1. Klausur zur Vorlesung
Algorithmentechnik
Wintersemester 2005/2006

Lösung!

Beachten Sie:

- Bringen Sie den Aufkleber mit Ihrem Namen und Matrikelnummer auf diesem Deckblatt an und beschriften Sie jedes Aufgabenblatt mit Ihren Namen und Matrikelnummer.
- Schreiben Sie die Lösungen auf die Aufgabenblätter und Rückseiten. Zusätzliches Papier erhalten Sie bei Bedarf von der Aufsicht.
- Zum Bestehen der Klausur sind **20** der möglichen **60** Punkte hinreichend.
- Es sind keine Hilfsmittel zugelassen.

Aufgabe	Mögliche Punkte						Erreichte Punkte					
	a	b	c	d	e	Σ	a	b	c	d	e	Σ
1	2	-	-	-	-	2		-	-	-	-	
2	3	3	2	-	-	8				-	-	
3	4	2	-	-	-	6			-	-	-	
4	3	3	2	-	-	8				-	-	
5	1	2	3	-	-	6				-	-	
6	1	1	1	3	3	9						
7	1	3	-	-	-	4			-	-	-	
8	2	3	-	-	-	5			-	-	-	
9	12x1					12						
Σ						60						

Problem 1: 2 Punkte

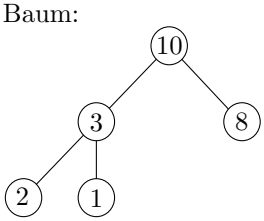
Gegeben seien die folgenden HEAP-Operationen. Geben Sie nach Schritt 5 und nach Schritt 8 jeweils den aktuellen Zustand des HEAPs als Baum und als Array A an.

1. Einfügen von 8
2. Einfügen von 3
3. Einfügen von 10
4. Einfügen von 2
5. Einfügen von 1

Lösung: Array:

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

Baum:

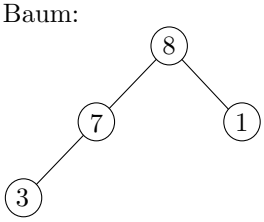


6. Einfügen von 7
7. Löschen von $A[4]$ (mit Wiederherstellung der HEAP-Eigenschaft)
8. Löschen des Maximums (mit Wiederherstellung der HEAP-Eigenschaft)

Lösung: Array:

1	2	3	4	5	6	7
8	7	1	3			

Baum:



Problem 2: 3+3+2=8 Punkte

a) Gegeben seien die folgenden Rekursionsformeln. Ordnen Sie diese nach ihrem asymptotischen Verhalten in aufsteigender Reihenfolge (die "schnellste" zuerst) und begründen Sie Ihre Ordnung mit Hilfe geeigneter Schranken. Es gelte jeweils $T_i(1) = c$, für $c > 0$, $i = 1, \ldots, 3$.

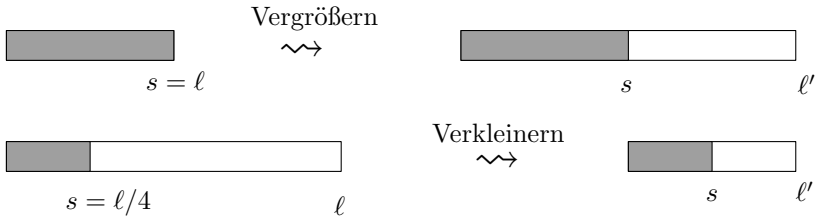
(i) $T_1(n) = 4T_1(n/4) + n$

(ii) $T_2(n) = T_2(n-1) + 2n$

(iii) $T_3(n) = 2T_3(n-1) + n$

Lösung: $T_1 <_{\text{asymptotisch}} T_2 <_{\text{asymptotisch}} T_3$
Für T_1 gilt nach Mastertheorem: $T_1(n) \in \Theta(n \log n)$
Für T_2 gilt: $T_2(n) = \sum_{i=0}^{n-2} 2(n-i) = \sum_{j=2}^{n} 2 * j \in \Theta(n^2)$
Für T_3 gilt: $T_3(n) = \sum_{i=0}^{n-2} 2^{i+1}(n-i) >= 2^{n-1} \in \Omega(2^n)$

b) Ein STACK soll mit Hilfe eines Arrays implementiert werden. Er soll theoretisch beliebig groß werden können und trotzdem nicht zu viel Speicher verschwenden. s bezeichne die Anzahl der momentan im STACK enthaltenen Elemente, ℓ die Länge des Arrays. Zu Beginn sind $s = 0$ und $\ell = 1$. (Das als letztes abgelegte Element steht an Stelle s im Array.)



