Name:
Vorname:
Matrikelnummer:

Klausur-ID:

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dennis Hofheinz

16. März 2017

Klausur Algorithmen I

Aufgabe 1.	Kleinaufgaben	13 Punkte
Aufgabe 2.	Binäre Heaps	11 Punkte
Aufgabe 3.	Minimale Spannbäume	13 Punkte
Aufgabe 4.	Kürzeste Wege	12 Punkte
Aufgabe 5.	Dynamische Programmierung	11 Punkte

Bitte beachten Sie:

- Bringen Sie den Aufkleber mit Ihrem Namen, Ihrer Matrikelnummer und Ihrer Klausur-ID oben links auf dem Deckblatt an.
- Merken Sie sich Ihre Klausur-ID und schreiben Sie auf **alle Blätter** der Klausur und Zusatzblätter Ihre Klausur-ID und Ihren Namen.
- Die Klausur enthält 21 Blätter. Die Bearbeitungszeit beträgt 120 Minuten.
- Die durch Übungsblätter gewonnenen Bonuspunkte werden erst nach Erreichen der Bestehensgrenze hinzugezählt. Die Anzahl der Bonuspunkte entscheidet nicht über das Bestehen der Klausur.
- Als Hilfsmittel ist ein beidseitig handbeschriebenes DIN-A4 Blatt zugelassen.
- Bitte kennzeichnen Sie deutlich, welche Lösung gewertet werden soll. Bei mehreren angegebenen Möglichkeiten wird jeweils die schlechteste Alternative gewertet.

Aufgabe	1	2	3	4	5	Summe
max. Punkte	13	11	13	12	11	60
Punkte						

Bonuspunkte:	Summe:	**Note:**

Aufgabe 1. Kleinaufgaben [13 Punkte]

Bearbeiten Sie die folgenden Aufgaben.

a. Ist die Höhe eines (a,b)-Baumes mit n Einträgen immer in $O(\log n)$? Begründen Sie Ihre Antwort kurz. (Es gelte $b \geq 2a - 1$ und $a \geq 2$.) [1 Punkt]

b. Geben Sie die Kreis-Eigenschaft, die zur Berechnung minimaler Spannbäume genutzt wurde, wieder. Sie dürfen davon ausgehen, dass alle Kantengewichte paarweise verschieden sind. [1 Punkt]

c. Nennen Sie einen Vor- und einen Nachteil von Hashtabellen mit verketteten Listen gegenüber Hashtabellen mit linearer Suche. [1 Punkt]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

d. Nennen Sie einen Vor- und einen Nachteil von doppelt verketteten Listen gegenüber einfach verketteten Listen. [1 Punkt]

e. Welche Eigenschaften machen einen Graphen zu einem DAG? [1 Punkt]

f. Beweisen oder widerlegen Sie: $(\log_{10} n)^n \in \mathrm{O}\left(n^{\log_2 n}\right)$. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

g. Ist die folgende Aussage korrekt? „Die Laufzeit von Dijkstras Algorithmus liegt für alle in der Vorlesung genannten Implementierungen im Worst Case in $\Omega((m+n)\log n)$", wobei m die Anzahl der Kanten und n die Anzahl der Knoten des Graphs sei. Falls ja, begründen Sie kurz. Falls nein, geben Sie die verbesserte Laufzeit an und benennen Sie, wie diese erreicht werden kann. [1 Punkt]

h. Sortieren Sie diese Folge mittels der Array-Implementierung von K-Sort (*KSortArray*) aus der Vorlesung aufsteigend: $\langle 5,2,9,3,5,1,8,2,0,10\rangle$. Es sei $K=8$, und der Schlüssel eines Elements sei der Rest bei der Division durch 8, d.h. $key(x)=x \mod 8$.
Geben Sie den Inhalt des Arrays c (des Hilfsarrays) an, unmittelbar *bevor* die ersten Elemente ins Ausgabearray geschrieben werden. Geben Sie weiterhin den Inhalt des Arrays b (des Ausgabearrays) und des Arrays c an, nachdem der Algorithmus beendet ist. [3 Punkte]

Hilfsarray c unmittelbar bevor ins Ausgabearray geschrieben wird:

0	1	2	3	4	5	6	7

Hilfsarray c nachdem der Algorithmus beendet ist:

0	1	2	3	4	5	6	7

Ausgabearray b nachdem der Algorithmus beendet ist:

1	2	3	4	5	6	7	8	9	10

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

i. Gegeben sei die folgende Rekurrenz:

$$T(n) = \begin{cases} 1 & \text{für } n = 1 \\ 4T(\frac{n}{2}) + n^2 & \text{für } n > 1 \end{cases}.$$

Zeigen Sie durch vollständige Induktion, dass

$$T(n) = n^2(\log_2(n) + 1)$$

gilt, falls $n = 2^k$ für ein $k \in \mathbb{N}_0$.

Hinweis: Es gilt $\log_2(2) = 1$. [2 Punkte]

Aufgabe 2. Binäre Heaps [11 Punkte]

Hinweis: In dieser Aufgabe handelt es sich bei Heaps stets um Min-Heaps.
a. Gegeben sei folgender binärer Heap im impliziter Darstellung. Zeichnen Sie den Heap in Baumdarstellung. [1 Punkt]

1	3	6	4	8	9	7

b. Sei $A[1 \ldots n]$ ein Array, das einen impliziten binären Heap darstellt. Nennen Sie die Heap-Eigenschaft bezüglich A. [1 Punkt]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 2

c. Führen Sie auf folgendem Array die Methode *buildHeapBackwards* aus der Vorlesung aus um einen binären Heap aufzubauen. Geben Sie dabei den Array nach jeder Veränderung an. Einträge, die sich nicht verändern, dürfen Sie dabei leer lassen. In diesem Fall werten wir die letzte Zahl, die in der Spalte darüber vorkommt. [2 Punkte]

9	7	5	1	8	3	6

Platz für Korrekturen:

d. Führen Sie auf folgendem binärem Heap die Methode *deleteMin* aus der Vorlesung aus. Geben Sie dabei das Array nach jeder Veränderung an. Einträge, die sich nicht verändern, dürfen Sie dabei leer lassen. In diesem Fall werten wir die letzte Zahl, die in der Spalte darüber vorkommt. [2 Punkte]

1	3	5	4	9	6	7	8	11	12

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 2

Platz für Korrekturen:

1	3	5	4	9	6	7	8	11	12

e. Führen Sie auf folgendem binärem Heap die Methode *insert*(1) aus der Vorlesung aus um die Zahl 1 in den Heap einzufügen. Geben Sie dabei das Array nach jeder Veränderung an. Einträge, die sich nicht verändern, dürfen Sie dabei leer lassen. In diesem Fall werten wir die letzte Zahl, die in der Spalte darüber vorkommt. [2 Punkte]

2	3	7	4	6	10	9	5	8	11	13	

Platz für Korrekturen:

(weitere Teilaufgaben auf den nächsten Blättern)

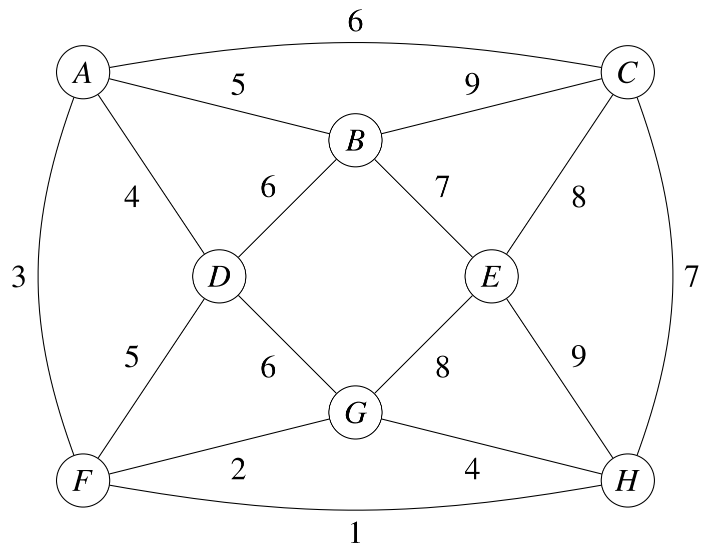
Fortsetzung von Aufgabe 2

f. Nennen Sie einen Vorteil von Heapsort gegenüber Mergesort und einen Vorteil von Mergesort gegenüber Heapsort. [1 Punkt]

