22

Klausur Algorithmen

Name: George, Lars

Matrikelnummer: 744665

Aufgabe	Erreichte Punktzahl
1 (6 Punkte)	6
2 (9 Punkte)	~~8~~ 4,5 Rip.
3 (8 Punkte)	8
4 (6 Punkte)	6
5 (7 Punkte)	7
6 (10 Punkte)	10
7 (8 Punkte)	8
8 (8 Punkte)	8
9 (6 Punkte)	6
10 (6 Punkte)	5,5
Summe 74 Punkte	~~72,5~~ 69 Rip.

Bearbeitungszeit: 90 Minuten

Note: 1,0

Gesamtnote: 1,0 9.02.08 Rip.

Bitte kreuzen Sie an, wenn einer der beiden nachfolgenden Fälle auf Sie zutrifft:

- ☐ **Letzter Versuch**
- ☐ **Zeitablauf**

Die Klausur ist bestanden, wenn mindestens 37 Punkte erreicht werden.

1. Aufgabe (6 Punkte) 6

Betrachten Sie nachfolgende Funktion (i : gebe die Zeilennummern an, auf die Sie sich bei der Laufzeitanalyse beziehen können)

```
1:      public static void mysterious(int[] A) {
2:              int i = 1;
3:              while (i < A.length) {
4:                      for (int j = 0; j < A.length / 2; j++) {
5:                              int tmp = A[j];
6:                              A[j] = A[A.length - 1 - j];
7:                              A[A.length - 1 - j] = tmp;
8:                      }
9:                      i *= 2;
10:             }
11:     }
```

- Schätzen Sie mit Hilfe der O-Notation die Laufzeit der Funktion im „worst-case" ab.
- Gibt es Unterschiede in der Größenordnung der Laufzeit im „worst-case" und im „best-case"? Wenn ja, welche gibt es?

Begründen Sie Ihre Aussagen!

3: $\log n$ (n wird immer verdoppelt bei der while Prüfung) $\boxed{n = \text{Länge des Arrays } A}$ ✓

4: $\frac{n}{2}$ (für die Hälfte der Elemente von A) ✓

$O(\log n \cdot \frac{n}{2})$ ✓

Es gibt keinen Unterschied der Laufzeit im "worst-case" zu der im "best-case", da sie nur von der Länge des Arrays abhängig ist und somit konstant für Arrays gleicher Länge ✓

~~5~~ 4,5

2. Aufgabe (9 Punkte)
Geben Sie an, welche der nachfolgenden Aussagen wahr, welche falsch sind. Für falsche Antworten werden jeweils 2 Punkte abgezogen!

- $2^{n+1} \in O(2^n)$ wahr ✓
- $n! \in O(n^n)$ wahr ✓
- $3\sqrt{n} \in O(\log n)$ falsch ✓
- $3n^2 - 20n \in \Omega(n^2)$ falsch f. −2
- $\log n^n \in O(\log n)$ falsch ✓
- $n^3 - n^2 + n \in \Theta(n^3)$ falsch f. −2

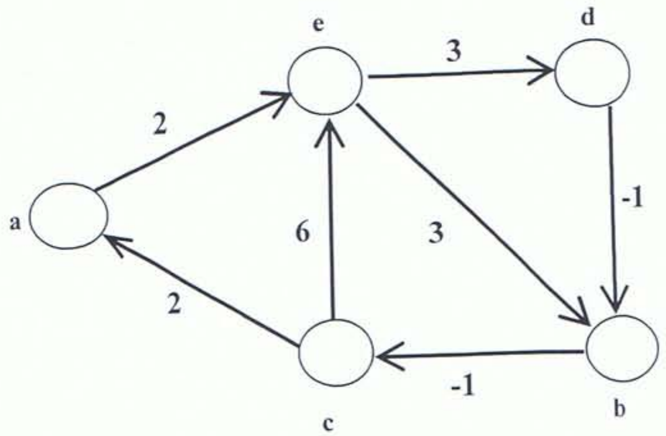
Geben Sie eine möglichst einfache, scharfe Funktion $g(n)$ an, so dass $f(n) \in O(g(n))$ mit $f(n) = n! + 3 \log n$.

