Name:

Vorname:

Matrikelnummer:

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders

ID 5

30.7.2013

Klausur Algorithmen I

Aufgabe 1.	Kleinaufgaben	15 Punkte
Aufgabe 2.	Dijkstras Algorithmus	10 Punkte
Aufgabe 3.	Zusammenhang	10 Punkte
Aufgabe 4.	Polyhalmadrom	10 Punkte
Aufgabe 5.	Minimale Spannbäume	8 Punkte
Aufgabe 6.	Hashing	7 Punkte

Bitte beachten Sie:

- Als Hilfsmittel ist nur **ein** DIN-A4-Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- **Schreiben** Sie auf **alle** Blätter Ihren Namen und Ihre Matrikelnummer.
- Merken Sie sich Ihre Klausur-ID 5 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		15	10	10	10	8	7	60
Punkte	EK							
	ZK							

Bonuspunkte:	Summe:	**Note:**

Aufgabe 1. Kleinaufgaben [15 Punkte]

a. Gegeben sind Teile der Union-Find Datenstruktur aus der Vorlesung. Fügen Sie Zeilen in den unten dargestellten Code ein, so dass die Union bzw. Link Operation immer den kleineren Baum (damit ist die Anzahl der Elemente der assoziierten Menge gemeint) an den größeren Baum anhängt.
Hinweis: dieses Vorgehen bezeichnet man auch als Union-by-Size.

[3 Punkte]

```
Class UnionFind(n : ℕ)
    parent=⟨1,2,...,n⟩ : Array [1..n] of 1..n

    Function find(i : 1..n) : 1..n
        if parent[i] = i then return i
        else return find(parent[i])

    Procedure link(i, j : 1..n)
        assert i und j sind Repräsentanten von verschiedenen Mengen

    Procedure union(i, j : 1..n)
        if find(i) ≠ find(j) then link(find(i), find(j))
```

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

b. Ein gerichteter Graph $G = (V, E)$ ist durch folgendes Adjazenzarray abgespeichert. Zeichnen Sie den Graphen und nummerieren Sie die Knoten in Ihrer Zeichnung. [2 Punkte]

	1	2	3	4	5	6	7	8
V	1	2	4	5	8	10	10	11

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

c. Lösen Sie die beiden folgenden Rekurrenzen im Θ-Kalkül:

$$U(n) = n + 20U(n/4), \qquad U(1) = 5$$
$$V(n) = 3n + V(n/4), \qquad V(1) = 42$$

mit $n = 4^k$ und $k \in \mathbb{N}_{>0}$. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

d. Zeigen Sie oder widerlegen Sie, dass $f(n) + g(n) = \mathrm{O}(g(f(n)))$ gilt. [2 Punkte]

e. Zeigen oder widerlegen Sie für jedes $k \in \mathbb{N}_+$, dass $\sum_{i=0}^{k} c_i n^i = \Theta(n^k)$ für $c_i \in \mathbb{N}, c_k \neq 0$ gilt. [2 Punkte]

(weitere Teilaufgaben auf dem nächsten Blatt)

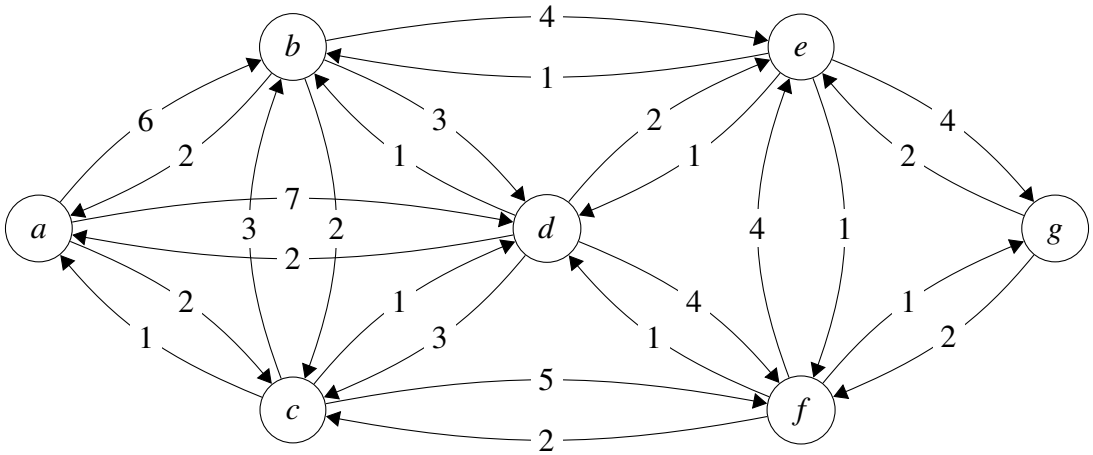
Fortsetzung von Aufgabe 1

f. Warum benötigt vergleichsbasiertes Sortieren im schlimmsten Fall mindestens $\Omega(n \log n)$ Vergleiche? Begründen Sie kurz. [2 Punkte]

g. Ordnen Sie die folgenden fünf Sortieralgorithmen aufsteigend nach worst-case Komplexität: Heap-Sort, Merge-Sort, Quick-Sort, Insertion-Sort und Bucket-Sort für Schlüsselwerte in $O(n)$. [2 Punkte]

Aufgabe 2. Dijkstras Algorithmus [10 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. Verwenden Sie für jeden Schleifendurchlauf eine Zeile in folgender Tabelle und tragen Sie die Werte der dabei auftretenden Variablen ein. Die ersten beiden Zeilen sind bereits vorgegeben.

