

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:
$$lcm(n_1, n_2) = \frac{n_1 \cdot n_2}{gcd(n_1, n_2)} \quad .$$

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

<code>print(int(LCM(140, 72)))</code>	2520
---------------------------------------	------

Vorbelegung

<code>def LCM(a, b): pass</code>
--

Testfälle

<code>print(int(LCM(140, 72)))</code>	2520
<code>print(__student_answer__.find("import"))</code>	-1
<code>print(int(LCM(37, 973)))</code>	36001
<code>print(int(LCM(2335, 258)))</code>	602430
<code>print(int(LCM(13663, 15449)))</code>	211079687

Musterantwort

<code>def GCD(a, b): if a < b: a, b = b, a</code>
--

```
    r = a % b
    if r is 0:
        return b
    else:
        return GCD(r, b)

def LCM(a, 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 isEmpty(), 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

<pre>q = Queue() print(q.isEmpty())</pre>	True
<pre>q = Queue() q.enqueue("Kekse") q.enqueue("Kuchen") print(q.head())</pre>	Kekse

Vorbelegung

```
class Queue:
    """Klasse, die eine selbstgebaute Queue darstellt.
    """

    def isEmpty(self):
        """Prueft, ob die Queue leer ist.
        :return: True, wenn die Queue leer ist;
                False, sonst
        """
        pass

    def head(self):
        """Gibt den Wert des ersten Elements in der Queue
        zurueck.
        :return: Den Wert des ersten Elementes in der Queue
        """
        pass

    def enqueue(self, x):
        """Haengt ein Element an die Queue an.
```

```

        :param x: Anzuhaengendes Element
        """
        pass

    def dequeue(self):
        """Entfernt das erste Element aus der Queue
        :return: Erstes Element
        """
        pass

```

Testfälle

<code>print(__student_answer__.find("import"))</code>	-1
<code>q = Queue() print(q.isEmpty())</code>	True
<code>q = Queue() q.enqueue("Kekse") q.enqueue("Kuchen") print(q.head())</code>	Kekse
<code>q = Queue() for i in range(1,101): q.enqueue(i) for i in range(40): q.dequeue() print(q.head())</code>	41
<code>q = Queue() q.enqueue("Kakao") print(q.isEmpty())</code>	False
<code>q = Queue() q.enqueue("Kekse") q.enqueue("Kuchen") q.dequeue() print(q.head())</code>	Kuchen
<code>q = Queue() q.enqueue("Torten") print(q.dequeue()) print(q.isEmpty())</code>	Torten True
<code>q = Queue() r = Queue() q.enqueue("Kamillentee") print(r.isEmpty()) r.enqueue(q.dequeue()) print(r.isEmpty())</code>	True False
<code>q = Queue() q.enqueue("6QLGYHCXUVDK56PCKH81KW54MRBFZNLAMTMNKF58") print(q.head())</code>	6QLGYHCXUVDK56PCK H81KW54MRBFZNLAMT MNKF58 6QLGYHCXUVDK56PCK

<pre>q.enqueue("{}OTP564GCIB3IZS3N26E5".format(q.dequeue())) print(q.head())</pre>	H81KW54MRBFZNLAMT MNKF580TP564GCIB3 IZS3N26E5
<pre>q = Queue() for i in range(1,51): q.enqueue(i) for i in range(25): q.dequeue() for i in range(1,76): q.enqueue(i) print(q.head())</pre>	26

Musterantwort

```
class Queue:
    """Klasse, die eine selbstgebaute Queue darstellt.
    """
    def __init__(self):
        self.lst = []

    def isEmpty(self) -> bool:
        """Prueft, ob die Queue leer ist.
        :return: True, wenn die Queue leer ist;
                False, sonst
        """
        return len(self.lst) == 0

    def head(self):
        """Gibt den Wert des ersten Elements in der Queue
        zurueck.
        :return: Den Wert des ersten Elementes in der Queue
        """
        if not self.isEmpty():
            return self.lst[0]
        else:
            return None

    def enqueue(self, x):
        """Haengt ein Element an die Queue an.
        :param x: Anzuhaengendes Element
        """
        self.lst.append(x)

    def dequeue(self):
        """Entfernt das erste Element aus der Queue
        :return: Erstes Element
        """
        if not self.isEmpty():
            returnvalue = self.lst[0]
            for i in range(len(self.lst) - 1):
```

