Name:	
Vorname:	
Matrikelnummer:	

Klausur-ID:

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders

13.3.2015

Klausur Algorithmen I

Aufgabe 1.	Kleinaufgaben	16 Punkte
Aufgabe 2.	Alle Wege	10 Punkte
Aufgabe 3.	Maximum-Subarray Problem	8 Punkte
Aufgabe 4.	Intervall-Graphen	13 Punkte
Aufgabe 5.	Leitelement	6 Punkte
Aufgabe 6.	Dijkstras Algorithmus	7 Punkte

Bitte beachten Sie:

- Als Hilfsmittel ist nur **ein** DIN-A4-Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- **Schreiben** Sie auf **alle Blätter** der Klausur und Zusatzblätter Ihre **Klausur-ID**.
- Merken Sie sich Ihre **Klausur-ID** auf dem Aufkleber für den Notenaushang.
- Die Klausur enthält 16 Blätter.
- Die durch Übungsblätter gewonnenen Bonuspunkte werden erst nach Erreichen der Bestehensgrenze hinzugezählt. Die Anzahl Bonuspunkte entscheidet nicht über das Bestehen.

Aufgabe		1	2	3	4	5	6	Summe
max. Punkte		16	10	8	13	6	7	60
Punkte	EK							
	ZK							

Bonuspunkte:	Summe:	**Note:**

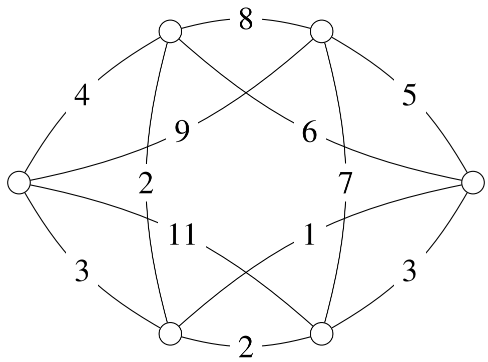
Aufgabe 1. Kleinaufgaben [16 Punkte]

a. Die Methoden einer Union-Find Datenstruktur seien wie folgt realisiert (erste Version aus der Vorlesung):

Function $find(i : 1..n)$
  if $parent[i] = i$ **then return** i
  else return $find(parent[i])$
Procedure $link(i, j : 1..n)$
  $parent[i] := j$
Procedure $union(i, j : 1..n)$
  if $find(i) \neq find(j)$ **then** $link(find(i), find(j))$

Welche worst-case Zeitkomplexität hat Kruskals Algorithmus mit dieser Union-Find Datenstruktur? [1 Punkt]

b. Markieren Sie in folgendem Graphen genau die Kanten, die zu einem *minimum spanning tree* (MST) gehören. [1 Punkt]



c. Zeigen oder widerlegen Sie, dass $\log_{\sqrt{n}} n = O(1)$ gilt. [1 Punkt]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

d. Zeigen oder widerlegen Sie, dass $\frac{\log_2 n}{\log_{n/2} n} = O(1)$ gilt. [2 Punkte]

e. Tragen Sie die folgenden Algorithmen und Datenstruktur-Operationen aus der Vorlesung entsprechend ihrer asymptotischer *worst-case Zeitkomplexität* in die Felder ein. [4 Punkte]

- *binarySearch()* auf einem Array mit festgelegter Größe n.
- *Dijkstra*s kürzeste Wege Algorithmus mit binärem Heap auf einem beliebigen ungerichteten Graphen mit n Knoten und $8n$ Kanten.
- *merge()* von zwei Listen der Länge n und $2n$.
- *last()* auf einer doppelt verketteten Liste der Länge n.
- *findComponents()* – Bestimmung aller Zusammenhangskomponenten in einem ungerichteten Graph mit $2n$ Knoten und n Kanten.
- *siftDown()* auf einem Heap mit n Elementen.
- *quickSort()* in der besten Variante aus der Vorlesung auf einem Array der Länge n.
- *insert()* in eine Hashtabelle mit linearer Suche.

