Aufgaben zur Veranstaltung
Algorithmen und Datenstrukturen, SS 2022

H. Pflug, J. Dietel FH Aachen, Campus Jülich; IT Center, RWTH Aachen

Klausurvorbereitung, Blatt 2 27.06.2022

Probeklausur

Aufgabe 1

Notieren Sie die Antworten in ein oder mehreren Stichworten oder einem kurzen Satz:

a) Die Java-Klasse für eine doppelt verkettete Liste heißt:
LinkedList

b) Die Java-Klasse für eine Hashtabelle heißt:
HashMap

c) Insertion-Sort hat im Best-Case die minimale Laufzeitkomplexität:
O(n)

d) Die drei Grundoperationen für reguläre Ausdrücke heißen:
Verkettung, Oder, Hüllenbildung

e) Das Verfahren zum Graphendurchlauf, das dem Level-Order-Durchlauf bei einem Baum entspricht, heißt:
Breitensuche

f) Wie heißen in einem Baum Knoten, die keine Kinder besitzen?
Blätter

g) Welche Laufzeit hat folgender Algorithmus? Geben Sie die Zeit in der O(n)-Notation an.

```
for (int i=0; i<n; i++) {
  for (int j=0; j<n; j++) {
    k++;
  }
}
```

$O(n^2)$

h) Wie nennt man einen Baum, bei dem alle Ebenen komplett besetzt sind?
vollständig

i) Nennen Sie zwei in der Vorlesung behandelte Entwurfsprinzipien.
Möglich: Greedy, Backtracking, Divide and Conquer, Branch and Bound

j) Gegeben sei der reguläre Ausdruck

((A | B) | CB)* (A | C)

Welche der folgenden Buchstabenfolgen erfüllen den regulären Ausdruck?

ABC, ACAC, BCBBBAC, C, ACB
ABC, BCBBBAC, C

k) Welches Sortierverfahren verbirgt sich hinter dem folgenden Codeschnipsel?

```
//Codeschnipsel
  if (array[j]>m) {
    array[j+1] = array[j];
  } else {
    break;
  }
```

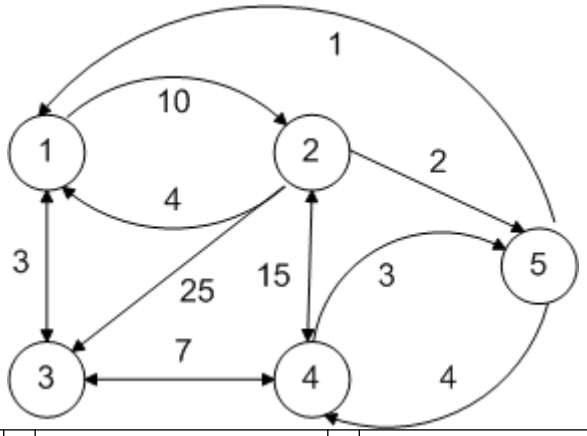
Insertionsort

Aufgabe 2

a) Stellen Sie die Adjazenzmatrix für den gegebenen Graphen auf.

	1	**2**	**3**	**4**	**5**
1	-	10	3	∞	∞
2	4	-	25	15	2
3	3	∞	-	7	∞
4	∞	15	7	-	3
5	1	∞	∞	4	-

b) Finden Sie mithilfe des Dijkstra-Algorithmus die minimalen Kosten, um von Knoten 2 zu jedem anderen Knoten des Graphen zu gelangen.



v_i	d				p			
	1	**3**	**4**	**5**	**1**	**3**	**4**	**5**
2	4	25	15	2	2	2	2	2
5	3	25	6		5		5	
1		6*	6			1		
3			6					
4								
	3	6	6	2	5	1	5	2

* Man könnte hier auch die 4 auswählen.

Aufgabe 3

Sortieren Sie die unten angegebene Zahlenfolge in aufsteigender Reihenfolge. Benutzen Sie den Quicksort-Algorithmus. Wählen Sie das rechte Element einer Teilliste als Pivot-Element. Machen Sie die Pivot-Elemente, die Teilfelder, sowie jeden Tauschvorgang kenntlich. Für Teilfelder von 2 Elementen brauchen Sie den Quicksort-Algorithmus nicht mehr zu verwenden, sondern dürfen die Elemente gegebenenfalls einfach umtauschen.

