Nachname:

Vorname:

Matrikelnummer:

**Lösungsvorschlag**

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders 22.09.2023

# Klausur Algorithmen II

| | | |
|---|---|---|
| Aufgabe 1. | Kleinaufgaben | 11 Punkte |
| Aufgabe 2. | Approximationsalgorithmen: Fahrradkurier | 11 Punkte |
| Aufgabe 3. | Party-Gnome | 8 Punkte |
| Aufgabe 4. | Randomisierte Algorithmen: Cuckoo Hashing | 9 Punkte |
| Aufgabe 5. | Stringology | 10 Punkte |
| Aufgabe 6. | Geometrische Algorithmen: Isolation von Bergen | 11 Punkte |

Bitte beachten Sie:

- Als Hilfsmittel ist nur **ein** DIN-A4 Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- Schreiben Sie Ihre Antworten nur in blau oder schwarz, mit dokumentenechtem Stift.
- **Schreiben** Sie auf **alle** Blätter der Klausur und Zusatzblätter Ihre **Matrikelnummer**.
- Die Klausur enthält **17 Seiten**.
- Zum Bestehen der Klausur sind 30 Punkte hinreichend.

Lösungsvorschlag

EK

ZK

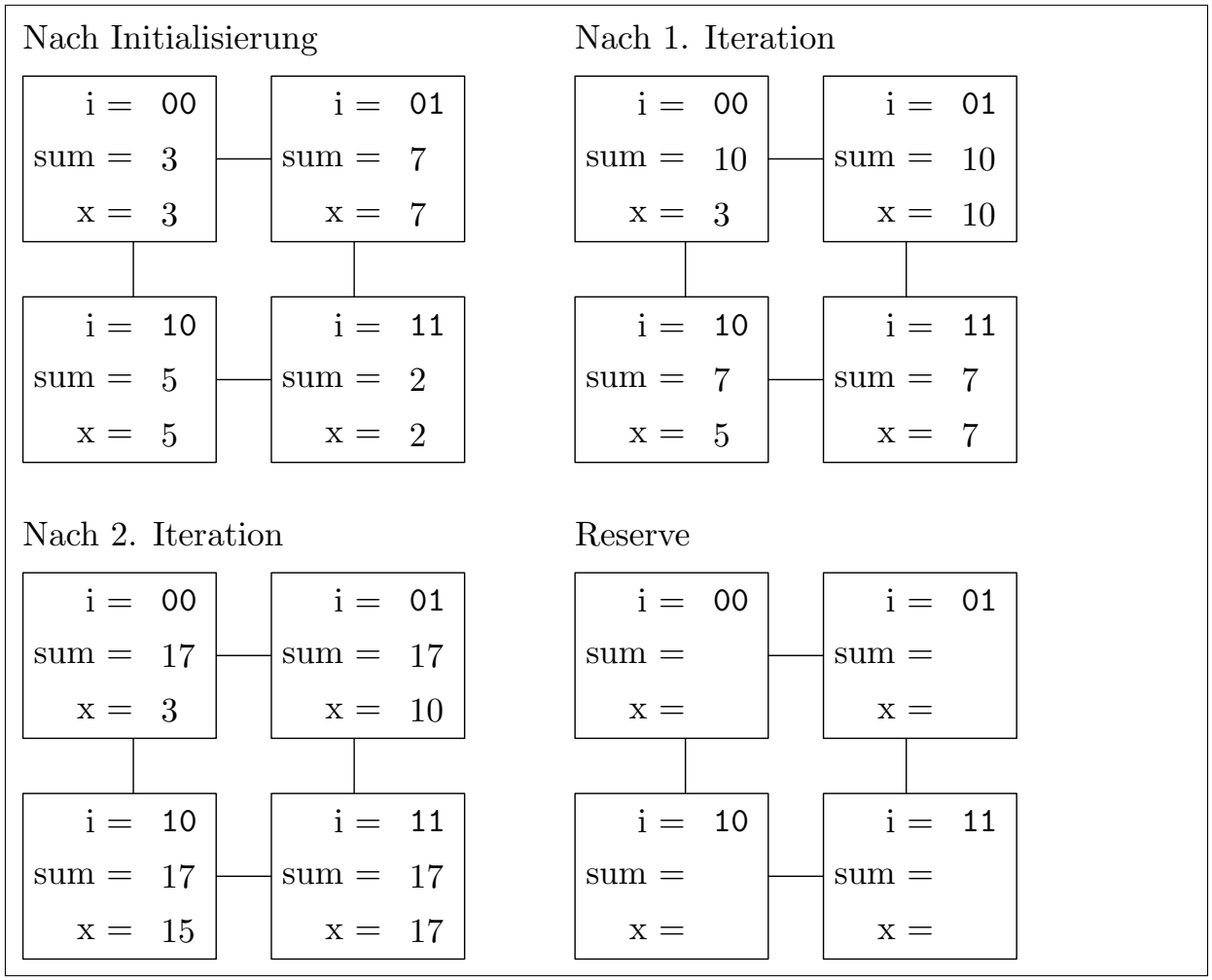
**Aufgabe 1.** Kleinaufgaben [11 Punkte]

**a.** Gegeben sei ein Hyperwürfel-Verbindungsnetzwerk mit 4 Rechenknoten. Führen Sie darauf den Hyperwürfel-Algorithmus für inklusive Präfixsummen aus der Vorlesung aus. Geben Sie den Zustand der Variablen auf allen Rechenknoten nach der Initialisierung, sowie nach jeder Iteration des Algorithmus an. Verwenden Sie dazu die Darstellung aus der Vorlesung.