```
        self.lst[i] = self.lst[i+1]
    del self.lst[-1]
    return returnvalue
else:
    raise IndexError
```

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:**

- * emptystack()
- * head()
- * push(int i)
- * pop()

Die Methode erhält als Eingabe eine Liste mit Strings, bei der die Zahlen und Operatoren bereits getrennt sind (z.B.: input = ["1", "2", "*", "3", "4", "*", "+"]). Strings können Sie mit Hilfe von "int(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>input = ["2", "1", "-"] s = CustomStack() result = Calculator(input, s); print(int(result));</pre>	1
<pre>input = ["1", "2", "*", "3", "4", "*", "+"] s = CustomStack() result = Calculator(input, s) print(int(result))</pre>	14
<pre>input = ["4", "2", "/", "4", "4", "4", "/", "2", "*", "/", "-"] s = CustomStack() result = Calculator(input, s) print(int(result))</pre>	0

<pre>input = ["278241", "489", "/", "6456", "*", "46546", "+"] s = CustomStack() result = Calculator(input, s) print(int(result))</pre>	3720010
---	---------

Vorbelegung

<pre>def Calculator(inputarray, stack): return 0</pre>
--

Testfälle

<pre>input = ["2", "1", "-"] s = CustomStack() result = Calculator(input, s); print(int(result));</pre>	1
<pre>input = ["1", "2", "*", "3", "4", "*", "+"] s = CustomStack() result = Calculator(input, s) print(int(result))</pre>	14
<pre>input = ["4", "2", "/", "4", "4", "4", "/", "2", "*, "/", "-"] s = CustomStack() result = Calculator(input, s) print(int(result))</pre>	0
<pre>input = ["278241", "489", "/", "6456", "*", "46546", "+"] s = CustomStack() result = Calculator(input, s) print(int(result))</pre>	3720010
<pre>input = ["1", "2", "*", "3", "4", "*", "+"] s = CustomStack() result = Calculator(input, s) print(s._compareHistory(["[1]", "[1, 2]", "[1]", [], "[2]", "[2, 3]", "[2, 3, 4]", "[2, 3]", "[2]", "[2, 12]", "[2]", [], "[14]", []]))</pre>	True
<pre>input = ["4", "2", "/", "4", "4", "4", "/", "2", "*, "/", "-"] s = CustomStack() result = Calculator(input, s) print(s._compareHistory(["[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>	True

<pre> input = ["278241", "489", "/", "6456", "*", "46546", "+"] s = CustomStack() result = Calculator(input, s) print(s._compareHistory(["[278241]", "[278241, 489]", "[278241]", "[]", "[569]", "[569, 6456]", "[569]", "[]", "[3673464]", "[3673464, 46546]", "[3673464]", "[]", "[3720010]", "[]"]))) </pre>	True
<pre> input = ["1231", "123", "3210", "35", "+", "+", "+"] s = CustomStack() result = Calculator(input, s) print(int(result)) print(s._compareHistory(["[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> input = ["2874", "1757", "+", "8989", "4768", "+", "+"] s = CustomStack() result = Calculator(input, s) print(int(result)) print(s._compareHistory(["[2874]", "[2874, 1757]", "[2874]", "[]", "[4631]", "[4631, 8989]", "[4631, 8989, 4768]", "[4631, 8989]", "[4631]", "[4631, 13757]", "[4631]", "[]", "[18388]", "[]"]))) </pre>	18388 True
<pre> input = ["561", "232", "+", "11158", "+"] s = CustomStack() result = Calculator(input, s) print(int(result)) print(s._compareHistory(["[561]", "[561, 232]", "[561]", "[]", "[793]", "[793, 11158]", "[793]", "[]", "[11951]", "[]"]))) </pre>	11951 True
<pre> input = ["8541", "5965", "*", "646", "+", "6", "+", "84585", "-"] s = CustomStack() result = Calculator(input, s) print(int(result)) print(s._compareHistory(["[8541]", "[8541, 5965]", "[8541]", "[]", "[50947065]", "[50947065, 646]", "[50947065]", "[]", "[50947711]", "[50947711, 6]", "[50947711]", "[]", "[50947717]", "[50947717, 84585]", "[50947717]", "[]", "[50863132]", "[]"]))) </pre>	50863132 True
<pre> n1 = getRandom() n2 = getRandom() n3 = getRandom() s = CustomStack() </pre>	3

<pre>input = [str(n1), str(n1), "/", str(n2), str(n2), "/", str(n3), str(n3), "/", "+", "+"] print(int(Calculator(input, s)))</pre>	
<pre>n1 = getRandom() n2 = getRandom() n3 = getRandom() s = CustomStack() input = [str(n1), str(n2), str(n3), "-", "-"] print(int(Calculator(input, s)) == int(n1 - (n2 - n3)))</pre>	True

Musterantwort

```
# This implementation assumes input is a correct formula
def Calculator(inputarray, stack):
    for i in inputarray:
        try:
            stack.push({
                "+": lambda x, y: y+x,
                "-": lambda x, y: y-x,
                "*": lambda x, y: y*x,
                "/": lambda x, y: y/x,}[i](stack.pop(),
stack.pop()))
        except KeyError:
            stack.push(int(i))
    return stack.pop()
```

Aufgabe X(-X)

Aufgabe

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

Die Liste hat folgende Operationen, welche als Eingabe Indizes 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>lts = ListToSort([2, 3, 1, 5, 99, 33]) lts = listSorter(lts) print(lts.isSorted()) print(lts.isSame([1, 2, 3, 5, 33, 99]))</pre>	<pre>True True</pre>
<pre>lts = ListToSort([randrange(0, 9999999, 1)]) lts = listSorter(lts) print(lts.isSorted())</pre>	<pre>True</pre>
<pre>lts = ListToSort([7, 7, 7, 7, 7, 7, 7]) lts = listSorter(lts) print(lts.isSorted())</pre>	<pre>True</pre>

Vorbelegung

