2, 4, 1]", "[2, 4]", "[2, 4, 2]", "[2, 4]", "[2]", "[2, 2]", "[2]", "[]", "[0]", "[]"}));</pre>	
<pre>IntegerStack s = new IntegerStack(); String[] input = {"278241", "489", "/", "6456", "", "46546", "+"}; Calculator(input, s); System.out.println(s.compareHistory(new String[] {"[278241]", "[278241, 489]", "[278241]", "[]", "[569]", "[569, 6456]", "[569]", "[]", "[3673464]", "[3673464, 46546]", "[3673464]", "[]", "[3720010]", "[]"}));</pre>	true
<pre>IntegerStack s = new IntegerStack(); String[] input = {"1231", "123", "3210", "35", "+", "+", "+"}; System.out.println(Calculator(input, s)); System.out.println(s.compareHistory(new String[] {"[1231]", "[1231, 123]", "[1231, 123, 3210]", "[1231, 123, 3210, 35]", "[1231, 123, 3210]", "[1231, 123]", "[1231, 123, 3245]", "[1231, 123]", "[1231]", "[1231, 3368]", "[1231]", "[]", "[4599]", "[]"}));</pre>	4599 true
<pre>IntegerStack s = new IntegerStack(); String[] input = {"2874", "1757", "+", "8989", "4768", "+", "+"}; System.out.println(Calculator(input, s)); System.out.println(s.compareHistory(new String[] {"[2874]", "[2874, 1757]", "[2874]", "[]", "[4631]", "[4631, 8989]", "[4631, 8989, 4768]", "[4631, 8989]", "[4631]", "[4631, 13757]", "[4631]", "[]", "[18388]", "[]"}));</pre>	18388 true
<pre>IntegerStack s = new IntegerStack(); String[] input = {"561", "232", "+", "11158", "+"}; System.out.println(Calculator(input, s)); System.out.println(s.compareHistory(new String[] {"[561]", "[561, 232]", "[561]", "[]", "[793]", "[793, 11158]", "[793]", "[]", "[11951]", "[]"}));</pre>	11951 true
<pre>IntegerStack s = new IntegerStack(); String[] input = {"8541", "5965", "*", "646", "+", "6", "+", "84585", "-"}; System.out.println(Calculator(input, s)); System.out.println(s.compareHistory(new String[] {"[8541]", "[8541, 5965]", "[8541]", "[]", "[50947065]", "[50947065, 646]", "[50947065]", "[]", "[50947711]", "[50947711, 6]", "[50947711]", "[]", "[50947717]", "[50947717,</pre>	50863132 true

84585]", "[50947717]", "[]", "[50863132]", "[]"}));	
IntegerStack s = new IntegerStack(); int n1 = getRandom(); int n2 = getRandom(); int n3 = getRandom(); String[] input = {"" + n1, "" + n1, "/", "" + n2, "" + n2, "/", "" + n3, "" + n3, "/", "+", "+"}; System.out.println(Calculator(input, s));	3
IntegerStack s = new IntegerStack(); int n1 = getRandom(); int n2 = getRandom(); int n3 = getRandom(); String[] input = {"" + n1, "" + n2, "" + n3, "-", "-"}; System.out.println(Calculator(input, s) == (n1 - (n2 - n3)));	true

Musterantwort

```
// This implementation assumes input is a correct formula
public int Calculator(String[] input, IntegerStack s) {
    for(int i = 0; i < input.length; ++i) {
        switch(input[i]) {
            case "+":
                int n1 = s.pop();
                int n2 = s.pop();
                s.push(n2 + n1);
                break;
            case "-":
                n1 = s.pop();
                n2 = s.pop();
                s.push(n2 - n1);
                break;
            case "*":
                n1 = s.pop();
                n2 = s.pop();
                s.push(n2 * n1);
                break;
            case "/":
                n1 = s.pop();
                n2 = s.pop();
                s.push(n2 / n1);
                break;
            default:
                s.push(Integer.parseInt(input[i]));
                break;
        }
    }
}
```

```
    return s.pop();  
}
```

Aufgabe 3-1

Aufgabe

Implementieren sie einen beliebigen Suchalgorithmus aus der Vorlesung, welcher die übergebene Liste sortiert.

Die Liste hat folgende Operationen, welche als Eingabe Indexe der Listenelemente bekommt:

- `Swap(int x, int y) <-` Tauscht die Items an der Position x und y
- `Larger(int x, int y) <-` Liefert true zurück, falls das Element an Position x echt größer als das Element an Position y ist
- `Smaller(int x, int y) <-` Liefert true zurück, falls das Element an Position y echt größer als das Element an Position x ist
- `Equal(int x, int y) <-` Liefert true zurück, falls das Element an Position y gleich groß wie das Element an Position x ist
- `Length() <-` Liefert die Länge der Liste zurück

Beispiele

<pre>ListToSort lts = new ListToSort(Arrays.asList(2, 3, 1, 5, 99, 33)); lts = Sort(lts); System.out.println(lts.isSorted()); System.out.println(lts.isSame(Arrays.asList(1, 2, 3, 5, 33, 99)));</pre>	true true
<pre>ListToSort lts = new ListToSort(Arrays.asList(7, 7, 7, 7, 7, 7)); lts = Sort(lts); System.out.println(lts.isSorted());</pre>	true

Vorbelegung

<pre>public ListToSort Sort(ListToSort list) { return null; }</pre>

Testfälle

<pre>ListToSort lts = new ListToSort(Arrays.asList(2, 3, 1, 5, 99, 33));</pre>	true true
--	--------------

