Prof. Dr. Ir. Joost-Pieter Katoen Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Aufgabe 1 ($\mathcal{O}$-Notation): (9 + 4 + 7 = 20 Punkte)

a) Geben Sie für die folgenden Paare von Mengen jeweils die Teilmengenbeziehung mit Hilfe der Symbole $\subset$, $\supset$ und $=$ an.

(1)	$\mathcal{O}(n^n)$	$o(n^n \cdot 2n + 3)$	**(2)**	$\Omega(\log_2(n^2))$	$\Omega(\log_2 n)$
(3)	$\mathcal{O}(n!)$	$\mathcal{O}(n^n)$	**(4)**	$\mathcal{O}(2^n)$	$\mathcal{O}(3^n)$
(5)	$\mathcal{O}(f(n)) \cap \Omega(f(n))$ für eine beliebige Funktion f	$\Theta(f(n))$	**(6)**	$\omega(n^2 + 3)$	$\Omega(3n^2 + 2n + 4)$

b) Beweisen oder widerlegen Sie die folgende Aussage: $\sum_{i=1}^{n} \frac{i}{n} \in \Theta(n)$

c) Beweisen oder widerlegen Sie die folgende Aussage: $\left(\frac{n}{2}\right)^{\frac{n}{2}} \in \Theta(n!)$

Lösung:

a)

(1)	$\mathcal{O}(n^n)$	$\subset$	$o(n^n \cdot 2n + 3)$	**(2)**	$\Omega(\log_2(n^2))$	$=$	$\Omega(\log_2 n)$
(3)	$\mathcal{O}(n!)$	$\subset$	$\mathcal{O}(n^n)$	**(4)**	$\mathcal{O}(2^n)$	$\subset$	$\mathcal{O}(3^n)$
(5)	$\mathcal{O}(f(n)) \cap \Omega(f(n))$	$=$	$\Theta(f(n))$	**(6)**	$\omega(n^2 + 3)$	$\subset$	$\Omega(3n^2 + 2n + 4)$

b) Die Funktion $f(n) = \sum_{i=1}^{n} \frac{i}{n}$ lässt sich wie folgt umformen:

$$f(n) = \sum_{i=1}^{n} \frac{i}{n} = \frac{1}{n} \cdot \sum_{i=1}^{n} i \overset{\text{kl. Gauß}}{=} \frac{1}{n} \cdot \frac{n \cdot (n+1)}{2} = \frac{n+1}{2} \in \Theta(n)$$

q.e.d.

c) Wir widerlegen die Aussage, indem wir zeigen, dass $n! \notin \mathcal{O}\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right)$ gilt.

$$n! = 1 \cdot 2 \cdot 3 \ldots \cdot \underbrace{\frac{n}{2} - 1 \cdot \frac{n}{2} \ldots \cdot n}_{> \left(\frac{n}{2}-1\right)^{\frac{n}{2}+1}} > \left(\frac{n}{2} - 1\right)^{\frac{n}{2}+1} \notin \mathcal{O}\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right)$$

$\Rightarrow n! \notin \mathcal{O}\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) \Rightarrow \left(\frac{n}{2}\right)^{\frac{n}{2}} \notin \Theta(n!)$.

Name: **Matrikelnummer:**

Aufgabe 2 (Sortieren): (4 + 3 + 3 + 7 + 3 = 20 Punkte)

a) Sortieren Sie das folgende Array mittels des Quicksort-Algorithmus aus der Vorlesung (d. h. das Pivot-Element ist das jeweils letzte Element des aktuell zu sortierenden Array-Bereichs).

1	9	2	8	4	7	5

Geben Sie das vollständige Array nach jeder Partitionierung an (d. h. Sie dürfen nicht die Ergebnisse zweier rekursive Aufrufe in einem Schritt angeben) und kennzeichnen Sie das für diese Partitionierung gewählte Pivot-Element durch eine Umkreisung im vorhergehenden Array.