7	9	11	6	3	2	1	5	12	4	8
7	6	3	2	1	5	4	8	12	11	9
3	2	1	4	7	5	6		9	11	12
1	2	3		5	6	7				

Aufgabe 4

Gegeben sei das folgende Feld:

20	16	21	3	6	11	42

Sortieren Sie das Feld mit Heap-Sort.

a) Wenden Sie einen der in der Vorlesung vorgestellten Algorithmen an, um das Feld in einen Heap zu verwandeln. Geben Sie jeweils nach dem Einfügen einer Zahl den entstandenen Heap an, wahlweise als Feld (Tabellenvorlage) oder als Baum.

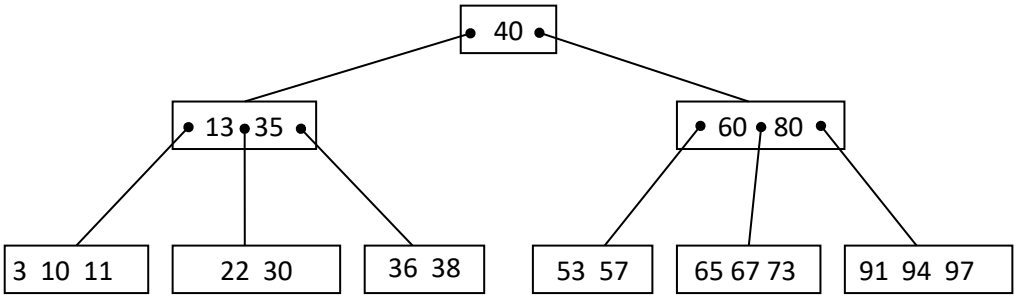
20						
20	16					
21	16	20				
21	16	20	3			
21	16	20	3	6		
21	16	20	3	6	11	
42	16	21	3	6	11	20

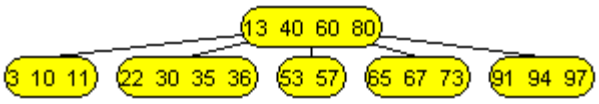
b) Sortieren Sie die Elemente durch Auflösung des Heaps. Geben Sie für jeden Zwischenschritt das aktuelle Feld an (einschließlich der bereits sortierten Elemente). Tipp: Sie können sich zusätzlich als Hilfestellung auch jeweils den Heap in Baumform notieren.

42	16	21	3	6	11	20
21	16	20	3	6	11	42
20	16	11	3	6	21	42
16	6	11	3	20	21	42
11	6	3	16	20	21	42
6	3	11	16	20	21	42
3	6	11	16	20	21	42

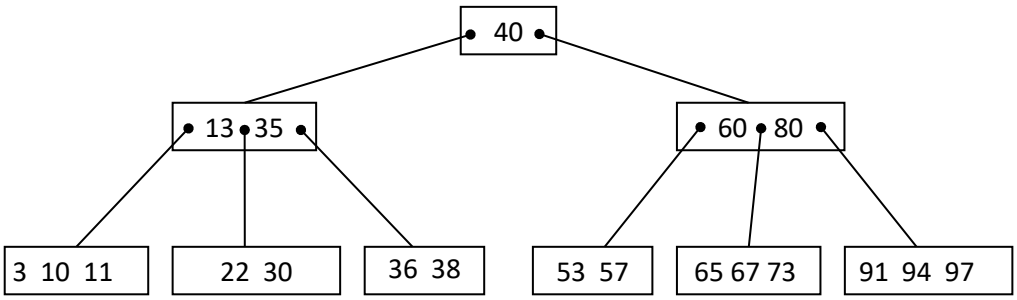
Aufgabe 5

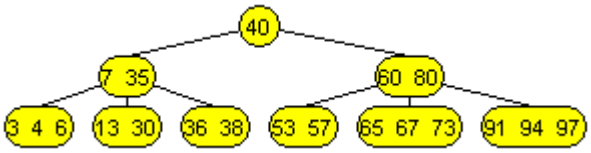
a) Gegeben sei folgender B-Baum der Ordnung 2. Löschen Sie die **38**.