$\Theta(1)$	
$\Theta(\log n)$	
$\Theta(n)$	
$\Theta(n \log n)$	
$\Theta(n^2)$	

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

f. Führen Sie least-significant-digit (LSD) Radixsort mit Dezimalziffern (Radix $K = 10$) auf dem folgenden Array durch. Tragen Sie nach jeder Runde das Array in eine der Schablonen ein, markieren Sie die *Bucketgrenzen der Ziffern* durch Trennstriche im Array und kennzeichnen Sie die zu bewertende Lösung deutlich. Rechenschritte zwischen den Arrays werden bei der Bewertung nicht berücksichtigt. [3 Punkte]

Eingabe:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
54	40	52	2	41	12	9	16	31	25	18	24	1	42	37	51

Bucket-Zähler zum Rechnen von Zwischenschritten (nicht bewertet):

0	1	2	3	4	5	6	7	8	9

Ergebnis nach der ☐ Ziffer:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bucket-Zähler zum Rechnen von Zwischenschritten (nicht bewertet):

0	1	2	3	4	5	6	7	8	9

Ergebnis nach der ☐ Ziffer:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Weitere Kopie, falls benötigt:

Bucket-Zähler zum Rechnen von Zwischenschritten (nicht bewertet):

0	1	2	3	4	5	6	7	8	9

Ergebnis nach der ☐ Ziffer:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

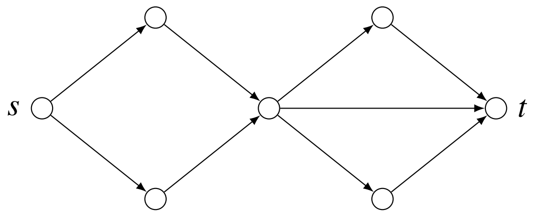
(weitere Teilaufgabe auf dem nächsten Blatt)

Fortsetzung von Aufgabe 1

g. Entwerfen Sie eine FIFO Queue mit Hilfe von zwei Stacks S_1 und S_2, so dass die Operationen *enqueue* und *dequeue* in amortisert konstanter Zeit laufen. Gehen Sie davon aus, dass sie in konstanter Zeit die Anzahl der Elemente eines Stacks abfragen können. Begründen Sie kurz warum die geforderte Laufzeit erreicht wird. [4 Punkte]

Aufgabe 2. Alle Wege [10 Punkte]

Betrachten Sie gerichtete azyklische zusammenhängende Graphen $G = (V, E)$ mit $V = \{1, 2, \ldots, n\}$.
a. Bestimmen Sie in folgendem Beispielgraph die Anzahl *aller* nicht notwendigerweise kantendisjunkter (paarweise verschiedener) Wege von s nach t. [1 Punkt]



Wege

b. Entwerfen Sie einen Algorithmus, der in $O(|V| + |E|)$ für zwei gegebene Knoten $s, t \in V$ die *Anzahl aller Wege* von s nach t bestimmt. Verwenden Sie hierbei das folgende Tiefensuchschema, und füllen Sie nur die unterstrichenen Prozeduren mit Pseudocode. [8 Punkte]

Tiefensuchschema für $G = (V, E)$ mit Startknoten $s \in V$
unmark all nodes
<u>**init**(s, t)</u>; mark s; DFS($\perp, s$)
`return` <u>**result()**</u>

`Procedure` DFS(u, v : NodeId)
    `foreach` $(v, w) \in E$ `do`
        `if` w is not marked `then`
            <u>**traverseTreeEdge**(v, w)</u>
            mark w; DFS(v, w)
        `else`
            <u>**traverseNonTreeEdge**(v, w)</u>
    `if` $u \neq \perp$ `then`
        <u>**backtrack**(u, v)</u>

`Procedure` **init**($s, t : V$):

`Procedure` **traverseTreeEdge**($v, w : V$):

(weitere Prozeduren und Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 2

`Procedure` **traverseNonTreeEdge**$(v, w : V)$:

`Procedure` **backtrack**$(u, v : V)$:

`Function` **result**$() : \mathbb{N}$:

Begründen Sie kurz die Korrektheit und Laufzeit Ihres Algorithmus.

c. Beschreiben Sie kurz wie Sie Ihren Algorithmus aus Teilaufgabe b) erweitern würden, falls nicht bekannt ist ob der gerichtete Graph einen Kreis enthält. Nehmen Sie dabei an, dass Kreise nur auf Pfaden von s zu t liegen können. [1 Punkt]

Aufgabe 3. Maximum-Subarray Problem [8 Punkte]

Gegeben sei ein Array mit n ganzen Zahlen. Beim Maximum-Subarray Problem wird die maximale Summe eines *zusammenhängenden* Teilarrays $A[i..j]$ gesucht.

