68

# Klausur Algorithmen

## Gruppe D

**Name:** _____

**Matrikelnummer:** _____

| Aufgabe | Erreichte Punktzahl |
|---|---|
| 1 ( 6 Punkte) | |
| 2 ( 7 Punkte) | |
| 3 ( 5 Punkte) | |
| 4 ( 7 Punkte) | |
| 5 ( 7 Punkte) | |
| 6 ( 6 Punkte) | |
| 7 ( 10 Punkte) | |
| 8 ( 7 Punkte) | |
| 9 ( 7 Punkte) | |
| Summe 62 Punkte | |

**Bearbeitungszeit: 100 Minuten**

**Note:** _____

**Gesamtnote:** _____

**Bitte kreuzen Sie an, wenn Sie Ihren 3. Versuch durchführen:**

☐ **3. Versuch**

**Die Klausur ist bestanden, wenn mindestens 30 Punkte erreicht werden.**

## 1. Aufgabe (6 Punkte)

Betrachten Sie nachfolgenden Algorithmus in Pseudocode:

```
1:    // G = (V, E) sei Graph mit n Knoten und m Kanten
2:    // Q sei leere Queue
3:    int sum = 0;
4:
5:    for (v ϵ V)          // fuer alle Knoten
6:         Q.enqueue(v); // v zu Queue hinzufügen
7:
8:    while (Q != leer) {
9:         v = Q.dequeue();  // Knoten von Queue nehmen
10:        sum++;
11:        for (u ϵ Adj(v))     // fuer alle Nachbarn von v
12:              sum++;
13:   }
```

Analysieren Sie die Laufzeit unter der Annahme, dass die Queue-Operationen optimal realisiert werden und der Graph geeignet durch Adjazenzlisten realisiert ist!
Schätzen Sie dazu mit Hilfe der O-Notation die Worst-Case-Laufzeit des Algorithmus in Abhängigkeit von der Problemgröße, d.h. Anzahl Kanten **m** und Anzahl Knoten **n**, ab.
Begründen Sie Ihre Aussage, indem Sie zu jeder Anweisung den Aufwand für die einfache Ausführung angeben sowie die Anzahl der Ausführungen im Worst-Case.

## 2. Aufgabe (7 Punkte)

a) Sei $f(n) = 3n^3 + 4\sqrt{n}$ und
$g_1(n) = 5n$
$g_2(n) = 4n^3 + 3n^2$
$g_3(n) = 2n^4 + \log n$

Füllen Sie folgende Tabelle aus, indem Sie „ja“ oder „wahr“ bei wahren Aussagen eintragen, „nein“ oder „falsch“ bei falschen Aussagen eintragen. Bei Eintrag einer falschen Antwort wird 1/2 Punkt abgezogen.

untere Schranke obere Schranke

| | $f(n) \in \Omega(g_i(n))$ | $f(n) \in \Theta(g_i(n))$ | $f(n) \in O(g_i(n))$ |
|---|---|---|---|
| $i = 1$ | wahr | falsch | falsch |
| $i = 2$ | wahr | wahr | wahr |
| $i = 3$ | falsch | falsch | wahr |

**b)** Geben Sie eine möglichst einfache, scharfe Funktion $g(n)$ an, so dass $f(n) \in \Omega(g(n))$ mit $f(n) = \frac{1}{2} \cdot n \cdot \sqrt{n} + (4 \log n)/n$

