NAME: VORNAME:

MATRIKELNUMMER:

RWTH Aachen **SS 2016**
Lehrgebiet Theoretische Informatik 2. Klausur (Gruppe A)
Rossmanith–Hensel–Kuinke–Sánchez 19. September 2016

Klausur zur Vorlesung Datenstrukturen und Algorithmen

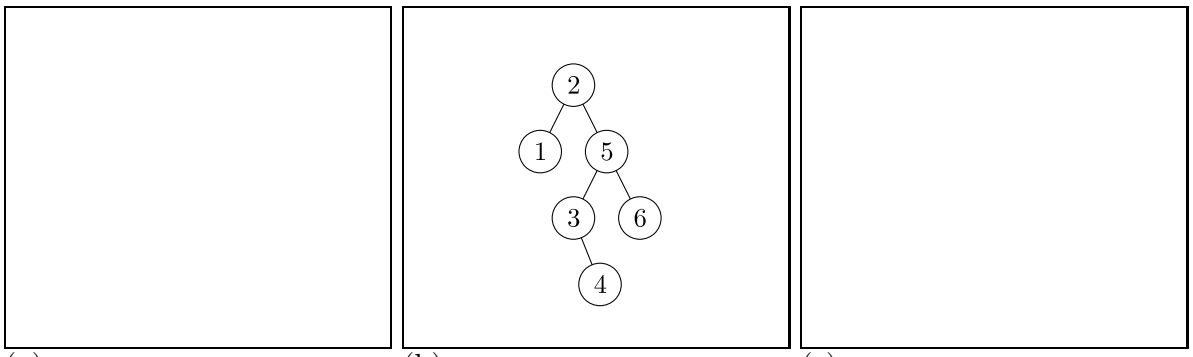
Hinweise zu allen Aufgaben: Tragen Sie die Ergebnisse auf dem Aufgabenblatt erst dann ein, wenn Sie mit der Aufgabe fertig sind und den Lösungsweg noch einmal überprüft haben. Wenn das Ergebnis falsch ist, kann gegebenenfalls der Lösungsansatz bewertet werden. Kennzeichnen Sie daher auf den Blankoblättern jeden Lösungsweg eindeutig mit dem jeweiligen Aufgabennummern und -buchstaben. Andernfalls kann der Lösungsansatz nicht berücksichtigt werden. Schreiben Sie deutlich, besonders in den Ergebniskästen; unleserliches wird nicht gewertet! Viel Erfolg!

Aufgabe 1 (14 Punkte)

a) Wie sieht ein anfangs leerer Splaybaum aus, wenn die Schlüssel 2, 3, 4, 1, 5, 6 in dieser Reihenfolge eingefügt werden?

b) Sehen Sie sich diesen Splaybaum an (0 Punkte).

c) Löschen Sie den Schlüssel 4 aus dem Splaybaum in b).

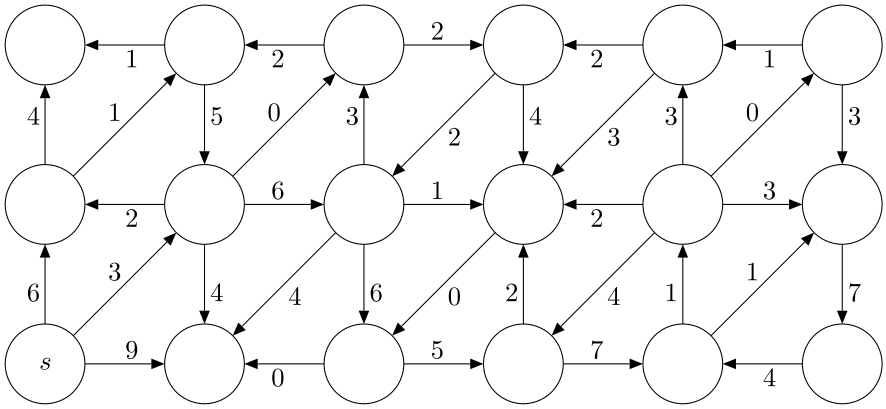


(a) (b) (c)

Schreiben Sie alle Zwischenschritte nieder und tragen Sie die Endergebnisse der Teilaufgaben am Ende in die obigen Kästen ein.