Die Zellen in Spalte Q sollen *alle in* der Queue gespeicherten Knoten v als Paare (p, v) mit Priorität p in Reihenfolge enthalten. Bei $d[\cdot]$ und $parent[\cdot]$ brauchen Sie nur *geänderte Werte* notieren. Wird ein Knoten v vom Algorithmus „*scanned*“ („*settled*“, „*set*“), so tragen Sie in $d[v]$ ein „s“ ein. [6 Punkte]

	geänderte Variablen $d[\cdot]$							geänderte Variablen $parent[\cdot]$						
vollständiger Inhalt Q	a	b	c	d	e	f	g	a	b	c	d	e	f	g
Initialisierung	0	∞	∞	∞	∞	∞	∞	a	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$(0, a)$	s	6	2	7					a	a	a			

Fortsetzung von Aufgabe 2

	geänderte Variablen $d[\cdot]$							geänderte Variablen $parent[\cdot]$						
vollständiger Inhalt Q	a	b	c	d	e	f	g	a	b	c	d	e	f	g

Bei Bedarf erhalten Sie von der Aufsicht weitere Blätter mit Tabellen.

b. Was ist die asymptotische worst-case Laufzeit von Dijkstras Algorithmus im O-Kalkül in Abhängigkeit von Knoten- und Kantenzahl, wenn ein binärer Heap verwendet wird? [1 Punkt]

c. Der Algorithmus von Dijkstra funktioniert nur für nicht-negative Kantengewichte. Es ist aber sicherlich möglich einen Graphen mit negativen Kantengewichten durch Addition einer Konstanten c zu allen Kantengewichten in einen Graph mit nicht-negativen Kantengewichten zu transformieren.

Wieso kann man den Algorithmus von Dijkstra mit dieser Technik nicht auf beliebige Kantengewichte verallgemeinern? Geben Sie ein Gegenbeispiel ohne negative Kreise an. [3 Punkte]

Aufgabe 3. Zusammenhang [10 Punkte]

Gegeben sei ein ungerichteter, zusammenhängender Graph $G = (V, E)$ und eine Zahl $k > 1$. Das Graphpartitionierungsproblem möchte eine Aufteilung der Knotenmenge V in k Teilmengen (Blöcke) $V_1, \ldots, V_k$ finden, die folgende Bedingungen erfüllt:

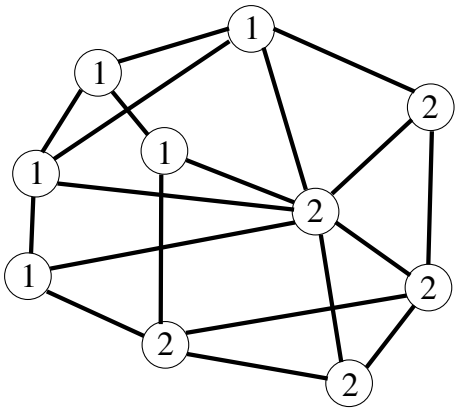
1. $\forall i \in \{1, \ldots, k\} : V_i \neq \emptyset$ – keine leeren Teilmengen
2. $\forall i, j \in \{1, \ldots, k\} : i \neq j \Rightarrow V_i \cap V_j = \emptyset$ – keine Überlappungen
3. $\bigcup_{i=1}^{k} V_i = V$ – volle Überdeckung
4. $\forall i \in \{1, \ldots, k\} : |V_i| \leq \lceil \frac{|V|}{k} \rceil$ – Balancebedingung

Die Zielfunktion ist häufig die Minimierung der Anzahl Schnittkanten $|\mathscr{E}|$ mit

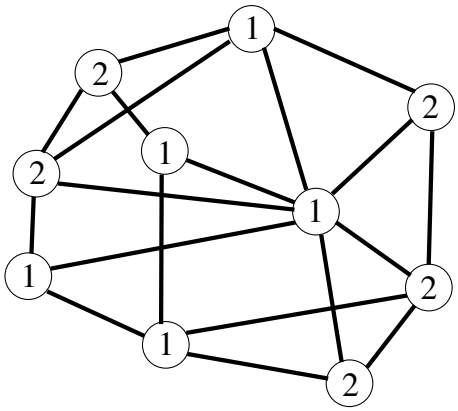
$$\mathscr{E} = \{\{u, v\} \in E \mid u \in V_i, v \in V_j, i \neq j\}.$$

In der Praxis werden aus verschiedenen Gründen häufig **zusammenhängende** Blöcke benötigt. Ein Block V_i heißt zusammenhängend, wenn der knoteninduzierte Teilgraph $G[V_i]$[1] zusammenhängend ist. Eine Partition heißt zusammenhängend, wenn alle Blöcke zusammenhängend sind.

a. Geben Sie jeweils für die folgenden beiden 2-Partitionen des Beispielgraphen an, ob alle Blöcke zusammenhängend sind. Begründen Sie jeweils kurz.
Hinweis: Die Blocknummer eines Knotens ist im Knoten dargestellt. [2 Punkt]



a)



b)