$g(n) = n!$ ✓

Geben Sie eine möglichst einfache, scharfe Funktion $k(n)$ an, so dass $h(n) \in \Omega(k(n))$ mit $h(n) = n \cdot \log n + 4n^2$.

$k(n) = n \log n$ (✓) zu ungenau −0,5

3. Aufgabe (8 Punkte) 8



Betrachten Sie den oben aufgeführten Graphen.
Stellen Sie den Verlauf des Bellman-Ford-Algorithmus zur Bestimmung der kürzesten Wege dar. Startknoten von dem aus die kürzesten Wege bestimmt werden sollen sei Knoten `e`. Geben Sie die `pred`- und `dist`-Werte in nachfolgenden Tabellen nach den **ersten beiden** Durchläufen der äußeren `for`-Anweisung an. Die Kanten in der inneren `for`-Anweisung sollen in lexikographischer Reihenfolge durchlaufen werden (d.h. (a, e), (b,c), (c,a), (c,e), …)

Initialisierung

	a	b	c	d	e
pred	null	null	null	null	null
dist	∞	∞	∞	∞	0

Werte nach 1. Durchlauf äußere for-Anweisung

	a	b	c	d	e
pred	null	e	null	e	null
dist	∞	3	∞	3	0

Werte nach 2. Durchlauf äußere for-Anweisung

	a	b	c	d	e
pred	c	d	b	e	null
dist	4	2	2	3	0

4. Aufgabe (6 Punkte) 6

Stellen Sie den Verlauf des Aufteilungsschrittes von Quicksort (ohne die nachfolgenden Rekursionen) für folgende Zahlenfolgen dar. Dabei soll als Pivotelement das Element am rechten Rand gewählt werden. Stellen Sie die Zahlenfolgen nach jedem Austauschschritt dar.

a) 5, 6, 10, 22, 12, 17, 2, 8, 19, 9

b) 4, 7, 3, 5, 8

c) 6, 10, 4, 5, 2

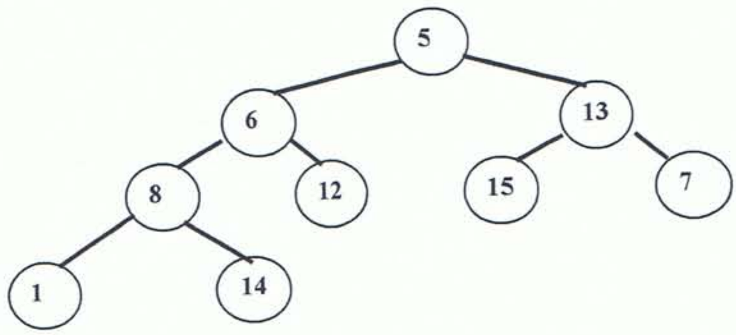
a) 5, 6, 10, 22, 12, 17, 2, 8, 19, $\boxed{9}$
5, 6, 8, 22, 12, 17, 2, 10, 19, $\boxed{9}$
5, 6, 8, 2, 12, 17, 22, 10, 19, $\boxed{9}$
5, 6, 8, 2, $\boxed{9}$, 17, 22, 10, 19, 12 ✓