Aufgabe 2 (15 Punkte)

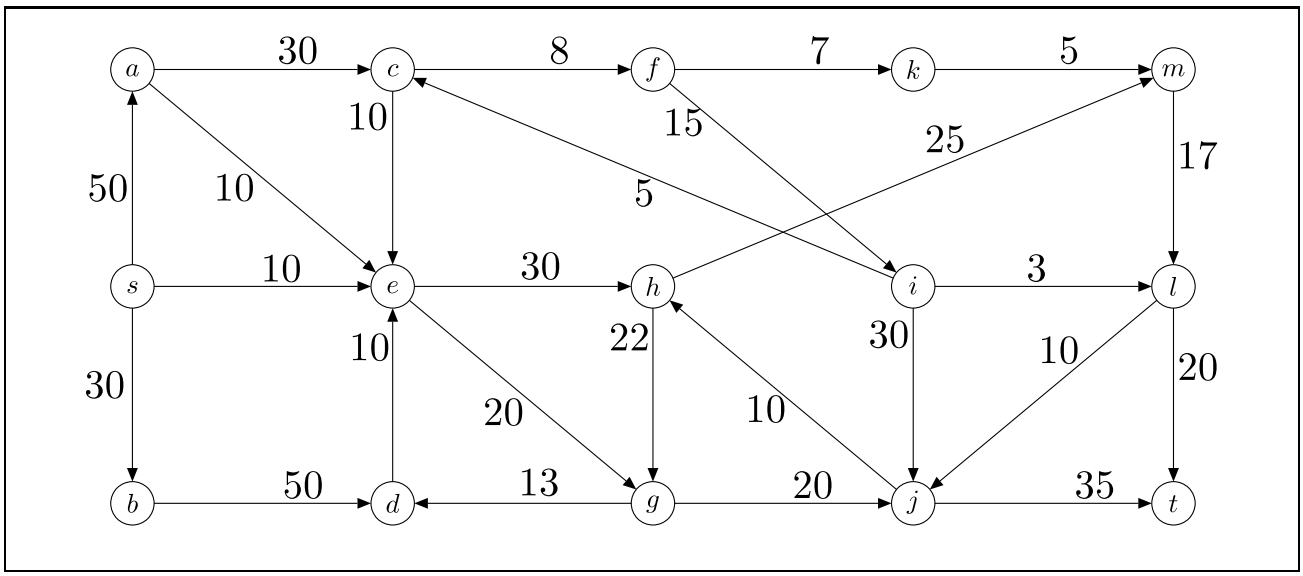
a) Wenden Sie den Algorithmus von Dijkstra auf den folgenden Graphen an.



Tragen Sie am Ende in die Zeichnung in jeden Knoten seinen Abstand zu s ein.

b) Aus wie vielen Knoten besteht die größte starke Zusammenhangskomponente?

Aufgabe 3 (15 Punkte)
Es sei folgendes Flußnetzwerk gegeben:



a) Was ist die Kapazität des Schnitts $(\{s, a, b, c, d, e, f, h, g\}, \{k, i, j, m, l, t\})$? ☐

b) Finden Sie einen maximalen Fluß und zeichnen Sie diesen in das Netzwerk ein.

c) Was ist der Wert Ihres Flußes? ☐

d) Geben Sie einen Schnitt mit minimaler Kapazität an:
$($☐, ☐$)$.

e) Wie groß ist die Kapazität dieses Schnitts? ☐

Aufgabe 4 (22 Punkte)
Hinweis: Beschreiben Sie alle Algorithmen und geben Sie diese *nicht* nur als (Pseudo-) code an. Es ist immer nach worst-case Laufzeiten gefragt.

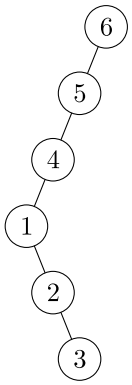
a) Es seien ein Array A der Länge n und ein Array B der Länge m gegeben. Sie enthalten jeweils keine Zahlen doppelt, aber die gleiche Zahl kann in A und B vorkommen. Entwerfen Sie einen Algorithmus der in $O((n+m)\log n)$ Schritten ein Array C ausgibt, das alle Elemente von A enthält, die *nicht* auch in B vorkommen. Begründen Sie die Laufzeit Ihres Algorithmus.

b) Es seien ein Array A der Länge n und eine natürliche Zahl $k \geq 1$ gegeben. Die Zahlen in A sind alle paarweise verschieden. Entwerfen Sie einen Algorithmus der in $O(n \log k)$ Schritten die kleinsten k Zahlen von A in aufsteigender Reihenfolge ausgibt. Begründen Sie die Laufzeit Ihres Algorithmus.

