Aufgabe 1 (Laufzeitanalyse): (4 + 6 + 6 + 3 = 19 Punkte)

a) Beweisen oder widerlegen Sie:

$$\log(n) \in \mathcal{O}(\sqrt{n})$$

Falls allgemeine Rechenregeln benutzt werden, müssen diese beim Namen genannt oder formuliert werden.

b) Beweisen oder widerlegen Sie:

$$\sum_{i=1}^{n}(2 \cdot i - 1) \in \Theta(n^2)$$

Falls allgemeine Rechenregeln benutzt werden, müssen diese beim Namen genannt oder formuliert werden.

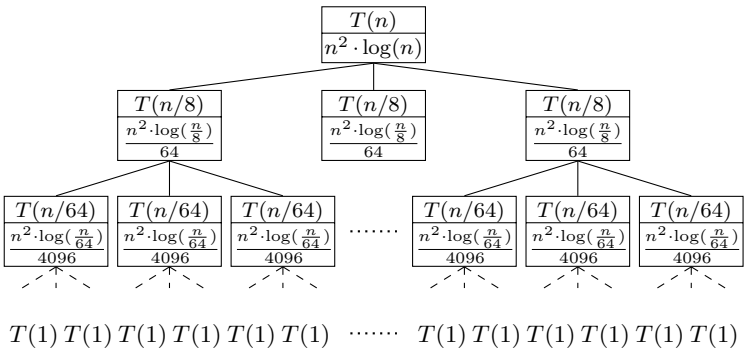
c) Bestimmen Sie die Komplexitätsklasse der folgenden Rekursionsgleichung mithilfe des Mastertheorems oder begründen Sie, warum das Mastertheorem nicht anwendbar ist.

$$T(n) = \begin{cases} 3 \cdot T(\frac{n}{27}) + \sqrt[3]{n+1} & \text{für } n > 1 \\ 5 & \text{sonst} \end{cases}$$

Erinnerung Mastertheorem: $E = \log_b(a)$ (alternativ $b^E = a$) und

$f(n) \in \mathcal{O}(n^{E-\epsilon}), \epsilon > 0$	$\rightarrow T(n) \in \Theta(n^E)$
$f(n) \in \Theta(n^E)$	$\rightarrow T(n) \in \Theta(n^E \cdot \log(n))$
$f(n) \in \Omega(n^{E+\epsilon}), \epsilon > 0$ und $a \cdot f(n/b) \leq d \cdot f(n), d < 1$	$\rightarrow T(n) \in \Theta(f(n))$

d) Gegeben sei der folgende Rekursionsbaum:



Ergänzen Sie die folgenden Aussagen:

- Der Rekursionsbaum hat die Tiefe _____ .
- Der Rekursionsbaum hat _____ Blätter.
- Auf der Ebene i summiert sich die Laufzeit zu _____ .

Lösung: _____

a) Behauptung: Die Aussage gilt.

Beweis:
Zu zeigen: $\limsup_{n\to\infty} \frac{\log(n)}{\sqrt{n}} < \infty$

Es gilt:

$$\lim_{n\to\infty} \frac{\log(n)}{\sqrt{n}} \overset{L'Hôpital}{=} \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\cdot\sqrt{n}}} = \lim_{n\to\infty} \frac{2\cdot\sqrt{n}}{n} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$$

Damit gilt auch $\limsup_{n\to\infty} \frac{\log(n)}{\sqrt{n}} = 0 < \infty$ und die Aussage gilt per Definition. □

b) Behauptung: Die Aussage gilt.

Beweis:
Zu zeigen:

$$\exists c_1, c_2 \in \mathbb{R}^{>0}, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N} \text{ mit } n \geq n_0 : c_1 \cdot n^2 \leq \sum_{i=1}^{n}(2\cdot i - 1) \leq c_2 \cdot n^2$$