$g(n) = n \cdot \sqrt{n}$

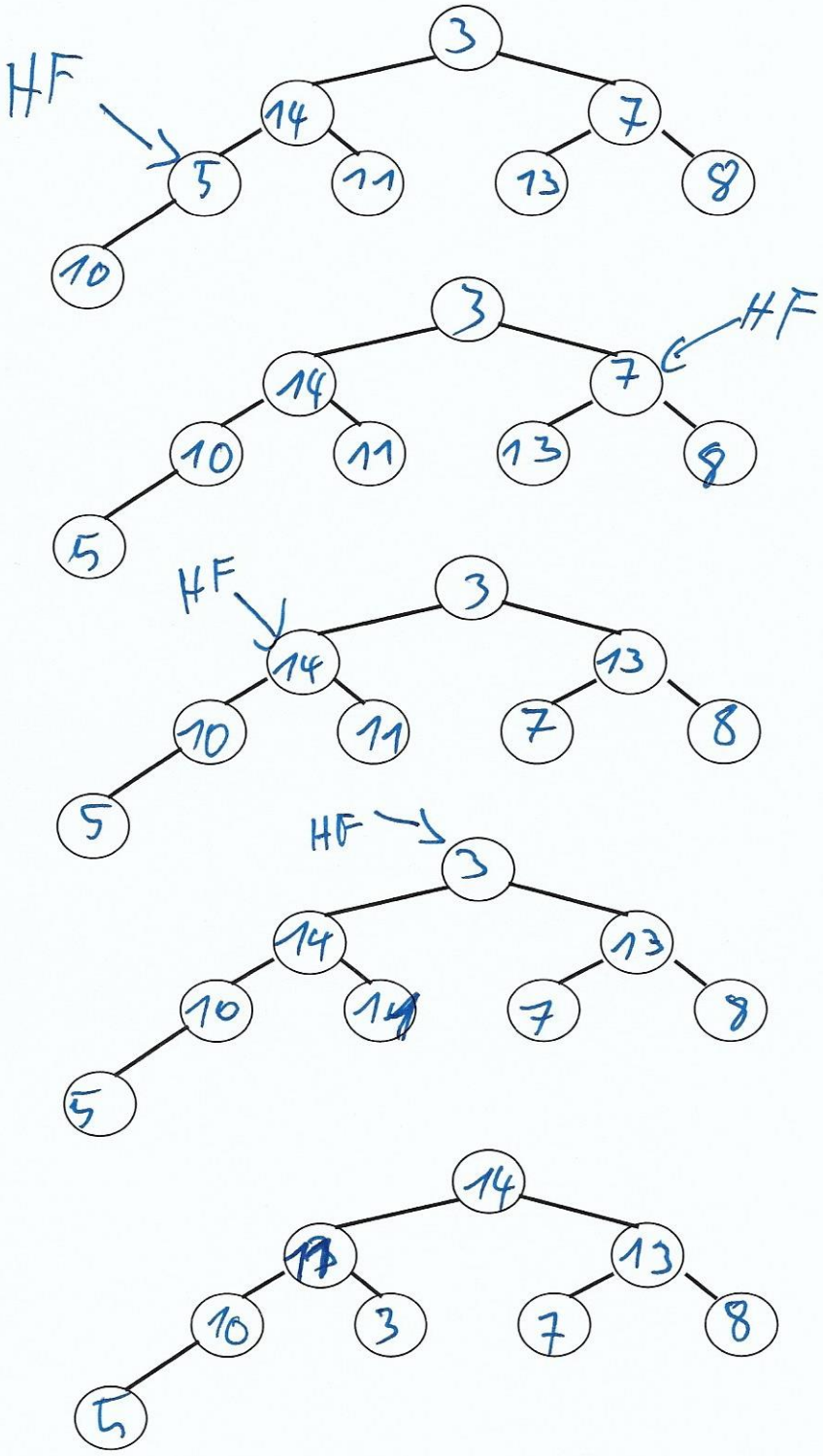
welcher Term

**c)** Geben Sie eine möglichst einfache, scharfe Funktion $k(n)$ an, so dass $h(n) \in O(k(n))$ mit $h(n) = 6 \cdot 2^n + n^2 + 6n$

$k(n) = 2^n$

welcher Term wächst am schnellsten

**3. Aufgabe (5 Punkte)**
Stellen Sie den Verlauf des Verfahrens **BuildHeap** zum Aufbau eines Heaps für die Zahlenfolge
3, 14, 7, 5, 11, 13, 8, 10 mit Hilfe eines Binärbaumes dar. In diesem Verfahren wird für bestimmte Knoten das Verfahren **Heapify** durchgeführt. Geben Sie zur Darstellung des Verfahrens jeweils an, auf welchen Knoten Sie **Heapify** anwenden und anschließend das Ergebnis nach vollständiger Abarbeitung (inklusive aller Rekursionen) von **Heapify**.



