32

Klausur Algorithmen

Gruppe B

Name [redacted]

Matr [redacted]

Aufgabe	Erreichte Punktzahl
1 (7 Punkte)	6,5
2 (6 Punkte)	5,0
3 (5 Punkte)	5,0
4 (7 Punkte)	6,5
5 (6 Punkte)	4,5
6 (7 Punkte)	7,0
7 (10 Punkte)	9,0
8 (7 Punkte)	7,0
9 (7 Punkte)	6,0
Summe 62 Punkte	56,5

Bearbeitungszeit: 100 Minuten

Note: 1,0

Gesamtnote: 1,0

Rp. 8.2.17

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

☐ **3. Versuch**

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

1. Aufgabe (7 Punkte)

6,5

a) Sei $f(n) = 3n^2 + 2\log n$ und
$g_1(n) = n^2 + 3n$
$g_2(n) = 5n^3 + \sqrt{n}$
$g_3(n) = 6n$

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.

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

b) Geben Sie eine möglichst einfache, scharfe Funktion $g(n)$ an, so dass $f(n) \in \Omega(g(n))$ mit $f(n) = 3n \cdot \log n + 4\sqrt{n}/n$

$g(n) = n \cdot \log n + \frac{\sqrt{n}}{n}$ -0,5

~~einfacher, aber~~ einfacher, aber nicht mehr so scharf: $g(n) = n \cdot \log n$

f.

c) Geben Sie eine möglichst einfache, scharfe Funktion $k(n)$ an, so dass $h(n) \in O(k(n))$ mit $h(n) = 2n^3 + \frac{1}{2}n^2 + 3\log n$

$k(n) = n^3$ ✓

2. Aufgabe (6 Punkte) 5

Betrachten Sie nachfolgenden Algorithmus in Pseudocode:

```
1:    // G = (V, E) sei Graph mit n Knoten und m Kanten
2:    // S sei leerer Stack
3:
4:    for (v ∈ V)  // fuer alle Knoten
5:         S.push(v); // v zu Stack hinzufügen
6:
7:    while (S != leer) {
8:         v = S.pop();
9:         System.out.print(v + ": ");
10:        for (u ∈ Adj(v))    // fuer alle Nachbarn von v
11:             System.out.print(u + ", ");
12:
13:        System.out.println(". ");
14:   }
```

O(1) } O(n) ✓

O(1) ✓

O(1)

O(1) } sc O(1) / wc O(n)

O(1)

sc O(n) / wc O(n²)

Analysieren Sie die Laufzeit unter der Annahme, dass die Stackoperationen 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.

Die einfache Ausführung ist immer von konstanter Laufzeit.

Die äußere for-Schleife hat immer die Laufzeit O(n). −1

Die innere for-Schleife hat im best case, also kein Nachbar, O(1),
im worst case, n ~~Nachbarn~~ Nachbarn, O(n)

Die while Schleife wird O(n)-mal durchgeführt.