[1] $G[V_i] = (V_i, \{\{u, v\} \in E \mid u, v \in V_i\})$

(Teilaufgaben **b.** und **c.** befinden sich auf dem nächsten Blatt)

Fortsetzung von Aufgabe 3

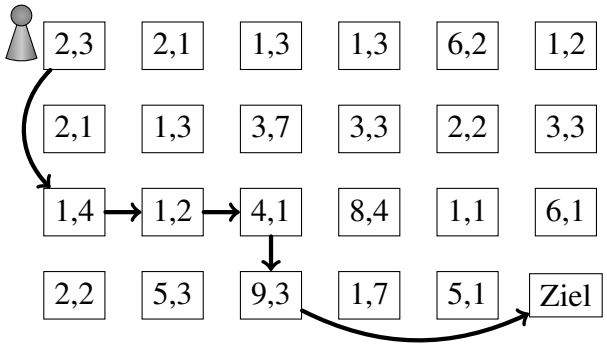
b. Gegeben ist nun eine k-Partition $V_1, \ldots, V_k$ $(k > 1)$ von einem zusammenhängenden Graphen $G = (V, E)$. Geben Sie einen Algorithmus an, der in Zeit $O(|E|\alpha_T(2|E|, |V|))$ entscheidet, ob die Partition zusammenhängend ist. α_T bezeichnet dabei die Ackermann-Funktion.

Hinweis: Sie können davon ausgehen, dass man zu einem Knoten $v \in V$ in konstanter Zeit die Blocknummer abfragen kann. Weiterhin werden auch schnellere Algorithmen als Lösung akzeptiert. [6 Punkte]

c. Zeigen Sie, dass ihr Algorithmus aus Teilaufgabe **b.** das gewünschte Laufzeitverhalten aufweist. [2 Punkte]

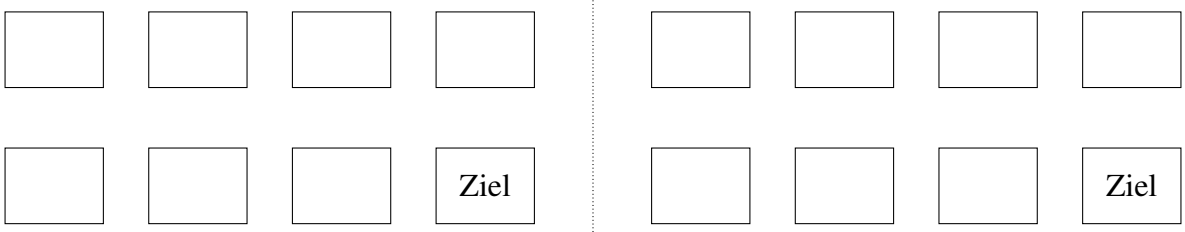
Aufgabe 4. Polyhalmadrom [10 Punkte]

Polyhalmadrom ist ein Spiel mit Karten und kleinen Figuren. Auf den Karten sind zwei Zahlen x_1, x_2 gedruckt und die Karten werden in einem rechteckigen Spielfeld mit m Zeilen und n Spalten zufällig angeordnet. Ein Spieler erhält eine kleine Figur, die zu Beginn auf die Karte am oberen linken Eck des Spielfelds gesetzt wird. Die Figur darf in jedem Zug wahlweise x_1 oder x_2 Karten weiterlaufen, aber nur nach rechts oder nach unten und nur innerhalb des Spielfeldes. Ziel des Spiels ist es, die Figur in **möglichst vielen Zügen** zur unteren rechten Ecke zu ziehen.



Beispiel eines Polyhalmadrom Spiels mit einer (nicht maximalen) Zugfolge

a. Zeigen Sie, dass eine *greedy-basierte* Suche, die nur die kleinere der beiden Zahlen betrachtet, nicht immer eine längstmögliche Zugfolge berechnet. Konstruieren Sie hierzu auf folgendem 2×4 Spielfeld ein Gegenbeispiel, in dem eine greedy-basierte Suche (siehe Pseudocode unten) eine suboptimale Zugfolge von Start bis Ziel berechnet. [2 Punkte]



Fall Sie beide Spielfelder verwenden, machen Sie deutlich welche Lösung zu werten ist.

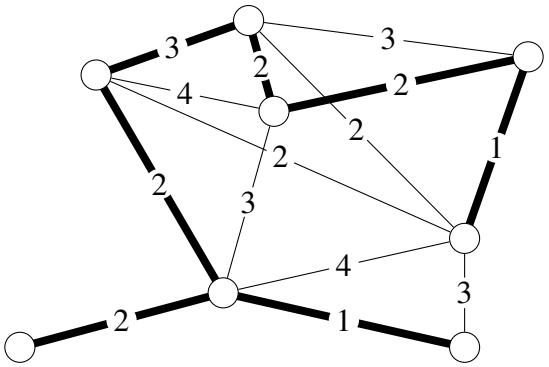
```
Function GreedyPolyhalmadrom(A : (m × n)-Matrix, i, j : ℕ, k : ℕ) : boolean
    if i > m ∨ j > n then return false
    if (i, j) = (m, n) then
        Print „Ziel vom Start in k Zügen erreichbar.“
        return true
    x := min(A[i, j].x1, A[i, j].x2)
    if GreedyPolyhalmadrom(A, i + x, j, k + 1) then
        Print „Zug (i, j) → (i + x, j)“
        return true
    if GreedyPolyhalmadrom(A, i, j + x, k + 1) then
        Print „Zug (i, j) → (i, j + x)“
        return true
    return false
```