Beispiel:
Für das Array | 1 | 3 | 2 | *wird die 2 als Pivot-Element markiert (also* | 1 | 3 | (2) | *) und wir erhalten als Ergebnisarray* | 1 | 2 | 3 |.

b) Geben Sie die asymptotische Worst-Case Laufzeit (Θ) von Quicksort an. Begründen Sie Ihre Antwort (ein Verweis darauf, dass dies in der Vorlesung gezeigt wurde, reicht nicht aus).

c) Füllen Sie die folgende Tabelle durch Angabe der entsprechenden Worst-Case Komplexitäten (Θ) aus. Sie brauchen Ihre Antworten **nicht** zu begründen.

d) Geben Sie die asymptotische Worst-Case Laufzeit (Θ) des Aufrufs `pentaSort(E, start, end)` für den folgenden Sortier-Algorithmus in Abhängigkeit der Differenz $n =$ `end` $-$ `start` an, wobei `end` $\geq$ `start` gilt. Geben Sie dazu zunächst eine **Rekursionsgleichung** basierend auf dem angegebenen Code an und **lösen Sie diese anschließend auf**.

```
void pentaSort(int E[], int start, int end) {
    int n = end - start;
    if (n < 5) {
        // nutze InsertionSort fuer kleine Eingaben
        insertionSort(E, start, end);
    } else {
        int fifth = n / 5;   // (abgerundete) Integer-Division
        pentaSort(E, start, end - fifth);
        pentaSort(E, start + fifth, end);
        pentaSort(E, start, end - fifth);
    }
}
```

Hinweis: Sie dürfen Bestandteile der Rekursionsgleichung, welche einer konstanten Laufzeit entsprechen, zu einer Konstanten c zusammenfassen (insbesondere Rundungen bzw. Gauß-Klammern) und müssen eventuell auftretende Logarithmen nicht ausrechnen.

e) Ist der `pentaSort` Sortier-Algorithmus mit dem initialen Aufruf `pentaSort(E, 0, n)` für ein Array `E` mit der Länge `n` stabil? Begründen Sie Ihre Antwort.

Lösung:

a)

1	9	2	8	4	7	(5)

1	4	(2)	5	9	7	8

1	2	4	5	9	7	(8)

1	2	4	5	7	8	9

b) Die Worst-Case Laufzeit von Quicksort ist $\Theta(n^2)$. Quicksort teilt den aktuellen Arraybereich anhand eines Pivot-Elements in zwei Teile (Elemente kleiner als das Pivot-Element und größer als oder gleich dem Pivot-Element). Falls immer das größte Element des aktuellen Arraybereichs als Pivot-Element gewählt wird, findet ein rekursiver Aufruf auf einem um 1 kleineren Arraybereich statt, während der zweite rekursive Aufruf auf einem leeren Arraybereich stattfindet. Das ergibt die Rekursionsgleichung $T(n) = T(n-1) + n + c$ mit $T(1) = 1$ für eine Konstante c und diese ist in $\Theta(n^2)$.

c)

Algorithmus	Laufzeit Worst-Case	Platz Worst-Case
Insertion-Sort	$\Theta(n^2)$	$\Theta(1)$ (zus. Speicher) $\Theta(n)$ (Platzkompl.) in-place
Merge-Sort	$\Theta(n \cdot \log(n))$	$\Theta(n)$
Heap-Sort	$\Theta(n \cdot \log(n))$	$\Theta(1)$ (zus. Speicher) $\Theta(n)$ (Platzkompl.) in-place

d) Die asymptotische Worst-Case Laufzeit ergibt sich gemäß der Rekursionsgleichung