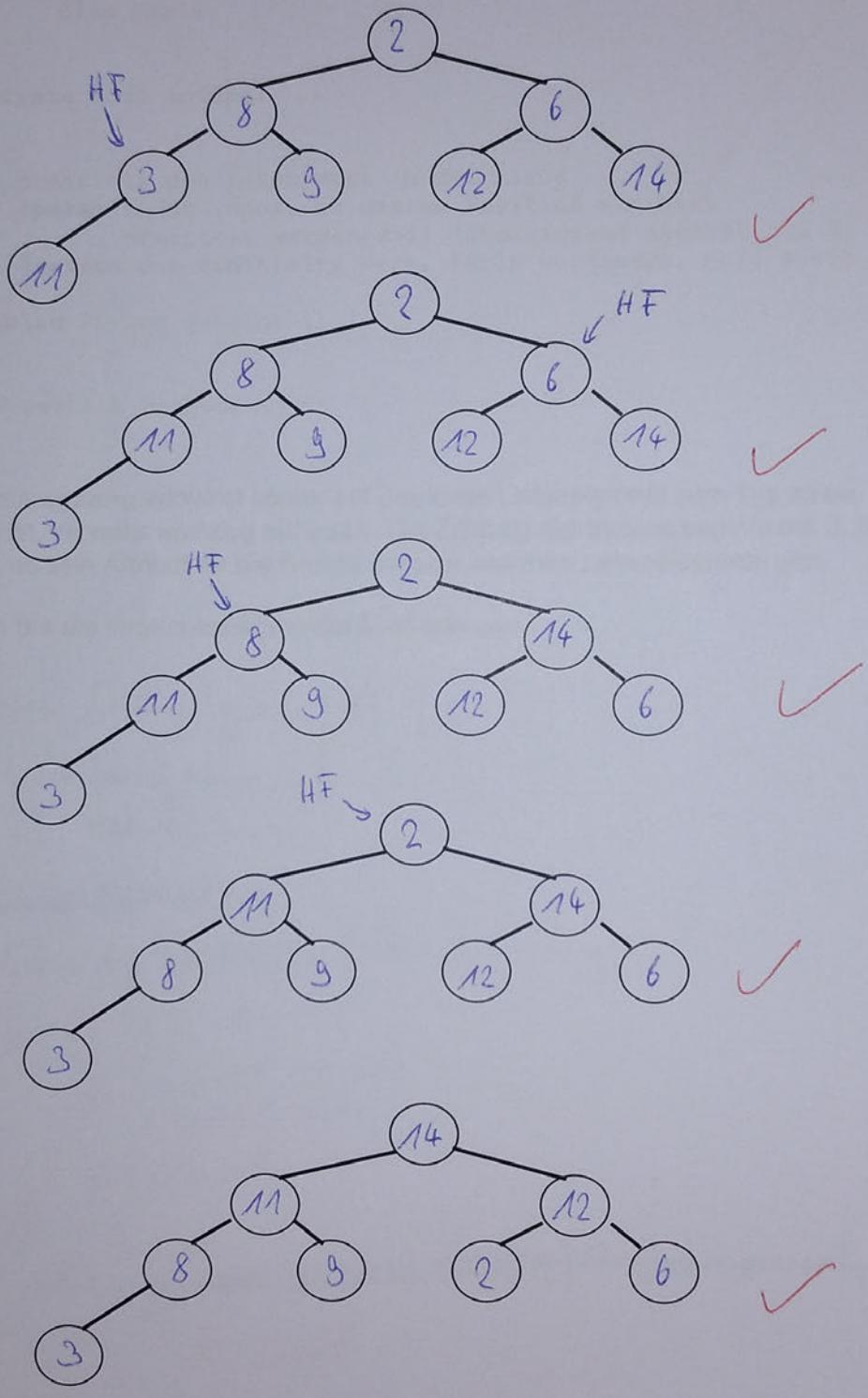
Zu ungenau! Besser wäre für alle Durchläufe von while bestimmen ⇒ O(m)

Total: best case: $O(n + n) = O(n)$

worst case: $O(n + n^2) = O(n^2)$ (✓)

3. Aufgabe (5 Punkte) (5)

Stellen Sie den Verlauf des Verfahrens **BuildHeap** zum Aufbau eines Heaps für die Zahlenfolge
2, 8, 6, 3, 9, 12, 14, 11 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) 6,5

Betrachten Sie nachfolgenden Ausschnitt einer Klasse für eine einfach verketteten Liste mit Strings.

```
public class EinfachVerketteteListe {
      class Elem {
            String wert;
            Elem next;
      }

      private Node anfang;

      /**
       * ermittelt den i-ten Wert in der Liste
       * @param i der Index, an dessen Position ein Wert
       *        ermittelt werden soll (Indizierung startet bei 0)
       * @return der ermittelte Wert, falls vorhanden, null sonst
       */
      public String get(int i) {
            // …
      }
      // weitere Methoden
}
```

