Klausur AuD April 2022 / Teil 1 / moodle-Aufgabe 2:

In der Vorlesung haben Sie das Verfahren Quadratic Probing Hashing als typisches Design für Hashfunktionen kennengelernt:

F (i, 13, k) := ((K mod 13) + (i-1)²) mod 13

Fügen Sie die folgenden Werte in dieser Reihenfolge ein:

9, 21, 27, 20, 1, 10, 0, 2, 7

Für nicht belegte Felder ist ein '-' einzutragen. Unten finden Sie unbewertete Felder für Notizen.

0	1	2	3	4	5	6

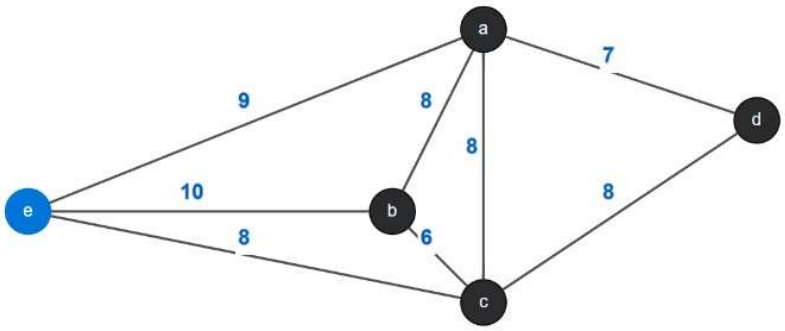
7	8	9	10	11	12

Unbewertete Felder für Notizen:
(Diese Felder können Sie für Zwischenergebnisse oder Ähnliches nutzen)

<die weiteren Felder hier ausgelassen>

Klausur AuD April 2022 / Teil 1 / moodle-Aufgabe 3:

Aus der Vorlesung kennen Sie den Algorithmus von Prim.



Geben Sie ausgehend vom Startknoten *e* die Warteschlange *Q* sowie die Kantenmenge *E* zum Zeitpunkt *i = 2* an.

Hinweis: Sie können die Warteschlange *Q* und die Kantenmenge *E* zum Zeitpunkt *i = 1* eintragen, damit Sie selbst den Überblick behalten. Diese Iteration wird jedoch NICHT bewertet.

Lösung:

Iteration i = 1:

Q = (, , , ,)

E = { }

Iteration i = 2:

Q = (, , , ,)

E = { }

Klausur AuD April 2022 / Teil 1 / moodle-Aufgabe 4:

Aus der Vorlesung kennen Sie den Algorithmus *String Matching basierend auf einem endlichen Automaten* über dem Alphabet = *(e, u, i, t, s, g, l, w, z, a)*. Gesucht sind alle Vorkommen von *T* = *uzae* in *S* = *uzauzuzaeu*, also Automat:

State	e	u	i	t	s	g	l	w	z	a
0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	2	0
2	0	1	0	0	0	0	0	0	0	3
3	4	1	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0

Bekanntlich ist *q* die Länge des zur Zeit längsten gültigen Kandidaten*, und R* enthält die Startindizes aller vollständigen Matches (Indizes beginnen mit 1) einfach nur als Zahlen. "Induktionsschritt *i*" heißt: De ersten *i* Zeichen in *S* sind behandelt.

ACHTUNG: Für ein leeres Feld geben Sie unbedingt "-" ein.

Induktionsschritt	q	R
0	1	
1	2	
2	3	
3	1	
4	2	
5	1	
6		
7		
8		
9		

Klausur AuD April 2022 / Teil 1 / moodle-Aufgabe 5:

Aus Vorlesung und Übungen kennen Sie folgende Klasse:

```
public class ListItem <T> {
  public T key;
  public ListItem<T> next;
}
```

Schreiben Sie alles, was in folgender Methodendefinition ausgelassen und durch ... angedeutet ist, also die Anweisungen in:

```
public <T> void mergeLists
            ( ListItem<ListItem<T>> lst, Comparator<T> cmp,
              ListItem<T> tail ) { ... }
```

Methode **mergeLists** darf ohne Nachprüfung davon ausgehen, dass **lst** auf den Kopf einer korrekt gebildeten Liste von Listen verweist (*Terminologie*: die Elemente der *Hauptliste* sind die *Einzellisten*), die auch leer sein kann. Ebenso darf sie davon ausgehen, dass jede Einzelliste von **lst** aufsteigend sortiert bzgl. **cmp** ist. Zudem darf sie davon ausgehen, dass **tail** ungleich **null** ist.

Methode **mergeLists** soll an **tail** eine korrekt gebildete Liste anhängen, die auf jedes **T**-Objekt, auf das in **lst** verwiesen wird, genau so viele Verweise enthält wie in allen Einzellisten von **lst** zusammen (einfacher gesagt: die Einzellisten sollen zu einer Liste "gemerged" werden). Darüber hinaus soll die Rückgabe keine Elemente enthalten. Die Rückgabe soll aufsteigend sortiert gemäß **cmp** sein.

***Verbindliche Anforderung*:** Die Liste von Listen, auf die **lst** verweist, wird durch **mergeLists** nicht verändert.