<pre> lts = Sort(lts); System.out.println(lts.isSorted()); System.out.println(lts.isSame(Arrays.asList(1, 2, 3, 5, 33, 99))); </pre>	
<pre> ListToSort lts = new ListToSort((Arrays.asList())); lts = Sort(lts); System.out.println(lts.isSorted()); </pre>	true
<pre> ArrayList<Integer> dummy = new ArrayList<>(256); for (int i = 0; i < 256; i++) { dummy.add(randrange(0, Integer.MAX_VALUE)); } ListToSort lts = new ListToSort(dummy); lts = Sort(lts); System.out.println(lts.isSorted()); </pre>	true
<pre> ArrayList<Integer> dummy = new ArrayList<>(256); for (int i = 0; i < 256; i++) { dummy.add(randrange(Integer.MIN_VALUE, Integer.MAX_VALUE)); } ListToSort lts = new ListToSort(dummy); lts = Sort(lts); System.out.println(lts.isSorted()); </pre>	true
<pre> ArrayList<Integer> dummy = new ArrayList<>(128); for (int i = 0; i < 128; i++) { dummy.add(i); } ListToSort lts = new ListToSort(dummy); lts = Sort(lts); System.out.println(lts.isSorted()); </pre>	true
<pre> ArrayList<Integer> dummy = new ArrayList<>(128); for (int i = 0; i < 128; i++) { dummy.add(127-i); } ListToSort lts = new ListToSort(dummy); lts = Sort(lts); System.out.println(lts.isSorted()); </pre>	true
<pre> ArrayList<Integer> dummy = new ArrayList<>(128); for (int i = 0; i < 128; i++) { dummy.add(9999); } ListToSort lts = new ListToSort(dummy); lts = Sort(lts); System.out.println(lts.isSorted()); </pre>	true
<pre> int rand = -randrange(1, 128); ArrayList<Integer> dummy = new ArrayList<>(128); for (int i = 0; i < 128; i++) { dummy.add(rand); } ListToSort lts = new ListToSort(dummy); </pre>	true

<pre>lts = Sort(lts); System.out.println(lts.isSorted());</pre>	
<pre>ListToSort lts = new ListToSort((Arrays.asList(42))); lts = Sort(lts); System.out.println(lts.isSorted());</pre>	true
<pre>ListToSort lts = new ListToSort((Arrays.asList(randrange(0, 9999999))))); lts = Sort(lts); System.out.println(lts.isSorted());</pre>	true
<pre>ListToSort lts = new ListToSort(Arrays.asList(7, 7, 7, 7, 7, 7)); lts = Sort(lts); System.out.println(lts.isSorted());</pre>	true

Musterantwort

```
public ListToSort Sort(ListToSort list) {
    // Implementation of BubbleSort
    for(int i=1; i < list.Length(); i++) {
        for(int j = list.Length()-1; j > i-1; j--) {
            if (list.Smaller(j, j-1)) {
                list.Swap(j, j-1);
            }
        }
    }
    return list;
}
```

Aufgabe 4-1

Aufgabe

In diesen Aufgaben wird es um Suchbäume gehen. Unsere Suchbäume sind aus "Nodes" aufgebaut, analog zur Vorlesung. Ein Node sieht wie folgt aus:

```
public class Node {  
    public int key;  
    public Node left;  
    public Node right;  
    public Node parent;  
}
```

In der Vorlesung haben Sie bereits einen Algorithmus für das Suchen in einem Suchbaum gesehen.

Sie sollen für die Suchbäume verschiedene Methoden implementieren. Implementieren Sie für diese Aufgabe eine Methode "insert", die eine "Node" im Suchbaum einfügt und den neuen Suchbaum zurück gibt. Dabei können die Knoten nach belieben wieder verwendet werden. Die Funktion bekommt dabei die Wurzel des Suchbaums übergeben sowie den neuen Knoten.

Beispiele

<pre>// build tree Node root = new Node(); root.key = 2; // build target Node target = new Node(); target.key = 1; // Call insert root = insert(root, target); // Compare Node NewRoot = new Node(); NewRoot.key = 2; NewRoot.left = new Node(); NewRoot.left.key = 1; NewRoot.left.parent = NewRoot; System.out.println(compareTrees(root, NewRoot));</pre>	true
<pre>// build tree Node root = new Node(); root.key = 2; // build target Node target = new Node(); target.key = 3;</pre>	true

<pre>// Call insert root = insert(root, target); // Compare Node NewRoot = new Node(); NewRoot.key = 2; NewRoot.right = new Node(); NewRoot.right.key = 3; NewRoot.right.parent = NewRoot; System.out.println(compareTrees(root, NewRoot));</pre>	
<pre>// build target Node target = new Node(); target.key = 15418; // Call insert Node root = insert(null, target); // Compare Node NewRoot = new Node(); NewRoot.key = 15418; System.out.println(compareTrees(root, NewRoot));</pre>	true
<pre>// build tree Node n1 = new Node(); n1.key = 72; Node n2 = new Node(); n2.key = 38; n1.parent = n2; n2.right = n1; n1 = new Node(); n1.key = 42; n2.parent = n1; n1.left = n2; n2 = new Node(); n2.key = 97; n2.parent = n1; n1.right = n2; Node root = new Node(); root.key = 17; root.right = n1; n1.parent = root; n1 = new Node(); n1.key = 1; n1.parent = root; root.left = n1; // build target Node target = new Node(); target.key = 27; // Call insert root = insert(root, target); // Compare n1 = new Node(); n1.key = 72;</pre>	true