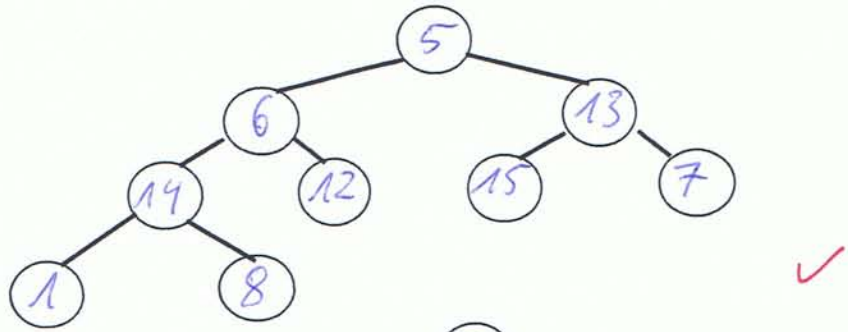
b) 4, 7, 3, 5, $\boxed{8}$ ✓

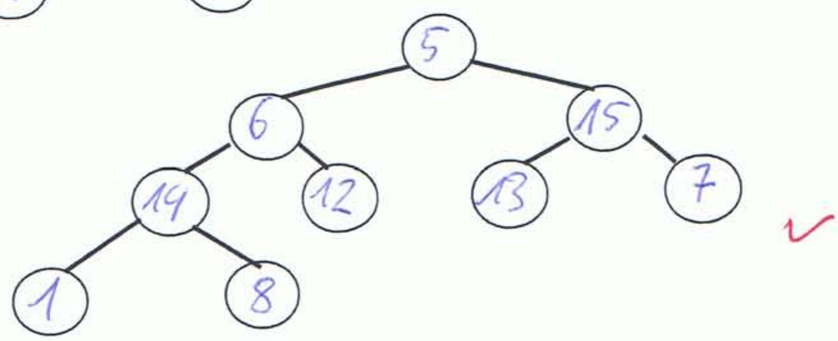
c) 6, 10, 4, 5, $\boxed{2}$
$\boxed{2}$, 10, 4, 5, 6 ✓

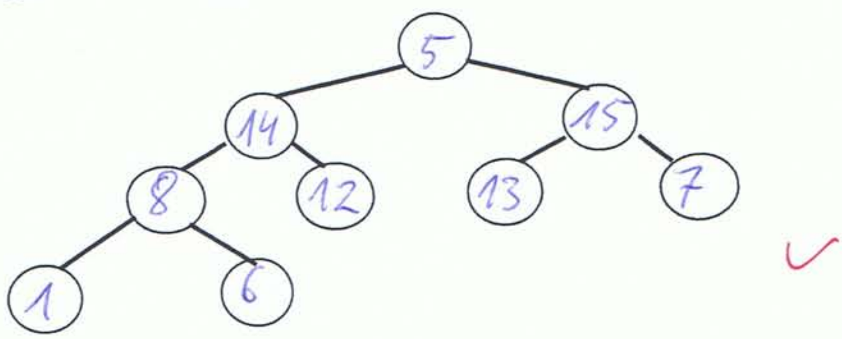
5. Aufgabe (7 Punkte)

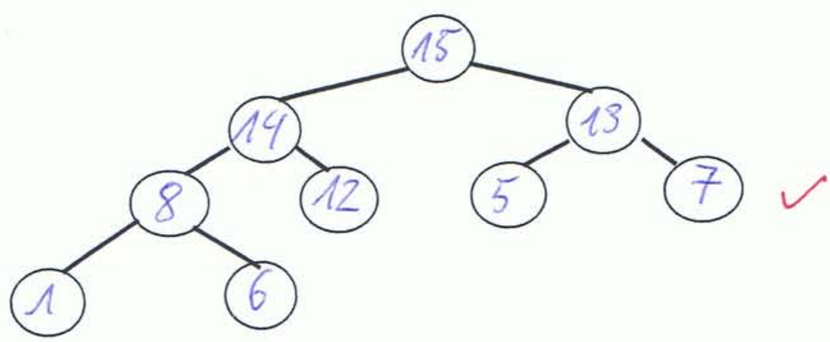
Wenden Sie auf nachfolgenden Binärbaum das Verfahren BuildHeap zur Erzeugung eines Heaps an.







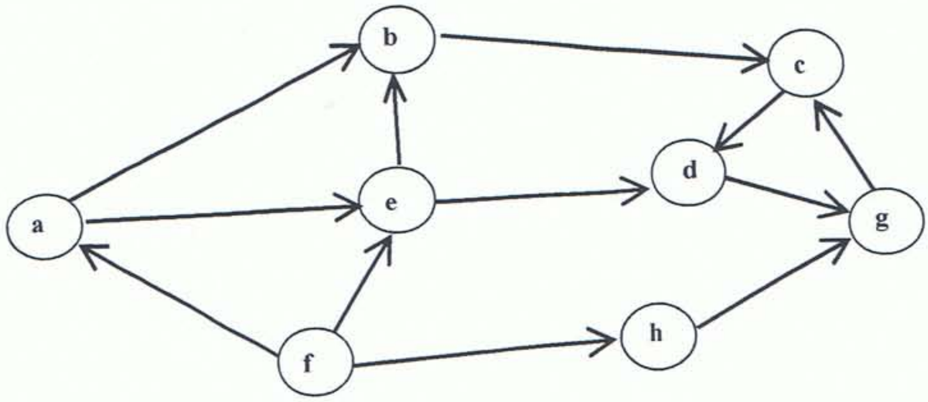




6. Aufgabe (10 Punkte)

10

a) Betrachten Sie nachfolgenden Graphen:



Führen Sie eine Tiefensuche startend mit Knoten **a** durch. Bei der Auswahl eines Nachbarknotens 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	3	4	10	13	5	14
last	12	9	8	7	11	16	6	15
pred	null	a	b	c	a	null	d	f

b) Geben Sie eine topologische Sortierung des Graphen aus a) an bzw. begründen Sie, falls dieser Graph nicht topologisch sortierbar ist.