Eingabe-Liste sortiert nach Prozessor-Index $i$: $[3,7,5,2]$

*Wenn Sie das Reserve-Bild beschriften, machen Sie deutlich, zu welcher Iteration es gehört. Andernfalls wird diese Teilaufgabe mit 0 Punkten bewertet.* [3 Punkte]

## Lösung

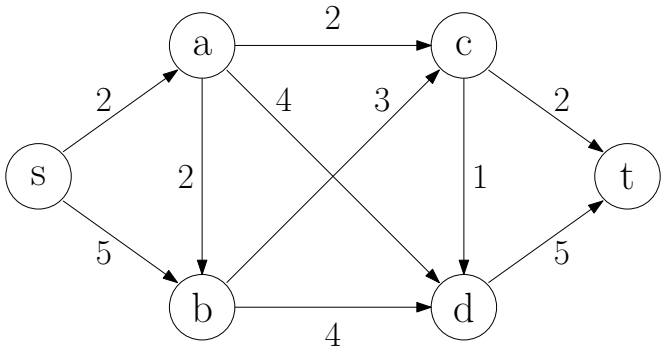


**b.** Geben Sie Work $W$ und Effizienz $E$ des Hyperwürfel-Präfixsummen-Algorithmus aus der Vorlesung an. Es seien $n$ Eingabe-Elemente und $p = n$ Rechenknoten gegeben, wobei $n$ eine Zweierpotenz ist. [2 Punkte]

## Lösung

- $T_{\text{seq}} = \Theta(n)$
- $T(p) = \Theta(\log p)$
- $W = p \cdot T(p) = \Theta(p \log p)$
- $S = T_{\text{seq}}/T(p) = \Theta(n/\log n)$
- $E = \Theta(1/\log n)$

**c.** Betrachten Sie das unten abgebildete Flussnetzwerk. Jede Kante $e$ ist mit ihrer Kapazität $c(e)$ beschriftet. Berechnen Sie mit dem Ford-Fulkerson-Algorithmus den maximalen Fluss von der Quelle $s$ zur Senke $t$. Geben Sie dabei für jeden Schritt den augmentierenden Pfad in Form einer Knotenliste und den Fluss, den Sie über diesen Pfad schieben, an. Sie benötigen nicht notwendigerweise alle Zeilen. Geben Sie zusätzlich den Wert des maximalen Flusses nach der Ausführung an. [3 Punkte]




| | Pfad | Fluss |
|---|---|---|
| 1. | | |
| 2. | | |
| 3. | | |
| 4. | | |
| 5. | | |

Maximaler Fluss: _____

## Lösung

Z.B.

- sact: 2
- sbdt: 4
- sbcadt: 1

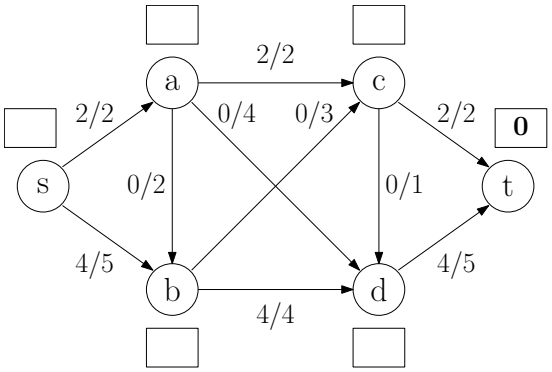
oder

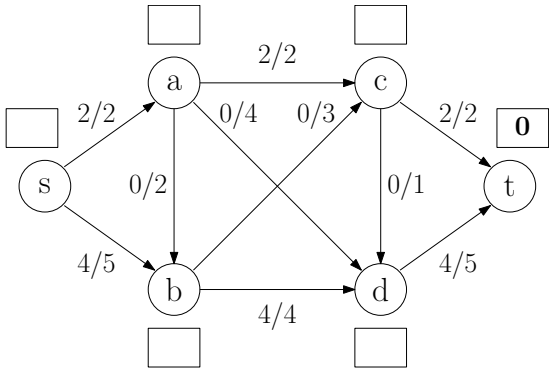
- sadt: 2
- sbct: 2
- sbdt: 3

Gesamtfluss: 7

**d.** Die unten gegebene Abbildung zeigt den Fluss $f$ auf dem Flussnetzwerk aus Teilaufgabe **c** nach einer Iteration des Dinitz-Algorithmus. Jede Kante $e$ ist mit $f(e)/c(e)$ beschriftet. Beschriften Sie das Kästchen für jeden Knoten $v$ mit seinem Level $d(v)$ im Layer-Graph der nächsten Iteration des Dinitz-Algorithmus.

*Zwei Kopien. Wenn Sie mehr als eine Abbildung beschriften, machen Sie deutlich, welche Kopie bewertet werden soll. Andernfalls wird diese Teilaufgabe mit 0 Punkten bewertet.* [2 Punkte]




## Lösung

| $v$ | s | a | b | c | d | t |
|---|---|---|---|---|---|---|
| $d(v)$ | 4 | 2 | 3 | 2 | 1 | 0 |

**e.** Sei $f(n,k)$ die Laufzeit eines Algorithmus mit Eingabegröße $n$. Geben Sie an, welche der folgenden Laufzeiten ein Problem *fixed-parameter tractable (FPT)* bezüglich $k$ machen. [1 Punkt]

| | $f(n,k)$ | ja | nein |
|---|---|---|---|
| 1. | $k!n^{42}$ | ☐ | ☐ |
| 2. | $2^{k\log n}$ | ☐ | ☐ |

## Lösung

- Ja, denn mit $g(k) = k!$ und $p(n) = n^{42}$ ist $p$ ein Polynom und $g$ nicht von $n$ abhängig.
- Nein, denn $2^{k\log n} = (2^{\log(n)})^k = (cn)^k$ für eine Konstante $c$, was sich nicht wie für FPT gefordert aufteilen lässt.