<pre> n2 = new Node(); n2.key = 38; n1.parent = n2; n2.right = n1; n1 = new Node(); // target! n1.key = 27; n1.parent = n2; n2.left = n1; n1 = new Node(); n1.key = 42; n2.parent = n1; n1.left = n2; n2 = new Node(); n2.key = 97; n2.parent = n1; n1.right = n2; Node NewRoot = new Node(); NewRoot.key = 17; NewRoot.right = n1; n1.parent = NewRoot; n1 = new Node(); n1.key = 1; n1.parent = NewRoot; NewRoot.left = n1; System.out.println(compareTrees(root, NewRoot)); </pre>	
<pre> Node[] test = getRandom(); Node root = insert(test[0], test[1]); System.out.println(compareTrees(root, test[2])); </pre>	true
<pre> Node[] test = getRandom(); Node root = insert(test[0], test[1]); System.out.println(compareTrees(root, test[2])); </pre>	true
<pre> Node[] test = getRandom(); Node root = insert(test[0], test[1]); System.out.println(compareTrees(root, test[2])); </pre>	true
<pre> Node[] test = getRandom(); Node root = insert(test[0], test[1]); System.out.println(compareTrees(root, test[2])); </pre>	true

Vorbelegung

```

// Node in parameter represents root of tree
public Node insert(Node root, Node insert) {
    return null;
}

```


Testfälle

<pre>// build tree Node root = new Node(); root.key = 2; // build target Node target = new Node(); target.key = 1; // Call insert root = insert(root, target); // Compare Node NewRoot = new Node(); NewRoot.key = 2; NewRoot.left = new Node(); NewRoot.left.key = 1; NewRoot.left.parent = NewRoot; System.out.println(compareTrees(root, NewRoot));</pre>	true
<pre>// build tree Node root = new Node(); root.key = 2; // build target Node target = new Node(); target.key = 3; // Call insert root = insert(root, target); // Compare Node NewRoot = new Node(); NewRoot.key = 2; NewRoot.right = new Node(); NewRoot.right.key = 3; NewRoot.right.parent = NewRoot; System.out.println(compareTrees(root, NewRoot));</pre>	true

Musterantwort

```
public Node insert(Node root, Node insert) {
    Node x = root;
    Node y = null; // saves Node.Parent
    while (x != null) {
        y = x;
        if (insert.key < x.key) {
```

```
        x = x.left;
    } else {
        x = x.right;
    }
}

insert.parent = y;
if (y == null) {
    return insert;
} else if (insert.key < y.key) {
    y.left = insert;
} else {
    y.right = insert;
}

return root;
}
```

Aufgabe 4-2

Aufgabe

In diesen Aufgaben wird es wieder um Suchbäume gehen. Zur Erinnerung: Unsere Suchbäume sind aus "Nodes" aufgebaut, analog zur Vorlesung. Ein Node sieht wie folgt aus:

```
public class Node {  
    public int key;  
    public Node left;  
    public Node right;  
    public Node parent;  
}
```

Implementieren Sie für diese Aufgabe eine Methode "delete", die eine "Node" im Suchbaum löscht und den neuen Suchbaum zurück gibt. Nutzen Sie dabei die Methode aus den Vorlesungsfolien, also die mit "TREE-MINIMUM". Dabei können die Knoten nach belieben wieder verwendet werden. Die Funktion bekommt dabei die Wurzel des Suchbaums übergeben sowie den zu löschenden Knoten.

Beispiele

<pre>// build tree Node root = new Node(); root.key = 2; root.left = new Node(); root.left.key = 1; root.left.parent = root; root.right = new Node(); root.right.key = 3; root.right.parent = root; // Call delete root = delete(root, root.right); // Compare Node NewRoot = new Node(); NewRoot.key = 2; NewRoot.left = new Node(); NewRoot.left.key = 1; NewRoot.left.parent = NewRoot; System.out.println(compareTrees(root, NewRoot));</pre>	true
<pre>// build tree Node root = new Node(); root.key = 2; root.left = new Node(); root.left.key = 1; root.left.parent = root; root.right = new Node(); root.right.key = 3;</pre>	true

<pre> root.right.parent = root; root.right.right = new Node(); root.right.right.key = 99; root.right.right.parent = root.right; // Call delete root = delete(root, root.right); // Compare Node NewRoot = new Node(); NewRoot.key = 2; NewRoot.left = new Node(); NewRoot.left.key = 1; NewRoot.left.parent = NewRoot; NewRoot.right = new Node(); NewRoot.right.key = 99; NewRoot.right.parent = NewRoot; System.out.println(compareTrees(root, NewRoot)); </pre>	
<pre> // build tree Node root = new Node(); root.key = 2; root.left = new Node(); root.left.key = 1; root.left.parent = root; root.right = new Node(); root.right.key = 50; root.right.parent = root; root.right.right = new Node(); root.right.right.key = 99; root.right.right.parent = root.right; root.right.left = new Node(); root.right.left.key = 25; root.right.left.parent = root.right; // Call delete root = delete(root, root.right); // Compare Node newRoot = new Node(); newRoot.key = 2; newRoot.left = new Node(); newRoot.left.key = 1; newRoot.left.parent = newRoot; newRoot.right = new Node(); newRoot.right.key = 99; newRoot.right.parent = newRoot; newRoot.right.left = new Node(); newRoot.right.left.key = 25; newRoot.right.left.parent = newRoot.right; System.out.println(compareTrees(root, newRoot)); </pre>	true

Vorbelegung

// Node in parameter represents root of tree
--

```
public Node delete(Node root, Node delete) {
    return null;
}
```