g. Sie möchten aus dem Algorithmus Heapsort aus der Vorlesung nun einen stabilen Sortieralgorithmus konstruieren. Erläutern Sie, was einen stabilen Sortieralgorithmus auszeichnet und wie Heapsort derart abgewandelt werden kann, dass ein stabiler Sortieralgorithmus entsteht. Hierbei soll die grundsätzliche Funktionsweise und Laufzeit von Heapsort erhalten bleiben, Ihr Algorithmus darf aber $O(n)$ zusätzlichen Speicherplatz verwenden, wobei n die Eingabegröße bezeichnet. [2 Punkte]

Aufgabe 3. Minimale Spannbäume [13 Punkte]

a. Führen Sie den Algorithmus von Kruskal an folgendem Graphen aus. Geben Sie die Kanten in der Reihenfolge an, in der Sie vom Algorithmus ausgewählt werden. Geben Sie die Kante zwischen Knoten u und v dabei als $\{u,v\}$ oder (u,v) an. [2 Punkte]



Kantenreihenfolge: ____________________

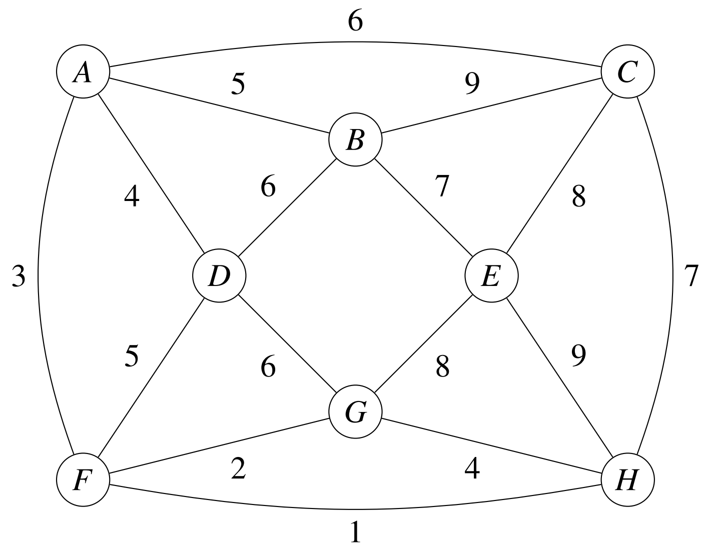
b. Welche beiden Beschleunigungstechniken wurden in der Vorlesung für die Union-Find-Datenstruktur eingeführt? Nennen und beschreiben Sie diese kurz. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 3

Gegeben sei nun ein Algorithmus, der wie folgt einen Spannbaum eines gewichteten ungerichteten Graphen berechnet. Der Algorithmus bekommt dabei nicht sofort Zugriff auf alle Knoten, sondern bekommt nach und nach Knoten und die zugehörigen Kanten gegeben, die den neuen Knoten mit dem bisherigen Teilgraphen verbinden. Die Knoten werden so nacheinander gegeben, dass stets mindestens eine Kante zwischen dem neuen Knoten und dem bisherigen Teilgraphen existiert. Sobald der Algorithmus einen neuen Knoten bekommt, wählt er von allen Kanten, die den neuen Knoten mit dem bisherigen Teilgraphen verbinden, diejenige mit dem geringsten Gewicht aus und fügt sie dem aktuellen Spannbaum hinzu.

c. Führen Sie den Algorithmus an folgendem Graphen aus. Die Knoten werden dem Algorithmus dabei in alphabetischer Reihenfolge (beginnend bei Knoten A) gegeben. Beachten Sie, dass der Algorithmus jeweils nur Kanten betrachtet, die den neuen Knoten mit dem bisherigen Teilgraphen verbindet. Geben Sie die Kanten in der Reihenfolge an, in der Sie vom Algorithmus ausgewählt werden. Geben Sie die Kante zwischen Knoten u und v dabei als $\{u,v\}$ oder (u,v) an. [2 Punkte]