Fortsetzung von Aufgabe 4

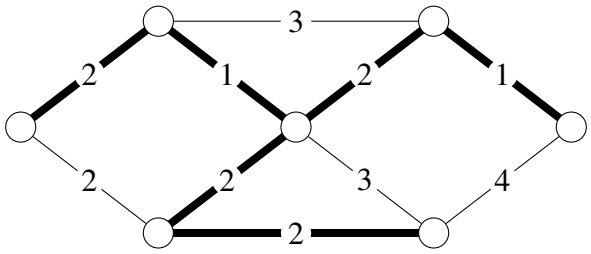
b. Geben Sie eine präzise Beschreibung eines Algorithmus an, der in $O(m \cdot n)$ Zeit immer eine längste Sprungfolge berechnet, oder feststellt, dass keine solche existiert.
Ein Algorithmus, der immerhin noch in $O(m^2 \cdot n)$ oder $O(m \cdot n^2)$ läuft, erreicht höchstens 3 Punkte. Eine Lösung ohne explizite Ausgabe einer längsten Zugfolge erreicht höchstens 5 Punkte. [8 Punkte]

Aufgabe 5. Minimale Spannbäume [8 Punkte]

a. Nennen Sie die Schnitteigenschaft (cut property) minimaler Spannbäume und markieren Sie in folgendem Graphen einen Schnitt, mit dem das Gewicht des Spannbaums reduziert werden kann, oder zeigen Sie, dass kein solcher Schnitt existiert. [2 Punkte]



b. Nennen Sie die Kreiseigenschaft (cycle property) minimaler Spannbäume und markieren Sie in folgendem Graphen einen Kreis, der diese Eigenschaft verletzt, oder zeigen Sie mit dieser Eigenschaft, dass der eingezeichnete Spannbaum minimal ist. [2 Punkte]



(Teilaufgabe **c.** befindet sich auf dem nächsten Blatt)

Fortsetzung von Aufgabe 5

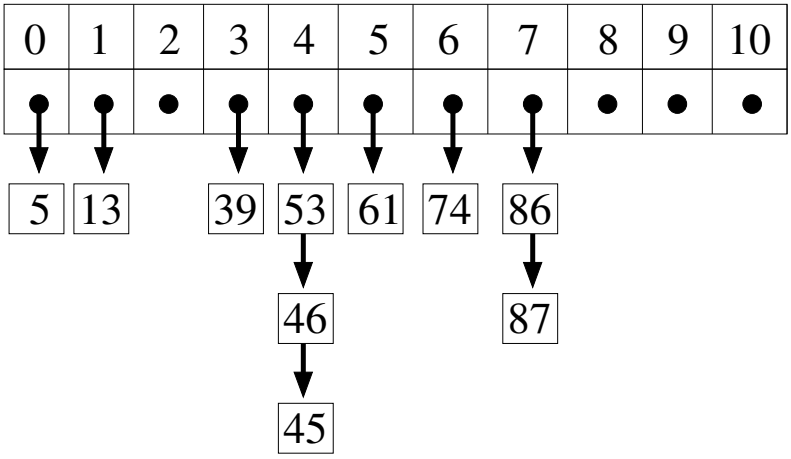
c. Sei ein ungerichteter Graph $G = (V, E)$ mit Kantengewichten $w : E \to \mathbb{N}$ gegeben, in dem die Gewichte aller Kanten paarweise verschieden sind. Zeigen Sie, dass der Graph einen eindeutig bestimmten minimalen Spannbaum besitzt. [4 Punkte]

Aufgabe 6. Hashing [7 Punkte]

Wir betrachten im Folgenden Hashtabellen mit n Buckets und zugehörigen Hashfunktionen[2],

$$h_n(x) = (x \text{ DIV } n) \text{ MOD } n\,.$$

Beispielsweise ist $h_{11}(74) = 6$. Zur Kollisionsauflösung werden einfach verkettete Listen verwendet. Folgende Hashtabelle hat beispielsweise die Größe $n = 11$ und Hashfunktion h_{11}:



a. Sei nun eine leere Hashtabelle mit $n = 11$ und Hashfunktion h_{11} gegeben.
Geben Sie eine Folge von *insert*-Operationen an, so dass die Tabelle nach Ausführen dieser Operationsfolge den obigen Zustand hat. [2 Punkte]

[2] Hierbei steht DIV für ganzzahlige Division und MOD für den Rest bei ganzzahliger Division.