Falls nach der Ausführung von PUSH kein Platz mehr im Array frei ist ($s = \ell$), so wird ein neues Array der Länge $\ell' = 2\ell$ angelegt, die Elemente des alten Arrays dorthin kopiert und der Platz für das alte Array freigegeben. Falls $s < \ell/4$ nach Ausführen von POP gilt, wird ein neues Array der Größe $\ell' = \ell/2$ reserviert, die Elemente des alten Arrays dorthin kopiert und das alte Array freigegeben.

Analysieren Sie die amortisierte Worst-case-Laufzeit einer Folge von n PUSH- und POP-Operationen. [*Hinweis*: Sie können annehmen, dass das Anlegen und Freigeben des Arrays keine Kosten hat, aber das Kopieren eines Elements Kosten 1 verursacht.]

Lösung: Wir zählen nach der Buchungsmethode für jede PUSH- oder POP-Operation 2 Kopiervorgänge. Dabei wird für eine PUSH-Operation Kosten von 3 veranschlagt und für eine POP-Operation Kosten 2. Falls bei PUSH das Array vergrößert wird, liegen zwischen dem letzten Vergrößern oder Verkleinern und dem aktuellen PUSH mindestens $\ell/2$ weitere PUSHs, die keine Veränderung der Größe des Arrays zur Folge hatten. Die Kosten ℓ des Vergrößerns sind damit schon bezahlt worden.

Wird bei einem POP das Array verkleinert, muss es vorher mindestens $\ell/4$ POPs ohne Größenveränderung gegeben haben. Die Kosten von $\ell/4$ für das Kopieren sind damit ebenfalls bezahlt.

Folglich sind die Kosten für n Operationen höchstens $3n$.

c) Begründen Sie kurz, warum die Laufzeit in $\Omega(n^2)$ ist, wenn das Array in der vorherigen Teilaufgabe schon bei $s < \ell/2 - 10$ statt bei $s < \ell/4$ auf die Größe $\ell/2$ verkleinert wird.

Lösung: Hier kann der Aufwand $\Omega(n^2)$ sein, wie folgendes Beispiel zeigt:

- $n/2$ PUSHs gefolgt von
- $\left.\begin{array}{l}11 \text{ POPs} \\ 11 \text{ PUSHs}\end{array}\right\} n/44$ mal.

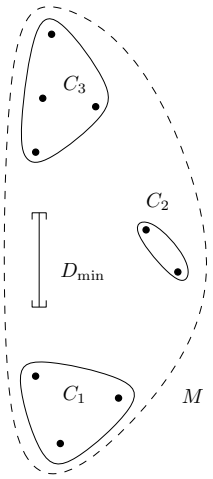
Der Aufwand für diese n Operationen ist mindestens $n/2 + n/44 \cdot (n/2 - 11) \in \Omega(n^2)$.

Problem 3: Bottom-Up Clustern von Dokumenten 4+2=6 Punkte

Gegeben sei eine Menge M von n verschiedenen Dokumenten und dazu eine symmetrische *Ähnlichkeitsfunktion* $d : M \times M \to \mathbb{R}$. Die Menge M soll nun wie folgt in "ähnliche" Teilmengen $C_1, \ldots, C_k \subset M$ partitioniert werden.

Zu Beginn bildet jedes Element von M eine eigene, einelementige Teilmenge $C_i, i = 1, \ldots, n$. Dann werden iterativ immer die beiden ähnlichsten Teilmengen verschmolzen. Die Ähnlichkeit zweier Teilmengen C_p, C_q ist $\max_{v \in C_p, w \in C_q} d(v, w)$. Der Algorithmus soll abbrechen, falls nur noch eine Teilmenge vorhanden ist oder kein Paar von Teilmengen mehr mindestens Ähnlichkeit $D_{\min}$ hat.

Die Ähnlichkeiten aller Dokumentenpaare seien vorberechnet und in einem Array $D = D[1], \ldots, D[n^2]$ in nicht-aufsteigender Reihenfolge vorsortiert. Die beiden zu $D[\ell]$ korrespondierenden Dokumente seien in $V[\ell]$ und $W[\ell]$ gespeichert, es gilt also $D[\ell] = d(V[\ell], W[\ell])$ für alle $\ell \in \{1, \ldots, n^2\}$.