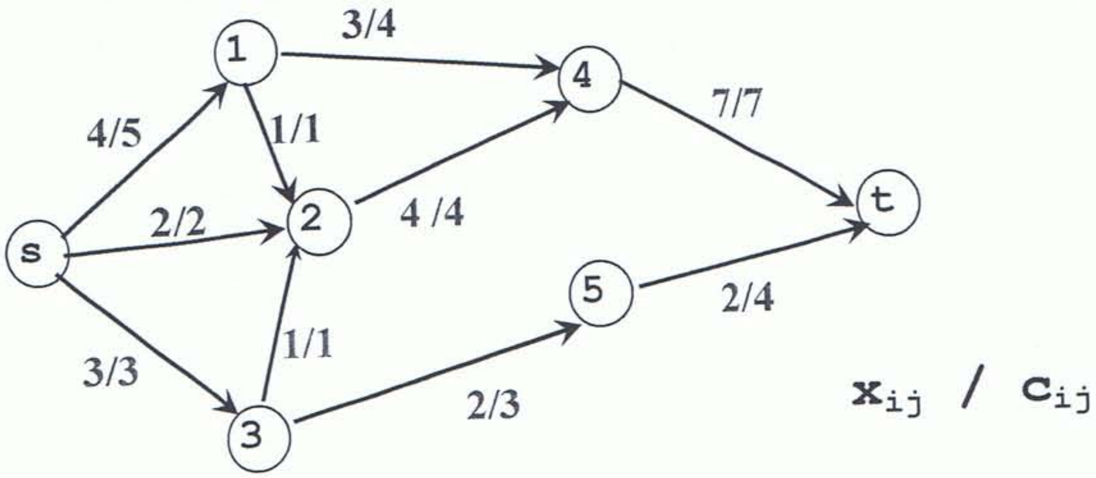
f → h a → e → b → c ⇄ d → g

Rückkante ⇒ topologische Sortierung nicht möglich!

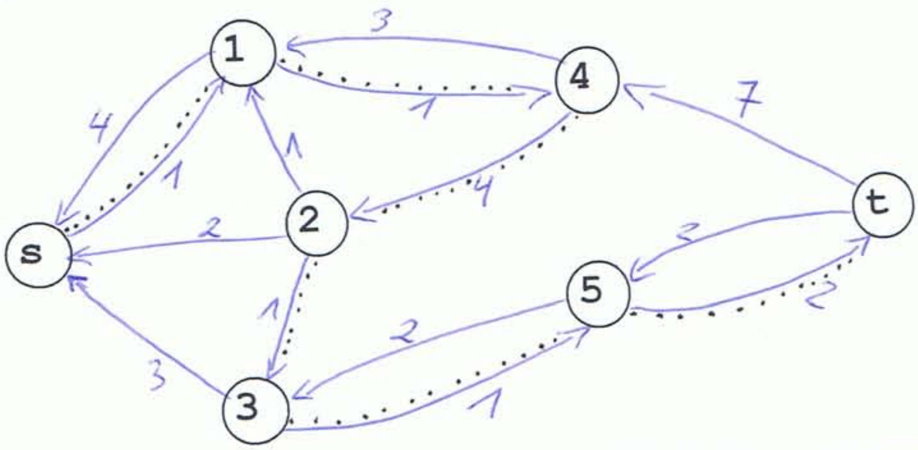
Die topologische Sortierung ist aufgrund des Kreises bei den Knoten c, d, g nicht möglich.