Lösungsvorschlag

EK

ZK

**Aufgabe 2.** Approximationsalgorithmen: Fahrradkurier [11 Punkte]

**a.** Gegeben sei ein Minimierungsproblem $P$ mit Bewertungsfunktion $w$ sowie ein Approximationsalgorithmus $A$ zur Lösung des Problems. Wann spricht man von einem konstanten Approximationsfaktor $\alpha$ für $A$? [1 Punkt]

## Lösung

Beschreibe $\mathrm{OPT}(I)$ die optimale Lösung der Probleminstanz $I$. $A$ approximiert eine optimale Lösung von $P$ mit konstantem Approximationsfaktor $\alpha \in \mathbb{R}$, wenn für alle Probleminstanzen $I$ gilt:

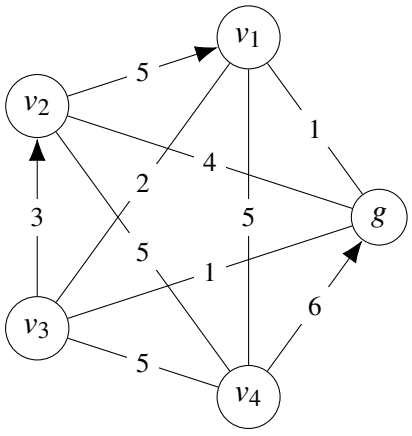
$$\frac{w(A(I))}{w(\mathrm{OPT}(I))} \leq \alpha$$

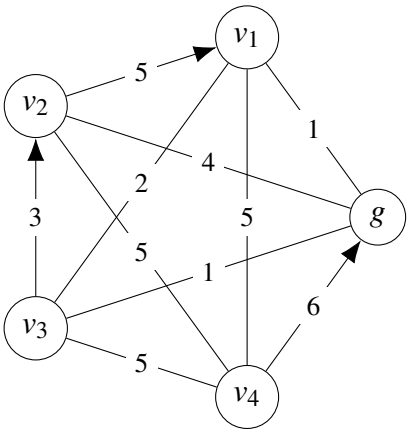
Ein Fahrradkurier plant seine Route für die Aufträge eines Tages.
Das Straßennetzwerk sei durch einen ungerichteten, zusammenhängenden Graph $G = (V,E)$ modelliert. Die Gewichtsfunktion $d : E \to \mathbb{R}$ gibt die Fahrtzeit für jeden Straßenabschnitt an.

Sei $R \subseteq V \times V$ die Menge an Aufträgen des Kuriers. Für jeden Auftrag $(s,t) \in R$ muss der Kurier ein Paket bei $s$ abholen und zu $t$ bringen, wobei stets $s \neq t$. Dabei kann der Kurier immer nur ein Paket gleichzeitig mit sich führen. Die Aufträge können in beliebiger Reihenfolge bearbeitet werden. Die Route muss bei der Geschäftsstelle des Kuriers $g \in V$ anfangen und enden.
Der Kurier möchte seine Route so planen, dass die insgesamt ohne Paket gefahrene Zeit minimiert wird. Wir nennen dieses Optimierungsproblem das *Fahrradkurier-Problem*.

**b.** Betrachten Sie das dargestellte Wegenetzwerk, wobei Kantenbeschriftungen die Fahrtzeiten angeben. Sei $R = \{(v_2,v_1),(v_3,v_2),(v_4,g)\}$.
Vervollständigen Sie in der Zeichnung die optimale Route, indem sie Pfeilspitzen an die verwendeten Kanten einzeichnen. Geben Sie auch die optimale Gesamtfahrtzeit ohne Paket an.

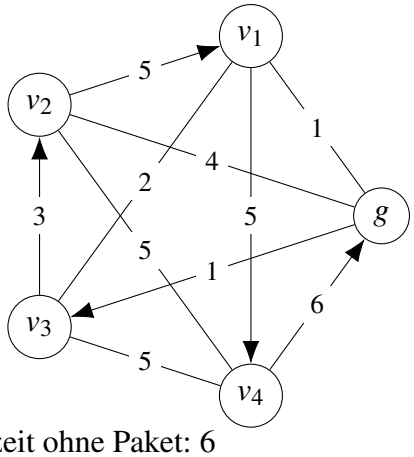
*Zwei Kopien. Wenn Sie mehr als einen Graph beschriften, machen Sie deutlich, welche Kopie korrigiert werden soll. Andernfalls wird diese Teilaufgabe mit 0 Punkten bewertet.* [2 Punkte]




Fahrtzeit ohne Paket: _____

## Lösung




Fahrtzeit ohne Paket: 6

**c.** Sei ein Algorithmus $A_{\text{opt}}$ bekannt, welcher eine optimale Lösung für das Handlungsreisendenproblem (TSP) auf *vollständigen gerichteten* Graphen findet. Wie kann $A_{\text{opt}}$ verwendet werden, um eine optimale Lösung für das Fahrradkurier-Problem zu finden?

*Hinweis*: Konstruieren Sie für eine Fahrradkurier-Instanz $I = (G = (V,E), d, g, R)$ einen vollständigen gerichteten Graphen $G'$, der die Aufträge des Kuriers repräsentiert. Sie dürfen annehmen, dass die kürzesten Fahrtzeiten zwischen allen Knotenpaaren in $G$ bekannt sind.