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

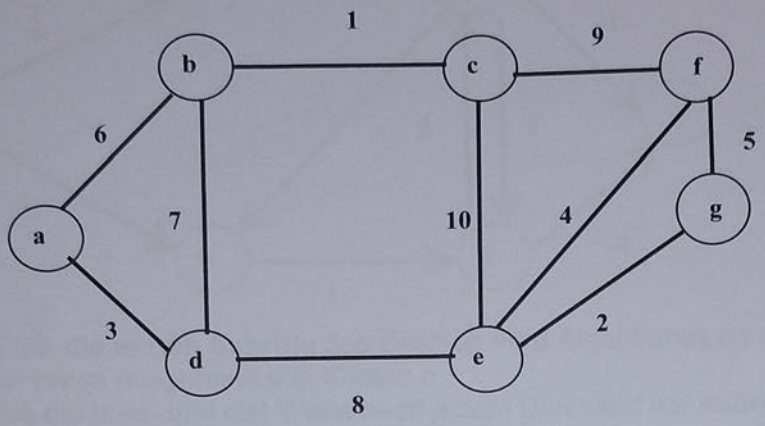
Ergänzen Sie die Implementierung der Methode **get**.

```
public String get(int i) {
  if (anfang == null)
      return null;
  ~~while (i > 0) {~~
  Elem e = ~~anfang;~~ (Elem) anfang;
  for (; i>0; i~~--~~--) {
      if (e.next == null)
          return null;
      e = e.next;
  }
  return ~~e.next;~~ (e.next == null) ? ~~null : e.next;~~
                  e;
}                 e.next
```

−0,5

5. Aufgabe (6 Punkte) 4,5

Betrachten Sie nachfolgenden Graphen. Bestimmen Sie in diesem Graphen einen minimal spannenden Baum.



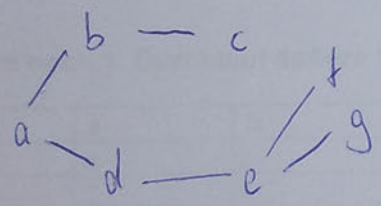
Verwenden Sie dazu

a) Kuskals Algorithmus
b) Prims Algorithmus; starten Sie dabei bei Knoten **c**.

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

a) Kruskal: ✓

$\{b,c\}, \{g,e\}, \{a,d\}, \{e,f\}, \{a,b\}, \{d,e\}$ ✓ ✓ ✓ ✓ ✓ ✓

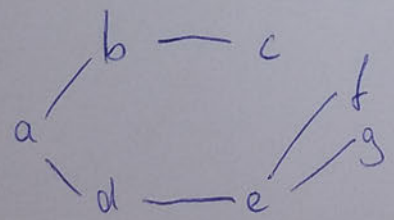


b) Prim: ✓

$\{b,c\}, \{a,b\}, \{a,d\}, \{d,e\}, \{g,e\}, \{e,f\}$ ~~$\{g,e\}, \{e,f\}$~~ $\{e,f\}, \{e,g\}$ ✓ ✓ ✓ ✓

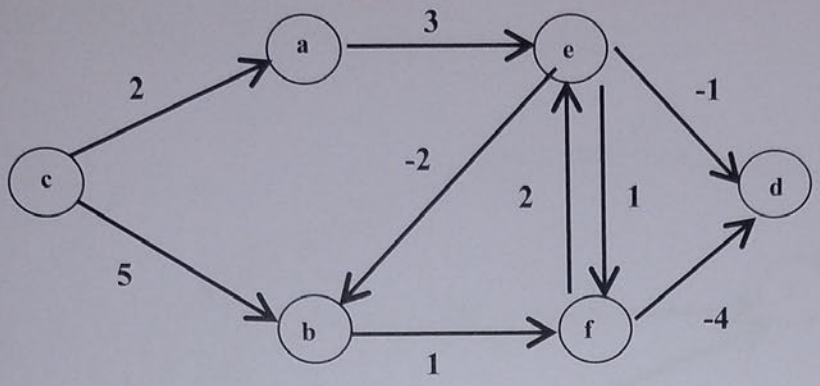
R. −1,5

Davon ausgegangen, dass die Queue lexikografisch ist.