(a) Schreiben Sie unter Verwendung einer geeigneten Datenstruktur aus der Vorlesung einen Algorithmus für das beschriebene Verfahren in Pseudocode. Nehmen Sie dazu an, die zur Datenstruktur gehörenden Operationen stünden zur Verfügung.

(b) Analysieren Sie die asymptotische Laufzeit Ihres Algorithmus und begründen Sie Ihre Antwort. (Sie brauchen nicht die Laufzeiten der Operationen der verwendeten Datenstruktur zu begründen.)

Lösung:

(a)

Algorithmus 1 : BOTTOM-UP-CLUSTERN

Eingabe : Menge M, Array D, Wert $D_{\min}$, Array V, Array W
Ausgabe : Keine
Seiteneffekte : Datenstruktur, welche die Partition verwaltet

```
1  n ← |M|
2  Für v ∈ M
3      MAKESET(v)
4  ℓ ← 1
5  Solange ℓ ≤ n² und D[ℓ] ≥ D_min wiederhole
6      v ← V[ℓ]
7      w ← W[ℓ]
8      p ← FIND(v)
9      q ← FIND(w)
10     Wenn p ≠ q
11         UNION(p, q)
12     ℓ ← ℓ + 1
```

(b) Es wird maximal n mal MAKESET, $2 \cdot n^2$ mal FIND und n mal UNION aufgerufen, jeweils angewendet auf eine Menge von maximal n Elementen. Die Laufzeit der FIND-Operationen dominiert die Laufzeit der MAKESET- und UNION-Operationen sowie die der restlichen Schritte des Algorithmus, die in $O(n^2)$ laufen. Als Folge von $2n^2$ Operationen in UNION-FIND ergibt sich somit eine Laufzeit von $O(n^2 \cdot G(n))$.

Problem 4: 3+3+2=8 Punkte

Es sei folgender Algorithmus gegeben, von dem nicht bekannt ist, was er berechnet:

Algorithmus 2 :

Eingabe : Ungerichteter, zusammenhängender Graph $G = (V, E)$, Kantengewichte $c\colon E \to \mathbb{R}^+$
Ausgabe : ???

```
1  Berechne einen beliebigen aufspannenden Baum T von G
2  S ← E \ T (Menge der Nichtbaumkanten)
3  R ← ∅
4  Solange S ≠ ∅ wiederhole
5  |  Wähle e ∈ S und entferne e aus S
6  |  e' ← Kante mit maximalem Gewicht auf C_e
7  |  R ← R ∪ e'
8  |  Wenn e ≠ e'
9  |  |  T ← (T \ e') ∪ e
10 Gib E \ R zurück
```

C_e sei dabei der durch T und e definierte Fundamentalkreis in G.

a) Zeigen Sie, dass die Laufzeit von Algorithmus 2 in $O(nm)$ ist. Geben Sie dazu für jeden Schritt eine Schranke für die Laufzeit an. Geben Sie außerdem kurz an, wie die Schritte 1 und 6 implementiert werden können.

Lösung:

1 $O(m + n)$, implementierbar durch BFS

2 $O(m)$

3 $O(1)$

4 Wird $|S| \in O(m)$ mal durchlaufen.

5 $O(1)$

6 Sei $e = \{v, w\}$. BFS in T, startend bei v bis w erreicht ist, um den Kreis C_e mit maximal n Kanten zu finden: $O(n)$

7 $O(1)$

8 $O(1)$

9 $O(1)$

10 $O(m)$

b) Was berechnet Algorithmus 2? Begründen Sie.

Lösung: Algorithmus 2 berechnet einen MST. Jeder Schleifendurchlauf ist eine Anwendung der "roten Regel". R ist die Menge der roten Kanten. In jedem Schleifendurchlauf wird die Kante mit maximalem Gewicht eines ungefärbten Kreises zu rot gefärbt, da T stets nur ungefärbte Kanten enthält. Der Algorithmus terminiert, wenn $m-(n-1)$ Kanten rot sind. Öfter kann die rote Regel nicht angewendet werden, also muss $E \setminus R = T$ am Ende ein (grüner) minimaler aufspannender Baum sein.

c) Gegeben sei nun ein Graph $G = (V, E)$ mit Kantengewichten $c\colon E \to \mathbb{R}^+$ und MST T.
Geben Sie einen Linearzeit-Algorithmus an, der zu einer nicht in G enthaltenen Kante e mit Kantengewicht $c(e) \in \mathbb{R}^+$ einen MST in $G' := (V, E')$ mit $E' = E \cup \{e\}$ zurückgibt und begründen sie dessen Korrektheit.