[4 Punkte]

## Lösung

Da die Zielfunktion gerade aus der insgesamt ohne Paket gefahrenen Zeit besteht und der Kurier einen Auftrag nach dem anderen bearbeiten muss, sind die Wege vom Ziel eines Auftrags zum Start des nächsten Auftrags wichtig, während die Wege zwischen dem Start und Ziel desselben Auftrages nicht relevant sind. Wir können die Fahrradkurier-Instanz also auf eine TSP-Instanz abbilden, indem wir jeden Auftrag als Knoten darstellen, und diese Knoten durch Kanten verbinden, deren Gewichte die Kürzesten-Wege-Distanzen vom Ziel eines Auftrags zum Start des nächsten Auftrags sind.

Bezeichne $\delta(u,v)$ die Kürzeste-Wege-Distanz zwischen $u, v \in V$. Transformiere zu TSP-Instanz $G' = (V', V' \times V')$ mit

- $V' := \{g\} \cup R$
- $d'((s_i,t_i),(s_j,t_j)) := \delta(t_i,s_j)$ für $1 \leq i,j \leq k$
- $d'(g,(s_i,t_i)) := \delta(g,s_i)$, $d'((s_i,t_i),g) := \delta(t_i,g)$

Verwende $A_{\text{opt}}$, um diese TSP-Instanz zu lösen und erhalte einen Kreis $C$ von Aufträgen (und $g$ an Anfang und Ende). Die optimale Lösung von $I$ arbeitet die Aufträge in der durch $C$ gegebenen Reihenfolge ab. Dabei fährt der Kurier zwischen den Aufträgen (also vom Ziel eines Auftrags zum Start des nächsten Auftrags) auf kürzesten Wegen in $G$.
Diese Lösung von $I$ ist optimal, denn gäbe es eine Abfolge von Aufträgen mit geringerer Fahrtzeit ohne Paket, so würde diese Abfolge auch eine bessere Lösung des TSP induzieren.

**d.** Um die Route möglichst abwechslungsreich zu gestalten, möchte der Kurier jeden Knoten (außer $g$) höchstens ein Mal besuchen. Zeigen Sie, dass es dann NP-schwer ist, dieses modifizierte Fahrradkurier-Problem innerhalb eines konstanten Faktors zu approximieren, wenn die Dreiecksungleichung nicht gilt.

*Hinweis*: Es ist NP-schwer, das TSP auf ungerichteten Graphen innerhalb eines konstanten Faktors zu approximieren, wenn die Dreiecksungleichung nicht gilt. [4 Punkte]

## Lösung

Reduziere TSP auf Fahrradkurier:

- Gegebene TSP-Instanz $G' = (V' = \{v_1, \ldots, v_n\}, V' \times V')$ mit Gewichtsfunktion $d'$, für die Dreiecksungleichung nicht gilt.
- Erzeuge Fahrradkurier-Instanz (ungerichteter Graph) $G = (V, E)$ mit
  - $V := \bigcup_{1 \leq i \leq n} \{s_i, t_i\}$,
  - $E := \{\{s_i, t_j\} \mid i, j \in \{1, \ldots, n\}\}$ und
  - Gewichtsfunktion $d$ mit $d(s_i, t_i) = 0$ und $d(s_i, t_j) = d'(v_i, v_j)$ für $i \neq j$.
- Aufträge $R := \{(s_i, t_i) \mid 1 \leq i \leq n\}$
- Geschäftsstelle an beliebigem $s_i$
- Transformation ist polynomiell ($n^2$)
- Lösung für Fahrradkurier = Lösung für TSP
- Wenn man Fahrradkurier in **P** approximieren kann, ergibt sich auch Approximation für TSP in **P**
- Widerspruch

Damit ist gezeigt, dass es Fahrradkurier-Instanzen gibt, für welche es NP-schwer ist, ihre Lösung innerhalb eines konstanten Faktors zu approximieren. Diese schwierig zu approximierenden Fahrradkurier-Instanzen enthalten insbesondere auch Instanzen, für welche die Dreiecksungleichung nicht gilt. Dies lässt sich daraus schließen, dass die Dreiecksungleichung für alle aus der Transformation entstehenden Instanzen nicht gilt, denn:

- Es gibt Dreieck $(v_1, v_2, v_3, v_1)$ in $G'$ mit $d'(v_1, v_3) > d'(v_1, v_2) + d'(v_2, v_3)$
- Im entsprechenden Kreis $(s_1, t_1, s_2, t_2, s_3, t_3, s_1)$ in $G$ gilt dann:

$$\begin{aligned} d(s_1, t_3) &= d'(v_1, v_3) \\ &> d'(v_1, v_2) + d'(v_2, v_3) \\ &= d(s_1, t_1) + d(t_1, s_2) + d(s_2, t_2) + d(t_2, s_3) \end{aligned}$$