***Hinweis*:** Für den gleichzeitigen Durchgang durch die Einzellisten richten Sie eine Liste **current** von geeignetem Elementtyp ein, so dass im Moment der Suche nach dem jeweils nächsten an **tail** anzuhängenden Element gilt: Seien $i_1,\ldots,i_k$

diejenigen Positionen von **lst**, bei denen mindestens ein Element der Einzelliste noch nicht in die Liste an **tail** kopiert worden ist. Für $j \in \{1,\ldots,k\}$ verweist das j-te Element von **current** auf das erste Element in der Einzelliste an Position i_j von **lst**, das noch nicht in die Liste an **tail** kopiert worden ist.

Klausur AuD April 2022 / Teil 1 / moodle-Aufgabe 6:

Folgende Klasse sei gegeben:

```
public class BinaryTreeNode <T> {
  public T key;
  public BinaryTreeNode<T> left;
  public BinaryTreeNode<T> right;
}
```

Schreiben Sie alles, was in folgender Methodendefinition ausgelassen und durch ... angedeutet ist, also die Anweisungen in:

```
public void makeLinear ( BinaryTreeNode<T> root ) { ... }
```

***Konkrete Aufgabe*:** Methode **makeLinear** darf ohne Nachprüfung davon ausgehen, dass **root** auf die Wurzel eines korrekt gebildeten binären Baumes verweist (der auch leer sein kann, d.h. **root** == **null** ist möglich). Unter exakter Beibehaltung der Schlüsselwertmenge soll sie diesen Baum so umstrukturieren, dass **node.right** == **null** für jeden Knoten **node** im Baum gilt.

***Verbindliche Anforderungen*:**

1. Methode **makeLinear** ist rein rekursiv, das heißt, Schleifen sind weder in **makeLinear** noch in davon direkt oder indirekt aufgerufenen Methoden erlaubt.
2. Die Reihenfolge der **key**-Werte wird beigehalten: Falls der Baum unmittelbar vor dem Aufruf von **makeLinear** Suchbaumeigenschaft gemäß irgendeiner Vergleichsoperation auf **T** hat, dann auch unmittelbar nach diesem Aufruf.

***Unverbindliche Hinweise*:**

1. Machen Sie sich klar: Falls der linke und der rechte Teilbaum von **root** schon jeweils in sich die Anforderungen erfüllt, dann muss der Baum "nur noch" so umstruktiert werden, dass der Baum eine lineare Liste aus **left**-Verweisen ist, die sich zusammensetzt aus dem rechten Teilbaum von **root** gefolgt von einem Knoten mit **root.key** gefolgt vom linken Teilbaum von **root**.
2. Richten Sie einen neuen Knoten ein, kopieren Sie **root.key** in diesen neuen Knoten, entfernen Sie einen geeignet gewählten anderen Knoten aus dem Baum, aber kopieren Sie vorher dessen **key**-Wert nach **root**.
3. Schreiben und verwenden Sie eine generische (rekursive!) Methode **leftmostNodeInTree**.

Klausur AuD April 2022 / Teil 2 / moodle-Aufgabe 2:

Folgende Klassen seien gegeben:

```
public class Arc <T> {
   public Node<T> head;
}

public class Node <T> {
   public List<Arc<T>> outgoingArcs;
}

public class Triple <T> {

   .....

   public Triple ( T t1, T t2, T t3 ) { ..... }
}
```

Die Elemente von **node.outgoingArcs** sind die aus einem Knoten **node** herauszeigenden Kanten, und **arc.head** ist der Zielknoten einer Kante **arc**.

Schreiben Sie alles, was in folgender Methodendefinition ausgelassen und durch ... angedeutet ist, also die Anweisungen in:

```
public List<Triple<Node<T>>>
void allTriangles ( List<Node<T>> startNodes ) { ... }
```

***Konkrete Aufgabe*:** Die Methode **allTriangles** darf ohne Nachprüfung davon ausgehen, dass alle Knoten in **startNodes** zu einem korrekt gebildeten gerichteten Graphen gehören. Für jeden Knoten v

in **startNodes** sollen alle Dreiecke von v in der Rückgabe enthalten sein. Darüber hinaus soll die Rückgabe nichts enthalten. Dabei ist ein Dreieck von v ein geordnetes Tripel von drei paarweise verschiedenen Knoten (v,w,x), so dass alle drei Kanten (v,w), (w,x) und (x,v) existieren. Der dynamische Typ der Rückgabe soll **LinkedList<Triple<Node<T>>>** sein (gemeint ist **java.util.LinkedList**).

***Hinweis*:** Machen Sie sich klar: Die Aufgabenstellung ist so formuliert, dass Sie für ein Dreieck (v,w,x) nicht prüfen müssen, ob schon vorher andere Dreiecke auf diesen Knoten gefunden wurden.