Testfälle

<pre>// build tree Node root = new Node(); root.key = 2; root.left = new Node(); root.left.key = 1; root.left.parent = root; root.right = new Node(); root.right.key = 3; root.right.parent = root; // Call delete root = delete(root, root.right); // Compare Node NewRoot = new Node(); NewRoot.key = 2; NewRoot.left = new Node(); NewRoot.left.key = 1; NewRoot.left.parent = NewRoot; System.out.println(compareTrees(root, NewRoot));</pre>	true
<pre>// build tree Node root = new Node(); root.key = 2; root.left = new Node(); root.left.key = 1; root.left.parent = root; root.right = new Node(); root.right.key = 3; root.right.parent = root; root.right.right = new Node(); root.right.right.key = 99; root.right.right.parent = root.right; // Call delete root = delete(root, root.right); // Compare Node NewRoot = new Node(); NewRoot.key = 2; NewRoot.left = new Node(); NewRoot.left.key = 1; NewRoot.left.parent = NewRoot; NewRoot.right = new Node(); NewRoot.right.key = 99; NewRoot.right.parent = NewRoot; System.out.println(compareTrees(root, NewRoot));</pre>	true
<pre>// build tree Node root = new Node();</pre>	true

```

root.key = 2;
root.left = new Node();
root.left.key = 1;
root.left.parent = root;
root.right = new Node();
root.right.key = 50;
root.right.parent = root;
root.right.right = new Node();
root.right.right.key = 99;
root.right.right.parent = root.right;
root.right.left = new Node();
root.right.left.key = 25;
root.right.left.parent = root.right;
// Call delete
root = delete(root, root.right);
// Compare
Node newRoot = new Node();
newRoot.key = 2;
newRoot.left = new Node();
newRoot.left.key = 1;
newRoot.left.parent = newRoot;
newRoot.right = new Node();
newRoot.right.key = 99;
newRoot.right.parent = newRoot;
newRoot.right.left = new Node();
newRoot.right.left.key = 25;
newRoot.right.left.parent = newRoot.right;
System.out.println(compareTrees(root, newRoot));

```

```

// build tree
Node n1 = new Node();
n1.key = 37;
n1.left = new Node();
n1.left.key = 32;
n1.left.parent = n1;
n1.right = new Node();
n1.right.key = 78;
n1.right.parent = n1;
Node target = new Node();
target.key = 21;
target.right = n1;
n1.parent = target;
target.left = new Node();
target.left.key = 13;
target.left.parent = target;
Node root = new Node();
root.key = 85;
root.left = target;
target.parent = root;
root.right = new Node();
root.right.key = 942;
root.right.parent = root;
root.right.left = new Node();

```

true

<pre> root.right.left.key = 136; root.right.left.parent = root.right; // Call delete root = delete(root, target); // Compare Node n2 = new Node(); n2.key = 37; n2.right = new Node(); n2.right.key = 78; n2.right.parent = n2; Node replaced = new Node(); replaced.key = 32; replaced.right = n2; n2.parent = replaced; replaced.left = new Node(); replaced.left.key = 13; replaced.left.parent = replaced; Node newRoot = new Node(); newRoot.key = 85; newRoot.left = replaced; replaced.parent = newRoot; newRoot.right = new Node(); newRoot.right.key = 942; newRoot.right.parent = newRoot; newRoot.right.left = new Node(); newRoot.right.left.key = 136; newRoot.right.left.parent = newRoot.right; System.out.println(compareTrees(root, newRoot)); </pre>	
<pre> Node root = new Node(); root.key = 55; root = delete(root, root); System.out.println(compareTrees(root, null)); </pre>	true
<pre> Node[] test = getRandom(); Node root = delete(test[0], test[1]); System.out.println(compareTrees(root, test[2])); </pre>	true
<pre> Node[] test = getRandom(); Node root = delete(test[0], test[1]); System.out.println(compareTrees(root, test[2])); </pre>	true
<pre> Node[] test = getRandom(); Node root = delete(test[0], test[1]); System.out.println(compareTrees(root, test[2])); </pre>	true
<pre> Node[] test = getRandom(); Node root = delete(test[0], test[1]); System.out.println(compareTrees(root, test[2])); </pre>	true
<pre> // build tree Node root = new Node(); root.key = 2; root.left = new Node(); root.left.key = 1; root.left.parent = root; </pre>	true

```

root.right = new Node();
root.right.key = 4;
root.right.parent = root;
root.right.left = new Node();
root.right.left.key = 3;
root.right.left.parent = root.right;
// Call delete
root = delete(root, root.right);
// Compare
Node NewRoot = new Node();
NewRoot.key = 2;
NewRoot.left = new Node();
NewRoot.left.key = 1;
NewRoot.left.parent = NewRoot;
NewRoot.right = new Node();
NewRoot.right.key = 3;
NewRoot.right.parent = NewRoot;
System.out.println(compareTrees(root, NewRoot));

```

Musterantwort

```

// Node in parameter represents root of tree
public Node delete(Node root, Node delete) {
    Node y = null; //y: node to delete (after swap)
    if (delete.left == null || delete.right == null) {
        y = delete;
    } else {
        y = treeMinimum(delete.right);
    }
    Node x = null; // non-null child of y
    if (y.left != null) {
        x = y.left;
    } else {
        x = y.right;
    }
    // now delete y
    if (x != null) {
        x.parent = y.parent;
    }
    if (y.parent == null) {
        root = x;
    } else {
        if (y == y.parent.left) {
            y.parent.left = x;
        } else {
            y.parent.right = x;
        }
    }
    // Copy data
    if (y != delete) {
        delete.key = y.key;
    }
}