$$T(n) = 3 \cdot T(\frac{n}{1,25}) + c$$

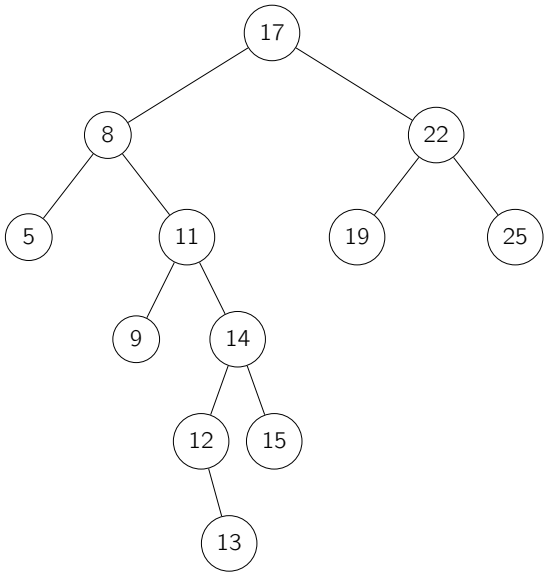
für eine Konstante c. Da $\frac{\log(3)}{\log(1,25)} > 1$, haben wir $c \in \mathcal{O}(n^{\frac{\log(3)}{\log(1,25)} - (1 - \frac{\log(3)}{\log(1,25)})})$. Dies ist der erste Fall des Master-Theorems und wir erhalten $T(n) \in \Theta(n^{\frac{\log(3)}{\log(1,25)}}) \approx \Theta(n^{4,9})$.

e) Der Algorithmus ist stabil, da im Basisfall ein stabiles Sortierverfahren genutzt wird und ansonsten keinerlei Tauschoperationen durchgeführt werden (insbesondere werden nur zusammenhängende Arraybereiche betrachtet).

Name: **Matrikelnummer:**

Aufgabe 3 (Bäume): (5 + 6 + 5 + 4 = 20 Punkte)

a) Überführen Sie den folgenden Baum der **Höhe 5** durch **maximal 4 Rotationen** in einen Baum der **Höhe 3**. Geben Sie dabei für jede Rotation an, um welchen Knoten rotiert wird, in welche Richtung rotiert wird und wie der resultierende Baum aussieht. Dabei dürfen der Knoten, um den rotiert wird, und die Rotationsrichtung für die erste Rotation im abgebildeten Baum markiert werden.



b) Ein AVL-Baum ist ein Binärbaum, bei dem sich für jeden Knoten die Höhe seiner Teilbäume höchstens um eins unterscheiden darf. Gegeben sei nun ein Binärbaum, der aus Instanzen der Klasse `Node` besteht:

```
class Node{
    Node left, right;
    int key;
}
```

Implementieren Sie eine Methode `check(Node tree)` in Pseudocode, welche überprüft, ob der durch den Wurzelknoten `tree` gegebene Baum ein AVL-Baum ist. Stellen Sie sicher, dass Ihre Methode eine Worst-Case Komplexität von $\Theta(n)$ besitzt, wobei n die Anzahl der Knoten des Baumes ist. Die Laufzeitschranke muss nicht bewiesen oder begründet werden.

Hinweis: Sie dürfen den Rückgabetyp der Methode frei wählen. Beispielsweise sind hier auch Tupel zulässig. Sie dürfen außerdem die Funktionen **max***, die das Maximum von zwei Zahlen bestimmt, und* **abs***, welche eine Zahl auf ihren Absolutbetrag abbildet, benutzen.*

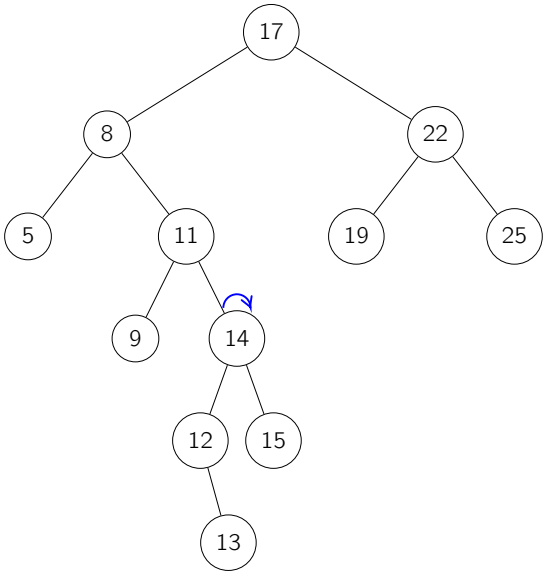
c) Für einen Binärbaum T bezeichne $\ell(T)$ die Anzahl der Blätter des Baumes.

