Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn

Klausur Algorithmen und Datenstrukturen

24. August 2022, 14:00 -17:00

Name: .....

Matrikel Nr.: .....

Unterschrift: .....

Erst öffnen wenn die Klausuraufsicht die Erlaubnis erteilt!

- Legen Sie Ihren **Studentenausweis** sichtbar vor sich.
- Schreiben Sie Ihren **Namen** und Ihre **Matrikelnummer** an die vorgesehenen Stellen.
- **Unterschreiben Sie** diese Seite um zu bestätigen, dass Sie die Fragen ohne unerlaubte Hilfsmittel beantwortet haben und die Klausuraufsicht über (gesundheitliche) Probleme informiert haben.
- Dies ist eine **Open Book** Klausur weshalb alle gedruckten und handgeschriebenen Materialien erlaubt sind. **Elektronische Hilfsmittel** sind **nicht** erlaubt.
- Schreiben Sie **lesbar** und nur mit **dokumentenechten** Stiften. Benutzen Sie **keine rote** Farbe und **keinen Bleistift**!
- Es wird **nur eine Lösung pro Aufgabe** gewertet. Vergewissern Sie sich, dass Sie zusätzliche Lösungen durchstreichen, andernfalls wird die schlechteste Lösung gewertet.
- Detaillierte Schritte können Ihnen zu **Teilpunkten** verhelfen falls das Endergebnis falsch ist.
- Die Schlüsselwörter **Zeigen Sie...**, **Beweisen Sie...**, **Begründen Sie...** oder **Leiten Sie ... her** zeigen an, dass Sie Ihre Antwort sorgfältig und gegebenenfalls formal begründen müssen.
- Die Schlüsselwörter **Geben Sie ... an** zeigen an, dass Sie lediglich die geforderte Antwort und keine Begründung liefern müssen.
- Die folgenden Regeln gelten **überall**, außer sie werden explizit außer Kraft gesetzt.
- Bei Laufzeitfragen ist nur die **asymptotische Laufzeit** notwendig.
- Wenn Sie einen Algorithmus angeben sollen, so können Sie Pseudocode angeben. Eine **ausreichend detaillierte** Beschreibung der Funktionsweise Ihres Algorithmus genügt jedoch.
- Algorithmen aus der Vorlesung **können als Blackbox** verwendet werden.
- **Lesen Sie jede Aufgabe sorgfältig** durch und stellen Sie sicher dass Sie diese verstanden haben!
- Falls Sie eine Frage zur Aufgabenstellung haben, geben Sie der Klausuraufsicht ein **Handzeichen**.
- Schreiben Sie Ihren Namen auf **alle Blätter**!
- 50 Punkte sind ausreichend zum **Bestehen der Klausur**.
- 100 Punkte sind ausreichend für die **beste Note**.

Aufgabe	1	2	3	4	5	6	7	8	**Total**
Maximum	22	12	12	15	15	12	12	20	120
Punkte									

Aufgabe 1: Kurze Fragen *(22 Punkte)*

(a) Im folgenden geht es darum die Schlüssel $k_1 = 12, k_2 = 15$ und $k_3 = 8$ mittels der in der Vorlesung beschriebenen Variante von *Cuckoo Hashing* in einer Hashtabelle der Größe $m = 7$ zu speichern. Geben seien hierzu die beiden Hashfunktionen $h_1(x) = 2 \cdot x + 1 \mod m$ und $h_2(x) = x + 3 \mod m$. Geben Sie in der linken Tabelle den Zustand der Hashtabelle nachdem k_1 und k_2 eingefügt wurden an. In der rechten Tabelle soll der finale Zustand - also nach hinzufügen von k_3 - zu sehen sein. *(4 Punkte)*

0	1	2	3	4	5	6

0	1	2	3	4	5	6

(b) Sei $f : \mathbb{R} \to \mathbb{R}$ eine stetige Funktion mit $f(0) = -1$ und $f(1) = 1$. Beschreiben Sie einen Algorithmus, welcher in $O(\log n)$ Zeit eine Nullstelle von f bis auf $1/n$ genau findet. Ihr Algorithmus soll also eine Zahl x ausgeben, so dass f im Intervall $(x - \frac{1}{n}, x + \frac{1}{n})$ eine Nullstelle hat. Wir nehmen dabei an, dass das Auswerten von f an einer Stelle x (d.h. die Berechnung von $f(x)$) konstante Zeit beansprucht.