```

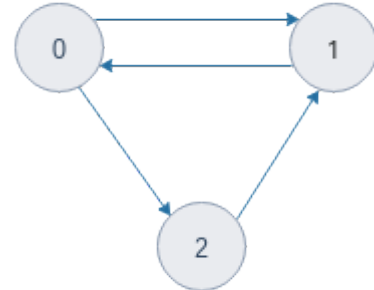


```
    }  
    // y should have no reference and should be deleted by the  
java gc  
    return root;  
}  
// Helper function  
public Node treeMinimum(Node n) {  
    while (n.left != null) {  
        n = n.left;  
    }  
    return n;  
}
```

Aufgabe 5-1

Aufgabe

In dieser Aufgabe geht es um die Implementierung von Graphen und sehr einfache Graphen-Algorithmen. In Java implementieren wir Graphen mit der Hilfe von Arrays. Wir schauen uns dabei Graphen mit ungewichteten Kanten an. Zur Abbildung eines Graphens in Java geben wir allen Knoten eine Nummer (startend mit 0), die dem Index der Kantenliste in der Liste entspricht. Beispiel:



```
int[][] graph = {  
    {1, 2},  
    {0},  
    {1}  
};
```

Jede dieser Kanten kann nun als gerichtet aufgefasst werden, ausgehen von dem Knoten mit der Indexnummer zu dem Knoten mit der entsprechenden Nummer (siehe Bild als graphische Visualisierung).

Nun wollen wir für die Graphen einige einfachen Algorithmen implementieren, dabei beginnen wir mit **Breitensuche**. Wie dieser Algorithmus abläuft ist in der Vorlesung zu finden, siehe auch Cormen et al, Kap. 22. Es sollen hier als Eingaben ungewichtete, gerichtete Graphen angenommen werden, es ist also nicht immer gegeben, dass es zu jeder Kante auch eine entgegengesetzt verlaufende Rückkante gibt.

Die Ausgaben soll dabei die Knoten des Graphen in der Reihenfolge enthalten, wie diese von der entsprechenden Suche abgearbeitet werden. Die Knoten sollen dabei in der Reihenfolge durchlaufen werden, in der sie in der Kantenliste angeordnet sind.

Sie können eine `java.util.ArrayList` oder einen `java.util.Stack` verwenden, falls Ihnen dies sinnvoll erscheint.

Beispiele

<pre>int[][] graph = { {1, 2}, {0, 3, 4}, {0, 5}, {1}, {1, 5}, {2, 4}, }; List<Integer> result = bfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 1, 2, 3, 4, 5]</pre>
---	-----------------------------------

<pre>int[][] graph = { {1,2}, {}, {} }; List<Integer> result = bfs(graph, 0); System.out.println(result);</pre>	[0, 1, 2]
<pre>int[][] graph = { {2,1}, {}, {} }; List<Integer> result = bfs(graph, 0); System.out.println(result);</pre>	[0, 2, 1]

Vorbelegung

```
public List<Integer> bfs(int[][] graph, int start) {
    List<Integer> result = new ArrayList<Integer>();
    return result;
}
```

Testfälle

<pre>int[][] graph = { {1,2}, {0,3,4}, {0,5}, {1}, {1,5}, {2,4}, }; List<Integer> result = bfs(graph, 0); System.out.println(result);</pre>	[0, 1, 2, 3, 4, 5]
<pre>int[][] graph = { {1,2}, {}, {} }; List<Integer> result = bfs(graph, 0); System.out.println(result);</pre>	[0, 1, 2]
<pre>int[][] graph = { {2,1}, {}, {} }; List<Integer> result = bfs(graph, 0); System.out.println(result);</pre>	[0, 2, 1]

<pre>int[][] graph = {{1, 2}, {0, 3, 4}, {0, 5}, {1}, {1, 5}, {2, 4}}; List<Integer> result = bfs(graph, 3); System.out.println(result);</pre>	<pre>[3, 1, 0, 4, 2, 5]</pre>
<pre>int[][] graph = {{1}, {0, 3, 4}, {0, 5}, {1}, {1, 5}, {2, 4}}; List<Integer> result = bfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 1, 3, 4, 5, 2]</pre>
<pre>int[][] graph = {{0}, {2}, {1}, {4}, {}, {}}; System.out.println(bfs(graph, 0)); System.out.println(bfs(graph, 1)); System.out.println(bfs(graph, 3)); System.out.println(bfs(graph, 4));</pre>	<pre>[0] [1, 2] [3, 4] [4]</pre>
<pre>int[][] graph = {{1, 2}, {0, 3, 4}, {0, 5}, {1}, {1, 5}, {2, 4}}; System.out.println(bfs(graph, 5));</pre>	<pre>[5, 2, 4, 0, 1, 3]</pre>
<pre>Random rand = new Random(); int[][] graph = new int[50][]; for(int i = 0; i < 50; ++i) { int n = rand.nextInt(20); int[] nodeList = new int[n]; for(int j = 0; j < n; ++j) { nodeList[j] = rand.nextInt(50); } graph[i] = nodeList; } int start = rand.nextInt(50); System.out.println(isCorrect(graph, start, bfs(graph, start)));</pre>	<pre>true</pre>
<pre>Random rand = new Random();</pre>	<pre>true</pre>

<pre>int[][] graph = new int[100][]; for(int i = 0; i < 100; ++i) { int n = rand.nextInt(40); int[] nodeList = new int[n]; for(int j = 0; j < n; ++j) { nodeList[j] = rand.nextInt(100); } graph[i] = nodeList; } int start = rand.nextInt(100); System.out.println(isCorrect(graph, start, bfs(graph, start)));</pre>	
<pre>Random rand = new Random(); int[][] graph = new int[200][]; for(int i = 0; i < 200; ++i) { int n = rand.nextInt(60); int[] nodeList = new int[n]; for(int j = 0; j < n; ++j) { nodeList[j] = rand.nextInt(200); } graph[i] = nodeList; } int start = rand.nextInt(200); System.out.println(isCorrect(graph, start, bfs(graph, start)));</pre>	true
<pre>int[][] graph = {{}}; System.out.println(bfs(graph, 0));</pre>	[0]