Kantenreihenfolge: ____

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 3

d. Geben Sie einen Graphen mit Reihenfolge der Knoten an, so dass der Algorithmus keinen minimalen Spannbaum berechnet. [2 Punkte]

e. Nach der Vorlesung kann die minimale Kante eines Schnittes jeweils für einen minimalen Spannbaum verwendet werden. Der Algorithmus wählt stets die kleinste Kante aus dem Schnitt zwischen dem neu hinzuzufügenden Knoten und dem bisherigen Teilgraphen aus. Warum impliziert dies nicht, dass der Algorithmus immer einen MST berechnet? [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

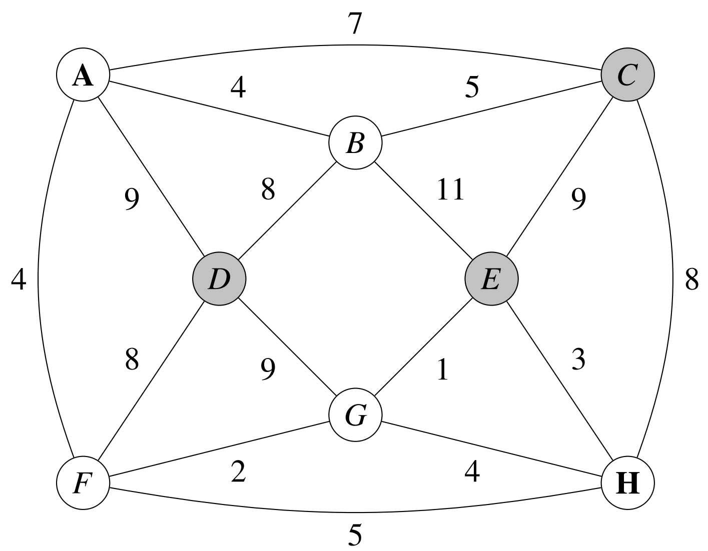
Fortsetzung von Aufgabe 3

f. Gibt es für jeden beliebigen gewichteten ungerichteten Graphen eine Reihenfolge der Knoten, so dass der Algorithmus den minimalen Spannbaum berechnet? Begründen Sie Ihre Antwort. Geben Sie gegebenfalls an, wie man die Reihenfolge der Knoten bestimmen kann. Beachten Sie, dass dem Algorithmus stets nur Knoten gegeben werden können, die durch eine Kante mit dem alten Teilgraph verbunden sind. Sie dürfen davon ausgehen, dass alle Kantengewichte paarweise verschieden sind. [3 Punkte]

Aufgabe 4. Kürzeste Wege [12 Punkte]

Ein Paketunternehmen, das sich auf Haus-zu-Haus-Paketzustellung spezialisiert hat, möchte die Auslieferung optimieren. Bei Abholung eines Paketes vom Starthaushalt bringt der Paketdienst das Paket zunächst zu einem Paketzentrum. Von dort bringt ein Zulieferer das Paket zum gewünschten Zielhaushalt. Dabei soll der Weg insgesamt minimiert werden. Es ist also jeweils ein kürzester Weg vom Start zum Ziel gesucht, der über mindestens ein Paketzentrum verläuft. Der Weg darf dabei sowohl über mehrere Paketzentren, als auch über andere Haushalte laufen. Dieses Problem sei wie folgt durch einen Graphen modelliert. Sei $G = (V, E)$ ein gewichteter ungerichteter Graph mit $n = |V|$ Knoten. Sei $P \subseteq V$ die Menge der $k = |P|$ Paketzentren, und die übrigen $V \setminus P$ Knoten die Menge der $n - k$ Haushalte. Weiter gebe es für zwei Knoten $u, v \in V$ genau dann eine Kante $\{u, v\} \in E$ mit Gewicht c, wenn der Paketbote über einen Weg der Länge c direkt von u nach v (und v nach u) fahren kann.