**4. Aufgabe (7 Punkte)**
Betrachten Sie nachfolgenden Ausschnitt einer Klasse für eine einfach verketteten Liste mit Strings.

```
public class SimpleLinkedList {
      class Node {
            Integer value;
            Node next;
      }

      private Node head;

      /**
       * ändert den Wert an Position index in der Liste
       * @param index der Index, an dessen Position ein Wert
       *        geändert werden soll (Indizierung startet bei 0)
       * @param newValue der neue Wert an Position index; wenn index kein
       *        gültiger Index, so bleibt die Liste unveraendert
       */
      public void update(int index, String newValue) {
            // …
      }
      // weitere Methoden
}
```

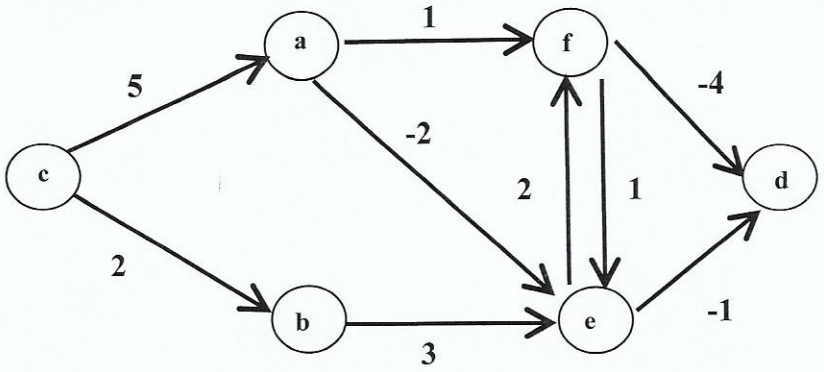
Integer

Das Attribut **head** verweist immer auf das erste Listenelement vom Typ **Node**. Falls die Liste leer ist, verweist **head** auf **null**. Die Zählung der Indices beginnt mit 0. Beachten Sie, dass es kein Attribut für die Anzahl der vorhandenen Listenelemente gibt.

Ergänzen Sie die Implementierung der Methode **update**.

```
public void update (int index, Integer newValue) {
    int i=0;
    while (i<index && current Node != null) {
        current Node = current Node.next();
    }
    if (current Node != null) {
        current Node.value = newValue;
    }
    return
```

## 5. Aufgabe (7 Punkte)




Wenden Sie **die ersten Schritte** des Bellman Ford Algorithmus an zur Bestimmung der kürzesten Wege ausgehend von Knoten **c**.
Geben Sie die pred- und dist-Werte nach jedem Durchlauf der äußeren for-Anweisung in nachfolgenden Tabellen an. Durchlaufen Sie dabei die Kanten des Graphen in lexikographischer Reihenfolge.

### Initialisierung

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| **pred** | | | – | | – | – |
| **dist** | ∞ | ∞ | 0 | ∞ | ∞ | ∞ |

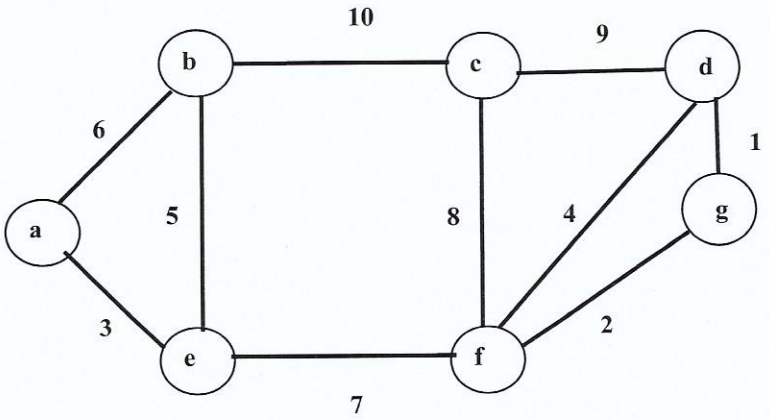
### Werte nach 1. Durchlauf äußere for-Anweisung