Begründen Sie die Korrektheit Ihres Algorithmus. *(6 Punkte)*

(c) Angenommen es gäbe eine Prioritätswarteschlange H deren Operationen folgende Laufzeit hätten: Sowohl `create` wie auch `insert`, `getMin` und `decreaseKey` haben konstante Laufzeit $O(1)$, während ein Aufruf von `deleteMin` eine Laufzeit von $O(\log n)$ hat (wenn n Elemente in der Datenstruktur gespeichert sind).

Welche asymptotische Laufzeit hätte Dijkstras Algorithmus (in Abhängigkeit der Kantenzahl m und der Knotenzahl n) wenn man H als zugrundeliegende Prioritätswarteschlange benutzt. Begründen Sie. *(5 Punkte)*

(d) Beweisen oder Widerlegen Sie folgende Aussage: Heapsort ist *stabil*. *(7 Punkte)*

Hinweise: Gehen Sie davon aus, dass Heapsort die Array-Implementierung aus der Vorlesung für den binären Heap benutzt. Ein Sortieralgorithmus heißt stabil, falls die ursprüngliche Reihenfolge von Elementen mit gleichem Schlüssel beibehalten bleibt. Bsp: Besteht das Array initial aus folgenden key-value-Paaren $[(3, a), (1, r), (1, b)]$*, ist* $[(1, r), (1, b), (3, a)]$ *eine stabile Sortierung, jedoch* $[(1, b), (1, r), (3, a)]$ *nicht.*

Musterlösung

(a)

			15	12		
0	1	2	3	4	5	6

	12		8	15		
0	1	2	3	4	5	6

(b)

$a = 0$
$b = 1$
for i in $\{1, \ldots, \log n\}$ **do**
  $x = \frac{a+b}{2}$
  if $f(x) > 0$ **then**
    $b = x$
  else if $f(x) < 0$ **then**
    $a = x$
  else
    return x
return $\frac{a+b}{2}$

Wir haben als Schleifeninvariante, dass $a < b$ und $f(a) < 0$ und $f(b) > 0$. Am Ende hat f also eine Nullstelle in (a, b). Das Intervall (a, b) schrumpft in jeder Iteration um die Hälfte. Am Ende gilt also $b - a < 1/2^{\log n} = 1/n$.

(c) Dijkstra ruft n mal insert/getMin/delMin auf, decreaseKey wird höchstens m mal aufgerufen (und 1 create, das aber für die Laufzeit keine Rolle spielt). Es folgt also folgende Laufzeit: $O(n \cdot \log n + m)$.

(d) Die Aussage ist falsch. Gegenbeispiel wäre das Array $[2, 1, 2]$. Der binäre Heap hätte nach dem Einfügen aller Werte die 1 in der Wurzel, die erste 2 als linkes Kind und die zweite 2 als rechtes Kind. Nun wird die 1 mittels delMin gelöscht. Beim Löschen wird das Wurzelelement mit dem letzten Eintrag, sprich der rechten 2, getauscht und anschließend das neue letzte Element gelöscht. Nun ist diejenige 2 im Wurzelknoten die ursprünglich die rechte 2 war, da die MinHeap Eigenschaft gilt, muss nicht repariert werden. Somit wird diese 2 später vor der anderen im (sortierten) Array stehen, was nicht der ursprünglichen Reihenfolge entspricht.

Aufgabe 2: Landau-Notation *(12 Punkte)*

(a) Gegeben folgende 5 Funktionen in Abhängigkeit von $n \in \mathbb{N}$. Geben Sie eine Sortierung der Funktionen bezüglich der $\mathcal{O}$-Notation an, sprich, für zwei aufeinanderfolgende Funktionen f, g in dieser Sortierung gilt jeweils $f(n) \in \mathcal{O}\big(g(n)\big)$. Die angegebenen Beziehungen müssen weder bewiesen noch begründet werden. *(4 Punkte)*

- $a(n) = 3 \cdot n^2 \cdot \log_2(n) + 4$
- $b(n) = 2^{2\log_2(n)}$
- $c(n) = a(n)/\sqrt{n+1}$
- $d(n) = \log_2(n!)$
- $e(n) = \sqrt{n^5}$

(b) Beweisen oder Widerlegen Sie anhand der Definition der Landau-Notation: $10 \cdot n^9 \in \Omega(n^{10})$ *(4 Punkte)*