***Erinnerung*:** Methode **add** von **LinkedList** hat einen Parameter vom formalen Typ **T** und kann als **void**-Methode verwendet werden. Sie können die verkürzte **for**-Schleife, die Sie u.a. von Arrays kennen, auch bei Referenzen von **List** anwenden; alternativ können Sie sich von jeder Liste auch einen **Iterator<T>** mit der parameterlosen Methode **iterator** holen und diese Liste mit den parameterlosen Methoden **hasNext** (boolesch) und **next** (Rückgabetyp **T**) durchlaufen.

Klausur AuD April 2022 / Teil 2 / moodle-Aufgabe 3:

Formulieren Sie das Sortierproblem, das durch Bubblesort gelöst wird, nach dem Schema (i) was ist eine zulässige Eingabe, (ii) was ist eine zulässige Ausgabe. Verwenden und erläutern Sie bei (ii) den Begriff "paarweiser Vergleich".

(iii) Formulieren Sie für Bubblesort die Schleifeninvariante der inneren Schleife.

(iv) Worauf muss bei Bubblesort geachtet werden, damit Stabilität garantiert ist?

Teil 2 / moodle-Aufgabe 4:

Formulieren Sie die Problemstellung *All Pairs Shortest Paths* nach dem Schema (i) was ist eine zulässige Eingabe, (ii) was ist die Ausgabe im Fall, dass es keine negativen Zykel gibt. Gehen Sie bei (ii) auch auf den optionalen, zusätzlichen Output ein, aus dem im Nachgang die eigentlichen Pfade berechnet werden können.

(iii) Formulieren Sie die Schleifeninvariante für den Algorithmus von *Bellman-Ford* (inklusive optionalem, zusätzlichem Output).

(iv) Wie müssen alle Datenstrukturen also initialisiert werden, damit die Invariante vor dem ersten Schleifendurchlauf erfüllt ist?

Klausur AuD April 2022 / Teil 2 / moodle-Aufgabe 5:

Gegeben sei eine Subroutine (in Java: Methode), die bei zwei Listen der Längen $m \geq 0$ und $n \geq m$ prüft, ob alle Elemente der ersten Liste auch in der zweiten Liste vorkommen. Wir unterscheiden im Folgenden:

I: Beide Listen sind nach derselben Sortierlogik sortiert.

II: Die Elemente beider Listen sind in keiner bestimmten Reihenfolge.

***Konkrete Aufgaben*:** Die Subroutine sei möglichst effizient implementiert und benutze nur konstant viel zusätzlichen Speicher.

(i) Wie sieht der Best Case bei I und II aus, (ii) wie sieht der Worst Case bei I und II aus?

(iii) Was ist die asymptotische Komplexität im Best Case bei I und II, (iv) was ist die asymptotische Komplexität im Worst Case bei I und II? Verwenden Sie keine O-Notation, sondern Begriffe wie "konstant", "logarithmisch in m", "linear in n", "m * n"', "quadratisch in m" usw.

Klausur AuD April 2022 / Teil 2 / moodle-Aufgabe 6:

Betrachten Sie einen generischen Algorithmus A (in Java: Methode), der drei Eingaben hat: einen Array a der Länge $n1$ mit generischem Komponententyp $T1$, eine natürliche Zahl $n2 \leq n1/2$ und eine Funktion $F: T1 \rightarrow T2$. Er liefert einen Outputstream s mit generischem Komponententyp $T2$ zurück. Algorithmus A speichert alle Komponenten von a in einen Heap durch $n1$-fache Anwendung der Ihnen aus der Vorlesung bekannten Einfügeoperation "insert" auf Heaps. Dann richtet A den Outputstream s ein, ruft $n2$-mal "extract-min" auf dem Heap auf, wendet F auf jeden der davon zurückgelieferten Werte an und speichert die Ergebnisse nach s. Sei $C1$ die Worst-Case-Komplexität der im Heap verwendeten Vergleichsoperation, $C2$ die Worst-Case-Komplexität der Einrichtung des Outputstreams und $C3$ die Worst-Case-Komplexität einer Anwendung von F.

***Konkrete Aufgabe*:** Gesucht ist die Worst-Case-Komplexität von A als Funktion von $n1$, $n2$, $C1$, $C2$ und $C3$. Formulieren Sie eine Formel für "..." in $\Theta(...)$. (Verwenden Sie auf der Tastatur * für Multiplikation und log(x) für den Logarithmus von x.)

Klausur AuD April 2022 / Teil 2 / moodle-Aufgabe 7:

Begründen Sie mathematisch: Für $a > b > 1$ wächst die Funktion $f1: n \longrightarrow a_n$ asymptotisch echt schneller als die Funktion $f2: n \longrightarrow b_n \cdot n$.

***Unverbindlicher Hinweis*:** Die Anwendung der Regel von L'Hopital, wie Sie es in Altklausuren gesehen haben, ist hier zielführend. Sie können aber auch direkter argumentieren, auf Basis der Eigenschaften der Exponentialfunktionen. Falls Sie L'Hopital verwenden, müssen Sie die Ableitungsfunktionen nicht exakt bestimmen, sondern es reicht, sie stattdessen mit Begriffen wie "konstant", "logarithmisch", "linear" usw. einzuordnen -- aber dann auch explizit hinzuschreiben, was aus dieser Einordnung folgt!