Enthält das Array nur positive Zahlen so ist das Problem einfach: die maximale Summe ist die Summe über alle Elemente. Enthält das Array ausschließlich negative Zahlen, soll das Ergebnis 0 sein. Schwierig wird das Problem dann, wenn das Array sowohl positive, als auch negative Zahlen enthält.

a. Markieren Sie im nachfolgenden Array das Teilarray, dessen Summe maximal ist und tragen Sie die maximale Summe in das vorgesehene Feld ein. [1 Punkt]

1	2	3	4	5	6	7	8	9	10
-2	1	-3	9	-5	2	7	-5	4	-3

maximale Summe = $\boxed{}$

b. Der nachfolgende Algorithmus löst das Maximum-Subarray Problem:

Function maxSubArraySum(A : **Array** $[1..n]$ **of** $\mathbb{Z}$) : $\mathbb{Z}$
  maxSubArraySumRec$(A, 1, n)$

Function maxSubArraySumRec(A : **Array** $[1..n]$ **of** $\mathbb{Z}$, $l : \mathbb{Z}$, $r : \mathbb{Z}$) : $\mathbb{Z}$
  if $l > r$ **then return** 0
  if $l = r$ **then return** $\mathbf{max}(0, A[l])$
  $m := (l+r)/2$
  $l_{max} := 0, \quad sum := 0$
  for $i := m$ **downto** l **do**
    $sum := sum + A[i]$
    $l_{max} := \mathbf{max}(l_{max}, sum)$
  $r_{max} := 0, \quad sum := 0$
  for $i := m+1$ **to** r **do**
    $sum := sum + A[i]$
    $r_{max} := \mathbf{max}(r_{max}, sum)$
  $c_{max} := l_{max} + r_{max}$
  $a_{max} :=$ maxSubArraySumRec(A, l, m)
  $b_{max} :=$ maxSubArraySumRec$(A, m+1, r)$
  return $\mathbf{max}(a_{max}, \mathbf{max}(b_{max}, c_{max}))$

In dieser Teilaufgabe interessieren wir uns für die *asymptotische Anzahl* von $\mathbf{max}(\dots)$ Operationen, die ein Aufruf von maxSubArraySum(A) für ein Array A mit n ganzen Zahlen benötigt. Geben Sie hierfür eine Rekurrenz $T(n)$ an. Bestimmen und beweisen Sie für $T(n)$ eine geschlossene Form. Gehen Sie dabei davon aus, dass n eine Zweierpotenz ist. [3 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 3

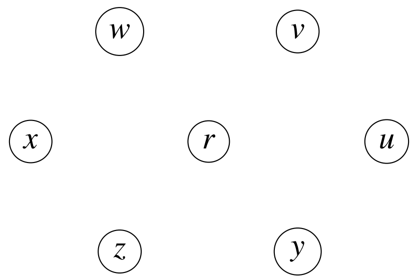
c. Entwerfen Sie einen Algorithmus, der das Maximum-Subarray Problem in Zeit $\mathrm{O}(n)$ löst.

[4 Punkte]

Aufgabe 4. Intervall-Graphen [13 Punkte]

In der Vorlesung wurden Intervall-Graphen definiert als Graphen, deren Knoten Intervalle $[a,b] \subseteq \mathbb{R}$ sind, und zwischen zwei Knoten genau dann eine Kanten existiert, wenn sich die Intervalle überlappen. Formal: $G = (V,E)$ mit
$V = \{[a_1,b_1],\ldots,[a_n,b_n]\}$
$E = \{\{[a_i,b_i],[a_j,b_j]\} \mid [a_i,b_i] \cap [a_j,b_j] \neq \emptyset\}$

a. Gegeben sei folgender Intervall-Graph: $V = \{u,v,w,x,y,z,r\}$ mit $u = [-42,0]$, $v = [0,0]$, $w = [-1,1]$, $x = [\frac{2}{3},\frac{3}{2}]$, $y = [1.5,3]$, $z = [1.\overline{1},\pi]$ und $r = [-\infty,\infty]$. Zeichnen Sie die Kanten in den folgenden Graphen ein. [2 Punkte]



b. Eine *Clique* in einem Graphen $G = (V,E)$ ist eine Teilmenge $U \subseteq V$, in der jeder Knoten mit jedem Knoten verbunden ist. Eine *größte Clique* ist eine solche Teilmenge maximaler Kardinalität, das heißt es gibt keine größere Knotenteilmenge mit dieser Eigenschaft. Finden Sie *eine* größte Clique in dem oben angegebenen Graphen und markieren Sie die dazugehörigen Knoten deutlich. [1 Punkt]