Lösung:

Algorithmus 3 : Algorithmus $\mathcal{B}$

Eingabe : Ungerichteter Graph $G = (V, E)$, Kantengewichte $c\colon E \to \mathbb{R}^+$, minimaler aufspannender Baum T von G, Kante $e \notin E$
Ausgabe : Minimaler aufspannender Baum T' in $G \cup e$

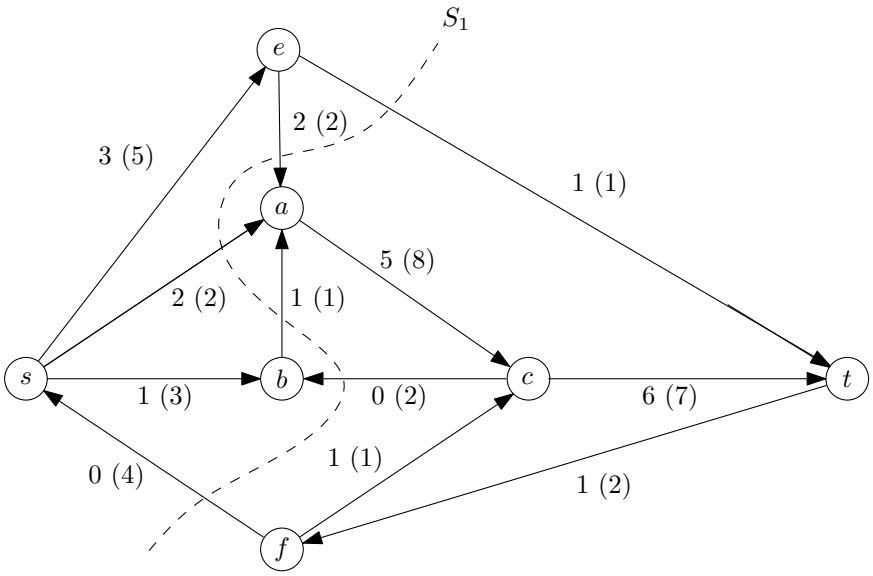
```
1 e' ← Kante maximalen Gewichts in C_e
2 Wenn e' ≠ e
3 |  T' ← T \ e' ∪ e
4 sonst
5 |  T' ← T
6 Gib T zurück
```

T kann wieder als ungefärbter Baum aufgefasst werden, die restlichen Kanten sind rot. Eine weiteres Anwenden der roten Regel auf C_e muss also zum korrekten Ergebnis führen.

Problem 5: 1+2+3=6 Punkte

(a) Zeichnen Sie in den untenstehenden Graphen einen minimalen s-t-Schnitt ein (die Zahlen in Klammern geben die Kantenkapazitäten an, die Zahlen vor den Klammern einen s-t-Fluss). Argumentieren Sie, warum Ihr Schnitt minimal ist.

Lösung:



Die Werte vor den Klammern geben einen Fluss an. Dieser hat den Wert 6, was auch die Kapazität des Flusses ist. Aus dem Max-Flow-Min-Cut-Theorem folgt die Minimalität des Schnittes.

(b) Beschreiben Sie die Arbeitsweise eines effizienten Algorithmus, der aus einem gegebenen Netzwerk $(D; s; t; c)$ mit maximalem s-t-Fluss f einen minimalen s-t-Schnitt $(S, V \setminus S)$ berechnet. (Eine exakte Formulierung in Pseudocode ist nicht erforderlich.)

Lösung:

Algorithmus 4 : Algorithmus $\mathcal{S}$

Eingabe : Netzwerk $(D = (V, E); s; t; c)$, $V = \{1, \ldots, n\}$, $E = \{1, \ldots m\}$, Kapazitäten $c[1 \ldots m]$, maximaler Fluss $f[1 \ldots m]$

Ausgabe : Minimaler Schnitt $(S, V \setminus S)$

1 Führe einen Breitensuche durch, die von s startet und nur Vorwärtskanten e_v mit $f(e) < c(e)$ und Rückwärtskanten e_r mit $f(e) > 0$ (i.e. Residualkanten) benutzt. Die Menge aller erreichten Knoten bildet S.

2 Gib $(S, V \setminus S)$ zurück;

Die Kapazität der Kanten, die von S nach $V \setminus S$ führen ist gerade $c(S, V \setminus S)$. Alle diese Kanten sind saturiert, also gilt $c(S, V \setminus S) = w(f)$.