a. Gegeben sei folgender Graph. Hierbei sei $P = \{C, D, E\}$ (im Graphen grau markiert) die Menge der $k = 3$ Paketzentren. Geben Sie einen kürzesten Weg von **A** nach **H** an, der über mindestens ein Paketzentrum verläuft. Markieren Sie den Weg entweder *eindeutig erkennbar* im Graphen, oder geben Sie die Knotenreihenfolge an. (Sobald Sie in irgendeiner Art und Weise eine Kante markiert haben, die nicht zum Pfad gehört, geben Sie bitte die Knotenreihenfolge an.) [2 Punkte]



Knotenreihenfolge: ____

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 4

b. Geben Sie nun für eine allgemeine Menge P von Paketzentren einen Algorithmus an, der für ein Knotenpaar $\{u, v\}$ einen Pfad findet, der über mindestens eines der $k = |P|$ Paketzentren verläuft. Ihr Algorithmus soll dabei in zwei Phasen vorgehen: In der *Vorberechnungsphase* darf Ihr Algorithmus (innerhalb der vorgegebenen Zeit- und Platzschranken) Daten vorberechnen. In dieser Phase ist dem Algorithmus der Graph bekannt, **nicht aber die später folgenden Anfragen**. In der auf die Vorberechnungsphase folgende *Anfragephase* bekommt Ihr Algorithmus dann eine Reihe von Anfragen, d.h. von Knotenpaaren $\{u, v\}$ und muss (wieder innerhalb der unten gegebenen Schranken) einen Pfad zwischen beiden Knoten finden. Die Nebenbedingungen für die beiden Phasen lauten:

- In der **Vorberechnungsphase** darf Ihr Algorithmus maximal $\mathrm{O}(|P|(|V| + |E|)\log(|V|))$ Zeit und maximal $\mathrm{O}(|P| \cdot |V|)$ zusätzlichen Speicher verwenden. *Erinnerung:* In dieser Phase sind die Paare $\{u, v\}$, die später angefragt werden, noch nicht bekannt!

- In der **Anfragephase** muss Ihr Algorithmus dann für jedes angefragte Paar $\{u, v\}$ in Zeit $\mathrm{O}(|P| + |V|)$ einen kürzesten Weg von u nach v ausgeben, der über mindestens ein Paketzentrum verläuft.

Geben Sie einen solchen Algorithmus *in Pseudocode* an. (Für informelle Lösungen können gegebenenfalls noch Teilpunkte vergeben werden.)
Für Ihren Algorithmus dürfen Sie davon ausgehen, dass der Graph als Adjazenzarray A vorliegt. Außerdem bekommen Sie ein weiteres Array P und dessen Länge k gegeben, in dem die Indizes der Knoten gespeichert sind, die die Paketzentren darstellen.

Hinweis: Sie dürfen Algorithmen, die aus der Vorlesung bekannt sind, als Unterroutinen für Ihre Lösung verwenden, ohne deren Implementierung anzugeben. [6 Punkte]

(zusätzlicher Platz auf dem nächsten Blatt)

Fortsetzung von Aufgabe 4

Zusätzlicher Platz für Aufgabe b.:

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 4

c. Begründen Sie, warum Ihr Algorithmus aus **b.** die geforderte Zeitkomplexität erreicht. [2 Punkte]

d. Um auch Einbahnstraßen zu berücksichtigen, betrachten wir nun gerichtete Kanten. Genauer sei $G = (V, E)$ ein gewichteter *gerichteter* Graph mit $n = |V|$ Knoten. Weiter gebe es für zwei Knoten $u, v \in V$ genau dann eine Kante $(u, v) \in E$ mit Gewicht c, wenn der Paketbote über einen Weg der Länge c direkt von u nach v fahren kann.
Skizzieren Sie kurz, wie Sie Ihren Algorithmus aus Teilaufgabe **b.** abändern müssen, um das gegebene Problem auf einem gerichteten Graphen zu lösen. Dabei sollen sich die asymptotischen Laufzeitschranken nicht ändern. [2 Punkte]

Aufgabe 5. Dynamische Programmierung [11 Punkte]