6. Aufgabe (7 Punkte)

7



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

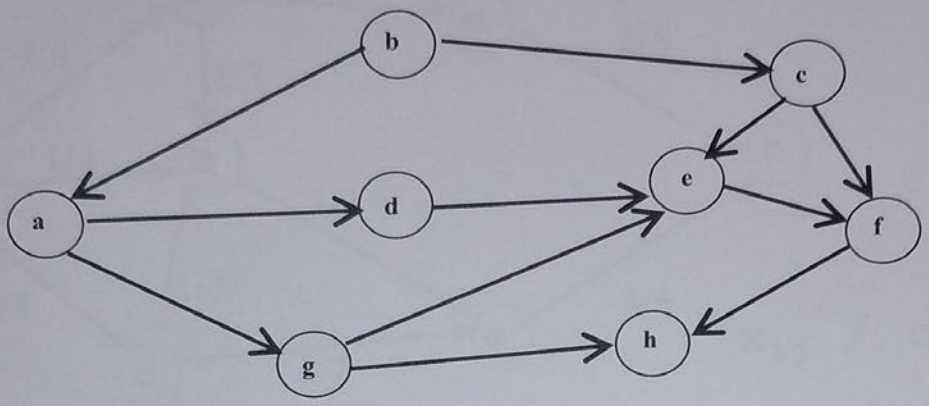
	a	b	c	d	e	f
pred	c	c	–	–	–	–
dist	2	5	0	∞	∞	∞

Werte nach 2. Durchlauf äußere for-Anweisung

	a	b	c	d	e	f
pred	c	~~c~~ e	–	~~e~~ f	a	b
dist	2	~~5~~ 3	0	~~4~~ 2	5	6

7. Aufgabe (10 Punkte) (9)

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:

a, d, g, e, h, f, b, c

	a	b	c	d	e	f	g	h
pred	-	-	-	a	d	~~a~~ e	a	~~f~~ g
dist	0	∞	~~∞~~	1	2	~~4~~ 3	1	4 2

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	13	14	2	3	4	11	5
last	~~10~~ 16	16	15	9	8	7	12	6
pred	-	-	b	a	d	e	a	f