(c) Gegeben sei folgende Erweiterung des Max-Flow-Problems: In dem Netzwerk D gebe es nun k Quellen $s_1, \ldots, s_k$ und ℓ Senken $t_1, \ldots, t_\ell$. Die Flusserhaltungsbedingungen müssen für alle Knoten

außer $s_1, \ldots, s_k, t_1, \ldots, t_\ell$ erfüllt sein, ansonsten ist alles wie im gewöhnlichen Flussproblem. Der Wert des Flusses sei

$$w(f) := \sum_{i=1}^{k} \left(\sum_{(s_i,v)\in E} f(s_i, v) - \sum_{(v,s_i)\in E} f(v, s_i) \right) .$$

Wie kann dieses Flussproblem mit Hilfe eines gewöhnlichen Flussproblems gelöst werden? Zeigen Sei die Korrektheit Ihres Ansatzes.

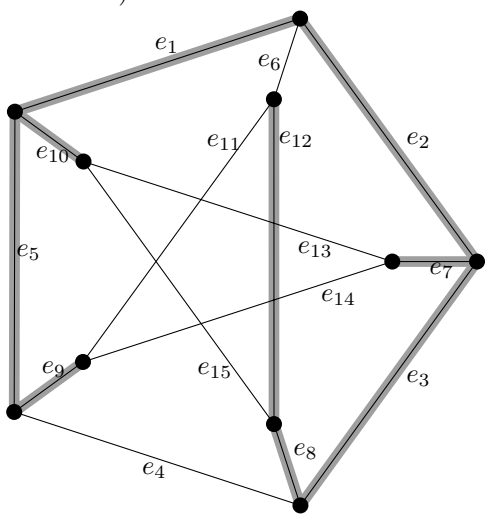
Lösung: Wir führen zwei neue Knoten s und t ein und Kanten von s zu jeder Quelle s_i und von jeder Senke t_j zu t. Die neuen Kanten haben jeweils Kapazität $\sum_{e\in E} c(e)$. f' kann als Fluss in D aufgefasst werden und umgekehrt ergibt jeder Fluss in D einen s-t-Fluss in dem erweiterten Netzwerk. Für den maximalen s-t-Fluss f' in dem neuen Netzwerk gilt

$$w(f') = \sum_{1}^{k} (s, s_i) = \sum_{1}^{k} \left(\sum_{(s_i,v)\in E} f(s_i, v) - \sum_{(v,s_i)\in E} f(v, s_i) \right)$$

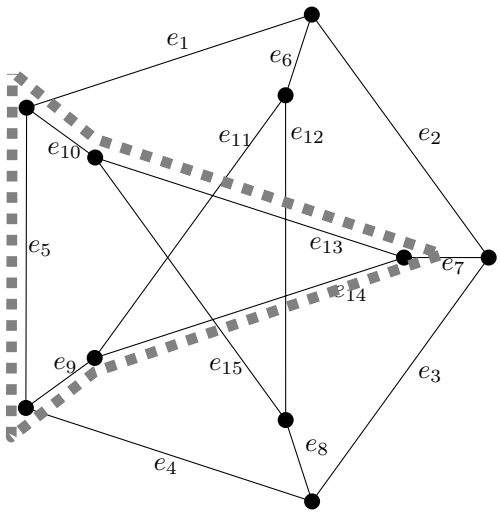
Folglich entspricht der maximale s-t-Fluss einen maximalen Fluss in D.

Problem 6: Kreisbasen von PETE 1+1+1+3+3=9 Punkte

Betrachten Sie den Petersen-Graph PETE, der im Folgenden zweimal abgebildet ist. Das Gewicht jeder Kante sei 1. (Benutzen Sie zum Benennen von Kanten nur den Index i nicht e_i. Das spart Zeit und ist besser lesbar.)



PETE mit Baum T.



PETE mit Kreis C.

(a) Welche Dimension hat der Kreisraum dieses Graphen?

(b) Erstellen Sie eine Fundamentalbasis $\mathcal{F}$ von PETE zu dem dick grau eingezeichneten aufspannenden Baum T in der linken Abbildung.

(c) Stellen Sie den dick gestrichelt eingezeichneten Kreis C in der rechten Abbildung mit Hilfe Ihrer Basis $\mathcal{F}$ aus Teilaufgabe (b) dar.