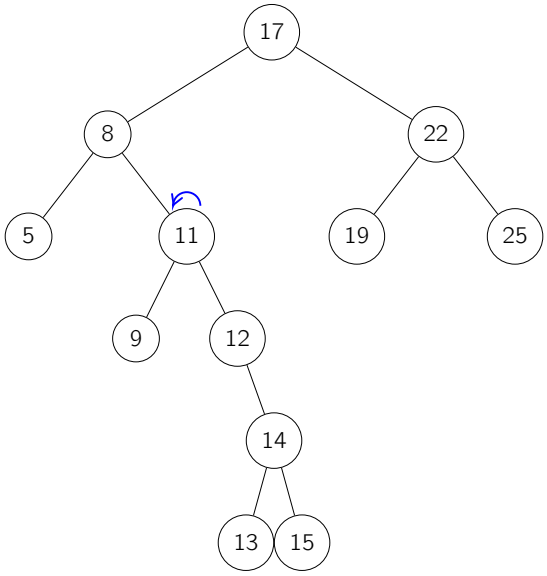
Beweisen Sie per Induktion: Für einen Binärbaum T der Höhe h, dessen innere Knoten alle 2 Nachfolger haben, gilt $\ell(T) \geq h + 1$.

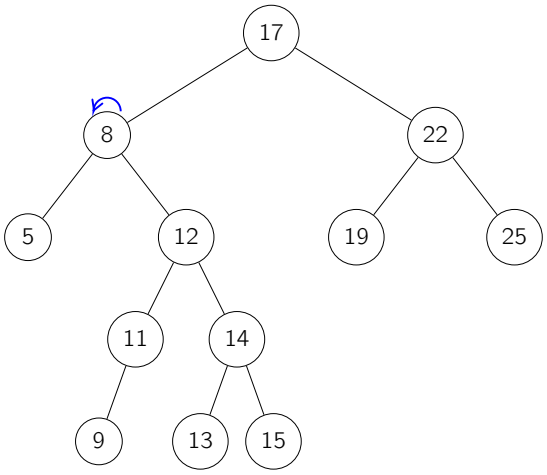
d) Beschreiben Sie ein Verfahren, welches einen Wert in einen existierenden Max-Heap mit n Elementen (repräsentiert durch seine Arraydarstellung) einfügt, so dass wieder ein Max-Heap entsteht. Ihr Verfahren soll eine Worst-Case Komplexität in $\mathcal{O}(\log n)$ besitzen. Gehen Sie dabei davon aus, dass das Array, welches den Heap enthält, die Länge $n + 1$ besitzt und die ersten n Elemente die Einträge des Heaps enthalten. Die Laufzeitschranke muss nicht bewiesen oder begründet werden.

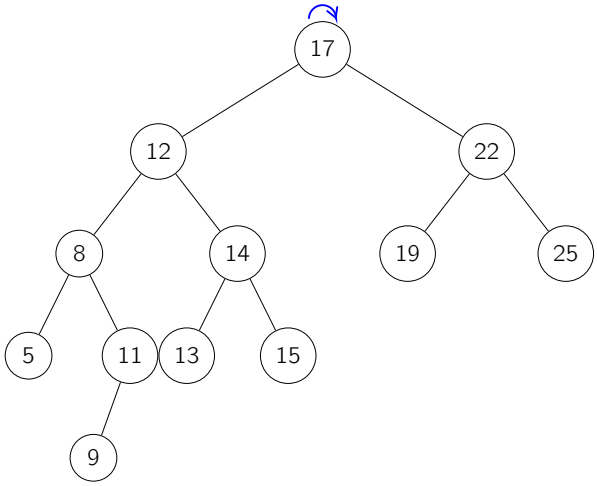
Hinweis: Achten Sie dabei darauf, die Stellen des Arrays, auf die zugegriffen wird, präzise anzugeben.