Musterantwort

```
public List<Integer> bfs(int[][] graph, int start) {
    List<Integer> visited = new ArrayList<Integer>();
    List<Integer> queue = new ArrayList<Integer>();
    queue.add(start);

    while(!queue.isEmpty()){
        int next = queue.remove(0);
        if(!visited.contains(next)) {
            visited.add(next);
            for(int vertex: graph[next]) {
                if(!visited.contains(vertex)) {
                    queue.add(vertex);
                }
            }
        }
    }
    return visited;
}
```

Aufgabe 5-2

Aufgabe

Als zweiter Algorithmus soll nun - analog zur ersten Aufgabe - eine **Tiefensuche** implementiert werden.

Auch hier sollen die Knoten in der Reihenfolge besucht werden, in der sie in der Kantenliste auftauchen.

Sie können eine `java.util.ArrayList` oder einen `java.util.Stack` verwenden, falls Ihnen dies sinnvoll erscheint.

Beispiele

<pre>int[][] graph = { {1,2}, {0,3,4}, {0,5}, {1}, {1,5}, {2,4}, }; List<Integer> result = dfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 1, 3, 4, 5, 2]</pre>
<pre>int[][] graph = { {1,2}, {}, {} }; List<Integer> result = dfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 1, 2]</pre>
<pre>int[][] graph = { {2,1}, {}, {} }; List<Integer> result = dfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 2, 1]</pre>

Vorbelegung

<pre>public List<Integer> dfs(int[][] graph, int start) { List<Integer> result = new ArrayList<Integer>(); return result; }</pre>

Testfälle

<pre>int[][] graph = { {1,2}, {0,3,4}, {0,5}, {1}, {1,5}, {2,4}, }; List<Integer> result = dfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 1, 3, 4, 5, 2]</pre>
<pre>int[][] graph = { {1,2}, {}, {} }; List<Integer> result = dfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 1, 2]</pre>
<pre>int[][] graph = { {2,1}, {}, {} }; List<Integer> result = dfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 2, 1]</pre>
<pre>int[][] graph = {{1, 2}, {0, 3, 4}, {0, 5}, {1}, {1, 5}, {2, 4} }; List<Integer> result = dfs(graph, 3); System.out.println(result);</pre>	<pre>[3, 1, 0, 2, 5, 4]</pre>
<pre>int[][] graph = {{1}, {0, 3, 4}, {0, 5}, {1}, {1, 5}, {2, 4}, }; List<Integer> result = dfs(graph, 0); System.out.println(result);</pre>	<pre>[0, 1, 3, 4, 5, 2]</pre>
<pre>int[][] graph = {{0}, {2}, {1}, {4}, {}, {} }; };</pre>	<pre>[0] [1, 2] [3, 4] [4]</pre>

<pre> System.out.println(dfs(graph, 0)); System.out.println(dfs(graph, 1)); System.out.println(dfs(graph, 3)); System.out.println(dfs(graph, 4)); </pre>	
<pre> int[][] graph = {{1, 2}, {0, 3, 4}, {0, 5}, {1}, {1, 5}, {2, 4} }; System.out.println(dfs(graph, 5)); </pre>	<pre> [5, 2, 0, 1, 3, 4] </pre>
<pre> Random rand = new Random(); int[][] graph = new int[50][]; for(int i = 0; i < 50; ++i) { int n = rand.nextInt(20); int[] nodeList = new int[n]; for(int j = 0; j < n; ++j) { nodeList[j] = rand.nextInt(50);; } graph[i] = nodeList; } int start = rand.nextInt(50); System.out.println(isCorrect(graph, start, dfs(graph, start))); </pre>	true
<pre> Random rand = new Random(); int[][] graph = new int[100][]; for(int i = 0; i < 100; ++i) { int n = rand.nextInt(40); int[] nodeList = new int[n]; for(int j = 0; j < n; ++j) { nodeList[j] = rand.nextInt(100);; } graph[i] = nodeList; } int start = rand.nextInt(100); System.out.println(isCorrect(graph, start, dfs(graph, start))); </pre>	true
<pre> Random rand = new Random(); int[][] graph = new int[200][]; for(int i = 0; i < 200; ++i) { int n = rand.nextInt(60); int[] nodeList = new int[n]; for(int j = 0; j < n; ++j) { nodeList[j] = rand.nextInt(200);; } graph[i] = nodeList; } int start = rand.nextInt(200); System.out.println(isCorrect(graph, start, dfs(graph, start))); </pre>	true


```
int[][] graph = {{}};  
System.out.println(dfs(graph, 0));
```

```
[0]
```

Musterantwort

```
public List<Integer> dfs(int[][] graph, int start) {  
    List<Integer> visited = new ArrayList<Integer>();  
    Stack<Integer> stack = new Stack<Integer>();  
    stack.push(start);  
  
    while(!stack.isEmpty()){  
        int next = stack.pop();  
        if(!visited.contains(next)) {  
            visited.add(next);  
            for(int i = graph[next].length-1; i >= 0; --i) { //  
Need to traverse list in reverse order to visit vertex in correct  
order  
                if(!visited.contains(graph[next][i])) {  
                    stack.push(graph[next][i]);  
                }  
            }  
        }  
    }  
    return visited;  
}
```

Aufgabe 6-1

Aufgabe

HINWEIS: Von den beiden Teilaufgaben in diesem Block muss nur eine gelöst werden! Wer beide löst, bekommt Bonuspunkte.