(c) Beweisen oder Widerlegen Sie entweder anhand der Definition der Landau-Notation oder anhand der Grenzwert-Charakterisierung: $\sum_{i=0}^{n} (2 \cdot (i+1)) \in o(n^3)$ *(4 Punkte)*

Musterlösung

(a) $d(n), c(n), b(n), a(n), e(n)$

(b) Die Aussage gilt nicht. Angenommen es gäbe ein c und n_0 so dass die Aussage gilt, dann

$$\begin{aligned} 10n^9 &\geq cn^{10} \\ 10 &\geq cn \\ \frac{10}{c} &\geq n \end{aligned}$$

Wähle also z.B. $n = \max\{10/c, n_0\} + 1$ um einen Widerspruch zur Annahme zu konstruieren.

(c) Die Aussage stimmt:

$$\sum_{i=0}^{n} (2 \cdot (i+1)) = 2 \sum_{i=0}^{n+1} i = 2\frac{(n+1)(n+2)}{2} = n^2 + 3n + 2 \leq 6n^2$$

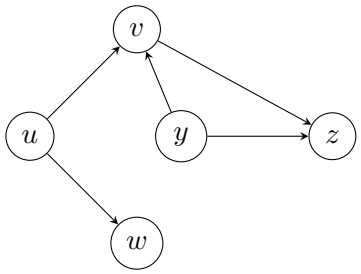
Da $6n^2 \leq cn^3 \Leftrightarrow n \geq 6/c$ (für beliebiges c), wähle $n_0 = \lceil 6/c \rceil$, dann gilt für alle $n \geq n_0$: $\sum_{i=0}^{n} (2 \cdot (i+1)) \leq 6n^2 \leq cn^3$.

Alternativ: Berechne Summe wie oben, dann:

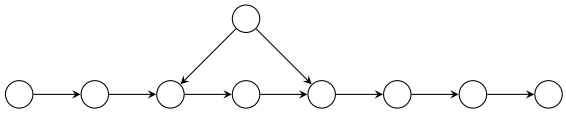
$$\lim_{n\to\infty} \frac{\sum_{i=0}^{n} (2 \cdot (i+1))}{n^3} = \lim_{n\to\infty} \frac{n^2 + 3n + 2}{n^3} = \lim_{n\to\infty} \frac{1}{n} + \frac{3}{n^2} + \frac{2}{n^3} = 0$$

Aufgabe 3: Topologische Sortierung *(12 Punkte)*

(a) Geben Sie eine topologische Sortierung des folgenden Graphen an. *(3 Punkte)*



(b) Wie viele topologische Sortierungen gibt es für den folgenden Graphen? *(3 Punkte)*



(c) Gegeben seien zwei gerichtete Graphen ohne Zyklen (DAGs) $G = (V, E)$ und $G' = (V, E')$ mit gleicher Knotenmenge V, so dass G ein Teilgraph von G' ist, d.h. $E \subseteq E'$. Auf welchem Graphen gibt es mehr Möglichkeiten, die Knotenmenge topologisch zu sortieren? Begründen Sie Ihre Antwort. *(6 Punkte)*

Musterlösung

(a) Z.B. u, w, y, v, z

(b) 3

(c) Eine topologische Sortierung eines Graphen bleibt gültig, wenn man Kanten aus dem Graph entfernt. Daher ist jede topologische Sortierung in G' auch eine in G und es gibt mehr Möglichkeiten, G topologisch zu sortieren.

Aufgabe 4: Rabin-Karp mit Quersummen *(15 Punkte)*

In dieser Aufgabe arbeiten wir mit einer modifizierten Variante des aus der Vorlesung bekannten Rabin-Karp Algorithmus. Sei $x = x_1x_2x_3 \dots x_r$ die Dezimaldarstellung (also zur Basis 10) von x. Dann definieren wir die Quersumme $\overline{x} := \sum_{i=1}^{r} x_i$ (Beispiel: $\overline{1927} = 1 + 9 + 2 + 7 = 19$).

Der hier verwendete Rabin-Karp benutzt als Hashfunktion $h(x) = \overline{x} \mod M$, wobei M ein beliebiger Modulus sein kann. Stimmen Hashwert von Muster und Textausschnitt überein, wird wie auch in der Vorlesung, mithilfe von `TestPosition` auf tatsächliche Gleichheit geprüft.