Lösungsvorschlag

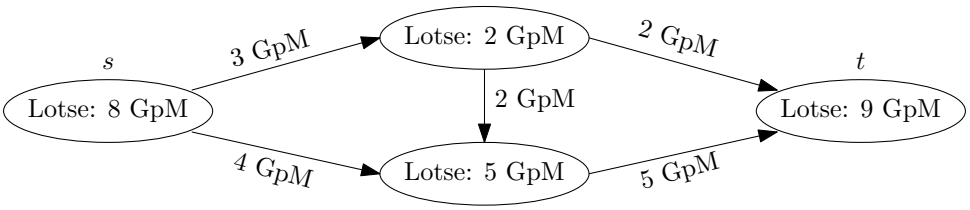
EK

ZK

**Aufgabe 3.** Party-Gnome [8 Punkte]

Es ist lange Nacht der Gnome in Gnomhausen. Eine Party geht gerade zu Ende, weshalb alle Gnome von dieser Party zu einer neuen Party wollen. Gnomhausens Wegenetzwerk besteht aus Tunneln und Kreuzungen. Jeder Tunnel kann nur von einer begrenzten Anzahl von Gnomen pro Minute (GpM) passiert werden. Außerdem wurde jedem Tunnel eine feste Richtung zugewiesen, in der die Gnome ihn begehen dürfen.
Wenn ein Gnom auf eine Tunnelkreuzung trifft, muss er den dort stationierten Gnom-Lotsen fragen, in welchen Tunnel er sich als nächstes begeben soll, bevor er weitergehen kann. Jeder Gnom-Lotse kann nur einer begrenzte Anzahl an Anfragen pro Minute beantworten. Die Start- und Ziel-Party finden jeweils an einer Kreuzung mit eigenem Gnom-Lotsen statt, welchen die Gnome wie an jeder anderen Kreuzung befragen müssen. Die Anzahl der Gnome bei der Start-Party sei unbeschränkt.

**a.** Betrachten Sie das unten abgebildete Wegenetzwerk mit Start-Party $s$ und Ziel-Party $t$. Jede Kante repräsentiert einen Tunnel in Pfeilrichtung und jeder Knoten repräsentiert eine Kreuzung. Wie viele Gnome können höchstens pro Minute bei der Zielparty $t$ ankommen? [2 Punkte]




Maximale GpM bei $t$: _____

## Lösung

Es können maximal 6 GpM über die Wege laufen. Würde man nur die Tunnels betrachten, würde sich 7 ergeben, aber die Gnome werden durch die Lotsen gebremst.

**b.** Sei $n$ die Anzahl der Kreuzungen und $m \geq n$ die Anzahl der Tunnel. Geben Sie einen Algorithmus an, welcher in Laufzeit $\mathcal{O}\left(n^2\sqrt{m}\right)$ ermittelt, wie viele Gnome pro Minute höchstens bei der Ziel-Party ankommen können. Begründen Sie die Laufzeit des Algorithmus. [4 Punkte]

## Lösung

Interpretiere das Wegenetzwerk als Flussnetzwerk $(G = (V,E), c, s, t)$. Für jede Kreuzung $k$, füge einen Eingangsknoten $e_k$ und einen Ausgangsknoten $a_k$ zu $V$ hinzu. Füge außerdem eine Kante $(e_k, a_k)$ zu $E$ hinzu, wobei $c(e_k, a_k)$ die maximalen GpM des Gnom-Lotsen bei $k$ beschreibt.
Für jeden Tunnel, der von Kreuzung $k$ zu Kreuzung $k'$ führt, füge eine Kante $(a_k, e_{k'})$ zu $E$ hinzu. Der Wert $c(a_k, e_{k'})$ ist die Zahl der Gnome, die pro Minute den Tunnel passieren können. Setze $s := e_{\text{start}}$ und $t := a_{\text{ziel}}$, wobei start und ziel jeweils die Kreuzungen bezeichnen, an denen Start- und Ziel-Party stattfinden.

Das entstandene Flussnetzwerk hat $n' = 2n$ Knoten sowie $m' = m + n$ Kanten, wobei $m' \in \mathcal{O}(m)$, da es keine isolierten Knoten gibt (Kreuzungen können nicht ohne angrenzende Tunnel existieren). Berechne einen maximalen Fluss $f$ mit dem Preflow-Push Algorithmus in Laufzeit $\mathcal{O}\left(n^2\sqrt{m}\right)$. Der Wert von $f$ beschreibt, wie viele GpM die Ziel-Party erreichen können.

**c.** Es gebe nun beliebig viele Start-Partys an je unterschiedlichen Kreuzungen. Die Zahl der Kreuzungen sei weiter $n$ und die Zahl der Tunnel $m \geq n$. Geben Sie einen Algorithmus an, der in Laufzeit $\mathcal{O}\left(n^2\sqrt{m}\right)$ ermittelt, wie viele Gnome pro Minute höchstens bei der Ziel-Party ankommen können. Begründen Sie die Laufzeit des Algorithmus. Die Anzahl Gnome bei jeder Start-Party sei unbeschränkt. [2 Punkte]

## Lösung

Lösung wie vorher nur mit zusätzlicher Superquelle $s$ und Kanten $(s, e_k)$ mit $c(s, e_k) = \infty$ für jede Kreuzung $k$, an der eine Startparty liegt. Führt zu $\mathcal{O}(1)$ zusätzlichen Knoten und $\mathcal{O}(n)$ zusätzlichen Kanten, also Laufzeit wie vorher.

Lösungsvorschlag

EK

ZK

**Aufgabe 4.** Randomisierte Algorithmen: Cuckoo Hashing [9 Punkte]

**a.** Gegeben seien folgende Schlüssel mit jeweils zwei Hashfunktions-Werten $h_1$ und $h_2$. Diese sollen in eine Cuckoo-Hashtabelle mit 2 Hashfunktionen und einem Schlüssel pro Zelle eingefügt werden. Fügen Sie mit einer beliebigen gültigen Strategie alle Schlüssel ein und geben Sie den finalen Zustand an.