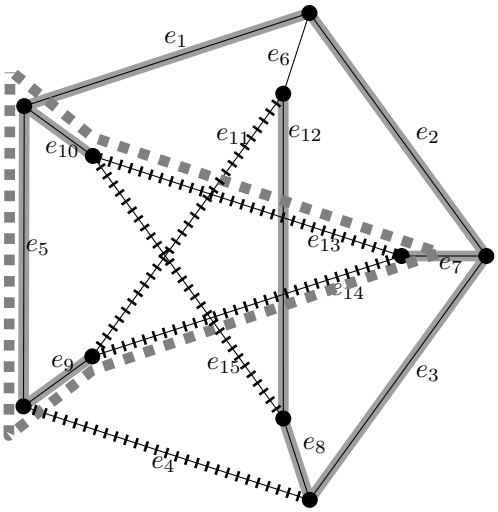
(d) Ist Ihre Basis $\mathcal{F}$ eine minimale Kreisbasis? Wenn ja, begründen Sie diese Behauptung. Wenn nicht, geben Sie eine minimale Kreisbasis an und begründen Sie, warum diese eine minimale Kreisbasis ist.

(e) Sei $\mathcal{A}$ ein Algorithmus, der eine minimale Kreisbasis approximiert, indem er die Fundamentalbasis zu einem minimalen aufspannenden Baum berechnet. Zeigen Sie, dass man im Allgemeinen keine relative Gütegarantie angeben kann für das Gewicht einer Kreisbasis, die von Algorithmus $\mathcal{A}$ berechnet wird.

Lösung:

(a) Es gilt $|E| = 15, |V| = 10$, somit folgt für die Dimension dim des Kreisraumes: $\dim = m - n + \mathcal{K} = 15 - 10 + 1 = 6$.

(b)



Der Baum zur Fundamentalbasis ist dick grau gezeichnet, die Nichtbaumkanten dick grau gepunktet.

$$\begin{aligned}\mathcal{F} = (&C_4 = \{1,2,3,4,5\},\\ &C_6 = \{2,3,6,8,12\},\\ &C_{11} = \{1,2,3,5,8,9,11,12\},\\ &C_{13} = \{1,2,7,10,13\},\\ &C_{14} = \{1,2,5,7,9,14\},\\ &C_{15} = \{1,2,3,8,10,15\})\end{aligned}$$

(c) $(0,0,0,1,1,0)$ oder $C_{13} \oplus C_{14}$

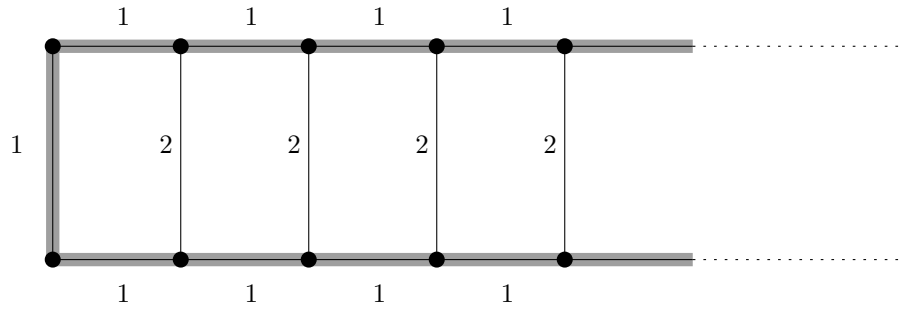
(d) Nein, da folgende Basis geringeres Gewicht hat:
MCB: $\mathcal{F}' = (C_4, C_6, C_{13}, C_{13} \oplus C_{14}, C_6 \oplus C_{11}, C_{15} \oplus C_{13})$
Die Elemente von $\mathcal{F}'$ sind linear unabhängig, da sie in ihrer Darstellung durch $\mathcal{F}$ jeweils ein exklusives Element von $\mathcal{F}$ haben. Wegen $|\mathcal{F}'| = 6$ ist $\mathcal{F}'$ sogar Kreisbasis. In Pete gibt es keinen Kreis mit weniger als 5 Kanten und jede Kante hat Gewicht 1. Da jeder Kreis in $\mathcal{F}'$ 5 Kanten hat, ist $\mathcal{F}'$ also sogar MCB.