(a) Sei der Suchtext $T = 12518230051251$ der Länge $n = 14$ und das Muster $P = 251$ der Länge $m = 3$. Die zu benutzende Hashfunktion sei $h(x) = \overline{x} \mod 5$. Geben Sie für alle $s \in \{0, 1, \dots, 11\}$ den Hashwert $h(T[s, s+1, s+2])$ an. *(9 Punkte)*

Hinweis: $T[s, s+1, s+2]$ ist das Teilwort aus T an Positionen s, $s+1$ und $s+2$. Bsp. $T[3, 4, 5] = 182$.

(b) Wie oft muss bei der Ausführung von Rabin-Karp die Funktion `TestPosition` für das Beispiel aus der a) aufgerufen werden? *(2 Punkte)*

(c) Geben Sie eine Möglichkeit an um $h(T[s + 1, s + 2, \dots, s + m])$ mit der Hashfunktion $h(x) = \overline{x} \mod M$ in konstanter Zeit zu berechnen, unter der Bedingung, dass der Hashwert des vorherigen Teilstrings $h(T[s, s + 1, \dots, s + m - 1])$ bekannt ist. *(4 Punkte)*

Hinweis: Die Laufzeit einer einzelnen Addition, Subtraktion, Multiplikation, Division oder Modulorechnung wird als konstant angenommen.

Musterlösung

(a) Tabelle:

s	$T[s, s+1, s+2]$	$h(T[s, s+1, s+2])$
0	125	3
1	251	3
2	518	4
3	182	1
4	823	3
5	230	0
6	300	3
7	005	0
8	051	1
9	512	3
10	125	3
11	251	3

(b) Da $h(P) = 3$, folgt dass es 7 `TestPosition` aufrufe gibt.

(c) Da $h(T[s, s + 1, \dots, s + m - 1]) = (T[s] + T[s + 1] + ... + T[s + m - 1]) \mod M$, muss man nur $T[s]$ subtrahieren und $T[s + m]$ addieren, sprich:

$$h(T[s + 1, s + 1, \dots, s + m]) = (T[s + m] - T[s] + h(T[s, s + 1, \dots, s + m - 1])) \mod M$$

Aufgabe 5: Längster Gemeinsamer Teilstring *(15 Punkte)*

Gegeben seien zwei Strings A, B der Länge n über den Zeichen $\{0, 1\}$. Wir möchten die Länge des längsten *gemeinsamen* Teilstrings von A und B berechnen. Ein Teilstring ist dabei eine zusammenhängende Teilfolge von Zeichen von A beziehungsweise B.

Geben Sie einen Algorithmus an der dieses Problem in Zeit $\mathcal{O}(n^2)$ löst und begründen Sie die Laufzeit. *(15 Punkte)*

Alternativ können Sei einen Algorithmus angeben der dieses Problem in Zeit $\mathcal{O}(n^3)$ löst. Dafür erhalten Sie maximal 5 Punkte.

Hinweise: Es gibt eine Lösung die Dynamisches Programmieren Bottom-Up benutzt. Merken Sie Sich für zwei Indizes i, j nur die maximale Länge eines gemeinsamen Teilstrings von A, B der jeweils beim Index i, j in A bzw. B endet.

Musterlösung

(a) Für jeden zusammenhängenden Teilstring in A führen wir eine Suche nach diesem Teilstring in B aus. Es gibt $\mathcal{O}(n^2)$ solche Teilstrings in A (denn jeder zusammenhängende Teilstring ist durch einen Start- und End-index definiert) und jede Suche nach einem Teilstring dauert $\mathcal{O}(n)$. Insgesamt also $\mathcal{O}(n^3)$.

(b) Sei $\ell(i, j)$ die Länge des längsten gemeinsamen Teilstrings von $A[0..i]$, $B[0..j]$, der bei $A[i]$ und $B[j]$ endet. Im Basisfall setzen wir $\ell(i, 0) = 1$ falls $A[i] = B[0]$ sonst $\ell(i, 0) = 0$. Analog setzen wir $\ell(1, j) = 1$ falls $A[0] = B[j]$ sonst $\ell(0, j) = 0$. Rekursiv setzen wir für $i, j \geq 1$

$$\ell(i,j) := \begin{cases} \ell(i-1, j-1) + 1, & \text{falls } A[i] = B[j] \\ 0, & \text{sonst.} \end{cases}$$

Wir berechnen nun diese Funktion komplett, bottom-up mittels dynamischer Progammierung. Wir speichern die Ergebnisse in einem 2-dimensionalen Array $L[i, j]$.