(Teilaufgaben **b.** und **c.** auf dem nächsten Blatt)

Fortsetzung von Aufgabe 6

b. Geben Sie für eine leere Hashtabelle der Größe n mit Hashfunktion h_n eine Folge von n **verschiedenen** *insert* Operationen und n **verschiedenen** *find* Operationen an, so dass die erwartete Laufzeit für die Operationsfolge nicht gilt. Begründen Sie kurz, warum Ihre Folge das gewünschte Verhalten liefert. [3 Punkte]

c. Nennen Sie zwei Vorteile von Hashing mit verketteten Listen gegenüber Hashing mit linearer Suche. [2 Punkte]

Konzeptpapier (Abgabe freiwillig)

Name:

Vorname:

Matrikelnummer:

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders

ID 6

30.7.2013

Klausur Algorithmen I

Aufgabe 1.	Kleinaufgaben	15 Punkte
Aufgabe 2.	Dijkstras Algorithmus	10 Punkte
Aufgabe 3.	Zusammenhang	10 Punkte
Aufgabe 4.	Polyhalmadrom	10 Punkte
Aufgabe 5.	Minimale Spannbäume	8 Punkte
Aufgabe 6.	Hashing	7 Punkte

Bitte beachten Sie:

- Als Hilfsmittel ist nur **ein** DIN-A4-Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- **Schreiben** Sie auf **alle** Blätter Ihren Namen und Ihre Matrikelnummer.
- Merken Sie sich Ihre Klausur-ID 6 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		15	10	10	10	8	7	60
Punkte	EK							
	ZK							

Bonuspunkte:	Summe:	**Note:**

Aufgabe 7. Kleinaufgaben [15 Punkte]

a. Gegeben sind Teile der Union-Find Datenstruktur aus der Vorlesung. Fügen Sie Zeilen in den unten dargestellten Code ein, so dass die Union bzw. Link Operation immer den kleineren Baum (damit ist die Anzahl der Elemente der assoziierten Menge gemeint) an den größeren Baum anhängt.
Hinweis: dieses Vorgehen bezeichnet man auch als Union-by-Size.

[3 Punkte]

```
Class UnionFind(n : ℕ)
    parent=⟨1,2,...,n⟩ : Array [1..n] of 1..n

    Function find(i : 1..n) : 1..n
        if parent[i] = i then return i
        else return find(parent[i])

    Procedure link(i, j : 1..n)
        assert i und j sind Repräsentanten von verschiedenen Mengen

    Procedure union(i, j : 1..n)
        if find(i) ≠ find(j) then link(find(i), find(j))
```

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 7

b. Ein gerichteter Graph $G = (V, E)$ ist durch folgendes Adjazenzarray abgespeichert. Zeichnen Sie den Graphen und nummerieren Sie die Knoten in Ihrer Zeichnung. [2 Punkte]

	1	2	3	4	5	6	7	8
V	1	2	4	5	8	10	10	11

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

c. Lösen Sie die beiden folgenden Rekurrenzen im Θ-Kalkül:

$$U(n) = n + 20U(n/4), \qquad U(1) = 5$$
$$V(n) = 3n + V(n/4), \qquad V(1) = 42$$

mit $n = 4^k$ und $k \in \mathbb{N}_{>0}$. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 7

d. Zeigen Sie oder widerlegen Sie, dass $f(n) + g(n) = \mathrm{O}(g(f(n)))$ gilt. [2 Punkte]

e. Zeigen oder widerlegen Sie für jedes $k \in \mathbb{N}_+$, dass $\sum_{i=0}^{k} c_i n^i = \Theta(n^k)$ für $c_i \in \mathbb{N}, c_k \neq 0$ gilt. [2 Punkte]

(weitere Teilaufgaben auf dem nächsten Blatt)

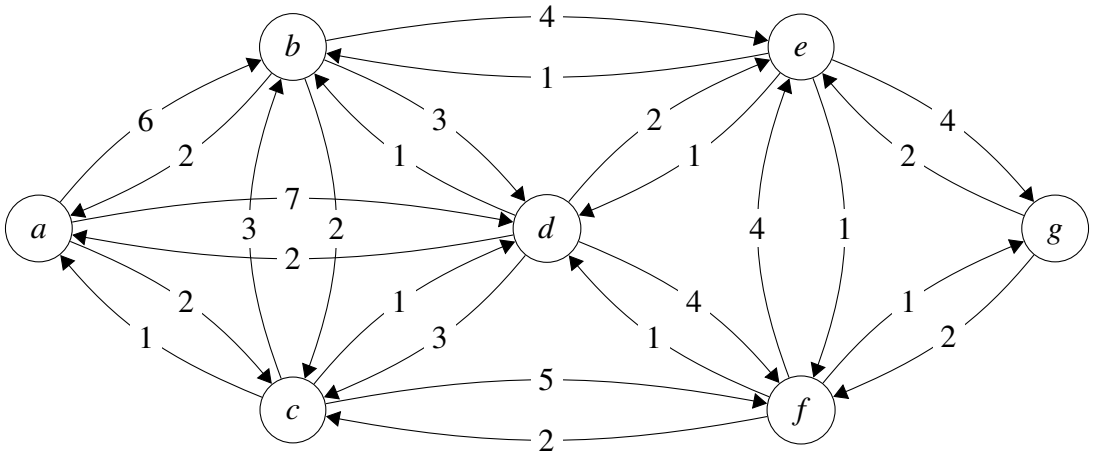
Fortsetzung von Aufgabe 7

f. Warum benötigt vergleichsbasiertes Sortieren im schlimmsten Fall mindestens $\Omega(n \log n)$ Vergleiche? Begründen Sie kurz. [2 Punkte]

g. Ordnen Sie die folgenden fünf Sortieralgorithmen aufsteigend nach worst-case Komplexität: Heap-Sort, Merge-Sort, Quick-Sort, Insertion-Sort und Bucket-Sort für Schlüsselwerte in $O(n)$. [2 Punkte]

Aufgabe 8. Dijkstras Algorithmus [10 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. Verwenden Sie für jeden Schleifendurchlauf eine Zeile in folgender Tabelle und tragen Sie die Werte der dabei auftretenden Variablen ein. Die ersten beiden Zeilen sind bereits vorgegeben.