*Zwei Kopien. Wenn Sie mehr als eine Kopie beschriften, machen Sie deutlich, welche Kopie bewertet werden soll. Andernfalls wird diese Teilaufgabe mit 0 Punkten bewertet.* [2 Punkte]

| Schlüssel | $h_1$ | $h_2$ |
|---|---|---|
| A | 5 | 2 |
| B | 3 | 4 |
| C | 3 | 7 |
| D | 1 | 7 |
| E | 3 | 5 |
| F | 5 | 4 |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

## Lösung

Löse durch scharfes Hinsehen, nicht durch Ausführen von 6 Insertions. Idee: Wenn ein Hashwert nur 1x vorkommt, kann ein Element sofort platziert werden, da es nicht mit anderen in die Quere kommt.

- Nur D hat den Hashwert 1
- Nur A hat den Hashwert 2
- Von den übrigen Elementen hat nur C den Hashwert 7
- Die übrigen 3 Elemente formen ein Dreieck, das in beide Richtungen orientiert werden kann.

Es ergeben sich 2 mögliche Lösungen:

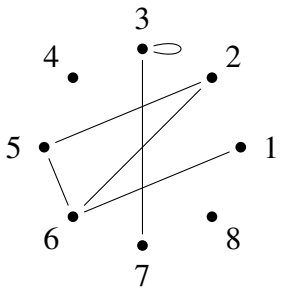
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| D | A | E | B | F | | C | |
| D | A | B | F | E | | C | |

**b.** Eine Cuckoo-Hashtabelle kann als (Multi-)Graph interpretiert werden, wobei jeder Knoten für eine Zelle steht und jede Kante die zwei Hashfunktions-Werte eines Schlüssels verbindet. Eine Cuckoo-Hashtabelle kann mit einer gegebenen Menge Schlüssel genau dann konstruiert werden, wenn der Graph nur aus Bäumen und Pseudobäumen besteht. Zeichnen Sie jeweils den Graph der folgenden beiden Schlüsselmengen. Sind die Tabellen erfolgreich konstruierbar?

*Hinweis*: Ein Pseudobaum ist ein Baum mit einer zusätzlichen Kante. [2 Punkte]

## Lösung

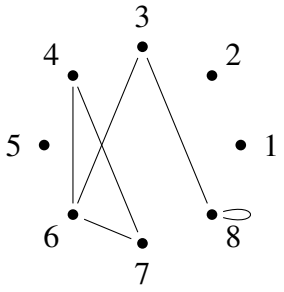
| Schlüssel | $h_1$ | $h_2$ |
|---|---|---|
| A | 6 | 5 |
| B | 5 | 2 |
| C | 1 | 6 |
| D | 2 | 6 |
| E | 7 | 3 |
| F | 3 | 3 |




Konstruierbar?

Ja

| Schlüssel | $h_1$ | $h_2$ |
|---|---|---|
| A | 8 | 3 |
| B | 4 | 6 |
| C | 6 | 3 |
| D | 6 | 7 |
| E | 8 | 8 |
| F | 7 | 4 |




Konstruierbar?

Nein

**c.** Soll eine Cuckoo-Hashtabelle stärker gefüllt werden als ein bestimmter Schwellwert, ist es unwahrscheinlich, dass die Konstruktion gelingt. Möchten wir trotzdem testen, ob die Konstruktion möglich ist, erreicht iteratives Einfügen aller Schlüssel auf Grund der Kollisionen nicht mehr erwartet lineare Laufzeit.
Geben Sie einen Algorithmus an, der mittels der Graph-Interpretation in Linearzeit entscheidet, ob die Hashtabelle für eine gegebene Schlüsselmenge konstruierbar ist. Ihr Algorithmus darf Laufzeit $\mathcal{O}(n)$ benötigen, wobei $n$ die Anzahl der Tabellen-Zellen ist. Begründen Sie die Laufzeit des Algorithmus. Gehen Sie dabei insbesondere darauf ein, wie die Laufzeit mit der von Ihnen gewählten Graphdatenstruktur erreicht wird. [5 Punkte]

## Lösung

- Idee: Bestimme den zugehörigen Graphen wie in Aufgabe b und zähle die Kanten in jeder Komponente.
- Wenn $n$ Tabellen-Zellen gegeben sind, wissen wir, dass die Konstruktion nur mit $\leq n$ Einträgen erfolgreich sein kann. Die Knoten sowie die Kanten des Graphen sind also beide durch $n$ beschränkt.
- In der Vorlesung wurde der Algorithmus für strenge Zusammenhangskomponenten auf gerichteten Graphen vorstellt. Auf ungerichteten Graphen wie in dieser Aufgabe ist die Betrachtung von *strengen* Zusammenhangskomponenten erst mal nicht sinnvoll. Um den Algorithmus aus der Vorlesung anwenden zu können, verdopple deshalb alle Kanten.
- Bestimme mit dem Algorithmus aus der Vorlesung die Zusammenhangskomponenten.
- Zähle in jeder Komponente, ob die Anzahl der Kanten exakt der doppelten Anzahl der Knoten entspricht.
- Verwenden wir zur Darstellung des Graphen beispielsweise eine Adjazenzliste, ist das Verdoppeln der Kanten in Linearzeit möglich. Wie bekannt aus der Vorlesung ist darauf auch das Finden der Zusammenhangskomponenten in Linearzeit möglich. Zum Zählen der Kanten in einer Komponente muss über die Knoten der Komponente iteriert werden und deren Kanten addiert werden. Jede Kante wird dabei genau 1x gezählt, insgesamt ist also auch dieser Schritt in Linearzeit möglich.

Ein alternativer Lösungsansatz verwendet Tiefensuche und zählt die Anzahl der `traverseNonTreeEdge`-Aufrufe pro Komponente.