c. Skizzieren Sie einen Algorithmus, der in $\mathrm{O}(|V|\log|V|)$ die Kardinalität einer größten Clique in einem Intervall-Graphen findet. [2 Punkte]

d. Zeichnen Sie einen Graphen, der sich nicht als Intervall-Graph darstellen lässt. [1 Punkt]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 4

e. Sei $G = (V,E)$ ein Graph. Eine *Sehne* in einem Kreis $C \subseteq V$ ist eine Kante s, die zwei Knoten des Kreises verbindet, ohne auf diesem Kreis zu liegen. Ein Kreis ist *sehnenfrei*, wenn er keine Sehne enthält. Beweisen oder widerlegen Sie: Jeder sehnenfreie Kreis in einem Intervall-Graphen enthält nicht mehr als drei Knoten. Hinweis: Stellen Sie Intervalle als Teilstrecken der Zahlengerade dar. [4 Punkte]

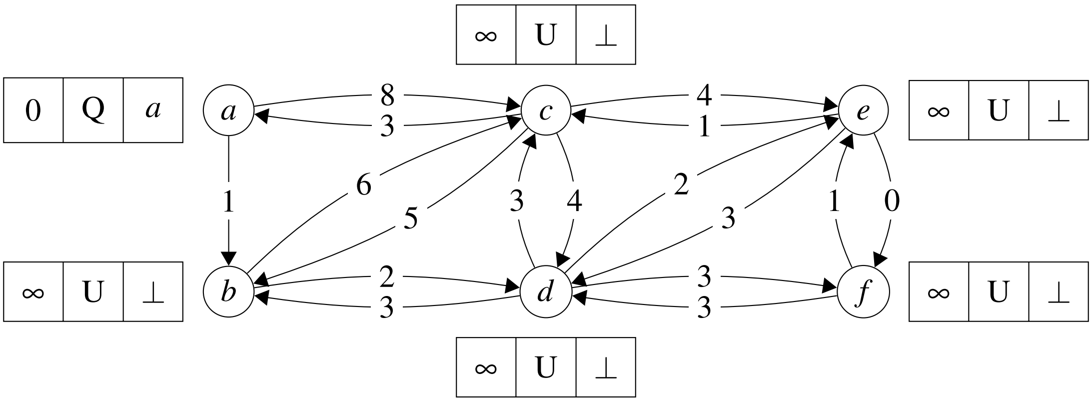
f. Das angesehene Hotel „Moselschlösschen" hat für die nächsten Monate n Buchungsanfragen erhalten. Diese liegen als Paare (b_i, e_i), $i \in \{1, n\}$ vor, die den jeweiligen An- und Abreisezeitpunkt festlegen. Der Manager des Hotels möchte aus Kostengründen für die Buchungen möglichst wenige Zimmer bereitstellen. Dank eines gewieften Studenten (der Teilaufgabe c gelöst hat) weiß er bereits, dass k Zimmer ausreichend sind. Jetzt muss er den Buchungen Zimmer zuordnen. Helfen Sie ihm, indem Sie einen Algorithmus mit Laufzeit $O(n \log n)$ skizzieren, der jeder Buchung genau ein Zimmer zuordnet, ohne ein Zimmer doppelt zu belegen. Sie können davon ausgehen, dass alle Zimmer gleichwertig sind. [3 Punkte]

Aufgabe 5. Leitelement [6 Punkte]