(e) In folgendem Graphen I_n ist der eindeutige MST dick grau eingezeichnet. Die eindeutige MCB besteht aus den einzelnen Kästchen. Sei n die Anzahl der Kanten mit Gewicht 2, dann gilt $w(\mathcal{A}) \in \Omega(n^2)$ und $w(MCB) \in O(n)$, und somit:

$$\mathcal{R}_{\mathcal{A}}(I_n) = \frac{w(\mathcal{A})}{w(MCB)} \in \Omega(n)$$

$\mathcal{R}_{\mathcal{A}}(I_n)$ kann also beliebig groß werden.
(Genauer:

$$\mathcal{R}_{\mathcal{A}}(I_n) = \frac{w(\mathcal{A})}{w(MCB)} = \frac{\sum_{i=1}^{n}(2+1+i\cdot 2)}{6\cdot n-1} = \frac{3\cdot n + n\cdot(n+1)}{6\cdot n-1} = \frac{n^2+4\cdot n}{6\cdot n-1} \quad \longrightarrow \infty \ )$$



Problem 7: LP

1+3=4 Punkte

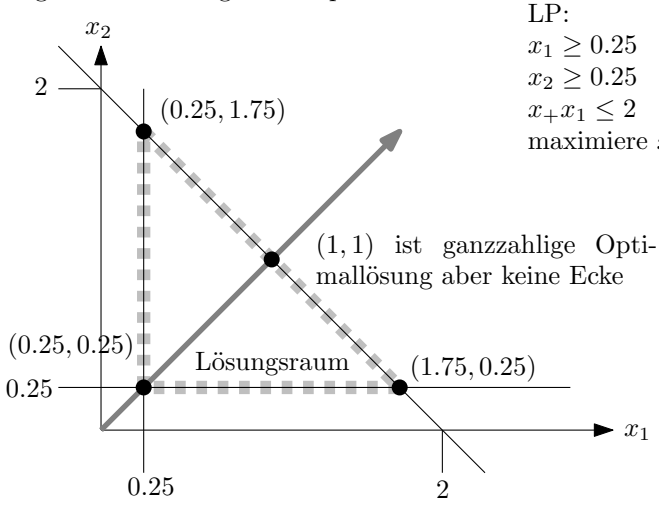
Sei ein lineares Programm P gegeben, für das eine *ganzzahlige* Lösung gefordert wird und dessen zulässiger Bereich ein Polytop ist.

(a) Angenommen es sei bekannt, dass alle Ecken des zulässigen Bereichs ganzzahlige zulässige Lösungen von P sind. Begründen Sie, warum man P mit dem Simplexverfahren lösen kann. *Lösung:* Da der zulässige Bereich ein Polytop ist, ist er beschränkt. Somit ist mindestens eine Ecke Optimallösung, und da alle Ecken ganzzahlig sind, ist jede solche Ecke Lösung von P. Das Simplexverfahren findet eine optimale Ecke und löst somit P.

(b) Angenommen es sei lediglich bekannt, dass es eine ganzzahlige Optimallösung von P gibt. Warum kann man P dann im Allgemeinen nicht mehr mit dem Simplexverfahren lösen? Zeigen Sie dies zusätzlich anhand eines konkreten Beispiels (keine Standardform erforderlich).

Lösung: Angenommen die Menge der Optimallösungen ist eine ℓ-Seite mit $\ell > 0$, dann ist im Polytop des zulässigen Bereichs zwar auch mindestens eine Ecke eine Optimallösung, aber nicht notwendigerweise eine ganzzahlige. Die Optimierungsrichtung steht senkrecht auf diese Seite. Sind

die einzigen ganzzahligen Optimallösungen (und somit die einzigen Lösungen für P) keine Ecken, so findet das Simplexverfahren diese Lösungen nicht, da dieses Verfahren nur Ecken absucht. Siehe folgende Abbildung als Beispiel:



LP:
$x_1 \geq 0.25$
$x_2 \geq 0.25$
$x_+x_1 \leq 2$
maximiere $x_1 + x_2$

Problem 8: 2+3 = 5 Punkte

Der Approximationsalgorithmus FIRST FIT DECREASING (FFD) für das Problem BIN PACKING sortiert zuerst die Elemente einer Instanz nach nicht-aufsteigenden Gewichten und führt dann FIRST FIT aus.

(a) Geben Sie eine Instanz I zu BIN PACKING an, für die gilt

$$\mathrm{FFD}(I) > \mathrm{OPT}(I) \ .$$

Lösung: $I = (e_1, e_2, e_3, e_4, e_5, e_6)$

$c(e_1) = 1/2$
$c(e_2) = c(e_3) = c(e_4) = 1/3$
$c(e_5) = c(e_6) = 1/4$

$\mathrm{FFD}(I) = 3 > \mathrm{OPT}(I) = 2$

(b) Zeigen Sie, dass es zu jedem gegebenen $k \in \mathbb{N}$ eine Instanz I_k gibt, für die $\mathrm{OPT}(I_k) \geq k$ und

$$\mathrm{FFD}(I_k) \geq \frac{7}{6}\mathrm{OPT}(I_k)$$

gilt.