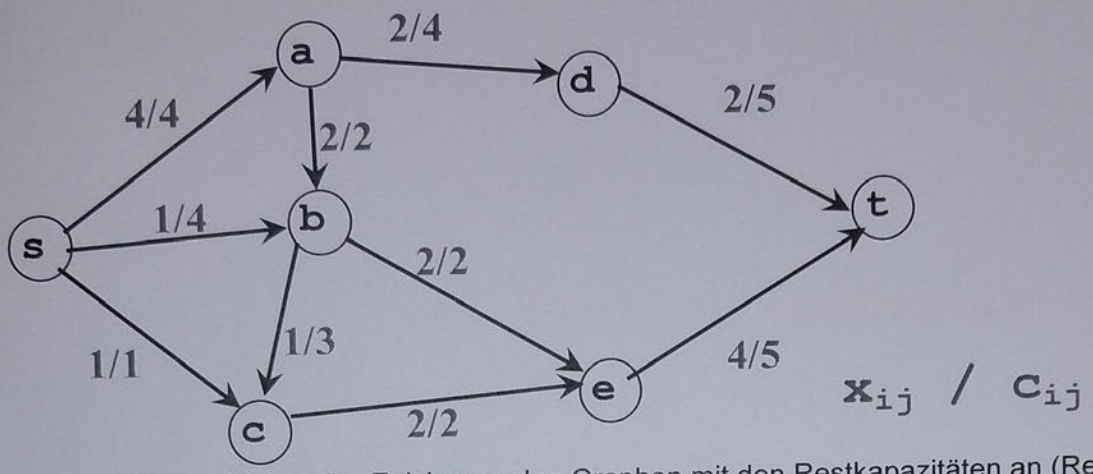
-1

a → d → e → f → h g b c

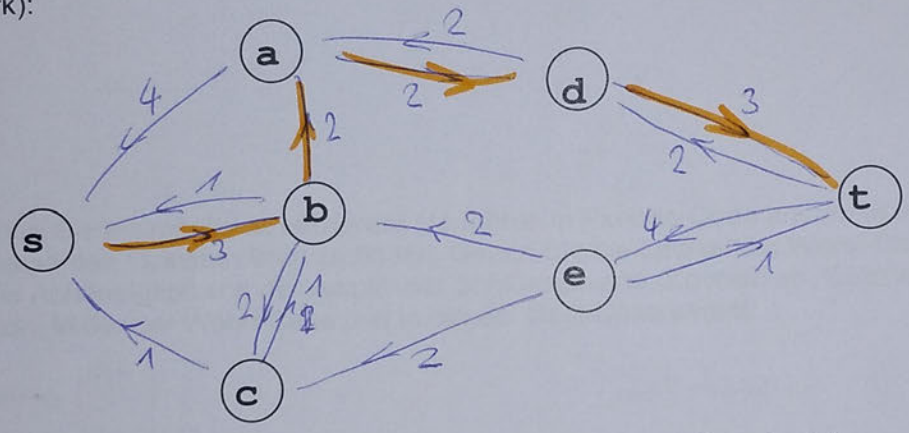
b → g

7

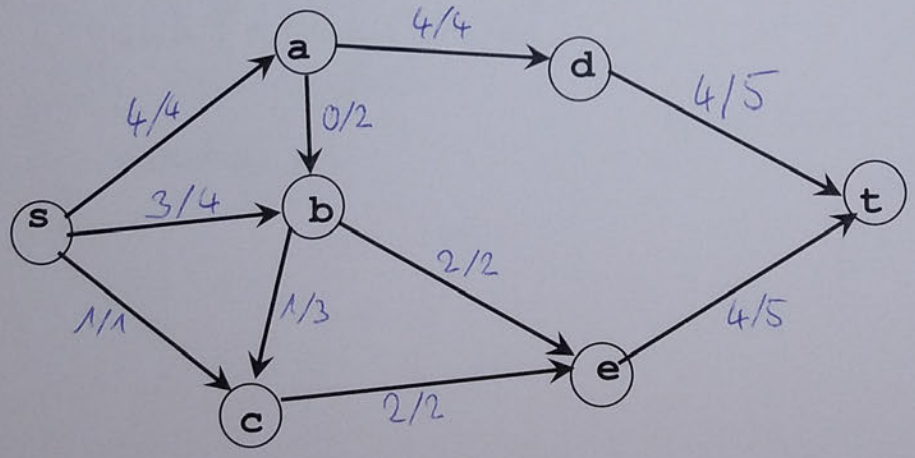
8. Aufgabe (7 Punkte)
Betrachten Sie nachfolgendes s-t-Netzwerk mit einem zulässigen Fluss **x**:



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:

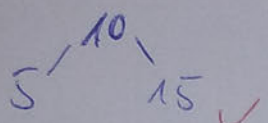


Geben Sie den Wert des aktuellen Flusses an, sowie einen Schnitt minimaler Kapazität und die Kapazität dieses Schnittes!

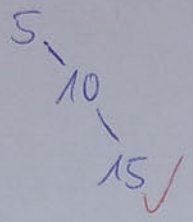
Der aktuelle Fluss-Wert ist 8, der Schnitt mit minimaler Kapazität ist $(S = \{s,a,b,c\}, T = \{d,e,t\})$, dessen Kapazität ist 8, der Fluss ist also maximal.

9. Aufgabe (7 Punkte) (6)

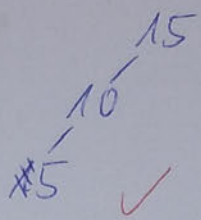
a) Geben Sie für die Schlüsselwerte {5, 10, 15} alle möglichen Suchbäume an!



✓

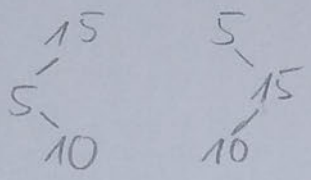


✓



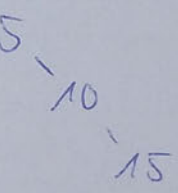
✓

es fehlen 2 Suchbäume −1

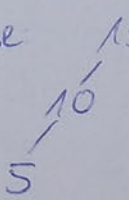


b) Geben Sie ein möglichst effizientes Verfahren in Pseudo-Code an, um in einem Suchbaum den kleinsten Wert zu finden. Geben Sie die Laufzeit im Worst-Case und im Best-Case in Abhängigkeit von der Anzahl der Schlüsselwerte (Knoten) an. Skizzieren Sie die Situation, in der der Worst-Case und in der der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

```
min (T) {
~~while (T.has~~
r = T.root;
while (r.hasLeftChild)
    r = r.LeftChild;
return r;
}
```
✓

Best-Case: 

→ O(1) ✓

Worst-Case 

→ O(n) ✓