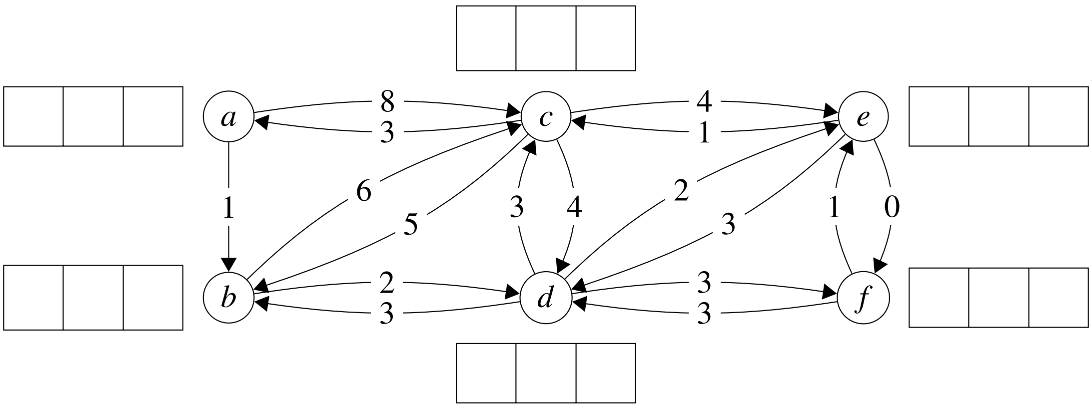
Gegeben sei ein Array A, das n Elemente enthält. Ein Element m nennt man das *Leitelement* von A, wenn es mindestens $\lfloor \frac{n}{2} + 1 \rfloor$ mal in A vorkommt. Skizzieren Sie einen Algorithmus, der prüft ob es ein Leitelement gibt und dieses gegebenenfalls bestimmt. Sie dürfen annehmen, dass die Elemente in $\mathrm{O}(1)$ Zeit vergleichbar und hashbar sind. Begründen Sie die Korrektheit, Laufzeit und den Platzverbrauch Ihrer Lösung.
Lösungen (samt Begründung) mit *deterministisch* $\mathrm{O}(n)$ Zeit und $\mathrm{O}(1)$ zusätzlichem Platz erhalten höchsten 6 Punkte, mit *erwartet* $\mathrm{O}(n)$ Zeit und $\mathrm{o}(n)$ Platz erhalten höchstens 4 Punkte, alle anderen Lösungen erhalten höchsten 2 Punkte.

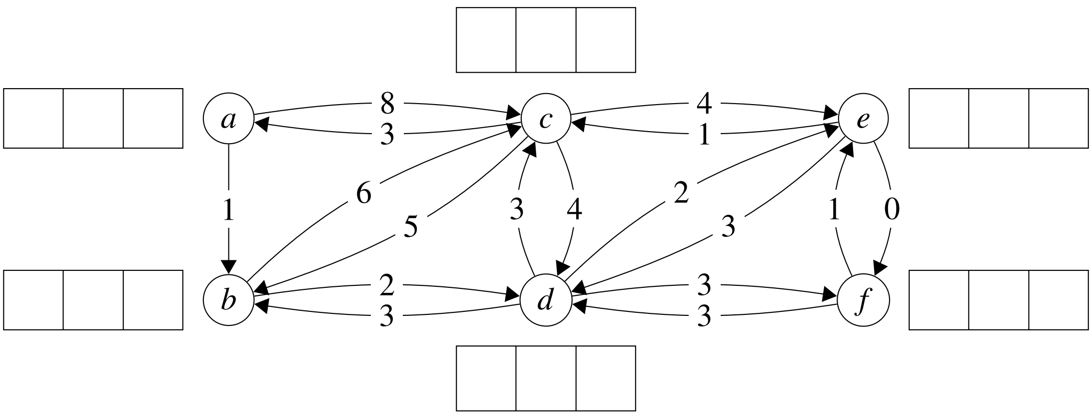
Aufgabe 6. Dijkstras Algorithmus [7 Punkte]

Gegeben sei der unten abgebildete gerichtete Graph $G = (V, E)$ mit Kantengewichten.