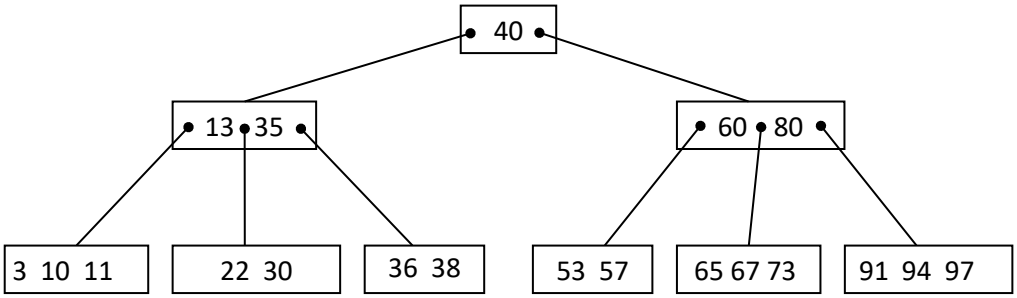


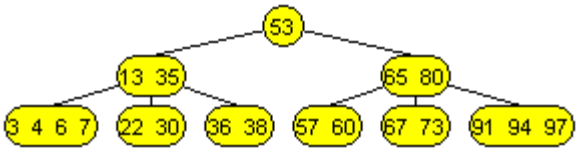
b) Gegeben sei folgender B-Baum der Ordnung 2. Löschen Sie die **22**.



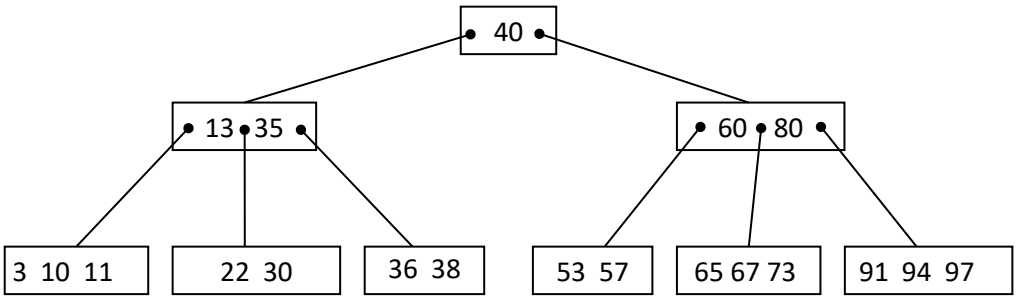


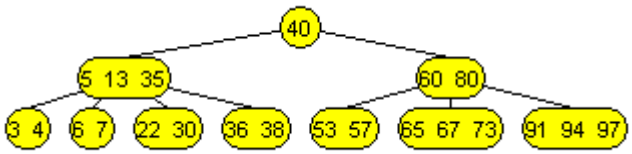
c) Gegeben sei folgender B-Baum der Ordnung 2. Löschen Sie die **40**.





d) Gegeben sei folgender B-Baum der Ordnung 2. Fügen Sie die **5** ein.





Aufgabe 6

Gegeben sei der folgende Code:

```
public static boolean analyzeString(String x) {
      for (int i=0; i<x.length()-1; i++) {
            if (x.substring(i+1).contains(""+x.charAt(i))) {
                  return false;
            }
      }
      return true;
}
```

a) Was prüft der angegebene Code?
Er prüft, ob Zeichen doppelt im übergebenen String vorkommen.

b) Welche Laufzeitkomplexität hat der Code in Abhängigkeit von der Länge des Strings x? Geben Sie den Worst-Case an. Begründen Sie Ihre Antwort.

Worst-Case: $O(n^2)$
Im Worst Case ist kein Zeichen doppelt und es kommt durch die Schleife und das implizite O(n) der substring-Methode (und der contains-Methode) zu $O(n^2)$.

c) Skizzieren Sie in Worten einen Algorithmus, der die Laufzeitkomplexität im Worst Case gegenüber dem angegebenen Code verbessert. Nennen Sie auch die neue Laufzeitkomplexität.

Es gibt zwei Möglichkeiten:
1. Man sortiert das Feld vorher und testet anschließend, ob zwei gleiche Buchstaben direkt hintereinander stehen. Das senkt den Worst Case auf O(n log n).
2. Man benutzt ein HashSet, in das man die Buchstaben nach der Reihe einfügt und vor dem Einfügen testet, ob sie schon vorhanden sind. Das senkt (allerdings nur bei einer guten Hash-Funktion) den Worst Case auf O(n).

Aufgabe 7:

Implementieren Sie die Methode

```
public static ArrayList<Integer> textSearch(String text, String pattern),
```

die ein übergebenes Pattern in einem übergebenen Text sucht und ein Array zurückgibt, in dem die Startindizes aller Vorkommen des Patterns im Text gespeichert sind.