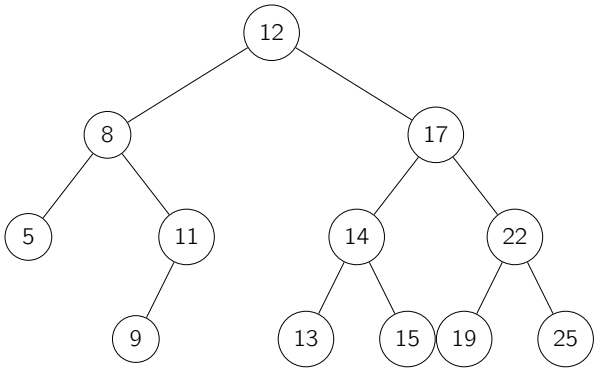
Lösung:












```
(int, bool) isAVLTree(Node tree) {
    int leftHeight, rightHeight;
    bool leftIsAVL, rightIsAVL;
    if (tree == null) return (-1, true);
    (leftHeight, leftIsAVL) = isAVLTree(tree.left);
    (rightHeight, rightIsAVL) = isAVLTree(tree.right);
    int newHeight = max(leftHeight, rightHeight) + 1;
    bool isAVL = leftIsAVL && rightIsAVL && (abs(leftHeight - rightHeight) <= 1);
```

```
        return (newHeight, isAVL);
    }
```

Beweis per Induktion über die Höhe h.

Induktionsanfang: $h = 0$:

Der einzige Baum T mit Höhe 0 ist der Baum, der nur aus der Wurzel besteht. Dieser Baum hat offensichtlich $\ell(T) \geq h + 1 = 1$ Blatt, nämlich die Wurzel selbst.

Induktionsannahme:

Für ein beliebiges, aber festes $h \in \mathbb{N}$ gelte, dass für alle Binärbäume der Höhe h, deren innere Knoten alle 2 Nachfolger haben, $\ell(T) \geq h + 1$ gilt.

Induktionsschluss: $h \to h + 1$:

Sei T ein Binärbaum der Höhe $h + 1$, dessen innere Knoten alle 2 Nachfolger haben und w die Wurzel von T. Da die Höhe von T größer als 1 ist, ist die Wurzel ein innerer Knoten und hat somit 2 Nachfolger. Dann muss einer der beiden Teilbäume die Höhe h haben, da sonst T nicht Höhe $h + 1$ hätte. Sei nun T_1 ein Teilbaum mit Höhe h und T_2 der andere Teilbaum. Dann gilt:

$$\ell(T) = \ell(T_1) + \ell(T_2) \overset{(IV)}{\geq} (h + 1) + \ell(T_2) \overset{(1)}{\geq} (h + 1) + 1 = h + 2$$

wobei (1) gilt, da T_2 mindestens ein Blatt haben muss.

Aus der Vorlesung ist bekannt, dass in der Arraydarstellung von Heaps die Kinder des Elements an Index i an Index $2 \cdot i + 1$ bzw. $2 \cdot i + 2$ stehen. Umgeformt bedeutet das, dass der Vater eines Elements an Index j an Index

$$\left\lfloor \frac{j - 1}{2} \right\rfloor \tag{1}$$

steht.

Ein Verfahren zum Lösen des Problems könnte wie folgt aussehen:

a) Schreibe das neue Element k an die letzte (freie) Stelle im Array, also an Index n.

b) Überprüfe, ob das Element an Position $\left\lfloor \frac{n-1}{2} \right\rfloor$ größer als das Element an Index n ist.

c) Ist dies der Fall, so ist der Heap wieder ein Max-Heap.

d) Ist dies nicht der Fall, so tausche die Elemente an diesen Indizes.

e) Fahre mit dem Tauschen "nach oben" solange fort wie nötig, wobei die zu untersuchenden Positionen durch die Gleichung (1) gegeben sind.

Da dieses Verfahren das Element im Worst-Case bis ganz nach oben tauschen muss und das Element dabei in jedem Tauschschritt eine Ebene aufsteigt, ergibt sich eine Worst-Case Komplexität von $\Theta(h) = \Theta(\log n)$, wobei h die Höhe des Heaps bezeichnet. Die logarithmische Schranke in n ergibt sich daraus, dass es sich bei Heaps per Definition um balancierte Bäume handelt.