In dieser Aufgabe soll die Edit Distance von zwei gegebenen Strings implementiert werden. Diese ist definiert als die minimale Anzahl von Einfüge-, Lösch- und Ersetzungsoperation von einzelnen Zeichen um den einen String in den anderen Umzuwandeln. Wichtig: es ist hier nach einer Implementation OHNE rekursive Aufrufe gefragt.

Beispiele

<code>System.out.println(editDistance("", ""));</code>	0
<code>System.out.println(editDistance("a", "a"));</code>	0
<code>System.out.println(editDistance("a", ""));</code>	1
<code>System.out.println(editDistance("", "a"));</code>	1
<code>System.out.println(editDistance("ac", "abc"));</code>	1
<code>System.out.println(editDistance("abc", "ac"));</code>	1
<code>System.out.println(editDistance("a", "b"));</code>	1

Vorbelegung

```
public int editDistance(String a, String b) {  
    return 0;  
}
```

Testfälle

<code>System.out.println(studentAnswer.split("editDistance\\(", -1).length-1);</code>	1
<code>System.out.println(editDistance("", ""));</code>	0
<code>System.out.println(editDistance("a", "a"));</code>	0
<code>System.out.println(editDistance("a", ""));</code>	1
<code>System.out.println(editDistance("", "a"));</code>	1
<code>System.out.println(editDistance("ac", "abc"));</code>	1
<code>System.out.println(editDistance("abc", "ac"));</code>	1
<code>System.out.println(editDistance("a", "b"));</code>	1
<code>System.out.println(editDistance("abc", ""));</code>	3

System.out.println(editDistance("", "abc"));	3
System.out.println(editDistance("", ""));	0
System.out.println(editDistance("cut", "cat"));	1
System.out.println(editDistance("abc", "abc"));	0
System.out.println(editDistance("geek", "gesek"));	1
System.out.println(editDistance("blablablabla", "abs"));	10
System.out.println(editDistance("abc", "abs"));	1
String s2 = "weqweqweqwesqwesqweweqweqweqweqseqwesqssswewqse" ; String t2 = "qswe"; System.out.println(editDistance(s2,t2));	42
String s = getRandom(5); String t = getRandom(5); System.out.println(isCorrect(s, t, editDistance(s,t)));	true
String s = getRandom(5); String t = getRandom(10); System.out.println(isCorrect(s, t, editDistance(s,t)));	true
String s = getRandom(50); String t = getRandom(100); System.out.println(isCorrect(s, t, editDistance(s,t)));	true
String s = "91336388630930104036509887910612766733414220177 834951661736334124642044989286936143277410676715 4961"; String t = "91336388630930136509887910612766733414220177834 900051661736334124642044989286936143277410676700 0961"; System.out.println(editDistance(s,t));	9

Musterantwort

```

/*
 * This function calculates and returns the edit distance
 * between tow given strings. this strings are given by the
 * parameters a and b. In the implementation the recursive
 * definition is implemented iterativly
 */

public int editDistance(String a, String b) {

```

```
int m = a.length();
int n = b.length();
int[][] edDis = new int[m+1][];
for(int i = 0; i <= m; ++i) {
    edDis[i] = new int[n+1];
    edDis[i][0] = i;
}
for(int i = 0; i <= n; ++i) {
    edDis[0][i] = i;
}

for(int i = 1; i <= m; ++i) {
    for(int j = 1; j <= n; ++j) {
        if(a.charAt(i-1) == b.charAt(j-1)) {
            edDis[i][j] = edDis[i-1][j-1];
        } else {
            edDis[i][j] = 1 + Math.min(Math.min(edDis[i][j-1],
edDis[i-1][j]), edDis[i-1][j-1]);
        }
    }
}
return edDis[a.length()][b.length()];
}
```

Aufgabe 6-2

Aufgabe

HINWEIS: Von den beiden Teilaufgaben in diesem Block muss nur eine gelöst werden! Wer beide löst, bekommt Bonuspunkte.

Das Knapsack-Problem ist ein Optimierungsproblem. Welche Gegenstände sollen aus einer vorhandenen Auswahl in einen begrenzten Rucksack(Knapsack) gepackt werden, um den zu transportierenden Wert zu maximieren? Diese Problem ist im allgemeinen NP-vollständig und daher wahrscheinlich nicht effizient lösbar. Unter der Annahme, dass die Volumina der Gegenstände ganzzahlig sind, finde sich ein Algorithmus, der diese Problem mittels dynamischer Programmierung löst. Gesucht ist nun die Implementierung eines Algorithmus, der für eine gegebenen Liste von Gegenständen und einem maximalen Rucksackvolumen den Wert ausgibt, der maximal transportiert werden kann. Die Liste aus Gegenständen besteht hierbei aus zwei Listen, eine enthält die Volumina und eine die Werte. Die Eingabe $([0.5, 0.3], [2, 5], 3)$ gibt also zwei Gegenstände, den ersten mit Volumen 2 und Wert 0.5, den zweiten mit Volumen 5 und Wert 0.3. Der letzte Wert gibt das Volumen des Knapsackes an, in diesem Fall wäre also die richtige Ausgabe des Algorithmus 0.5, da nur der 1. Gegenstand überhaupt in den Rucksack passt. Wäre ein größeres Volumen von 7 gegeben, wäre die richtige Ausgabe 0.8 Wichtig ist hierbei noch anzumerken, dass die Lösung NICHT 1.5 ist, es ist also nicht möglich den gleiche Gegenstand mehrfach zu verwenden.