```
def listSorter(listToSort: ListToSort) -> ListToSort:
    pass
```

Testfälle

<pre>lts = ListToSort([2, 3, 1, 5, 99, 33]) lts = listSorter(lts) print(lts.isSorted()) print(lts.isSame([1, 2, 3, 5, 33, 99]))</pre>	True True
<pre>lts = ListToSort([]) lts = listSorter(lts) print(lts.isSorted())</pre>	True
<pre>lst = [] for i in range(256): lst.append(randrange(0, 2**31, 1)) lts = ListToSort(lst) lts = listSorter(lts) print(lts.isSorted())</pre>	True
<pre>lst = [] for i in range(256): lst.append(randrange(-2**31, 2**31, 1)) lst.append(-1) lts = ListToSort(lst) lts = listSorter(lts) print(lts.isSorted())</pre>	True
<pre>lst = [] for i in range(128): lst.append(i) lts = ListToSort(lst) lts = listSorter(lts) print(lts.isSorted())</pre>	True
<pre>lst = [] for i in range(128, 0): lst.append(i) lts = ListToSort(lst) lts = listSorter(lts) print(lts.isSorted())</pre>	True
<pre>lst = [] for i in range(128): lst.append(1) lts = ListToSort(lst) lts = listSorter(lts) print(lts.isSorted())</pre>	True
<pre>lst = [] inval = randrange(-128, -1, 1) for i in range(128): lst.append(inval) lts = ListToSort(lst) lts = listSorter(lts) print(lts.isSorted())</pre>	True
<pre>lts = ListToSort([42]) lts = listSorter(lts)</pre>	True

<code>print(lts.isSorted())</code>	
<code>lts = ListToSort([randrange(0, 9999999, 1)])</code> <code>lts = listSorter(lts)</code> <code>print(lts.isSorted())</code>	True
<code>lts = ListToSort([7, 7, 7, 7, 7, 7, 7])</code> <code>lts = listSorter(lts)</code> <code>print(lts.isSorted())</code>	True

Musterantwort

```
def listSorter(listToSort: ListToSort) -> ListToSort:
    """
    This is an implimentation of bubble sort
    """
    changed = True
    while changed:
        changed = False
        for i in range(1, listToSort.Length()):
            if listToSort.Larger(i-1, i):
                listToSort.Swap(i-1, i)
                changed = True
    return listToSort
```

Aufgabe X(-X)

Aufgabe

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

```
class Node:
```

```
    * key
    * left
    * right
    * 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 root = Node() root.key = 2 # build target target = Node() target.key = 1 # Call insert root = insert(root, target) # Compare NewRoot = Node() NewRoot.key = 2 NewRoot.left = Node() NewRoot.left.key = 1 NewRoot.left.parent = NewRoot print(compare_trees(root, NewRoot))</pre>	True
<pre># build tree root = Node() root.key = 2</pre>	True

<pre># build target target = Node() target.key = 3 # Call insert root = insert(root, target) # Compare NewRoot = Node() NewRoot.key = 2 NewRoot.right = Node() NewRoot.right.key = 3 NewRoot.right.parent = NewRoot print(compare_trees(root, NewRoot))</pre>	
--	--

Vorbelegung

<pre># Node in parameter represents root of tree def insert(root, insert): return None</pre>
--

Testfälle

<pre># build tree root = Node() root.key = 2 # build target target = Node() target.key = 1 # Call insert root = insert(root, target) # Compare NewRoot = Node() NewRoot.key = 2 NewRoot.left = Node() NewRoot.left.key = 1 NewRoot.left.parent = NewRoot print(compare_trees(root, NewRoot))</pre>	True
<pre># build tree root = Node() root.key = 2 # build target</pre>	True

<pre> target = Node() target.key = 3 # Call insert root = insert(root, target) # Compare NewRoot = Node() NewRoot.key = 2 NewRoot.right = Node() NewRoot.right.key = 3 NewRoot.right.parent = NewRoot print(compare_trees(root, NewRoot)) </pre>	
<pre> # build target target = Node() target.key = 15418 # Call insert root = insert(None, target) # Compare NewRoot = Node() NewRoot.key = 15418 print(compare_trees(root, NewRoot)) </pre>	True
<pre> # build tree n1 = Node() n1.key = 72 n2 = Node() n2.key = 38 n1.parent = n2 n2.right = n1 n1 = Node() n1.key = 42 n2.parent = n1 n1.left = n2 n2 = Node() n2.key = 97 n2.parent = n1 n1.right = n2 root = Node() root.key = 17 root.right = n1 n1.parent = root n1 = Node() n1.key = 1 n1.parent = root </pre>	True