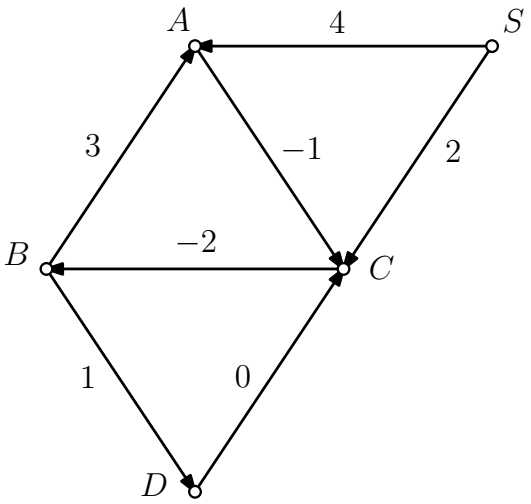
Algorithm 1 `compute-bottom-up`(A, B)

```
L ← Neues 2-dimensionales n × n-Array
for k = 0 to n − 1 do                                   ▷ Basisfälle
    L[0, k] ← (A[0] = B[k])
    L[k, 0] ← (A[k] = B[0])
for i = 1 to n − 1 do                                   ▷ Bottom Up Berechnung von L
    for j = 1 to n − 1 do
        if A[i] = B[j] then
            L[i, j] ← L[i−1, j−1] + 1
        else
            L[i, j] ← 0
m ← 0
for i = 1 to n − 1 do                                   ▷ grössten Wert berechnen
    for j = 1 to n − 1 do
        if L[i, j] > m then m ← L[i, j]
return m
```

Wir berechnen L gemäss der obigen Rekursion. Zum Schluss iterieren wir durch das gesamte Array L und merken uns den grössten Wert den wir antreffen und geben ihn aus. Beide Schritte dauern jeweils $\mathcal{O}(n^2)$.

Aufgabe 6: Kürzeste Pfade *(12 Punkte)*

Gegeben sei der folgende gerichtete Graph mit $n = 5$ Knoten als Schaubild.



(a) Hat dieser Graph einen negativen Zyklus? Falls ja markieren Sie die Kanten von einem negativen Zyklus (Umkreisung oder Färbung der Kanten). *(2 Punkte)*

(b) Führen Sie auf diesem Graph den *Bellman-Ford* Algorithmus mit Startknoten S aus und tragen Sie *nach jeder* Iteration der äußeren Schleife die bisher berechneten Distanzen von S *in der vorgegeben Tabelle* ein (siehe Lösungsblatt). *(10 Punkte)*

Wichtig: Iterieren Sie in der inneren Schleife des Bellman-Ford Algorithmus die Kanten aufsteigend nach Größe der Gewichte, d.h., stets in der Reihenfolge $-2, -1, 0, 1, 2, 3, 4$ *!*

Musterlösung

(a) Die Kanten $(C, B), (B, D), (D, C)$ bilden einen negativen Zyklus.

(b) Lösungstabelle:

	$\delta(S,S)$	$\delta(S,A)$	$\delta(S,B)$	$\delta(S,C)$	$\delta(S,D)$
Initial	0	∞	∞	∞	∞
$i = 1$	0	4	∞	2	∞
$i = 2$	0	3	0	2	1
$i = 3$	0	3	0	1	1
$i = 4$	0	2	-1	1	0

Aufgabe 7: Mystischer Algorithmus *(12 Punkte)*

Sei $G = (V, E)$ ein *ungerichteter* Graph mit n Knoten. Betrachten Sie den folgenden Algorithmus `Mystery` gegeben als Pseudocode, welcher als Eingabe G sowie einen Knoten $s \in V$ erhält.