Ein Chemiekonzern verwendet eine Reihe von k Chemikalien (die wir mit $1, 2, \ldots, k$ durchnummerieren), und Sie kennen deren jeweiligen Bedarf für die nächste Zeit. Der Bedarf an den Chemikalien sei durch den Vektor $\mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} \in \mathbb{R}^k_{\geq 0}$ gegeben, wobei der j-te Eintrag b_j dem Bedarf an Chemikalie j entspricht. Sie haben außerdem n Angebote als Vektoren $\mathbf{a}_i = \begin{pmatrix} a_{i,1} \\ \vdots \\ a_{i,k} \end{pmatrix} \in \mathbb{R}^k_{\geq 0}$ für $i \in \{1, \ldots, n\}$ vorliegen mit jeweiligen Preisen $p_i \in \mathbb{N}$ für $i \in \{1, \ldots, n\}$. Dabei entspricht $a_{i,j}$ der Menge der Chemikalie j im i-ten Angebot und p_i dem Preis des i-ten Angebot (für $j \in \{1, \ldots, k\}$ und $i \in \{1, \ldots, n\}$).
Sie können Angebote nur entweder komplett annehmen oder komplett ablehnen. Ein Angebot kann nur einmal, und nicht beliebig oft angenommen werden. Sie möchten Ihren gesamten Bedarf zu einem möglichst geringen Gesamtpreis decken. Dabei ist es unproblematisch, wenn Sie zu große Mengen einer Chemikalie einkaufen.

a. Der Konzern benötigt 20 Einheiten Sauerstoff, 40 Einheit Wasserstoff, 80 Einheiten Stickstoff, 5 Einheiten Helium und 5 Einheiten Krypton, so dass sich der Bedarf

$$\mathbf{b} = \begin{pmatrix} 20 \\ 40 \\ 80 \\ 5 \\ 5 \end{pmatrix}$$

ergibt. Es liegen folgende 5 Angebote vor:

$$\mathbf{a}_1 = \begin{pmatrix} 10 \\ 40 \\ 20 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{a}_2 = \begin{pmatrix} 5 \\ 0 \\ 60 \\ 5 \\ 0 \end{pmatrix}, \quad \mathbf{a}_3 = \begin{pmatrix} 30 \\ 0 \\ 0 \\ 5 \\ 0 \end{pmatrix}, \quad \mathbf{a}_4 = \begin{pmatrix} 5 \\ 5 \\ 30 \\ 0 \\ 5 \end{pmatrix} \quad \text{und } \mathbf{a}_5 = \begin{pmatrix} 0 \\ 90 \\ 80 \\ 0 \\ 5 \end{pmatrix}$$

mit Preisen $p_1 = 50,$ $p_2 = 30,$ $p_3 = 40$ $p_4 = 50$ und $p_5 = 100.$

Welche Angebote wählen Sie, um den Preis zu minimieren? Welcher Preis ergibt sich? [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 5

b. Stellen Sie ein ganzzahliges lineares Programm (ILP) auf, das die Aufgabe allgemein (*nicht* nur für die konkreten Werte aus **a.**) als Minimierungsproblem beschreibt. Geben Sie dazu die benötigten Variablen, die zu minimierende Kostenfunktion und alle erforderlichen Nebenbedingungen an. [3 Punkte]

c. Eine optimale Lösung des Minimierungsproblems lässt sich aus optimalen Lösungen von Teilproblemen zusammensetzen. $P[i, \mathbf{b}]$ bezeichne hierbei den minimalen Preis um den Bedarf $\mathbf{b}$ mit den Angeboten $\mathbf{a}_1, \ldots, \mathbf{a}_i$ abzudecken. Sollten die gegeben Angebote die geforderten Mengen insgesamt nicht erfüllen können, soll der Preis ∞ entsprechen. Geben Sie die Rekurrenz und den Basisfall für die Lösung des Minimierungsproblems auf diese Art und Weise an. [3 Punkte]

- Basisfall für $i = 1$:

  $P[1, \mathbf{b}] =$

- Rekurrenz für $i = n, \ldots, 2$:

  $P[i, \mathbf{b}] =$

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 5

d. Geben Sie nun basierend auf der Rekurrenz aus **c.** ein dynamisches Programm in Pseudocode an, das die Kosten der Optimallösung dieses Problems berechnet. Sollten die gegebenen Angebote die geforderten Mengen insgesamt nicht erfüllen können, soll Ihr Algorithmus ∞ als Kosten zurückgeben. [3 Punkte]

Konzeptpapier für Nebenrechnungen.