7. Aufgabe (8 Punkte)

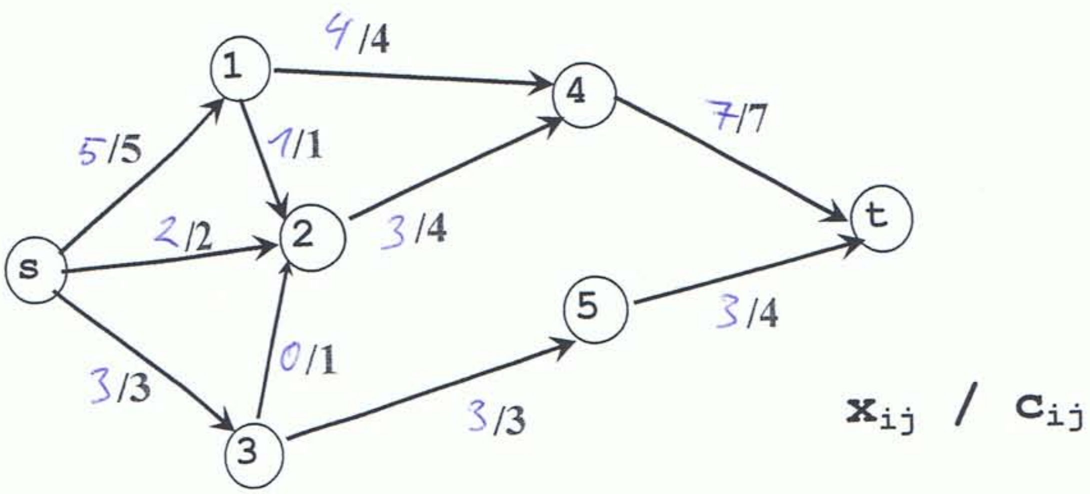
Betrachten Sie nachfolgendes s-t-Netzwerk mit einem zulässigen Fluss **x**:



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



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



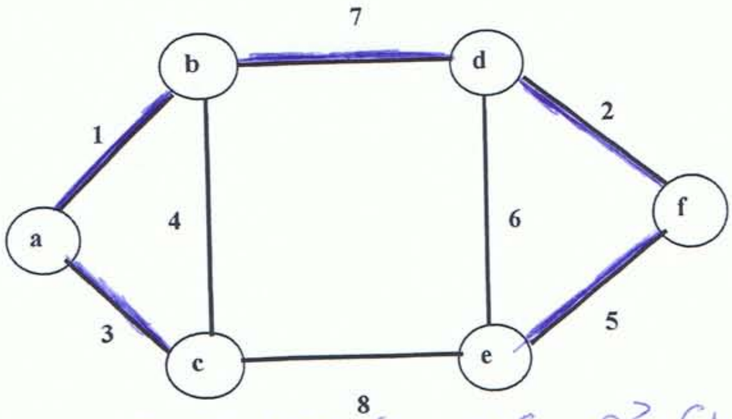
8. Aufgabe (8 Punkte) 8

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

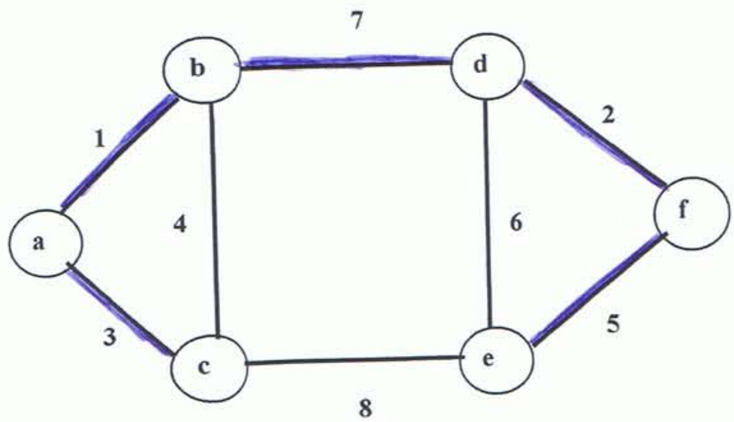
Verwenden Sie dazu

a) Kuskals Algorithmus
b) Prims Algorithmus; starten Sie dabei bei Knoten `f`.

Geben Sie jeweils die Reihenfolge an, in der die Kanten des minimal spannenden Baumes ausgewählt werden und zeichnen Sie den minimal spannenden Baum in dem Graphen ein.