Algorithm 2 `Mystery`(G, s)

```
D ← neues Dictionary
D[s] ← 0
for jeden Knoten u ∈ V \ {s} do
    D[u] ← −1
u ← s; i ← 1
while Solange es eine Kante {u, v} ∈ E gibt, so dass D[v] = −1 ist do
    v ← beliebiger Nachbar von u mit D[v] = −1
    D[v] ← D[u] + i
    u ← v; i ← i + 1
```

(a) Wie viele Iterationen kann die while Schleife in Abhängigkeit von n maximal durchlaufen? *(6 Punkte)*

(b) Was ist der größte mögliche Wert in D nach Ausführung von `Mystery`(G, s) asymptotisch in Abhängigkeit von n? *(6 Punkte)*

Musterlösung

(a) $n - 1$

(b) $\Theta(n^2)$

Aufgabe 8: Minimale Spannbäume *(20 Punkte)*

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph, der durch Adjazenzlisten gegeben ist. Angenommen wir wissen, dass G genau einen Zyklus enthält.

(a) Beschreiben Sie einen Algorithmus mit Laufzeit $O(|V|)$, welcher den Zyklus von G ausgibt. Die Ausgabe sollte also die Form $v_1, v_2, \ldots, v_k, v_1$ haben, wobei $v_1, v_2, \ldots, v_k$ paarweise verschieden sind mit $\{v_i, v_{i+1}\} \in E$ für $i = 1, \ldots, k-1$ und $\{v_1, v_k\} \in E$. Begründen Sie die Laufzeit. *(8 Punkte)*

Sei nun $G = (V, E, w)$ ein zusammenhängender, ungerichteter, gewichteter Graph, C ein Zyklus in G und $e \in C$ eine schwerste Kante im Zyklus, d.h. $w(e) \geq w(e')$ für alle $e' \in C$.

(b) Zeigen Sie, dass es einen minimalen Spannbaum von G gibt, welcher e nicht enthält. *(8 Punkte)*

Sei nun $G = (V, E, w)$ ein zusammenhängender, ungerichteter, gewichteter Graph, der durch Adjazenzlisten gegeben ist. Angenommen wir wissen, dass G genau einen Zyklus enthält.

(c) Geben Sie einen Algorithmus an, der in Laufzeit $O(|V|)$ einen minimalen Spannbaum von G berechnet. *(4 Punkte)*

Hinweis: Sie dürfen Teil (a) und (b) auch dann nutzen, wenn Sie sie nicht gelöst haben. Wenn es hilfreich für Sie ist, dürfen Sie annehmen, dass zusätzlich eine Adjazenzmatrix (mit den Gewichten der Kanten als Einträge) gegeben ist.

Musterlösung

(a) Man startet ausgehend von einem beliebigen Knoten s ein DFS. Sei P der Pfad grauer Knoten beginnend bei s, gespeichert als doppelt verkettete Liste. Wir updaten P nach jedem Schritt, d.h. wir hängen einen Knoten hinten an, wenn er grau gefärbt wird und löschen den letzten Knoten, wenn er schwarz gefärbt wird. Sobald man eine Rückwärtskante entdeckt, d.h. man von einem Knoten u ausgehend einen grauen Knoten v entdeckt, geben wir zunächst v und danach P von hinten nach vorne aus, bis wir in P wieder auf v treffen.

Mit Breitensuche: Jeder Knoten merkt sich, von welchem Knoten aus er markiert wurde. Wenn man dann von u aus auf einen Knoten v stößt, der bereits von w aus markiert wurde, geht man von v aus sowohl über u als auch über w den Pfad der Vorgänger zurück Richtung Wurzel s, bis man auf einen gemeinsamen Knoten x stößt. Der gesuchte Kreis ist dann der Pfad von x über u nach v zurück über w nach x.

Die Laufzeit wird dominiert durch den DFS/BFS welcher Laufzeit $O(m+n)$ hat. Da $m = n$ gilt, ist die Laufzeit $O(n)$.

(b) Sei T ein MST von G welcher $e = \{u, v\}$ enthält. $T \setminus \{e\}$ besteht dann aus zwei Zusammenhangskomponenten A (alle Knoten die von u aus erreichbar sind) und B (alle Knoten die von v aus erreichbar sind). Es gibt eine Kante $e' \in C$, welche eine Schnittkante von (A, B) ist, d.h. beide Komponenten verbindet. $(T \setminus \{e\}) \cup \{e'\}$ ist also ein Spannbaum und ist wegen $w(e') \leq w(e)$ nicht schwerer als T.

(c) Wir nutzen den Algorithmus von (a) um den Zyklus auszugeben. Dann gehen wir durch den Zyklus und löschen eine schwerste Kante. Der MST besteht dann aus allen übrigen Kanten.