Beispiele

<pre>double[] c = {0.5, 1.3}; int[] v = {2, 5}; int m = 3; System.out.println(knapSack(c,v,m));</pre>	0.5
<pre>double[] c = {6, 8}; int[] v = {6, 8}; int m = 0; System.out.println(knapSack(c,v,m));</pre>	0.0
<pre>double[] c = {6, 8}; int[] v = {6, 8}; int m = 10; System.out.println(knapSack(c,v,m));</pre>	8.0
<pre>double[] c = {1, 1, 99}; int[] v = {6, 4, 11}; int m = 10; System.out.println(knapSack(c,v,m));</pre>	2.0
<pre>double[] c = {6, 8}; int[] v = {6, 8}; int m = 17; System.out.println(knapSack(c,v,m));</pre>	14.0

Vorbelegung

```
public double knapSack(double[] values, int[] volumes, int capacity) {  
    assert values.length == volumes.length;  
    return 0.0;  
}
```

Testfälle

System.out.println(studentAnswer.split("knapSack\\(", -1).length-1);	1
double[] c = {0.5, 1.3}; int[] v = {2, 5}; int m = 3; System.out.println(knapSack(c,v,m));	0.5
double[] c = {}; int[] v = {}; int m = 10; System.out.println(knapSack(c,v,m));	0.0
double[] c = {6, 8}; int[] v = {6, 8}; int m = 0; System.out.println(knapSack(c,v,m));	0.0
double[] c = {6, 8}; int[] v = {6, 8}; int m = 10; System.out.println(knapSack(c,v,m));	8.0
double[] c = {1, 1, 99}; int[] v = {6, 4, 11}; int m = 10; System.out.println(knapSack(c,v,m));	2.0
double[] c = {6, 8}; int[] v = {6, 8}; int m = 17; System.out.println(knapSack(c,v,m));	14.0
double[] c = {2, 6, 11}; int[] v = {2, 6, 11}; int m = 18; System.out.println(knapSack(c,v,m));	17.0
double[] c = {2, 6, 100}; int[] v = {2, 6, 17}; int m = 18; System.out.println(knapSack(c,v,m));	100.0
double[] c = {2, 2, 2, 2, 2, 2}; int[] v = {1, 2, 3, 4, 5, 6}; int m = 9;	6.0

System.out.println(knapSack(c,v,m));	
double[] c = {10, 2, 10, 2, 2, 2}; int[] v = {1, 2, 3, 4, 5, 6}; int m = 9; System.out.println(knapSack(c,v,m));	22.0
double[] c = {10, 2, 10, 2, 2, 2}; int[] v = {1, 2, 3, 4, 5, 6}; int m = 100; System.out.println(knapSack(c,v,m));	28.0
double[] c = new double[100]; int[] v = new int[100]; for(int i = 0; i < 100; ++i){ c[i] = i; v[i] = i; } int m = 100; System.out.println(knapSack(c,v,m));	100.0
double[] c = new double[100]; int[] v = new int[100]; for(int i = 0; i < 100; ++i){ c[i] = 99-i; v[i] = 99-i; } int m = 100; System.out.println(knapSack(c,v,m));	100.0
double[] c = new double[100]; int[] v = new int[100]; for(int i = 0; i < 100; ++i){ c[i] = i*2; v[i] = i; } int m = 100; System.out.println(knapSack(c,v,m));	200.0
Random rand = new Random(); double[] c = new double[100]; int[] v = new int[100]; double sum = 0; for(int i = 0; i < 100; ++i){ int n = rand.nextInt(100); c[i] = n; v[i] = i; sum += n; } int m = 4951; System.out.println(knapSack(c,v,m) == sum);	true
Random rand = new Random(); double[] c = new double[100]; int[] v = new int[100]; double sum = 0; for(int i = 0; i < 100; ++i){	true

<pre> int n = rand.nextInt(100); c[i] = n; v[i] = i; sum += n; } int m = 4951; System.out.println(knapSack(c,v,m) == sum); </pre>	
<pre> Random rand = new Random(); double[] c = new double[100]; int[] v = new int[100]; for(int i = 0; i < 100; ++i){ int n = rand.nextInt(100); c[i] = n; v[i] = i; } int m = 4951/4; System.out.println(isCorrect(c, v, m, knapSack(c,v,m))); </pre>	true
<pre> Random rand = new Random(); double[] c = new double[100]; int[] v = new int[100]; for(int i = 0; i < 100; ++i){ int n = rand.nextInt(100); c[i] = n; v[i] = i; } int m = 4951/4; System.out.println(isCorrect(c, v, m, knapSack(c,v,m))); </pre>	true
<pre> Random rand = new Random(); double[] c = new double[100]; int[] v = new int[100]; for(int i = 0; i < 100; ++i){ int n = rand.nextInt(100); c[i] = n; v[i] = i; } int m = 4951/4; System.out.println(isCorrect(c, v, m, knapSack(c,v,m))); </pre>	true

Musterantwort

```

/* This function calculates and returns the max
 * value in knapsack with the capacity "capacity" wich is filled
 * with stuff of volume "volumes" and the value "values".
 */
public double knapSack(double[] values, int[] volumes, int
capacity) {

```



```

    assert values.length == volumes.length;

    int n = values.length;
    double[][] Result = new double[n+1][];

    for(int i = 0; i < n+1; ++i) {
        Result[i] = new double[capacity+1];
    }

    for(int i = n-1; i > -1; --i) {
        for(int j = 0; j < capacity+1; ++j) {
            if (volumes[i] <= j) {
                Result[i][j] = Math.max(values[i] + Result[i+1][j-
volumes[i]], Result[i+1][j]);
            } else {
                Result[i][j] = Result[i+1][j];
            }
        }
    }
    return Result[0][capacity];
}

```