Die Zellen in Spalte Q sollen *alle in* der Queue gespeicherten Knoten v als Paare (p, v) mit Priorität p in Reihenfolge enthalten. Bei $d[\cdot]$ und $parent[\cdot]$ brauchen Sie nur *geänderte Werte* notieren. Wird ein Knoten v vom Algorithmus „*scanned*“ („*settled*“, „*set*“), so tragen Sie in $d[v]$ ein „s“ ein. [6 Punkte]

	geänderte Variablen $d[\cdot]$							geänderte Variablen $parent[\cdot]$						
vollständiger Inhalt Q	a	b	c	d	e	f	g	a	b	c	d	e	f	g
Initialisierung	0	∞	∞	∞	∞	∞	∞	a	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$(0, a)$	s	6	2	7					a	a	a			

Fortsetzung von Aufgabe 8

	geänderte Variablen $d[\cdot]$							geänderte Variablen $parent[\cdot]$						
vollständiger Inhalt Q	a	b	c	d	e	f	g	a	b	c	d	e	f	g

Bei Bedarf erhalten Sie von der Aufsicht weitere Blätter mit Tabellen.

b. Was ist die asymptotische worst-case Laufzeit von Dijkstras Algorithmus im O-Kalkül in Abhängigkeit von Knoten- und Kantenzahl, wenn ein binärer Heap verwendet wird? [1 Punkt]

c. Der Algorithmus von Dijkstra funktioniert nur für nicht-negative Kantengewichte. Es ist aber sicherlich möglich einen Graphen mit negativen Kantengewichten durch Addition einer Konstanten c zu allen Kantengewichten in einen Graph mit nicht-negativen Kantengewichten zu transformieren.

Wieso kann man den Algorithmus von Dijkstra mit dieser Technik nicht auf beliebige Kantengewichte verallgemeinern? Geben Sie ein Gegenbeispiel ohne negative Kreise an. [3 Punkte]

Aufgabe 9. Zusammenhang [10 Punkte]

Gegeben sei ein ungerichteter, zusammenhängender Graph $G = (V, E)$ und eine Zahl $k > 1$. Das Graphpartitionierungsproblem möchte eine Aufteilung der Knotenmenge V in k Teilmengen (Blöcke) $V_1, \ldots, V_k$ finden, die folgende Bedingungen erfüllt:

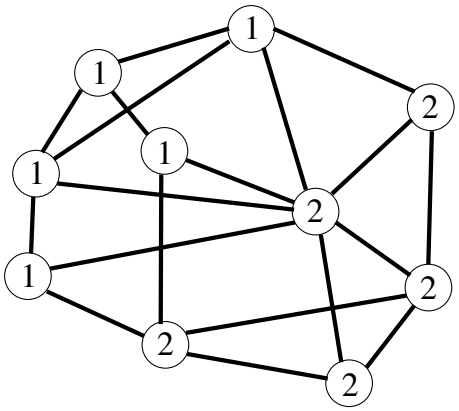
1. $\forall i \in \{1, \ldots, k\} : V_i \neq \emptyset$ – keine leeren Teilmengen
2. $\forall i, j \in \{1, \ldots, k\} : i \neq j \Rightarrow V_i \cap V_j = \emptyset$ – keine Überlappungen
3. $\bigcup_{i=1}^{k} V_i = V$ – volle Überdeckung
4. $\forall i \in \{1, \ldots, k\} : |V_i| \leq \lceil \frac{|V|}{k} \rceil$ – Balancebedingung

Die Zielfunktion ist häufig die Minimierung der Anzahl Schnittkanten $|\mathscr{E}|$ mit

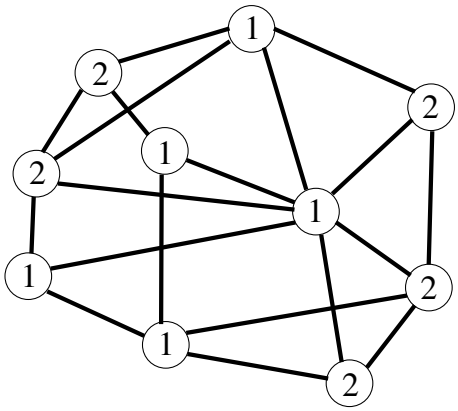
$$\mathscr{E} = \{\{u, v\} \in E \mid u \in V_i, v \in V_j, i \neq j\}.$$

In der Praxis werden aus verschiedenen Gründen häufig **zusammenhängende** Blöcke benötigt. Ein Block V_i heißt zusammenhängend, wenn der knoteninduzierte Teilgraph $G[V_i]$[3] zusammenhängend ist. Eine Partition heißt zusammenhängend, wenn alle Blöcke zusammenhängend sind.

a. Geben Sie jeweils für die folgenden beiden 2-Partitionen des Beispielgraphen an, ob alle Blöcke zusammenhängend sind. Begründen Sie jeweils kurz.
Hinweis: Die Blocknummer eines Knotens ist im Knoten dargestellt. [2 Punkt]



a)



b)