Name: **Matrikelnummer:**

Aufgabe 4 (Hashing): (2 + 8 + 6 + 4 = 20 Punkte)

Gegeben sei die Hashfunktion

$$h(k, i) = (k \bmod 11 + 1 \cdot i + 2 \cdot i^2) \bmod 11.$$

a) Um welche Art von Sondierung handelt es sich bei der gegebenen Hashfunktion?

b) Fügen Sie die Werte $11, 47, 39, 9, 58, 15, 7$ unter Verwendung der obigen Hashfunktion nacheinander in eine anfangs leere Hash-Tabelle der Größe 11 ein:

c) Robin Hood Hashing ist eine Variante von Hashing, bei der im Falle einer Kollision immer das Schlüsselelement mit der kleineren Anzahl an Sondierungschritten weiter gehasht wird. D. h. gibt es mit einem Element k_1 nach i Sondierungsschritten eine Kollision mit einem Element k_2 nach j Sondierungsschritten (also $h(k_1, i) = h(k_2, j)$) und ist $j < i$, dann wird mit k_2 weiter sondiert, ansonsten mit k_1.

Fügen Sie die Werte $22, 52, 40, 17, 13$ nach Robin Hood Hashing mit der Hashfunktion

$$h(k, i) = ((k \bmod 7) + i * (1 + k \bmod 5)) \bmod 7$$

in eine anfangs leere Hashtabelle der Größe 7 ein:

d) Beschreiben Sie die Vor- und Nachteile, welche entstehen, wenn man beim Robin Hood Hashing tauscht, falls $j \leq i$ gilt.

Lösung:

a) Es handelt sich um Hashing mit quadratischem Sondieren.

b)

0	1	2	3	4	5	6	7	8	9	10
11		58	47	15		39	7		9	

c)

0	1	2	3	4	5	6
	(22,0)		~~(52,0)~~		(40,0)	(17,1)
		(52,2)	(13,1)			

alternative Lösung:

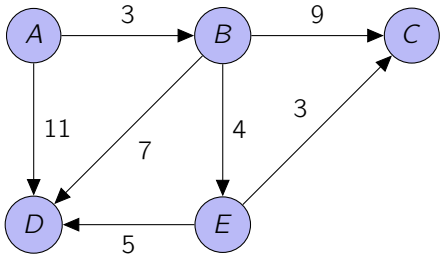
0	1	2	3	4	5	6
	(22,0)		~~(52,0)~~		(40,0)	
			~~(17,0)~~			~~(52,1)~~
		(52,2)	(13,1)			(17,1)

d) Es führt lediglich dazu, dass einmal mehr ein Element getauscht werden muss.

Name: **Matrikelnummer:**

Aufgabe 5 (Graphen): (5 + 7 + 5 + 3 = 20 Punkte)

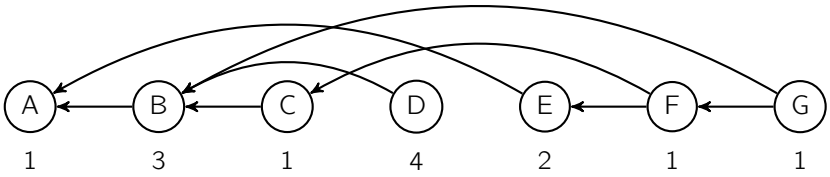
a) Ermitteln Sie mit Hilfe des Dijkstra-Algorithmus die kürzesten Wege vom **Startknoten A** zu allen anderen Knoten des folgenden Graphen. Verwenden Sie dazu die folgende Tabelle. Notieren Sie für jeden Rechenschritt den aktuell gewählten Knoten zur Verbesserung der Wege und die Länge der bis zu diesem Zeitpunkt möglichen kürzesten Wege für jeden noch nicht abgeschlossenen Knoten ($D[\dots]$). Streichen Sie Felder der Tabelle, die nicht mehr benötigt werden, durch (dies geschieht, sobald ein Knoten als Baum-Knoten klassifiziert wurde).