Das Pattern kann neben Zahlen und Groß- und Kleinbuchstaben auch das Zeichen ‚?' enthalten, das als Platzhalter für ein beliebiges Zeichen steht.

```
public static ArrayList<Integer> textSearch(String text, String pattern) {
            ArrayList<Integer> ret = new ArrayList<Integer>();
            weiterruecken:
            for (int i = 0; i <= text.length() - pattern.length(); i++) {
                  for (int j = 0; j < pattern.length(); j++) {
                        if (text.charAt(i + j) != pattern.charAt(j)
                                    && pattern.charAt(j) != '?') {
                              continue weiterruecken;
                        }
                  }
                  // Pattern passt
                  ret.add(i);
                  // Zwei Moeglichkeiten
                  // 1.) Pattern rueckt um 1 weiter
                  // 2.) Pattern rueckt um die Laenge des Patterns weiter
                  // Fuer Moeglichkeit 2 muesste i=i+j-1 eingefuegt werden.
            }
            return ret;
      }
```

Aufgabe 8:

Ein Knoten eines Binärbaums habe folgendes Aussehen:

```
public class Node {
  public String cont;
  public Knoten left;
  public Knoten right;
}
```

Schreiben Sie eine Funktion

```
public static boolean isComplete(Node root),
```

die überprüft, ob der Baum, dessen Wurzel der übergebene Knoten ist, vollständig ist oder nicht. Ist der Baum vollständig (alle Ebenen sind komplett besetzt), wird `true` zurückgegeben, ansonsten `false`.

Idee hier: Es wird überprüft, ob gilt: Anzahl Knoten = $2^{\text{Höhe+1}}$-1. Der Einfachheit halber wird nicht die Höhe berechnet, sondern die Anzahl der Level = Höhe+1.

```
public static boolean isComplete(Node root) {
      // 1<<x: Ergebnis ist 2 hoch x; << ist Linksshift-Funktion
      return getNodeCount(root) == 1 << getLevelCount(root) - 1;
}

public static int getNodeCount(Node k) {
      if (k == null) {
            return 0;
      }
      return getNodeCount(k.left) + getNodeCount(k.right) + 1;
}

public static int getLevelCount(Node k) {
      // LevelCount == Hoehe+1
      if (k == null) {
            return 0;
      }
      return Math.max(getLevelCount(k.left), getLevelCount(k.right)) + 1;
}
```

Aufgabe 9:

a) Implementieren Sie eine Klasse Kind mit folgenden Attributen:
`String name;`
`int alter;`
und einem passenden Konstruktor.

b) Implementieren Sie eine Klasse `Ringelreihen`, die eine Gruppe von Kindern kreisförmig anordnet (siehe Bild). Erweitern Sie eventuell die Klasse `Kind`. Die Klasse Ringelreihen soll einen Zeiger auf ein Kind des Kreises besitzen. Ferner soll die Klasse folgende Methoden enthalten:

```
public Ringelreihen(Kind k1, Kind k2) /*Erzeugt ein Ringelreihen mit zwei Kindern*

public void moveLeft()  /*verschiebt den Zeiger um ein Kind gegen die Uhrzeigerrichtung*/

public void moveRight() /*verschiebt den Zeiger um ein Kind in Uhrzeigerrichtung*/

public void insert(Kind k) /*fügt ein neues Kind in den Reigen (links vom aktuellen Kind) ein und setzt den Zeiger auf das neue Kind.*/

public int size()   /*Gibt die Anzahl der Kinder im Reigen zurück.*/

public void setToYoungest()  /*Setzt den Zeiger auf das jüngste Kind oder auf eines der jüngsten, falls es mehrere jüngste Kinder gibt.*/
```

Benutzen Sie keine vorgefertigten Java-Klassen, wie z.B. ArrayList oder LinkedList.

```
public static class Kind {
            public String name;
            public int alter;
            public Kind rechts;
            public Kind links;
      }

public static class Ringelreihen {
            public Kind zeiger;
            private int anzahl;

            public Ringelreihen(Kind k1, Kind k2) {
                  k1.links = k2;
                  k1.rechts = k2;
                  k2.links = k1;
                  k2.rechts = k1;
                  zeiger = k1;
                  anzahl = 2;
            }
```

```
        public void moveLeft() {
              zeiger = zeiger.links;
        }

        public void moveRight() {
              zeiger = zeiger.rechts;
        }

        public void insert(Kind k) {
              k.rechts = zeiger;
              k.links = zeiger.links;
              k.links.rechts = k;
              k.rechts.links = k;
              zeiger = k;
              anzahl++;
        }

        public int size() {
              return anzahl;
        }

        public void setToYoungest() {
              Kind jung = zeiger;
              Kind temp = zeiger.rechts;
              while (temp != zeiger) {
                    if (temp.alter < jung.alter) {
                          jung = temp;
                    }
                    temp = temp.rechts;
              }
              zeiger = jung;
        }
    }
}
```