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| **pred** | c | c | – | – | – | – |
| **dist** | 5 | 2 | 0 | ∞ | ∞ | ∞ |

### Werte nach 2. Durchlauf äußere for-Anweisung

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| **pred** | c | c | – | f | a | e |
| **dist** | 5 | 2 | 0 | 1 | 3 | 5 |

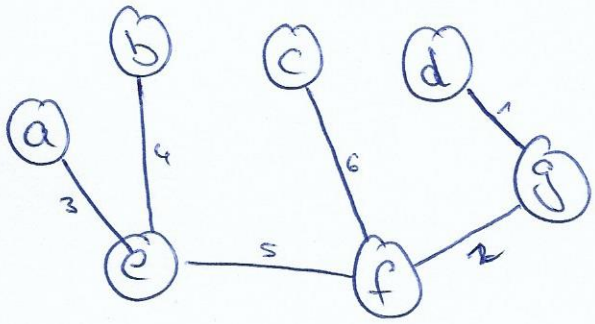
**6. Aufgabe (6 Punkte)**
Betrachten Sie nachfolgenden Graphen. Bestimmen Sie in diesem Graphen einen minimal spannenden Baum.



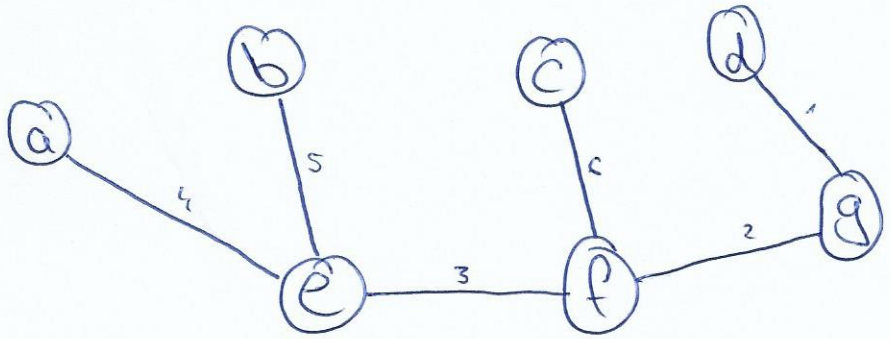

Verwenden Sie dazu
a) Kuskals Algorithmus
b) Prims Algorithmus; starten Sie dabei bei Knoten **d**.

Geben Sie jeweils die Reihenfolge an, in der die Kanten des minimal spannenden Baumes ausgewählt werden.

**a) Kruskal:** {d, g}, {f, g}, {a, e}, {b, e}, {e, f}, {c, f}

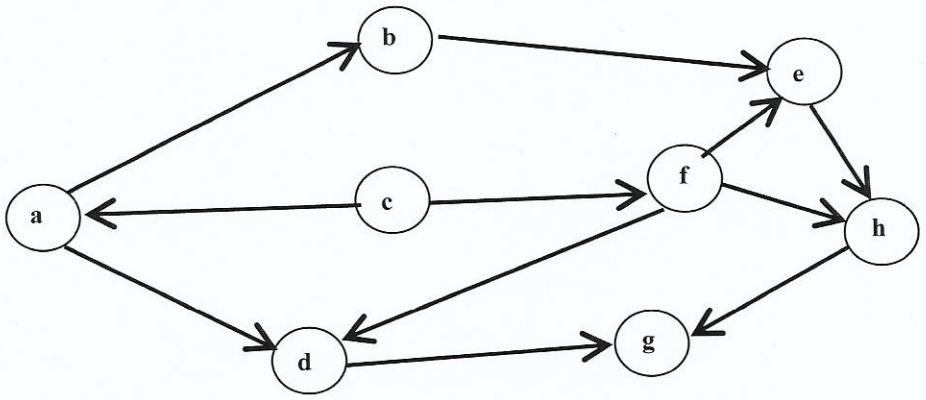



**b) Prim:**
{d, g}, {f, g}, {e, f}, {a, e}, {b, e}, {c, f}



**7. Aufgabe (10 Punkte)**

Betrachten Sie nachfolgenden Graphen:




a) Führen Sie eine **Breitensuche** startend bei Knoten **a** durch. Bei Auswahlmöglichkeit zwischen mehreren Knoten ist immer der lexikographisch kleinste zu wählen. Ermitteln Sie dazu die pred- und dist-Werte und tragen Sie diese in nachfolgender Tabelle ein:

Q a, b, d, e, g, h

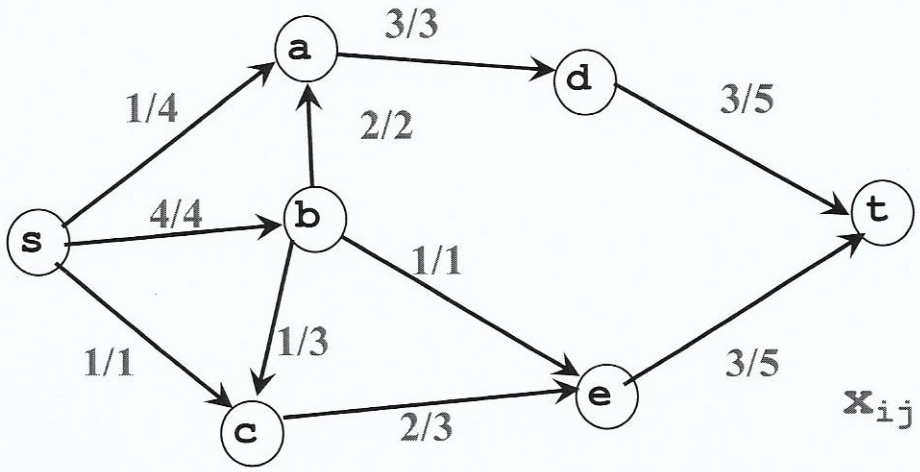
| | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| **pred** | – | a | – | a | b | – | d | e |
| **dist** | 0 | 1 | ∞ | 1 | 2 | ∞ | 2 | 3 |

b) Führen Sie eine **Tiefensuche** startend mit Knoten **a** durch. Bei der mehreren Auswahlmöglichkeiten eines Knotens wählen Sie immer den Knoten mit der lexikographisch kleinsten Beschriftung. Ermitteln Sie die **first**-, **last**- und **pred**-Werte und tragen Sie sie in nachfolgender Tabelle ein:

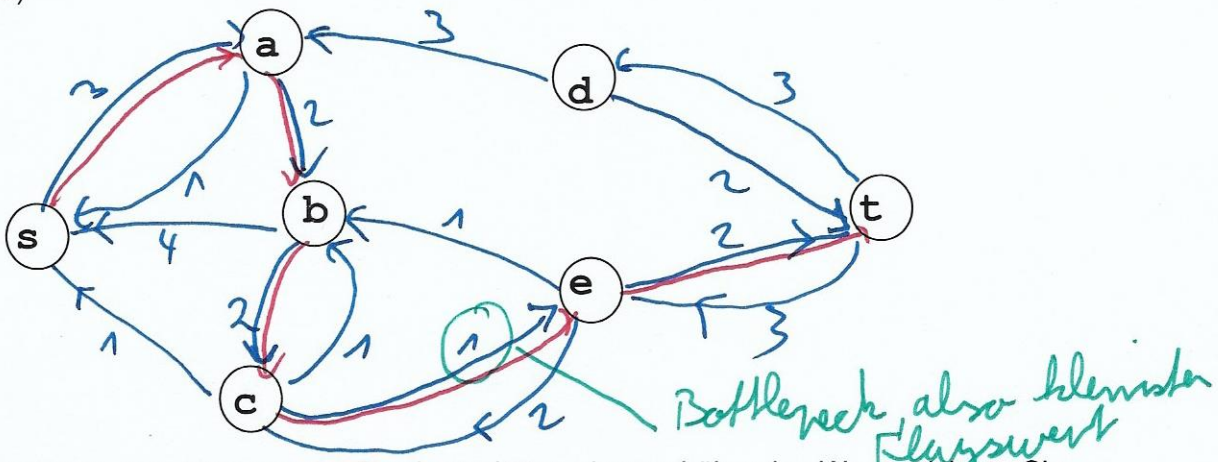
| | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| **first** | 1 | 2 | 13 | 10 | 3 | 14 | 5 | 4 |
| **last** | 12 | 9 | 16 | 11 | 8 | 15 | 6 | 7 |
| **pred** | – | a | – | a | b | c | h | e |

topologisch sortierbar?