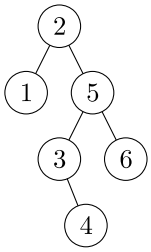
Lösung Klausur A

Aufgabe 1

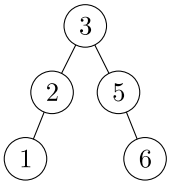
a)



b)

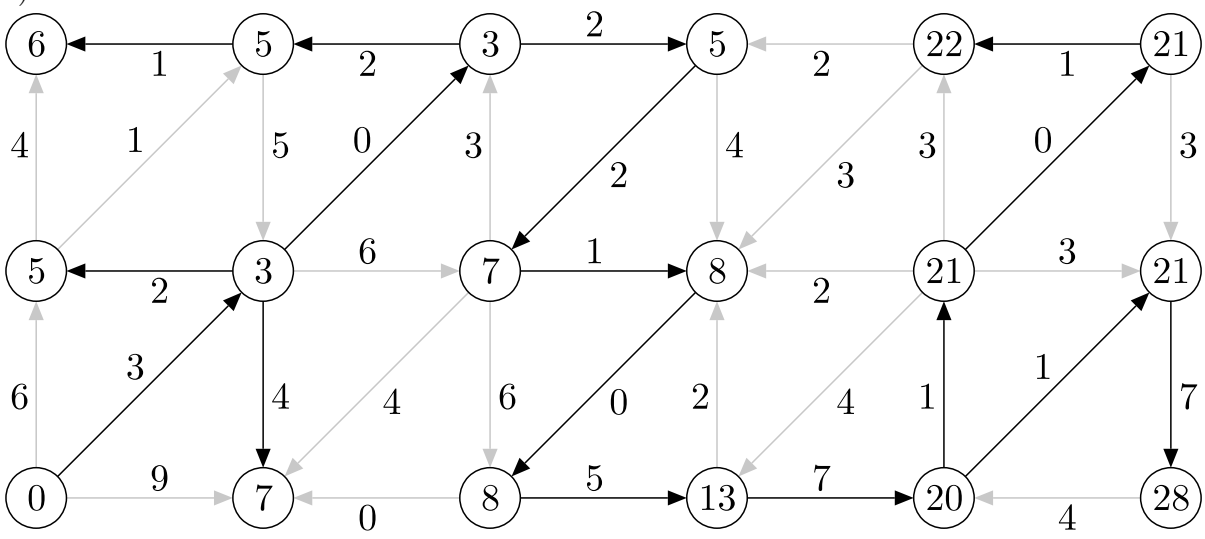


c)



Aufgabe 2

a)

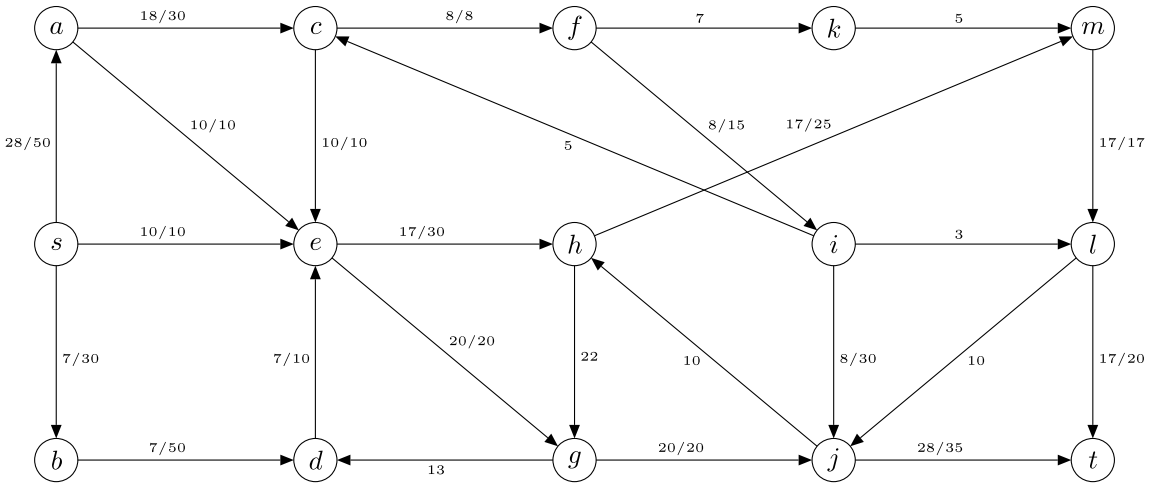


b) 15

Aufgabe 3

a) 67

b)