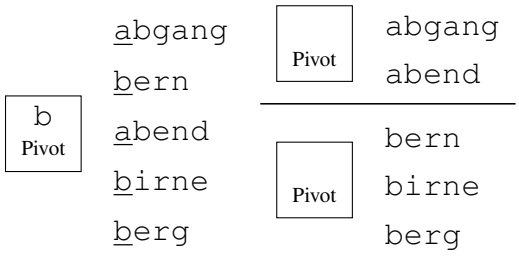
Lösungsvorschlag

EK

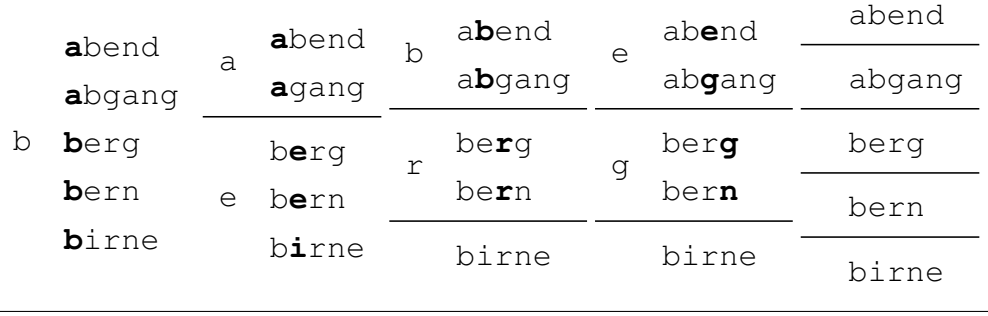
ZK

**Aufgabe 5.** Stringology [10 Punkte]

**a.** Sortieren Sie die folgenden Strings mithilfe von Multikey-Quicksort. Der erste Schritt ist bereits angegeben. Führen Sie den Algorithmus mit der gleichen Notation fort und geben Sie in jedem Schritt das Pivotelement/die Pivotelemente an und markieren Sie die Zeichen, die mit dem Pivotelement verglichen werden. [3 Punkte]




## Lösung




Bei der LZ**78**-Kompression wird ein \$-terminierter Text $T$ über dem Alphabet $\Sigma$ wie folgt in $z$ Faktoren zerlegt:

- $f_0 = \varepsilon$ ist ein leerer Faktor ($\varepsilon$ bezeichnet das leere Wort),
- $T = f_1 f_2 \dots f_z$ ist die Konkatenation aller $z$ nicht-leeren Faktoren und
- wenn die ersten $i < z$ Faktoren die ersten $j$ Zeichen des Textes faktorisieren, also $f_1 \dots f_i = T[1..j)$, dann ist der nächste Faktor $f_{i+1}$ das *längste* Präfix von $T[j..n]$, so dass

$$\exists k \in [0,i], \alpha \in \Sigma \cup \{\$\} \colon f_{i+1} = f_k \alpha.$$

Der neue Faktor entspricht also einem bereits existierenden Faktor, an den ein weiteres Zeichen angefügt wurde.

**Beispiel:** `aaabbbaabb$` wird in folgende Faktoren zerlegt: `a|aa|b|bb|aab|b$`.

**b.** Geben Sie die LZ**77**-Faktorisierung von `aaabbbaabb$` an. Verwenden Sie die gleiche Notation wie im obenstehenden Beispiel. [1 Punkt]

## Lösung

`a|aa|b|bb|aabb|$`

**c.** Geben Sie einen Text an, bei dem die LZ78-Faktorisierung mehr Faktoren hat als die LZ77-Faktorisierung. Begründen Sie kurz. [2 Punkte]

## Lösung

$T = \texttt{aaaaaa\$}$. LZ77: `a|aaaaa|$` (3 Faktoren) und LZ78: `a|aa|aaa|$` (4 Faktoren).

**d.** Zeigen Sie, dass für einen Text der Länge $n$ über einem konstanten Alphabet die LZ78-Faktorisierung bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt. [4 Punkte]

## Lösung

Sei $T = \texttt{a}^{n-1}\$$. LZ78: $\texttt{a}|\texttt{aa}|\texttt{aaa}|\ldots|\texttt{a}^{k-1}|\texttt{a}^{k}|\texttt{a}^{n-k(k+1)/2}\$$ für das größte $k$, so dass $k(k+1)/2 < n$ gilt. Somit ist die Anzahl der Faktoren $\Theta(\sqrt{n})$.

Lösungsvorschlag

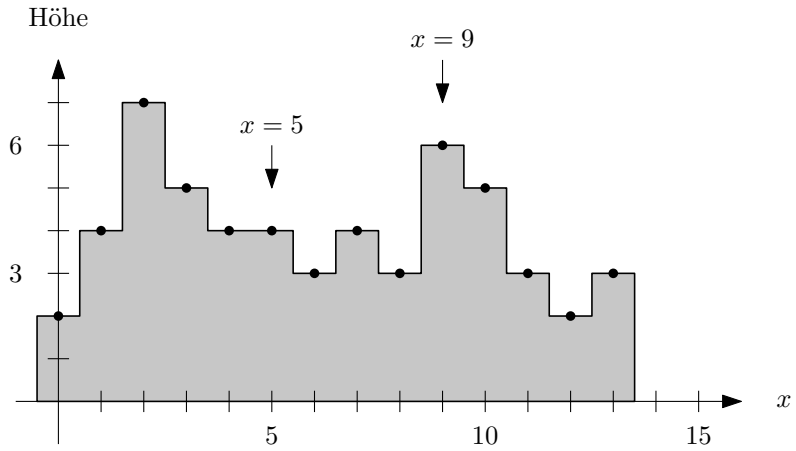
EK

ZK

**Aufgabe 6.** Geometrische Algorithmen: Isolation von Bergen [11 Punkte]

Die Isolation eines Berggipfels ist der kürzeste Abstand zu einem Höhenwert, der echt größer ist als der Gipfel. In dieser Aufgabe betrachten wir 2D Berge. Die Höhen der Berge seien gegeben als eine nach $x$-Wert sortierte Liste von ganzzahligen Höhenwerten der Länge $n$ (einer für jede $x$-Position). Zwischen den Höhenwerten sei als Stufenfunktion interpoliert. Wir definieren die Isolation des höchsten Gipfels als $\infty$.