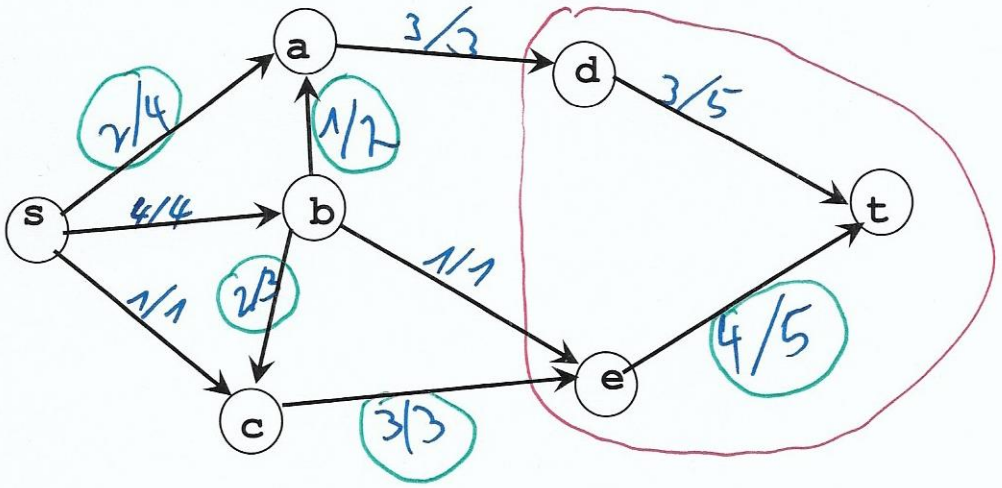
**8. Aufgabe (7 Punkte)**
Betrachten Sie nachfolgendes s-t-Netzwerk mit einem zulässigen Fluss **x**:




Geben Sie in nachfolgender Zeichnung den Graphen mit den Restkapazitäten an (Residual Network):




Ermitteln Sie in dem Graphen der Restkapazitäten einen erhöhenden Weg, zeichnen Sie diesen ein und geben Sie den resultierenden Fluss in folgender Zeichnung an:




7

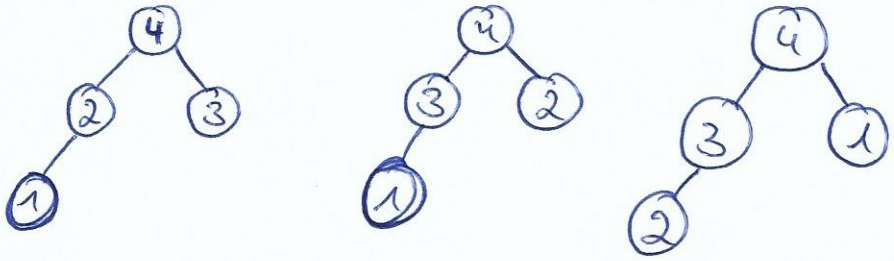
Geben Sie den Wert des aktuellen Flusses an, sowie einen Schnitt minimaler Kapazität und die Kapazität dieses Schnittes!

$S = \{s, a, b, c\}$

$T = \{d, e, t\}$

**9. Aufgabe (7 Punkte)**

**a)** Geben Sie für die Schlüsselwerte {1, 2, 3, 4} alle möglichen Heaps an!




**b)** Geben Sie ein möglichst effizientes Verfahren in Pseudo-Code an, um in einem Heap den kleinsten Wert zu finden. Geben Sie die Laufzeit im Worst-Case und im Best-Case in Abhängigkeit von der Anzahl der Schlüsselwerte an. Skizzieren Sie die Situation, in der der Worst-Case und in der der Best-Case eintritt.

Durchlauf alle Blätter und merke den kleinsten Wert

BestCase = WorstCase

$\frac{n}{2} \rightarrow O(n)$