Es gilt für $n \geq 1$:

$$\begin{aligned} & \sum_{i=1}^{n}(2\cdot i - 1) \\ = \; & 2\cdot(\sum_{i=1}^{n} i) - n \qquad | \text{ da } \sum_{i=1}^{n} i = \tfrac{n\cdot(n+1)}{2} \\ = \; & n^2 \end{aligned}$$

oder alternativ

$$\begin{aligned} a_n &= \sum_{i=1}^{n}(2\cdot i - 1) \qquad | \quad \text{da } a_n = a_1 + (n-1)\cdot d, \text{ wobei } d = a_{i+1} - a_i \\ &= \tfrac{n}{2}\cdot(2\cdot 1 + (n-1)\cdot 2) \\ &= n^2 \end{aligned}$$

oder alternativ (komplizierter)

$$\begin{aligned} & 2\cdot\sum_{i=1}^{n}(2\cdot i - 1) \\ = \; & 1 + 3 + \ldots + (2\cdot(n-1)-1) + (2\cdot n - 1) + 1 + 3 + \ldots + (2\cdot(n-1)-1) + (2\cdot n - 1) \\ = \; & (2\cdot n - 1) + 1 + (2\cdot(n-1)-1) + 3 + \ldots + 3 + (2\cdot(n-1)-1) + 1 + (2\cdot n - 1) \\ = \; & (2\cdot n - 1) + 1 + (2\cdot n - 3) + 3 + \ldots + 3 + (2\cdot n - 3) + 1 + (2\cdot n - 1) \\ = \; & \underbrace{2\cdot n \qquad + 2\cdot n \qquad + \ldots + 2\cdot n \qquad + 2\cdot n}_{n \text{ mal}} \\ = \; & 2\cdot n^2 \end{aligned}$$

oder alternativ (noch komplizierter)

$$\sum_{i=1}^{n}(2\cdot i-1)$$

$$= \begin{cases} 1+3+\ldots+(2\cdot\frac{n}{2}-1)+(2\cdot(\frac{n}{2}+1)-1)+\ldots+(2\cdot n-1), & n \text{ gerade} \\ 1+3+\ldots+(2\cdot(\frac{n+1}{2}-1)-1)+(2\cdot\frac{n+1}{2}-1)+(2\cdot(\frac{n+1}{2}+1)-1)+\ldots+(2\cdot n-1), & n \text{ ungerade} \end{cases}$$

$$= \begin{cases} (2\cdot n-1)+1+(2\cdot(n-1)-1)+3+\ldots+(2\cdot(\frac{n}{2}+1)-1)+(2\cdot\frac{n}{2}-1), & n \text{ gerade} \\ (2\cdot n-1)+1+(2\cdot(n-1)-1)+3+\ldots+(2\cdot(\frac{n+1}{2}+1)-1)+(2\cdot(\frac{n+1}{2}-1)-1) & \\ \quad+(2\cdot\frac{n+1}{2}-1), & n \text{ ungerade} \end{cases}$$

$$= \begin{cases} (2\cdot n-1)+1+(2\cdot n-3)+3+\ldots+(n+1)+(n-1), & n \text{ gerade} \\ (2\cdot n-1)+1+(2\cdot n-3)+3+\ldots+(n+2)+(n-2)+n, & n \text{ ungerade} \end{cases}$$

$$= \begin{cases} \overbrace{2\cdot n \quad +2\cdot n \quad +\ldots+2\cdot n}^{\frac{n}{2} \text{ mal}}, & n \text{ gerade} \\ \underbrace{2\cdot n \quad +2\cdot n \quad +\ldots+2\cdot n}_{\frac{n-1}{2} \text{ mal}}+n, & n \text{ ungerade} \end{cases}$$

$$= \begin{cases} 2\cdot n\cdot\frac{n}{2}, & n \text{ gerade} \\ 2\cdot n\cdot\frac{n-1}{2}+n, & n \text{ ungerade} \end{cases}$$

$$= n^2$$

Also gilt insbesondere $n^2 \leq \sum_{i=1}^{n}(2\cdot i-1) \leq n^2$ und somit ist für $c_1 = c_2 = 1$ und $n_0 = 1$ die Aussage bewiesen. □

c) Behauptung: $T(n) \in \Theta(\sqrt[3]{n}\cdot\log(n))$

Beweis:
Wir haben $a = 3$, $b = 27$ und $E = \frac{1}{3}$.
Wir zeigen $\sqrt[3]{n+1} \in \Theta(\sqrt[3]{n})$ für $n > 0$:

$$\exists c_1, c_2 > 0.\exists n_0.\forall n \geq n_0 : c_1\cdot\sqrt[3]{n} \leq \sqrt[3]{n+1} \leq c_2\cdot\sqrt[3]{n}.$$
$$\text{Wähle: } c_1 = 1, c_2 = 2 : \sqrt[3]{n} \leq \sqrt[3]{n+1} \leq 2\cdot\sqrt[3]{n}$$