<pre> root.left = n1 # build target target = Node() target.key = 27 # Call insert root = insert(root, target) # Compare n1 = Node() n1.key = 72 n2 = Node() n2.key = 38 n1.parent = n2 n2.right = n1 n1 = Node() # target! n1.key = 27 n1.parent = n2 n2.left = n1 n1 = Node() n1.key = 42 n2.parent = n1 n1.left = n2 n2 = Node() n2.key = 97 n2.parent = n1 n1.right = n2 NewRoot = Node() NewRoot.key = 17 NewRoot.right = n1 n1.parent = NewRoot n1 = Node() n1.key = 1 n1.parent = NewRoot NewRoot.left = n1 print(compare_trees(root, NewRoot)) </pre>	
<pre> test = get_random() root = insert(test[0], test[1]) print(compare_trees(root, test[2])) </pre>	True
<pre> test = get_random() root = insert(test[0], test[1]) print(compare_trees(root, test[2])) </pre>	True
<pre> test = get_random() </pre>	True

<pre> root = insert(test[0], test[1]) print(compare_trees(root, test[2])) </pre>	
<pre> test = get_random() root = insert(test[0], test[1]) print(compare_trees(root, test[2])) </pre>	True

Musterantwort

```

# Node in parameter represents root of tree
def insert(root: Node, insert: Node):
    x = root
    y = None # saves Node.parent
    while x is not None:
        y = x
        if insert.key < x.key:
            x = x.left
        else:
            x = x.right;

    insert.parent = y
    if y is None:
        return insert
    elif insert.key < y.key:
        y.left = insert
    else:
        y.right = insert;

    return root

```

Aufgabe 4-2

Aufgabe

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

class Node:

- * key
- * left
- * right
- * Parent

Implementieren Sie für diese Aufgabe eine Methode "delete", die eine "Node" im Baum löscht und den neuen Baum 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 Baums übergeben sowie den zu löschenden Knoten.

Beispiele

<pre># build tree root = Node() root.key = 2 root.left = Node() root.left.key = 1 root.left.parent = root root.right = Node() root.right.key = 3 root.right.parent = root # Call delete root = delete(root, root.right) # Compare NewRoot = Node() NewRoot.key = 2 NewRoot.left = Node() NewRoot.left.key = 1 NewRoot.left.parent = NewRoot print(compare_trees(root, NewRoot))</pre>	True
<pre># build tree root = Node() root.key = 2 root.left = Node() root.left.key = 1 root.left.parent = root root.right = Node() root.right.key = 3</pre>	True

<pre> root.right.parent = root root.right.right = Node() root.right.right.key = 99 root.right.right.parent = root.right # Call delete root = delete(root, root.right) # Compare NewRoot = Node() NewRoot.key = 2 NewRoot.left = Node() NewRoot.left.key = 1 NewRoot.left.parent = NewRoot NewRoot.right = Node() NewRoot.right.key = 99 NewRoot.right.parent = NewRoot print(compare_trees(root, NewRoot)) </pre>	
<pre> # build tree root = Node() root.key = 2 root.left = Node() root.left.key = 1 root.left.parent = root root.right = Node() root.right.key = 50 root.right.parent = root root.right.right = Node() root.right.right.key = 99 root.right.right.parent = root.right root.right.left = Node() root.right.left.key = 25 root.right.left.parent = root.right # Call delete root = delete(root, root.right) # Compare newRoot = Node() newRoot.key = 2 newRoot.left = Node() newRoot.left.key = 1 newRoot.left.parent = newRoot newRoot.right = Node() newRoot.right.key = 99 newRoot.right.parent = newRoot newRoot.right.left = Node() newRoot.right.left.key = 25 newRoot.right.left.parent = newRoot.right print(compare_trees(root, newRoot))# build tree </pre>	True

```

root = Node()
root.key = 2
root.left = Node()
root.left.key = 1
root.left.parent = root
root.right = Node()
root.right.key = 4
root.right.parent = root
root.right.left = Node()
root.right.left.key = 3
root.right.left.parent = root.right

# Call delete
root = delete(root, root.right);

# Compare
NewRoot = Node()
NewRoot.key = 2
NewRoot.left = Node()
NewRoot.left.key = 1
NewRoot.left.parent = NewRoot
NewRoot.right = Node()
NewRoot.right.key = 3
NewRoot.right.parent = NewRoot

print(compare_trees(root, NewRoot))

```

Vorbelegung

```

# Node in parameter represents root of tree
def delete(root, delete):
    return None

```

Testfälle

```

# build tree
root = Node()
root# build tree
root = Node()
root.key = 2
root.left = Node()
root.left.key = 1
root.left.parent = root
root.right = Node()
root.right.key = 3
root.right.parent = root
root.right.right = Node()
root.right.right.key = 99
root.right.right.parent = root.right