[3] $G[V_i] = (V_i, \{\{u, v\} \in E \mid u, v \in V_i\})$

(Teilaufgaben **b.** und **c.** befinden sich auf dem nächsten Blatt)

Fortsetzung von Aufgabe 9

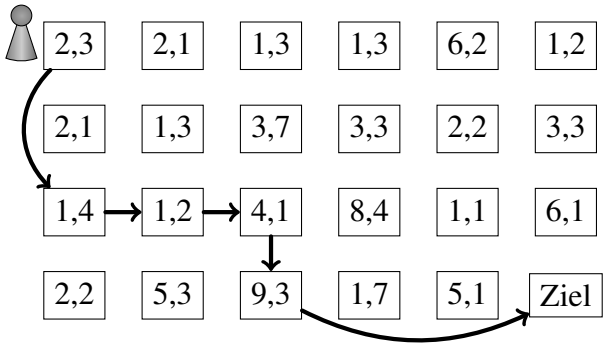
b. Gegeben ist nun eine k-Partition $V_1, \ldots, V_k$ $(k > 1)$ von einem zusammenhängenden Graphen $G = (V, E)$. Geben Sie einen Algorithmus an, der in Zeit $O(|E|\alpha_T(2|E|, |V|))$ entscheidet, ob die Partition zusammenhängend ist. α_T bezeichnet dabei die Ackermann-Funktion.

Hinweis: Sie können davon ausgehen, dass man zu einem Knoten $v \in V$ in konstanter Zeit die Blocknummer abfragen kann. Weiterhin werden auch schnellere Algorithmen als Lösung akzeptiert. [6 Punkte]

c. Zeigen Sie, dass ihr Algorithmus aus Teilaufgabe **b.** das gewünschte Laufzeitverhalten aufweist. [2 Punkte]

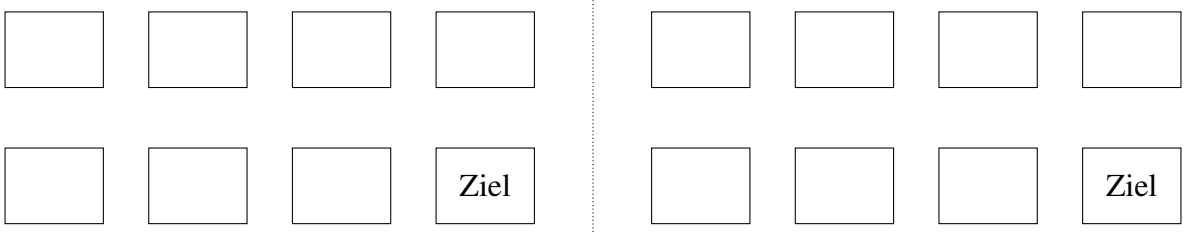
Aufgabe 10. Polyhalmadrom [10 Punkte]

Polyhalmadrom ist ein Spiel mit Karten und kleinen Figuren. Auf den Karten sind zwei Zahlen x_1, x_2 gedruckt und die Karten werden in einem rechteckigen Spielfeld mit m Zeilen und n Spalten zufällig angeordnet. Ein Spieler erhält eine kleine Figur, die zu Beginn auf die Karte am oberen linken Eck des Spielfelds gesetzt wird. Die Figur darf in jedem Zug wahlweise x_1 oder x_2 Karten weiterlaufen, aber nur nach rechts oder nach unten und nur innerhalb des Spielfeldes. Ziel des Spiels ist es, die Figur in **möglichst vielen Zügen** zur unteren rechten Ecke zu ziehen.



Beispiel eines Polyhalmadrom Spiels mit einer (nicht maximalen) Zugfolge

a. Zeigen Sie, dass eine *greedy-basierte* Suche, die nur die kleinere der beiden Zahlen betrachtet, nicht immer eine längstmögliche Zugfolge berechnet. Konstruieren Sie hierzu auf folgendem 2×4 Spielfeld ein Gegenbeispiel, in dem eine greedy-basierte Suche (siehe Pseudocode unten) eine suboptimale Zugfolge von Start bis Ziel berechnet. [2 Punkte]



Fall Sie beide Spielfelder verwenden, machen Sie deutlich welche Lösung zu werten ist.

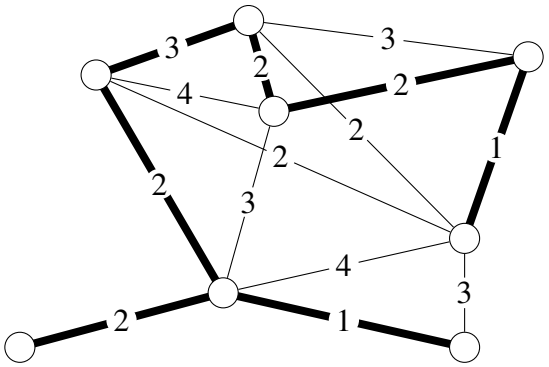
```
Function GreedyPolyhalmadrom(A : (m × n)-Matrix, i, j : ℕ, k : ℕ) : boolean
    if i > m ∨ j > n then return false
    if (i, j) = (m, n) then
        Print „Ziel vom Start in k Zügen erreichbar.“
        return true
    x := min(A[i, j].x1, A[i, j].x2)
    if GreedyPolyhalmadrom(A, i + x, j, k + 1) then
        Print „Zug (i, j) → (i + x, j)“
        return true
    if GreedyPolyhalmadrom(A, i, j + x, k + 1) then
        Print „Zug (i, j) → (i, j + x)“
        return true
    return false
```