**a.** Gegeben sei die folgende diskretisierte 2D Gebirgskette. Die Achsenbeschriftungen seien in Kilometern gegeben. Geben Sie die Isolation der Gipfel bei $x = 5$ und $x = 9$ an. [2 Punkte]




| Gipfel | Isolation |
|---|---|
| $x = 5$ | |
| $x = 9$ | |

## Lösung

| Gipfel | Isolation |
|---|---|
| $x = 5$ | 2 |
| $x = 9$ | 7 |

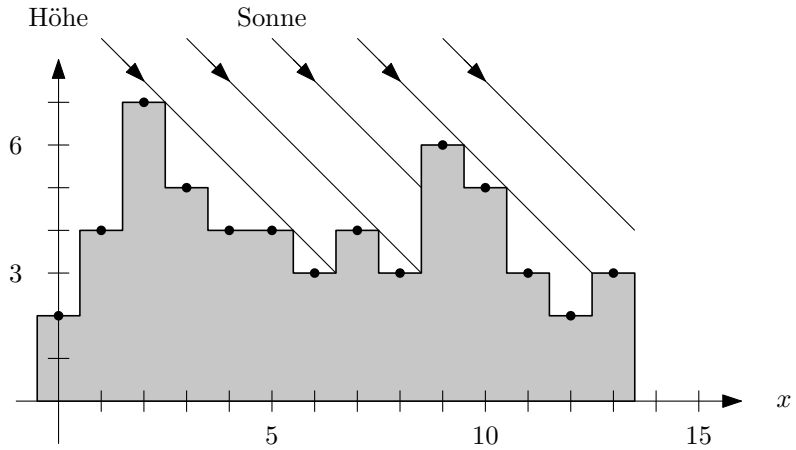
**b.** Geben Sie einen Algorithmus an, der in Zeit $\mathcal{O}(n)$ die Isolation jedes Berges aus einer gegebenen 2D-Bergkette bestimmt. Begründen Sie die Laufzeit des Algorithmus.

*Hinweis*: Eventuell ist mehr als ein Durchlauf durch die Eingabe notwendig. [5 Punkte]

## Lösung

- Führe einen linearen Sweep von links nach rechts aus. Behalte dabei einen Stack an Punkten. Sei $p$ der nächste besuchte Höhenwert.
- Solange $p$ größer ist als der oberste Wert auf Stack, führe `pop` aus.
- Für jeden gepoppten Punkt $q$ ist $p$ der nächst höhere Punkt nach rechts. Speichere für $q$ die entsprechende $x$-Distanz ab.
- Dann pushe $p$ auf den Stack und gehe weiter zum nächsten Punkt.
- Führe einen analogen Sweep auch von rechts nach links aus. Dadurch ist für jeden Punkt sein nächst höherer Punkt in beiden Richtungen bekannt.
- In einem letzten Durchlauf bestimme für jeden Punkt die kleinere der beiden $x$-Distanzen, um die Isolation zu erhalten.
- Es wird 3 mal durch alle $n$ Eingabewerte iteriert. In jedem der ersten beiden Durchläufe wird jeder Punkt einmal auf den Stack gelegt und einmal wieder entfernt. Die Berechnung der Minima ist in linearer Zeit möglich. Dadurch ergibt sich insgesamt die geforderte lineare Laufzeit.

**c.** Es scheine nun die Sonne parallel in einem 45°-Winkel auf eine solche Gebirgskette. Wir wollen herausfinden, welche der Berge im Schatten liegen und welche in der Sonne. Dabei ist ein Berg genau dann beschattet, wenn sein Mittelpunkt beschattet ist.
Im folgenden Beispiel ist z.B. $x = 5$ beschattet und $x = 7$ liegt in der Sonne.




Geben Sie einen Algorithmus an, der in Zeit $\mathcal{O}(n)$ für die Berge an jedem $x$-Wert bestimmt, ob dieser in der Sonne oder im Schatten liegt. Begründen Sie die Laufzeit des Algorithmus. [4 Punkte]

## Lösung

- Idee: Iteriere von links nach rechts über die Berge. Speichere dabei immer einen Berg als den aktuellen Verschatter.
- Bestimme für jeden neuen Berg, ob dieser im Schatten liegt. Ist der $x$-Abstand zum aktuellen Verschatter größer als die Höhendifferenz zum Verschatter, liegt der neue Berg in der Sonne. Ist der Abstand kleiner oder gleich, liegt der neue Berg im Schatten.
- Falls der aktuelle Berg in der Sonne liegt, wird der aktuelle Verschatter auf diesen aktualisiert.
- Es ist ein einzelner linearer Durchgang durch das Eingabe-Array nötig. In jedem Schritt werden nur Differenzen verglichen und möglicherweise der Verschatter aktualisiert. Es ergibt sich eine lineare Laufzeit.