Teilbeweise:

$$\sqrt[3]{n} \leq \sqrt[3]{n+1} \qquad (\forall n > 0)$$

Außerdem zu zeigen:

$$\begin{aligned} \sqrt[3]{n+1} &\leq 2\cdot\sqrt[3]{n} \\ n+1 &\leq 8\cdot n \\ 1 &\leq 7\cdot n \qquad (\forall n \geq \frac{1}{7} \rightarrow n_0 = 1) \end{aligned}$$

Alternativ via Limes:

$$\lim_{n\to\infty}\frac{\sqrt[3]{n+1}}{\sqrt[3]{n}} = \lim_{n\to\infty}\frac{n+1}{n} = \lim_{n\to\infty}\frac{1}{n}+\frac{n}{n} = 1 \qquad (\forall n > 0)$$

Da der Limes größer 0 und kleiner als ∞ ist, gilt die Aussage.
Somit ist die Behauptung gültig (2. Fall des Mastertheorems). □

d)
- Der Rekursionsbaum hat die Tiefe $\log_8(n)$.
- Der Rekursionsbaum hat $3^{\log_8(n)}$ Blätter.
- Auf der Ebene i summiert sich die Laufzeit zu $3^i \cdot \frac{1}{64^i} \cdot n^2 \cdot \log(\frac{n}{8^i})$.

Aufgabe 2 (Sortierverfahren): (3 + 3 + 4 + 3 = 13 Punkte)

a) Beweisen Sie, dass der Selectionsort-Algorithmus **aus der Vorlesung** nicht stabil ist.

b) Sortieren Sie das Array `[ 5,1,7,3,2 ]` durch Anwendung des Bubblesort-Algorithmus **aus der Vorlesung**. Geben Sie dazu das Array nach jeder Swap-Operation an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

c) Sortieren Sie das Array `[ 7,6,5,1,4,2,8,3 ]` durch Anwendung des Mergesort-Algorithmus **aus der Vorlesung**. Geben Sie dazu das Array nach jeder Merge-Operation an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

d) Stellen Sie die Heap-Eigenschaft im Array `[ 2,7,9,3,6,5,1,4 ]` her, wie es zu Beginn des Heapsort-Algorithmus **aus der Vorlesung** geschieht. Geben Sie dazu das Array nach jeder Swap-Operation an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

2	7	9	3	6	5	1	4

Lösung:

a) **Behauptung:** Selectionsort ist nicht stabil.

Beweis:
Betrachten wir folgendes Eingabearray: `[ 2`$_a$`,2`$_b$`,1 ]`
Bei diesem Array wird die erste 2_a mit dem letzten Element getauscht, sodass das Ergebnis `[ 1,2`$_b$`,2`$_a$` ]` ist, wobei allerdings die vormals vordere 2_a nun hinter der anderen 2_b steht. Dieses Gegenbeispiel zeigt, dass Selectionsort nicht stabil ist. □

b)

5	1	7	3	2
1	5	7	3	2
1	5	3	7	2
1	5	3	2	7
1	3	5	2	7
1	3	2	5	7
1	2	3	5	7

c)

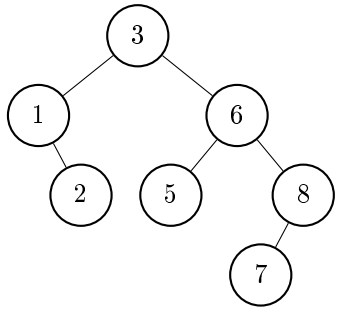
7	6	5	1	4	2	8	3
6	7	5	1	4	2	8	3
6	7	1	5	4	2	8	3
1	5	6	7	4	2	8	3
1	5	6	7	2	4	8	3
1	5	6	7	2	4	3	8
1	5	6	7	2	3	4	8
1	2	3	4	5	6	7	8

d)

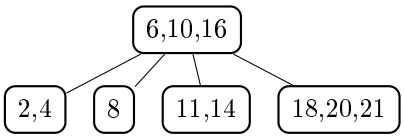
2	7	9	3	6	5	1	4
2	7	9	4	6	5	1	3
9	7	2	4	6	5	1	3
9	7	5	4	6	2	1	3

Aufgabe 3 (Bäume): (5 + 3 + 10 + 3 + 3 = 24 Punkte)

a) Löschen Sie den Wert 3 aus dem folgenden **AVL-Baum** und geben Sie die entstehenden Bäume nach jeder **Löschoperation** sowie jeder **Rotation** an:



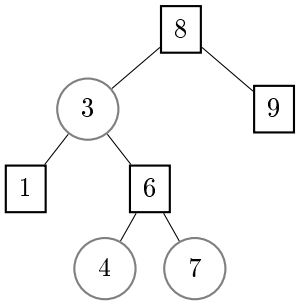
b) Fügen Sie den Wert 13 in den folgenden **2-3-4-Baum** ein und geben Sie den dabei entstehenden Baum an:



c) Fügen Sie den Wert 5 in den folgenden **Rot-Schwarz-Baum** ein und geben Sie die entstehenden Bäume nach

- jeder **Einfügeoperation**,
- jeder **Rotation** sowie
- jeder **Umfärbung** an.

Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.



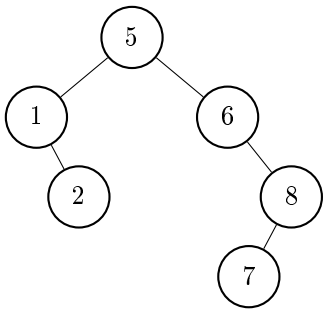
d) Geben Sie den kleinsten Baum (Anzahl Blätter minimal) an, der die **Heap-Eigenschaft** erfüllt und die Schlüssel $1, 2, 4, 4, 5, 6, 6, 6$ enthält.

e) Geben Sie eine **Einfügesequenz** der 5 Schlüssel von 0 bis 4 an, sodass der resultierende binäre Suchbaum **maximal unbalanciert** ist.

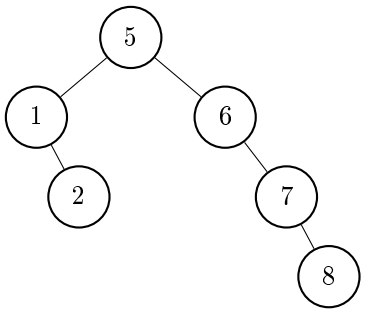
Geben Sie eine **Einfügesequenz** der 7 Schlüssel von 0 bis 6 an, sodass der resultierende binäre Suchbaum **minimal unbalanciert** ist.

Lösung:

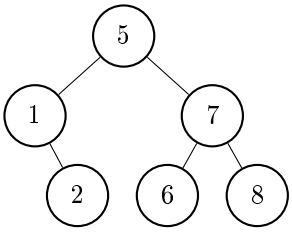
a) entferne 3



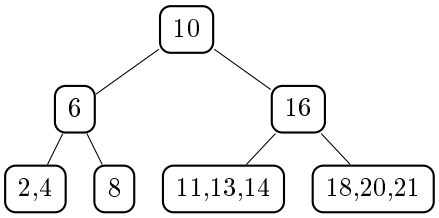
rotiere 8 nach rechts



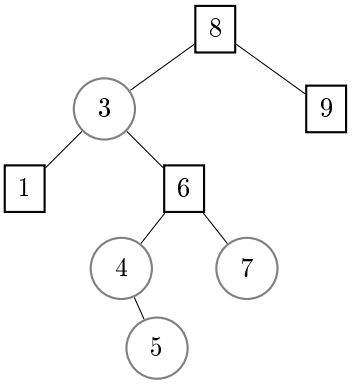
rotiere 6 nach links



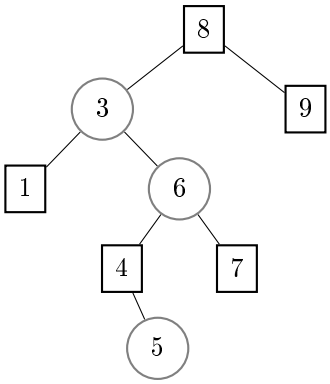
b) Schritt 1: Füge 13 ein



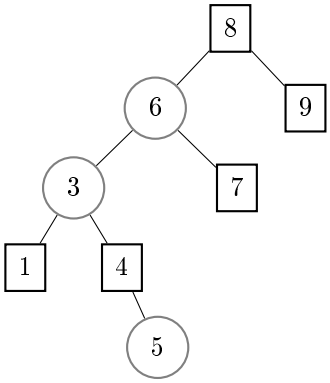
c) füge 5 ein



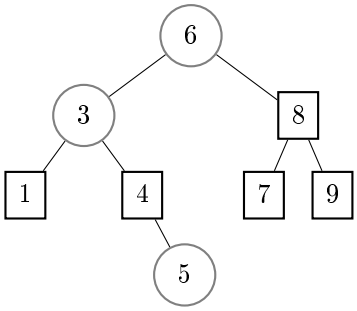
Fall 1: umfärben



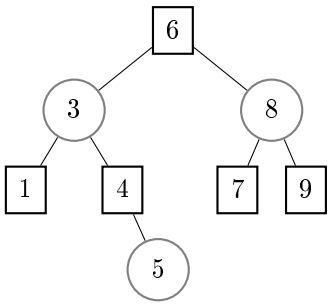
Fall 2: rotiere 3 nach links



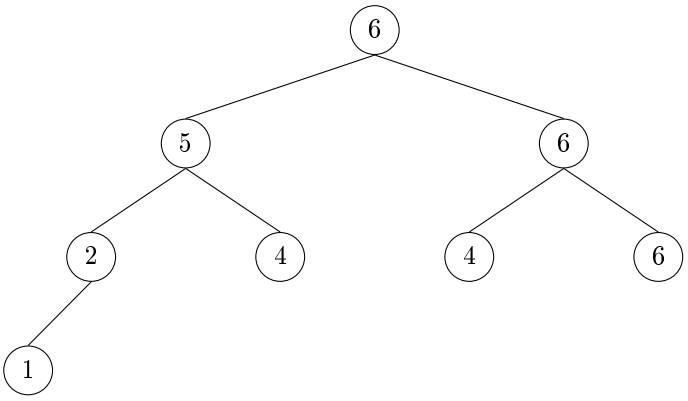
Fall 3: rotiere 8 nach rechts



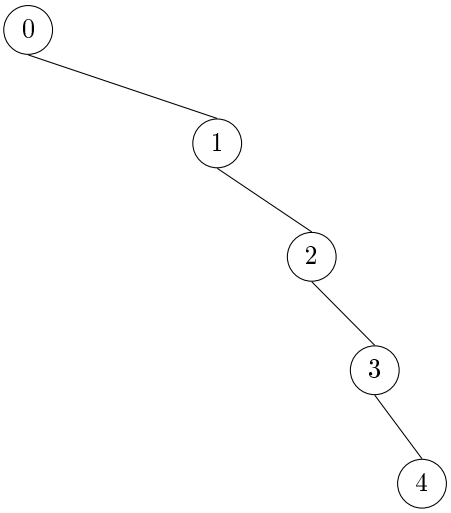
Fall 3: umfärben



d)



e) Einfügesequenz wäre $0, 1, 2, 3, 4$ oder alternativ geht auch $4, 3, 2, 1, 0$.



Einfügesequenz 3, $2, 5$ oder $5, 2$, danach die anderen Schlüssel in beliebiger Reihenfolge. Wenn 3 und dann einer der Knoten aus $\{5, 2\}$ eingefügt wurden kann natürlich auch erst der dazugehörige Teilbaum aufgebaut werden (bei 2: $0, 1$ in beliebiger Reihenfolge, bei 5: $4, 6$ in beliebiger Reihenfolge).

Aufgabe 4 (Hashing): (3 + 3 = 6 Punkte)

a) Fügen Sie die folgenden Werte in das unten stehende Array `a` der Länge 7 unter Verwendung der **Multiplikationsmethode** ($c = 0.25$) mit **linearer Sondierung** ein ($f(n,i) = \lfloor 7 \cdot (n \cdot 0.25 \mod 1) \rfloor + i \mod 7$), wobei $x \mod 1$ den Nachkommateil von x bezeichnet:

5, 7, 4, 11, 0, 12.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

b) Fügen Sie die folgenden Werte in das unten stehende Array `a` der Länge 11 unter Verwendung der **Divisionsmethode** mit **quadratischer Sondierung** ($c_1 = 0.0$, $c_2 = 1.0$) ein ($f(n,i) = ((n \mod 11) + 0.0 \cdot i + 1.0 \cdot i^2) \mod 11$):

1, 8, 7, 12, 18, 22.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]

Lösung:

a) m = 7, c = 0.25:

4	5	0	12		7	11

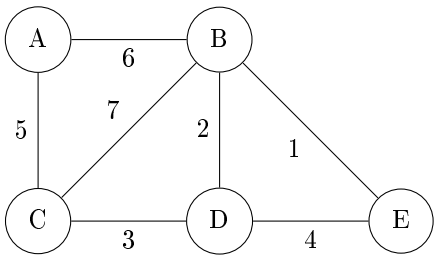
b) m = 11, $c_1 = 0.0$, $c_2 = 1.0$:

18	1	12		22			7	8		

Aufgabe 5 (Graphen): (6 + 8 + 8 + 8 = 30 Punkte)

a) Beweisen oder widerlegen Sie: Der Dijkstra Algorithmus zur Berechnung kürzester Pfade arbeitet korrekt auf gerichteten Graphen, welche zwar negative Kantengewichte enthalten, aber bei denen jeder Zyklus ein positives Gesamtgewicht hat.

b) Führen Sie Prim's Algorithmus auf dem folgenden Graphen aus.