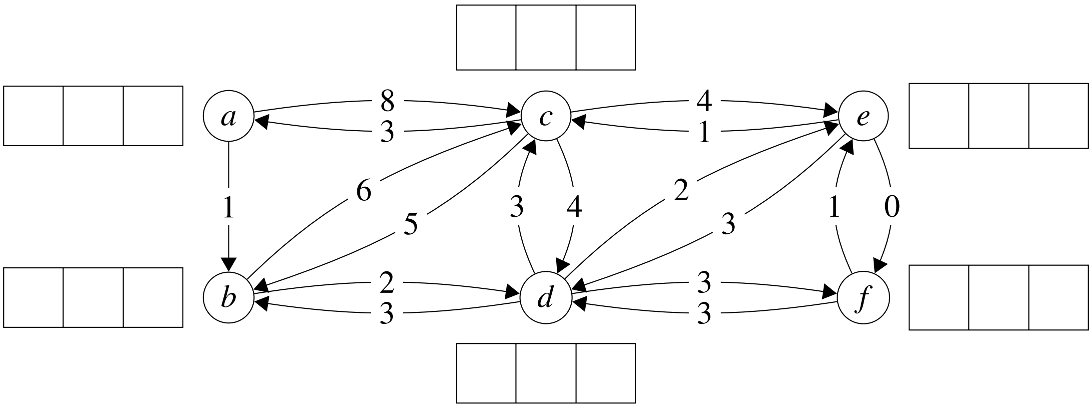
a. Führen Sie auf dem obigen Graphen den Algorithmus von Dijkstra mit Startknoten a durch. Geben Sie nach jedem Schleifendurchlauf für alle Knoten in den drei Kästchen die aktuelle Distanz, den Status des Knotens und den "Parent"-Knoten an. Bezeichnen Sie den Status mit einem 'Q', 'S' oder 'U', je nachdem ob der Knoten in der **Q**ueue ist, bereits **S**canned wurde oder noch **U**nerreicht ist. Verwenden Sie dazu die untenstehenden Kopien des Graphens. Die Initialisierung ist im obigen Graphen bereits vorgegeben. Beschriften Sie deutlich die Graphen, die gewertet werden sollen. Geben Sie gegebenenfalls die Schleifendurchlaufsnummern an. Sie erhalten weitere Blätter mit Graphen bei Bedarf von der Aufsicht. [4 Punkte]