```

True

```
# Call delete
root = delete(root, root.right)
```

```
# Compare
NewRoot = Node()
NewRoot.key = 2
NewRoot.left = Node()
NewRoot.left.key = 1
NewRoot.left.parent = NewRoot
NewRoot.right = Node()
NewRoot.right.key = 99
NewRoot.right.parent = NewRoot
```

```
print(compare_trees(root, NewRoot))t.key = 2
root.left = Node()
root.left.key = 1
root.left.parent = root
root.right = Node()
root.right.key = 3
root.right.parent = root
# Call delete
root = delete(root, root.right)
```

```
# Compare
NewRoot = Node()
NewRoot.key = 2
NewRoot.left = Node()
NewRoot.left.key = 1
NewRoot.left.parent = NewRoot

print(compare_trees(root, NewRoot))
```

```
# build tree
root = Node()
root.key = 2
root.left = Node()
root.left.key = 1
root.left.parent = root
root.right = Node()
root.right.key = 3
root.right.parent = root
root.right.right = Node()
root.right.right.key = 99
root.right.right.parent = root.right
```

```
# Call delete
root = delete(root, root.right)
```

```
# Compare
NewRoot = Node()
NewRoot.key = 2
NewRoot.left = Node()
```

True

<pre> NewRoot.left.key = 1 NewRoot.left.parent = NewRoot NewRoot.right = Node() NewRoot.right.key = 99 NewRoot.right.parent = NewRoot print(compare_trees(root, NewRoot)) </pre>	
<pre> # build tree root = Node() root.key = 2 root.left = Node() root.left.key = 1 root.left.parent = root root.right = Node() root.right.key = 50 root.right.parent = root root.right.right = Node() root.right.right.key = 99 root.right.right.parent = root.right root.right.left = Node() root.right.left.key = 25 root.right.left.parent = root.right # Call delete root = delete(root, root.right) # Compare newRoot = Node() newRoot.key = 2 newRoot.left = Node() newRoot.left.key = 1 newRoot.left.parent = newRoot newRoot.right = Node() newRoot.right.key = 99 newRoot.right.parent = newRoot newRoot.right.left = Node() newRoot.right.left.key = 25 newRoot.right.left.parent = newRoot.right print(compare_trees(root, newRoot)) </pre>	True
<pre> # build tree n1 = Node() n1.key = 37 n1.left = Node() n1.left.key = 32 n1.left.parent = n1 n1.right = Node() n1.right.key = 78 n1.right.parent = n1 target = Node() target.key = 21 </pre>	True

<pre> target.right = n1 n1.parent = target target.left = Node() target.left.key = 13 target.left.parent = target root = Node() root.key = 85 root.left = target target.parent = root root.right = Node() root.right.key = 942 root.right.parent = root root.right.left = Node() root.right.left.key = 136 root.right.left.parent = root.right # Call delete root = delete(root, target) # Compare n2 = Node() n2.key = 37 n2.right = Node() n2.right.key = 78 n2.right.parent = n2 replaced = Node() replaced.key = 32 replaced.right = n2 n2.parent = replaced replaced.left = Node() replaced.left.key = 13 replaced.left.parent = replaced newRoot = Node() newRoot.key = 85 newRoot.left = replaced replaced.parent = newRoot newRoot.right = Node() newRoot.right.key = 942 newRoot.right.parent = newRoot newRoot.right.left = Node() newRoot.right.left.key = 136 newRoot.right.left.parent = newRoot.right print(compare_trees(root, newRoot)) </pre>	
<pre> root = Node() root.key = 55 root = delete(root, root) </pre>	<p>True</p>

<code>print(compare_trees(root, None))</code>	
<code>test = get_random() root = delete(test[0], test[1]) print(compare_trees(root, test[2]))</code>	True
<code>test = get_random() root = delete(test[0], test[1]) print(compare_trees(root, test[2]))</code>	True
<code>test = get_random() root = delete(test[0], test[1]) print(compare_trees(root, test[2]))</code>	True
<code>test = get_random() root = delete(test[0], test[1]) print(compare_trees(root, test[2]))</code>	True
<code># build tree root = Node() root.key = 2 root.left = Node() root.left.key = 1 root.left.parent = root root.right = Node() root.right.key = 4 root.right.parent = root root.right.left = Node() root.right.left.key = 3 root.right.left.parent = root.right # Call delete root = delete(root, root.right); # Compare NewRoot = Node() NewRoot.key = 2 NewRoot.left = Node() NewRoot.left.key = 1 NewRoot.left.parent = NewRoot NewRoot.right = Node() NewRoot.right.key = 3 NewRoot.right.parent = NewRoot print(compare_trees(root, NewRoot))</code>	True