Schritt	0	1	2	3	4	5
Knoten						
$D[A]$						
$D[B]$						
$D[C]$						
$D[D]$						
$D[E]$						

b) Beweisen oder widerlegen Sie: Sei (S, T) ein minimaler Schnitt in einem Flussnetzwerk G mit nur einer Quelle s und einer Senke t. Dann ist die Senke t von jedem Knoten in T erreichbar ohne Knoten aus S zu passieren.

c) Gegeben sei die folgende topologische Sortierung eines Graphen, der eine Menge voneinander abhängiger Aufgaben modelliert, wobei die Gewichte der Knoten unter diesen notiert sind. Tragen Sie in die unten stehende Tabelle den frühestmöglichen Beendigungszeitpunkt (eft) für jede der Aufgaben A bis G ein und geben Sie einen kritischen Pfad an.



	A	B	C	D	E	F	G
eft							

kritischer Pfad:

d) Eine Firma möchte für eine gegebene Menge von Sprachen in der Lage sein, jede Sprache in jede andere zu übersetzen. Dazu möchte sie eine Menge von Übersetzern beschäftigen, die jeweils zwischen genau zwei Sprachen (in beide Richtungen) Texte übersetzen können. Die Übersetzer beziehen ein Monatsgehalt für ihre Tätigkeit.

Dieses Problem kann man als Graphen modellieren, in dem die Knoten den Sprachen entsprechen und die (ungerichteten) Kanten zwischen den Knoten die Übersetzungsmöglichkeiten darstellen. Die Monatsgehälter stehen als Kosten an den jeweiligen Kanten.

Nehmen Sie an, dass der Graph, der das Problem modelliert, zusammenhängend ist. Mit welchem Algorithmus aus der Vorlesung lässt sich herausfinden, welche Übersetzer die Firma beschäftigen muss, damit alle

Eingabesprachen in alle Ausgabesprachen übersetzt werden können und dabei die Summe der monatlichen Gehälter minimal ist? Begründen Sie Ihre Antwort.

Lösung:

Schritt	1	2	3	4	5
Knoten	A	B	E	D	C
$D[A]$	0	X	X	X	X
$D[B]$	3	3	X	X	X
$D[C]$	∞	12	10	10	10
$D[D]$	11	10	10	10	X
$D[E]$	∞	7	7	X	X

b) Beweis durch Widerspruch:
Angenommen (S, T) sei ein minimaler Schnitt und es gäbe ein $v \in T$, sodass v die Senke t nicht erreichen kann ohne einen Knoten aus S zu passieren. Sei $V \subset T$ die Menge aller Knoten aus T, welche von v erreichbar sind ohne Knoten aus S zu passieren. Wir haben also $t \notin V$. Da die Knoten in V keine Quellen sein können, muss es mindestens eine Kante (u, w) geben, sodass $u \in S$, $w \in V$ und $c(u, w) > 0$. Für die Knotenmenge $T \setminus V$ muss demnach gelten, dass sie eine geringere Summe an eingehender Kapazität hat als T. Demnach hat der Schnitt $(S \cup V, T \setminus V)$ eine geringere Kapazität als (S, T) im Widerspruch dazu, dass (S, T) ein minimaler Schnitt ist.

	A	B	C	D	E	F	G
eft	1	4	5	8	3	6	7

kritischer Pfad: A, B, D (Gewicht 8)

Das Problem lässt sich mit dem Prim-Algorithmus lösen.

Begründung:

- Ein Spannbaum garantiert, dass es zwischen jeweils zwei Knoten immer einen Pfad gibt. Das heißt, dass jede Sprache in jede andere übersetzt werden kann.
- Der Spannbaum ist kostenminimal, d. h. es gibt keine andere Auswahl von Kanten (also Übersetzern), die ebenfalls alle Sprachen in jede andere überführen können und ein kleineres Gesamtgehalt bekommen.