Hinweis: Man kommt mit zwei verschiedenen Gewichtswerten aus.

Lösung: (Ausführlicher als verlangt.) Sei $K \in \mathbb{N}$ so gewählt, dass $K \geq k$ und K sowohl durch 2 als auch durch 3 teilbar ist. Betrachte die Instanz I_k bestehend aus K Elementen mit Gewichten 0.4 und $2 \cdot K$ Elementen mit Gewichten 0.3. I_k enthält also $3 \cdot K$ Elemente. Der Algorithmus FFD verteilt die K Elemente von Gewichten 0.4 in die ersten $\frac{K}{2}$ Bins, in denen dann jeweils nur noch Platz für

Elemente von Gewicht höchstens 0.2 übrig ist. Deswegen werden die restlichen Elemente von Gewicht 0.3 auf weitere $\frac{2K}{3}$ Bins verteilt, die dann zu jeweils 0.9 gefüllt sind. Insgesamt ergibt sich

$$\mathrm{FFD}(I_k) = \frac{K}{2} + \frac{2K}{3} = \frac{7}{6}K$$

In einer optimalen Lösung enthält jedes Bin ein Element von Gewicht 0.4 und zwei Elemente von Gewicht 0.3, also insgesamt $\mathrm{OPT}(I_k) = K \geq k$ Bins. Es gibt also Instanzen I_k mit beliebig großem $\mathrm{OPT}(I_k) \geq k$, für die alle $\mathrm{FFD}(I_k) \geq \frac{7}{6}\mathrm{OPT}(I_k)$ gilt.

Problem 9: $12 \times 1 = 12$ Punkte

Kreuzen Sie für folgende Aussagen an, ob diese wahr oder falsch sind.

Hinweis: Für jede richtige Antwort gibt es einen Punkt, für jede falsche Antwort wird ein Punkt abgezogen. Es wird keine negative Gesamtpunktzahl für diese Aufgabe geben.

	Wahr	Falsch
$O(n \cdot (\log\log n) \cdot \sqrt{n}) \cap \Theta(n^{7/4}) = \emptyset$.	X	
Ein absteigend sortiertes Array repräsentiert einen Heap.	X	
Die Zahlenfolge $(12, 8, 2, 6, 4, \pi, 0, 3)$ ist ein 3-HEAP.		X
Der Algorithmus von Kruskal ist für dichte Graphen besser geeignet als der Algorithmus von Prim.		X
Wenn zu jedem Kantengewicht in G die gleiche positive Zahl $\lambda \in \mathbb{R}$ addiert wird, dann bleibt der minimale Schnitt gleich.		X
Wenn zu jedem Kantengewicht in G die gleiche positive Zahl $\lambda \in \mathbb{R}$ addiert wird, dann bleibt der minimale Spannbaum gleich.	X	
Gegeben ist ein Flussnetzwerk mit ganzzahligen Kapazitäten. Dann ist jeder maximale Fluss auf jeder Kante ganzzahlig.		X
Für einen Graphen $G = (V, E)$ und eine Kreisbasis $\mathcal{B}$ von G gilt stets: $\forall e \in E : \exists B \in \mathcal{B} : e \in B$.		X
Falls $\mathcal{P} \neq \mathcal{NP}$, dann gibt es einen polynomialen Algorithmus zur Lösung eines beliebigen LP.	X	
Falls $\mathcal{P} \neq \mathcal{NP}$, dann gibt es keinen polynomialen Algorithmus zur Lösung eines beliebigen ganzzahligen LP.	X	
QUICKSORT mit zufälliger Pivotelementwahl ist ein randomisierter Las-Vegas-Algorithmus.	X	
Sei Π ein Minimierungsproblem und $\mathcal{A}_\epsilon$ eine Familie von Approximationsalgorithmen für $\epsilon > 0$, die polynomial in $\lvert I\rvert$ und $1/\epsilon$ sind und für die gilt $\exists k > 1 : \forall I : \mathcal{A}_\epsilon(I) \leq k \cdot OPT(I)$. Dann ist $\mathcal{A}_\epsilon$ ein FPAS für Π.		X