Musterantwort

```
# Helper function
def tree_minimum(n):
    while n.left is not None:
        n = n.left
    return n
# Node in parameter represents root of tree
def delete(root: Node, delete: Node):
```

```

y = None # y: node to delete (after swap)
if delete.left is None or delete.right is None:
    y = delete
else:
    y = tree_minimum(delete.right)
x = None # non-null child of y
if y.left is not None:
    x = y.left
else:
    x = y.right
# now delete y
if x is not None:
    x.parent = y.parent
if y.parent is None:
    root = x
else:
    if y is y.parent.left:
        y.parent.left = x
    else:
        # build tree
root = Node()
root.key = 2
root.left = Node()
root.left.key = 1
root.left.parent = root
root.right = Node()
root.right.key = 4
root.right.parent = root
root.right.left = Node()
root.right.left.key = 3
root.right.left.parent = root.right

# Call delete
root = delete(root, root.right);

# Compare
NewRoot = Node()
NewRoot.key = 2
NewRoot.left = Node()
NewRoot.left.key = 1
NewRoot.left.parent = NewRoot
NewRoot.right = Node()
NewRoot.right.key = 3
NewRoot.right.parent = NewRoot

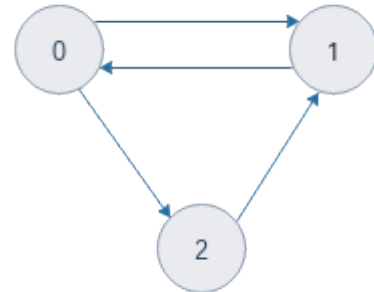
print(compare_trees(root, NewRoot))    y.parent.right = x
# Copy data
if y is not delete:
    delete.key = y.key
# y should have no reference and should be deleted by the
python gc
return root

```

Aufgabe 5-1

Aufgabe

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



```
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.

Beispiele

<pre>graph = [[1, 2], [0, 3, 4], [0, 5], [1], [1, 5], [2, 4],] result = bfs(graph, 0) print(result)</pre>	<pre>[0, 1, 2, 3, 4, 5]</pre>
---	-----------------------------------

<pre>graph = [[1,2], [], []] result = bfs(graph, 0) print(result)</pre>	[0, 1, 2]
<pre>graph = [[2,1], [], []] result = bfs(graph, 0) print(result)</pre>	[0, 2, 1]

Vorbelegung

```
def bfs(graph, start):
    return []
```

Testfälle

<pre>graph = [[1,2], [0,3,4], [0,5], [1], [1,5], [2,4],] result = bfs(graph, 0) print(result)</pre>	[0, 1, 2, 3, 4, 5]
<pre>graph = [[1,2], [], []] result = bfs(graph, 0) print(result)</pre>	[0, 1, 2]
<pre>graph = [[2,1], [], []] result = bfs(graph, 0) print(result)</pre>	[0, 2, 1]
<pre>graph = [[1, 2],</pre>	[3, 1, 0, 4, 2,

<pre> [0, 3, 4], [0, 5], [1], [1, 5], [2, 4],] result = bfs(graph, 3) print(result) </pre>	5]
<pre> graph = [[1], [0, 3, 4], [0, 5], [1], [1, 5], [2, 4],] result = bfs(graph, 0) print(result) </pre>	[0, 1, 3, 4, 5, 2]
<pre> graph = [[0], [2], [1], [4], [], []]] print(bfs(graph, 0)) print(bfs(graph, 1)) print(bfs(graph, 3)) print(bfs(graph, 4)) </pre>	[0] [1, 2] [3, 4] [4]
<pre> graph = [[1, 2], [0, 3, 4], [0, 5], [1], [1, 5], [2, 4]]] print(bfs(graph, 5)) </pre>	[5, 2, 4, 0, 1, 3]
<pre> graph = [] for i in range(50): nodeList = [] for i in range(random.randrange(0, 20, 1)): nodeList.append(random.randrange(0, 50, 1)) graph.append(nodeList) start = random.randrange(0, 50, 1) print(_isCorrect(graph, start, bfs(copy.deepcopy(graph), start))) </pre>	True
<pre> graph = [] for i in range(100): nodeList = [] for i in range(random.randrange(0, 40, 1)): nodeList.append(random.randrange(0, 100, </pre>	True

1)) graph.append(nodeList) start = random.randrange(0, 100, 1) print(_isCorrect(graph, start, bfs(copy.deepcopy(graph), start)))	
graph = [] for i in range(200): nodeList = [] for i in range(random.randrange(0, 60, 1)): nodeList.append(random.randrange(0, 200, 1)) graph.append(nodeList) start = random.randrange(0, 200, 1) print(_isCorrect(graph, start, bfs(copy.deepcopy(graph), start)))	True
print(bfs([[]], 0))	[0]

Musterantwort

```
def bfs(graph, start):
    visited, queue = list(), [start]
    while queue:
        next = queue.pop(0)
        if next not in visited:
            visited.append(next)
            for vertex in graph[next]:
                if vertex not in visited:
                    queue.append(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.

Beispiele

<pre>graph = [[1,2], [0,3,4], [0,5], [1], [1,5], [2,4],] result = dfs(graph, 0) print(result)</pre>	<pre>[0, 1, 3, 4, 5, 2]</pre>
<pre>graph = [[1,2], [], [],] result = dfs(graph, 0) print(result)</pre>	<pre>[0, 1, 2]</pre>
<pre>graph = [[2,1], [], [],] result = dfs(graph, 0) print(result)</pre>	<pre>[0, 2, 1]</pre>

Vorbelegung

```
def dfs(graph, start):  
    return []
```

Testfälle

<pre>graph = [[1,2], [0,3,4], [0,5], [1], [1,5], [2,4],] result = dfs(graph, 0) print(result)</pre>	<pre>[0, 1, 3, 4, 5, 2]</pre>
---	-----------------------------------