Fortsetzung von Aufgabe 10

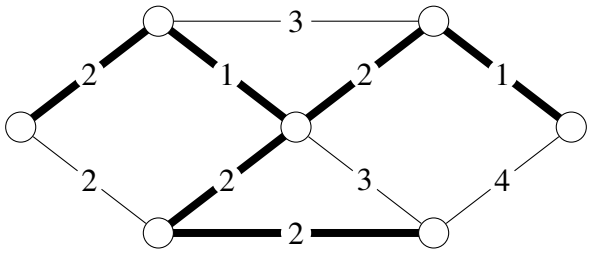
b. Geben Sie eine präzise Beschreibung eines Algorithmus an, der in $O(m \cdot n)$ Zeit immer eine längste Sprungfolge berechnet, oder feststellt, dass keine solche existiert.
Ein Algorithmus, der immerhin noch in $O(m^2 \cdot n)$ oder $O(m \cdot n^2)$ läuft, erreicht höchstens 3 Punkte. Eine Lösung ohne explizite Ausgabe einer längsten Zugfolge erreicht höchstens 5 Punkte. [8 Punkte]

Aufgabe 11. Minimale Spannbäume [8 Punkte]

a. Nennen Sie die Schnitteigenschaft (cut property) minimaler Spannbäume und markieren Sie in folgendem Graphen einen Schnitt, mit dem das Gewicht des Spannbaums reduziert werden kann, oder zeigen Sie, dass kein solcher Schnitt existiert. [2 Punkte]



b. Nennen Sie die Kreiseigenschaft (cycle property) minimaler Spannbäume und markieren Sie in folgendem Graphen einen Kreis, der diese Eigenschaft verletzt, oder zeigen Sie mit dieser Eigenschaft, dass der eingezeichnete Spannbaum minimal ist. [2 Punkte]



(Teilaufgabe **c.** befindet sich auf dem nächsten Blatt)

Fortsetzung von Aufgabe 11

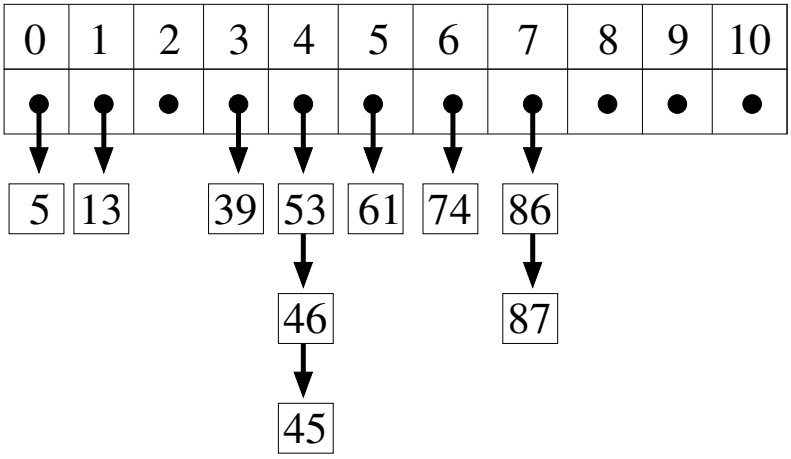
c. Sei ein ungerichteter Graph $G = (V, E)$ mit Kantengewichten $w : E \to \mathbb{N}$ gegeben, in dem die Gewichte aller Kanten paarweise verschieden sind. Zeigen Sie, dass der Graph einen eindeutig bestimmten minimalen Spannbaum besitzt. [4 Punkte]

Aufgabe 12. Hashing [7 Punkte]

Wir betrachten im Folgenden Hashtabellen mit n Buckets und zugehörigen Hashfunktionen[4],

$$h_n(x) = (x \text{ DIV } n) \text{ MOD } n\,.$$

Beispielsweise ist $h_{11}(74) = 6$. Zur Kollisionsauflösung werden einfach verkettete Listen verwendet. Folgende Hashtabelle hat beispielsweise die Größe $n = 11$ und Hashfunktion h_{11}:



a. Sei nun eine leere Hashtabelle mit $n = 11$ und Hashfunktion h_{11} gegeben.
Geben Sie eine Folge von *insert*-Operationen an, so dass die Tabelle nach Ausführen dieser Operationsfolge den obigen Zustand hat. [2 Punkte]

[4] Hierbei steht DIV für ganzzahlige Division und MOD für den Rest bei ganzzahliger Division.

(Teilaufgaben **b.** und **c.** auf dem nächsten Blatt)

Fortsetzung von Aufgabe 12

b. Geben Sie für eine leere Hashtabelle der Größe n mit Hashfunktion h_n eine Folge von n **verschiedenen** *insert* Operationen und n **verschiedenen** *find* Operationen an, so dass die erwartete Laufzeit für die Operationsfolge nicht gilt. Begründen Sie kurz, warum Ihre Folge das gewünschte Verhalten liefert. [3 Punkte]

c. Nennen Sie zwei Vorteile von Hashing mit verketteten Listen gegenüber Hashing mit linearer Suche. [2 Punkte]

Konzeptpapier (Abgabe freiwillig)