Insbesondere ist es nicht möglich, die Aufgabe mit dem Floyd-Algorithmus zu lösen, da dieser nur die kürzesten Pfade zwischen allen Knoten findet. Es war jedoch nicht gefragt, die Übersetzung zwischen zwei Sprachen billig zu halten, sondern die Gesamtkosten zu minimieren.

Name: **Matrikelnummer:**

Aufgabe 6 (Dyn. Programmierung/Geom. Algorithmen): (8 + 12 = 20 Punkte)

a) Füllen Sie die folgende Tabelle mit Zahlen und Pfeilmarkierungen gemäß dem Verfahren aus der Vorlesung zur Bestimmung der Longest-Common-Subsequence (LCS) für die Buchstabensequenzen FRAGE und DRANG. Geben Sie außerdem die LCS an und markieren Sie die Zellen in der Tabelle, welche zum Ablesen der LCS verwendet werden, durch Umkreisungen.

b) Betrachten Sie folgende Punkte im zweidimensionalen Raum:

$$(3,0); (6,3); (4,2); (3,6); (2,4); (2,2); (0,3)$$

Diese Punkte sind bereits gemäß ihrer Polarkoordinaten bzgl. des Punktes (3,0) sortiert. Bestimmen Sie die konvexe Hülle dieser Punkte mit Hilfe des Graham-Scans. Geben Sie zu jeder Determinantenberechnung an,

- welche drei Punkte an der Berechnung beteiligt sind,
- zwischen welchen Vektoren die Determinante berechnet wird,
- welchen Wert die Determinante hat und
- wie der Stack nach der entsprechenden Push- oder Pop-Operation aussieht.

Hinweis: Da die Punkte bereits sortiert sind, brauchen Sie keine erneute Sortierung vorzunehmen. Falls keine Determinante berechnet wird, brauchen Sie keine Stackänderung anzugeben. Es ist nicht nötig, eine Determinante zu berechnen, wenn der Stack weniger als drei Elemente enthält.

Lösung:

a)

		F	R	A	G	E
	0	0	0	0	0	0
D	0	↑0	↑0	↑0	↑0	↑0
R	0	↑0	↖1	←1	←1	←1
A	0	↑0	↑1	↖2	←2	←2
N	0	↑0	↑1	↑2	↑2	↑2
G	0	↑0	↑1	↑2	↖3	←3

LCS: RAG

b)

Name: **Matrikelnummer:**

Knoten	Vektoren und Determinante	Stack
(3, 6) (4, 2) (6, 3)	$\det(\begin{pmatrix} -2 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 4 \end{pmatrix}) = -8 - 1 = -9 \leq 0$	[(3, 6)] (6, 3) (3, 0)
(2, 4) (3, 6) (6, 3)	$\det(\begin{pmatrix} -3 \\ 3 \end{pmatrix}, \begin{pmatrix} -1 \\ -2 \end{pmatrix}) = 6 + 3 = 9 > 0$	(2, 4) (3, 6) (6, 3) (3, 0)
(2, 2) (2, 4) (3, 6)	$\det(\begin{pmatrix} -1 \\ -2 \end{pmatrix}, \begin{pmatrix} 0 \\ -2 \end{pmatrix}) = 2 - 0 = 2 > 0$	(2, 2) (2, 4) (3, 6) (6, 3) (3, 0)
(0, 3) (2, 2) (2, 4)	$\det(\begin{pmatrix} 0 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ 1 \end{pmatrix}) = 0 - 4 = -4 \leq 0$	[(0, 3)] (2, 4) (3, 6) (6, 3) (3, 0)
(0, 3) (2, 4) (3, 6)	$\det(\begin{pmatrix} -1 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ -1 \end{pmatrix}) = 1 - 4 = -3 \leq 0$	[(0, 3)] (3, 6) (6, 3) (3, 0)
(0, 3) (3, 6) (6, 3)	$\det(\begin{pmatrix} -3 \\ 3 \end{pmatrix}, \begin{pmatrix} -3 \\ -3 \end{pmatrix}) = 9 + 9 = 18 > 0$	(0, 3) (3, 6) (6, 3) (3, 0)