<pre> [1,2], [0,3,4], [0,5], [1], [1,5], [2,4],] result = dfs(graph, 0) print(result) </pre>	<pre> 2] </pre>
<pre> graph = [[1,2], [], [],] result = dfs(graph, 0) print(result) </pre>	<pre> [0, 1, 2] </pre>
<pre> graph = [[2,1], [], [],] result = dfs(graph, 0) print(result) </pre>	<pre> [0, 2, 1] </pre>
<pre> graph = [[1, 2], [0, 3, 4], [0, 5], [1], [1, 5], [2, 4],] result = dfs(graph, 3) print(result) </pre>	<pre> [3, 1, 0, 2, 5, 4] </pre>
<pre> graph = [[1], [0, 3, 4], [0, 5], [1], [1, 5], [2, 4],] result = dfs(graph, 0) print(result) </pre>	<pre> [0, 1, 3, 4, 5, 2] </pre>
<pre> graph = [[0], [2], [1], [4], [], [],] print(dfs(graph, 0)) print(dfs(graph, 1)) print(dfs(graph, 3)) </pre>	<pre> [0] [1, 2] [3, 4] [4] </pre>

<code>print(dfs(graph, 4))</code>	
<pre>graph = [[1, 2], [0, 3, 4], [0, 5], [1], [1, 5], [2, 4]] print(dfs(graph, 5))</pre>	<code>[5, 2, 0, 1, 3, 4]</code>
<pre>graph = [] for i in range(50): nodeList = [] for i in range(random.randrange(0, 20, 1)): nodeList.append(random.randrange(0, 50, 1)) graph.append(nodeList) start = random.randrange(0, 50, 1) print(_isCorrect(graph, start, dfs(copy.deepcopy(graph), start)))</pre>	True
<pre>Graph = [] for i in range(100): nodeList = [] for i in range(random.randrange(0, 40, 1)): nodeList.append(random.randrange(0, 100, 1)) graph.append(nodeList) start = random.randrange(0, 100, 1) print(_isCorrect(graph, start, dfs(copy.deepcopy(graph), start)))</pre>	True
<pre>graph = [] for i in range(200): nodeList = [] for i in range(random.randrange(0, 60, 1)): nodeList.append(random.randrange(0, 200, 1)) graph.append(nodeList) start = random.randrange(0, 200, 1) print(_isCorrect(graph, start, dfs(copy.deepcopy(graph), start)))</pre>	True
<code>print(dfs([[]], 0))</code>	<code>[0]</code>

Musterantwort

```
def dfs(graph, start):
    visited, queue = list(), [start]
    while queue:
        next = queue.pop()
        if next not in visited:
            visited.append(next)
```

```
        for vertex in reversed(graph[next]): # Need to reverse
list in order to visit vertex in correct order
            if vertex not in visited:
                queue.append(vertex)
    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>print(editDistance("", ""))</code>	0
<code>print(editDistance("a", "a"))</code>	0
<code>print(editDistance("a", ""))</code>	1
<code>print(editDistance("", "a"))</code>	1
<code>print(editDistance("ac", "abc"))</code>	1
<code>print(editDistance("abc", "ac"))</code>	1
<code>print(editDistance("a", "b"))</code>	1

Vorbelegung

```
def editDistance(a, b):  
    return 0
```

Testfälle

<code>print(__student_answer__.count("editDistance(", 0, len(__student_answer__)))</code>	1
<code>print(editDistance("", ""))</code>	0
<code>print(editDistance("a", "a"))</code>	0
<code>print(editDistance("a", ""))</code>	1
<code>print(editDistance("", "a"))</code>	1
<code>print(editDistance("ac", "abc"))</code>	1
<code>print(editDistance("abc", "ac"))</code>	1
<code>print(editDistance("a", "b"))</code>	1
<code>print(editDistance("abc", ""))</code>	3

<code>print(editDistance("", "abc"))</code>	3
<code>print(editDistance("", ""))</code>	0
<code>print(editDistance("cut", "cat"))</code>	1
<code>print(editDistance("abc", "abc"))</code>	0
<code>print(editDistance("geek", "gesek"))</code>	1
<code>print(editDistance("blablablabla", "abs"))</code>	10
<code>print(editDistance("abc", "abs"))</code>	1
<code>s2 = """weqweqweqwesqwesqweweqweqweqweqseqwesqsssweqws e""" t2 = """qswe""" print(editDistance(s2, t2))</code>	42
<code>s = getRandom(5) t = getRandom(5) print(_isCorrect(s, t, editDistance(s, t)))</code>	True
<code>s = getRandom(5) t = getRandom(10) print(_isCorrect(s, t, editDistance(s, t)))</code>	True
<code>s = getRandom(50) t = getRandom(100) print(_isCorrect(s, t, editDistance(s, t)))</code>	True
<code>s = "91336388630930104036509887910612766733414220177 834951661736334124642044989286936143277410676715 4961" t = "91336388630930136509887910612766733414220177834 9ööö516617363341246420449892869361432774106767öö ö961" print(editDistance(s, t))</code>	9

Musterantwort