c) 45

d) $\{s, a, b, c, d, e, g, h, m\}$, $\{f, k, i, j, l, t\}$

e) 45

Aufgabe 4

Wir zeigen verschiedene Lösungsansätze, die alle gleich richtig sind.

a)

1. Wir nehmen eine Datenstruktur T in der man in logarithmischer Zeit einfügen und löschen kann, wie z.B. einen AVL-Baum. Zuerst fügen wir alle Element von A in T ein. Das braucht Zeit $O(n \log n)$. Dann löschen wir alle Elemente in B aus T, was Zeit $O(m \log n)$ braucht. Am Ende geben wir alle Elemente aus, die noch in T sind. Da T maximal n Element enthält, können wir das in $O(n)$ Zeit machen. Also ist die gesamte Laufzeit:

$$O(n \log n) + O(m \log n) + O(n) = O((n + m) \log n)$$

2. Wir sortieren das Array A mit z.B. Heapsort in $O(n \log n)$. Dann führen wir für jedes Element in B eine binäre Suche in A aus, was Zeit $O(m \log n)$ braucht. Falls wir das Element finden makieren wir es als gefunden (z.B. in einem zweiten booleschen Array). Am Ende geben wir alle unmakierten Elemente von A in $O(n)$ aus. Die Gesamtaufzeit beträgt also:

$$O(n \log n) + O(m \log n) + O(n) = O((n + m) \log n)$$

3. Wir benutzen eine Hashtabelle und fügen alle Elemente von B in $O(m)$ Zeit ein. Dann iterieren wir über alle Elemente aus A und überprüfen für jedes in $O(1)$ ob es in der Hashtabelle enthalten ist, falls es nicht enthalten ist geben wir es aus. Die Laufzeit beträgt $O(n + m)$.

b) Beachte: Für die Laufzeitbetrachtung gilt $k \leq n$, da es nicht mehr als n Elemente gibt und daher eine obere Schranke für alle Datenstrukturen ist.

1. Wir nehmen eine anfangs leere Datenstruktur T in der wir in logarithmischer Zeit einfügen, so wie das größte Element finden und löschen können. z.B. einen AVL-Baum oder einen max-Heap. Wir fügen die ersten $k+1$ Elemente aus A in T ein. Jede dieser $k+1$ Einfügeoperationen braucht Zeit $O(\log k)$. Da T ein AVL-Baum ist, hat er Tiefe $O(\log k)$. Wir können also das größte Element in T in $O(\log k)$ finden, indem man von der Wurzel aus immer das rechte Kind besucht, bis es keines mehr gibt. Man kann dieses maximale Element dann in $O(\log k)$ Zeit löschen.

   Wir fügen dann immer das nächste Element aus A in T ein, finden und löschen das größte Element, bis wir das für die restlichen $n-(k+1)$ Elemente in A gemacht haben.

   T enthält jetzt die k kleinsten Elemente aus A. Da T ein Suchbaum mit k Elementen ist, können wir in $O(k)$ Zeit alle Elemente in T in aufsteigender Reihenfolge ausgeben.

   Die Gesamtlaufzeit ist somit:

$$(k+1) \cdot O(\log k) + (n-(k+1)) \cdot 2O(\log k) + O(k) = O(n \log k)$$

2. Wir suchen mit Quickselect das k-t größte Element von A in $O(n)$. Dann gehen wir in $O(n)$ über das Array und speichern alle Elemente die kleiner gleich diesem sind. Dadurch erhalten wir die k kleinsten Elemente die wir in $O(k \log k)$ mit z.B. Heapsort sortieren können. Die Gesamtlaufzeit beträgt somit $O(n + k \log k) = O(n \log k)$.