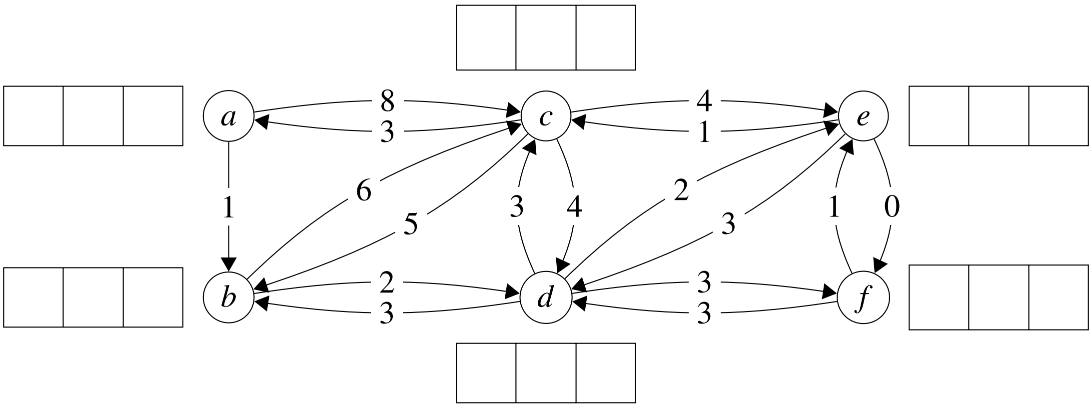


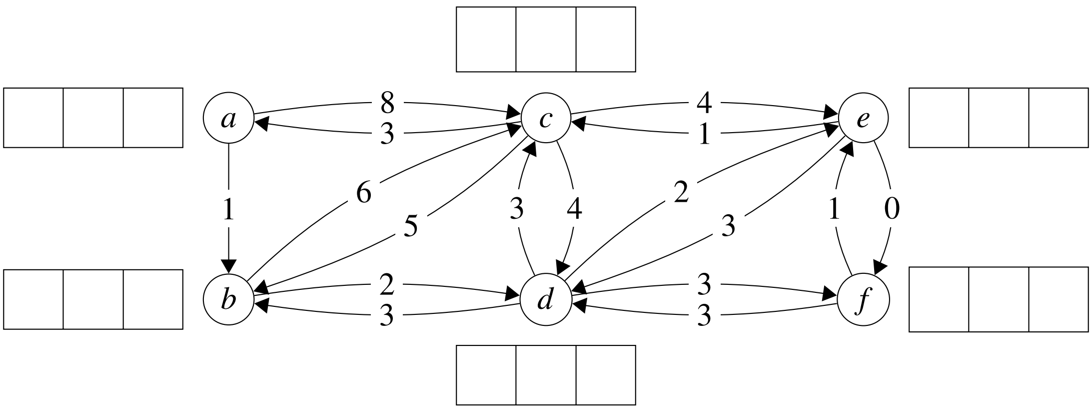


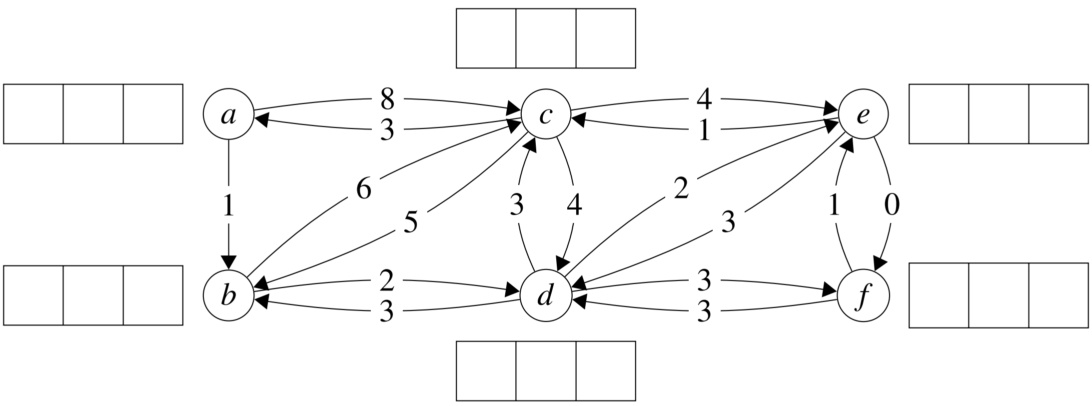
(weitere Graphen auf dem nächsten Blatt)

Fortsetzung von Aufgabe 6



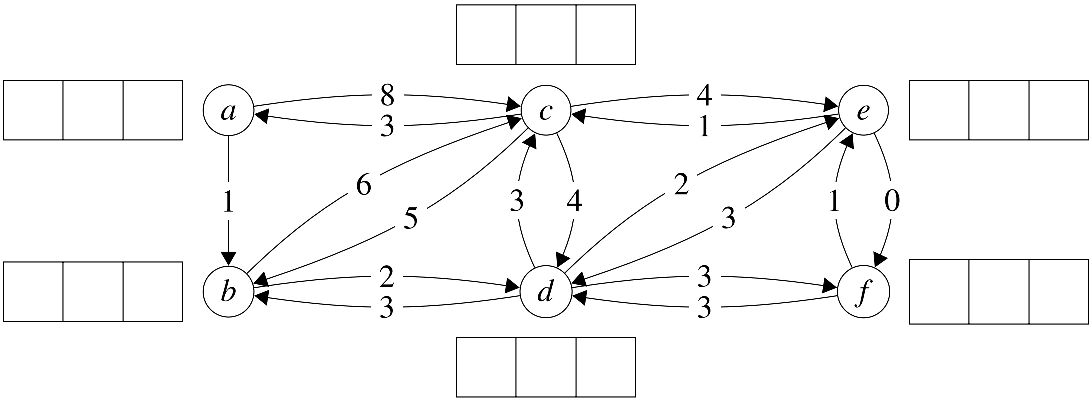




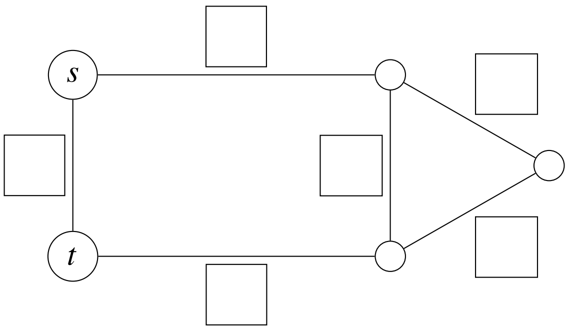


(weitere Graphen und Teilaufgaben auf dem nächsten Blatt)

Fortsetzung von Aufgabe 6



b. Zeichnen Sie in folgendem Graphen mit festgelegtem Startknoten s und Zielknoten t *Kantenrichtungen und Gewichte* ein, so dass der Graph einen negativen Zyklus enthält und Dijkstras Algorithmus beim Scannen von t mit korrekter Distanz dennoch terminiert werden kann. [1 Punkt]



c. Welche Eigenschaft eines Problems wird bei dem Prinizip der dynamischen Programmierung ausgenutzt? [1 Punkt]

d. Nennen Sie drei Algorithmen, die auf dynamischer Programmierung basieren. [1 Punkt]

Konzeptpapier (Abgabe freiwillig)