```
def editDistance(a, b):
    """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"""

    m = len(a)
    n = len(b)

    edDis = [0] * (m+1)
    for i in range(m+1):
        edDis[i] = [0] * (n+1)
```

```
for i in range(m+1):
    edDis[i][0] = i

for i in range(n+1):
    edDis[0][i] = i

for i in range(1,m+1):
    for j in range(1,n+1):
        if a[i-1]==b[j-1]:
            edDis[i][j] = edDis[i-1][j-1]
        else:
            edDis[i][j] = 1 + min(edDis[i][j-1],
                                   edDis[i-1][j],
                                   edDis[i-1][j-1])

return edDis[len(a)][len(b)]
```

Aufgabe X(-X)

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

<code>print(knapSack([0.5, 1.3], [2, 5], 3))</code>	0.5
<code>print(knapSack([6, 8], [6, 8], 0))</code>	0
<code>print(knapSack([6, 8], [6, 8], 10))</code>	8
<code>print(knapSack([1, 1, 99], [6, 4, 11], 10))</code>	2
<code>print(knapSack([6, 8], [6, 8], 17))</code>	14

Vorbelegung

```
def knapSack(value, volume, capacity):
    assert len(value) == len(volume)

    for x in volume:
        assert x == round(x)

    # TODO

    return 0
```

Testfälle

<code>print(__student_answer__.count("knapSack(", 0, len(__student_answer__)))</code>	1
<code>print(knapSack([0.5, 1.3],[2, 5],3))</code>	0.5
<code>print(knapSack([],[],10))</code>	0
<code>print(knapSack([6, 8],[6, 8],0))</code>	0
<code>print(knapSack([6, 8],[6, 8],10))</code>	8
<code>print(knapSack([1, 1, 99],[6, 4, 11],10))</code>	2
<code>print(knapSack([6, 8],[6, 8],17))</code>	14
<code>print(knapSack([2, 6, 11],[2, 6, 11],18))</code>	17
<code>print(knapSack([2, 6, 100],[2, 6, 17],18))</code>	100
<code>print(knapSack([2, 2, 2, 2, 2, 2],[1, 2, 3, 4, 5, 6],9))</code>	6
<code>print(knapSack([10, 2, 10, 2, 2, 2],[1, 2, 3, 4, 5, 6],9))</code>	22
<code>print(knapSack([10, 2, 10, 2, 2, 2],[1, 2, 3, 4, 5, 6],100))</code>	28
<code>print(knapSack(list(range(100)),list(range(100)),100))</code>	100
<code>print(knapSack(list(reversed(range(100))),list(reversed(range(100))),100))</code>	100
<code>print(knapSack(list(range(0,200,2)),list(range(100)),100))</code>	200
<code>ks = sum(list(range(100)))+1 c = list(range(100)) for x in c: x = random.uniform(0, 100) v = list(range(100)) ev = sum(c) print(knapSack(v,c,ks) == ev)</code>	True
<code>ks = sum(list(range(100)))+1 c = list(range(100)) for x in c: x = random.uniform(0, 100) v = list(range(100)) ev = sum(c) print(knapSack(v,c,ks) == ev)</code>	True
<code>ks = sum(list(range(100)))+1 ks = ks//4 c = list(range(100)) for x in c: x = random.uniform(0, 100) v = list(range(100)) print(_isCorrect(v,c,ks,knapSack(copy.deepcopy(v),copy.deepcopy(c),copy.deepcopy(ks))))</code>	True

<pre> ks = sum(list(range(100)))+1 ks = ks//4 c = list(range(100)) for x in c: x = random.uniform(0, 100) v = list(range(100)) print(_isCorrect(v,c,ks,knapSack(copy.deepcopy(v),copy.deepcopy(c),copy.deepcopy(ks)))) </pre>	True
<pre> ks = sum(list(range(100)))+1 ks = ks//4 c = list(range(100)) for x in c: x = random.uniform(0, 100) v = list(range(100)) print(_isCorrect(v,c,ks,knapSack(copy.deepcopy(v),copy.deepcopy(c),copy.deepcopy(ks)))) </pre>	True

Musterantwort

```

def knapSack(value, volume, capacity):
    """This function calculates and returns the max
    value in knapsack with the capacity "capacity" wich is filled
    with stuff of volume "volume" and the value "value"."""

    assert len(value) == len(volume)

    for x in volume:
        assert x == round(x)

    n = len(value)

    Result = [0] * (n+1)
    for i in range(n+1):
        Result[i] = [0] * (capacity + 1)

    for i in range(n+1):
        for j in range(capacity + 1):
            Result[i][j] = 0

    for i in range(n-1, -1, -1):
        for j in range(1, capacity + 1):
            if volume[i] <= j:
                Result[i][j] = max(value[i] + Result[i + 1][j -
volume[i]],
                                Result[i+1][j])
            else:
                Result[i][j] = Result[i+1][j]

    return Result[0][capacity]

```