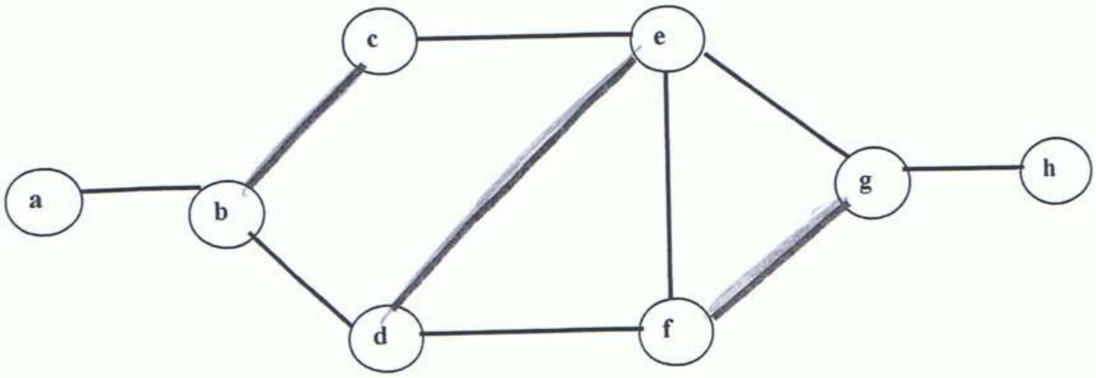
Kruskal: { {a,b}, {d,f}, {a,c}, {e,f}, {b,d} } ✓



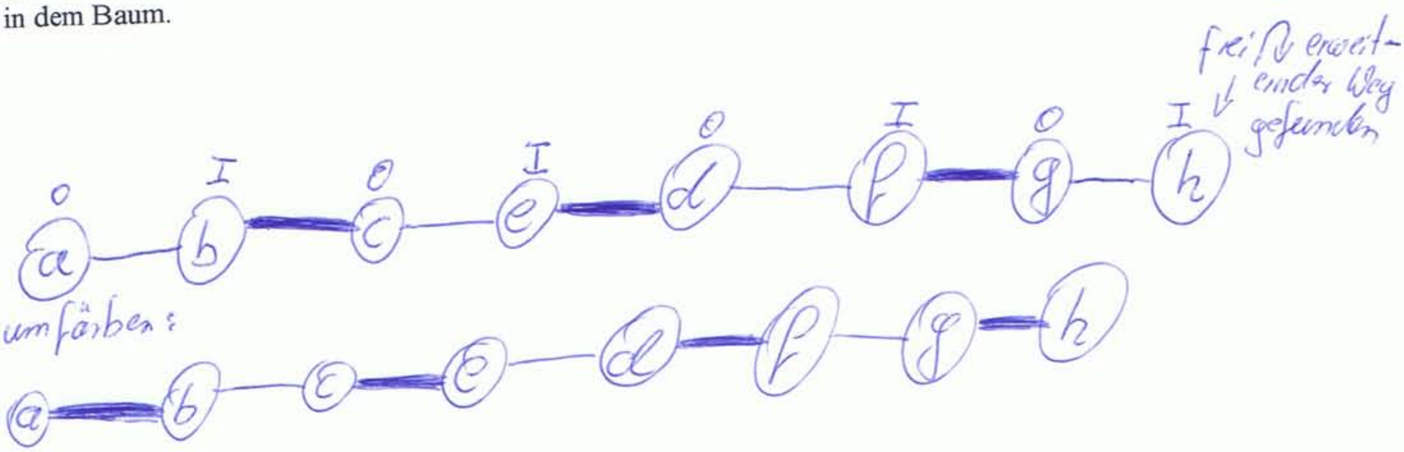
Prim: { {d,f}, {e,f}, {b,d}, {a,b}, {a,c} } ✓

9. Aufgabe (6 Punkte) 6

Betrachten Sie nachfolgenden Graph mit Matching $M = \{ \{b,c\}, \{d,e\}, \{f,g\} \}$:

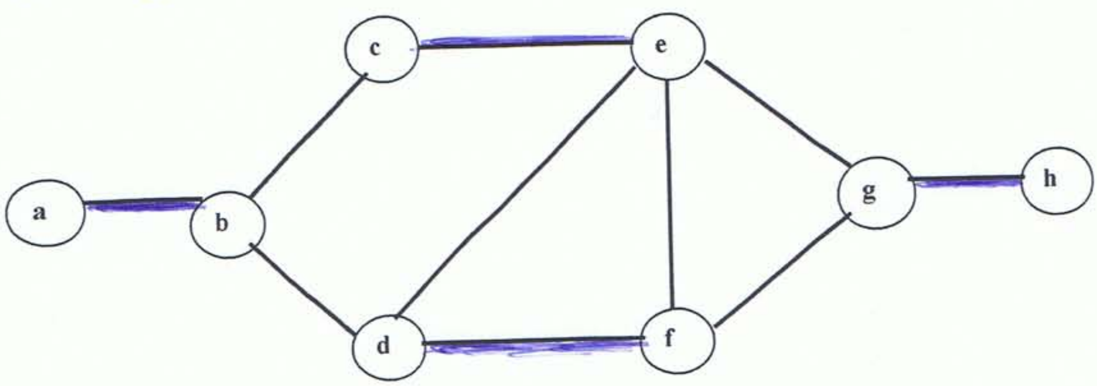


Ermitteln Sie nun einen alternierenden erweiternden Baum / Weg, nach der Methode „AlternierendenBaum". Geben Sie dabei auch die Label der Knoten an, sowie einen erweiternden Weg in dem Baum.



Geben Sie nun in nachfolgendem Graph das neue Matching nach Umfärben des erweiternden Wegs an:

$\mathcal{M} = \{ \{a,b\}, \{c,e\}, \{d,f\}, \{g,h\} \}$



✓

5,5

10. Aufgabe (6 Punkte)
Formulieren Sie eine Operation `delete` in einem Heap in Pseudo-Code. Diese Operation soll einen Knoten an einer beliebigen Position in einem Heap entfernen und dabei die Heap-Eigenschaft erhalten. Diese Operation soll möglichst effizient durchgeführt werden und die Korrektheit ist zu begründen.. Der Pseudo-Code soll dabei so genau formuliert werden, dass die grundsätzliche Vorgehensweise klar ist.
Geben Sie die Laufzeit dieser Operation an.

```
delete(int pos, []arr){
*  arr[pos] = arr[arr.length-1] // letztes Blatt an die
                                   Stelle wo gelöscht werden soll
   arr[arr.length-1] = null // nun doppeltes Blatt entfernen
   heapify(arr, arr[pos], arr.length-1);
}
```

*: if(pos ≤ arr.length)

Korrektes Programm auf der Rückseite!

Begründung: Das zu löschende Element wird so gesehen mit dem letzten Element des Heap ausgetauscht und dann „gelöscht". So ist es weiterhin ein (fast) vollständiger Baum. ~~Nun wird heapify auf das ausgetauschte Element angewand (ohne das nun letzte Element)~~ um die ~~Heapeigenschaft wieder her zu stellen.~~

~~Die Laufzeit ist hier nur von heapify maßgelich abhängig, welches im „worst-case" einmal durch alle Ebenen des Heaps muss, also liegt sie in O(log n)~~

Anmerkung auf der Rückseite

Um das Array konsistent zu halten muss es neu erstellt werden (mit einem Element weniger); Hierfür mussen alle Elemente bis auf das letzte kopiert werden, sprich eine Laufzeit $\underline{O(n)}$! (✓)

Arraykopie ist aber nicht unbedingt notwendig, wenn man einen Bereich für heapify angibt wie Sie es getan haben

korrekt:

```
delete(arr, pos){
  if(pos < arr.length){
    arr[pos] = arr[arr.length-1];
    heapify(arr, pos, arr.length-1);
    erstelle tmpArray: Elemente
    für(Anzahl ~~Elemente~~ von arr -1)
      tmpArray[i] = arr[i]
    arr = tmpArray; // Zeiger von arr auf das neue, kleinere
                       Array setzen
}
```

(✓)

Bedeutung von diesem Parameter hätte erläutert werden müssen! (Ist es die # der Elemente oder der größte Index?)

−0,5

Element an zu löschender Stelle mit letztem aus Heap überschreiben. Neues Array ohne das letzte (doppelte) Element anlegen. Heapify anwenden um Array wieder mit Heap-Eigenschaft zu belegen.