Der Startknoten hat hierbei den Schlüssel A. Geben Sie dazu **vor** jedem Durchlauf der äußeren Schleife an,

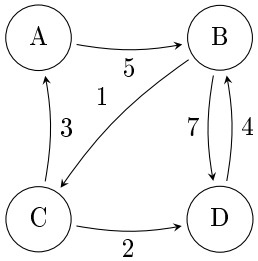
1) welche Kosten die Randknoten haben (d. h. für jeden Knoten `v` in `Q` den Wert `key[v]`)
2) und welchen Knoten `extractMin(Q)` wählt, indem Sie den Kosten-Wert des gewählten Randknoten in der Tabelle unterstreichen (wie es in der ersten Zeile bereits vorgegeben ist).

Geben Sie zudem den vom Algorithmus bestimmten minimalen Spannbaum an.

#Iteration	A	B	C	D	E
1	$\underline{0}$	∞	∞	∞	∞
2					
3					
4					
5					

Minimaler Spannbaum:

c) Betrachten Sie den folgenden Graphen:



Führen Sie den **Algorithmus von Floyd** auf diesem Graphen aus. Geben Sie dazu nach jedem Durchlauf der äußeren Schleife die aktuellen Entfernungen in einer Tabelle an. Die erste Tabelle enthält bereits die Adjazenzmatrix nach Bildung der reflexiven Hülle. Der Eintrag in der Zeile i und Spalte j ist also ∞, falls es keine Kante vom Knoten der Zeile i zu dem Knoten der Spalte j gibt, und sonst das Gewicht dieser Kante. Beachten Sie, dass in der reflexiven Hülle jeder Knoten eine Kante mit Gewicht 0 zu sich selbst hat.

①	A	B	C	D
A	0	5	∞	∞
B	∞	0	1	7
C	3	∞	0	2
D	∞	4	∞	0

②	A	B	C	D
A				
B				
C				
D				

③	A	B	C	D
A				
B				
C				
D				

④	A	B	C	D
A				
B				
C				
D				

⑤	A	B	C	D
A				
B				
C				
D				

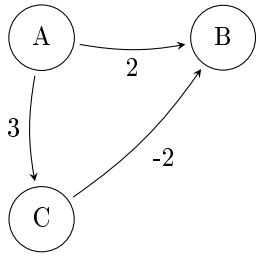
d) Betrachten Sie die folgende Situation: Sie koordinieren die Platzvergabe für Volkshochschulkurse. Als Grundlage für die Vergabe haben Sie eine Menge von n Kursen $K = \{k_1, \ldots, k_n\}$ und eine Menge von m Personen $P = \{p_1, \ldots, p_m\}$. Jede Person kann an höchstens einem Kurs teilnehmen und Sie haben von allen Personen Angaben darüber, welche Kurse sie interessieren. Diese Angaben liegen in Form einer Funktion $I : P \times K \to \{0, 1\}$ vor, wobei der Funktionswert 1 angibt, dass sich die jeweilige Person für den jeweiligen Kurs interessiert. Außerdem gibt die Kapazitätsfunktion $C : K \to \mathbb{N}$ zu jedem Kurs an, wieviele Personen maximal am jeweiligen Kurs teilnehmen können.

Geben Sie ein Verfahren **basierend auf Verfahren aus der Vorlesung** an, das berechnet, welche Personen welchen Kursen zugeteilt werden sollten, damit so viele Personen wie möglich einen Kurs besuchen, ohne die Kapazitäten der Kurse zu überschreiten und ohne dass eine Person mehreren Kursen zugeteilt wird. Ihr Verfahren muss eine Laufzeitkomplexität in $\mathcal{O}((n+m)^5)$ haben (Sie brauchen nicht zu beweisen, dass diese Laufzeitschranke eingehalten wird).

Lösung:

a) **Behauptung:** Die Aussage gilt nicht.

Beweis:
Betrachten wir folgenden Graphen:

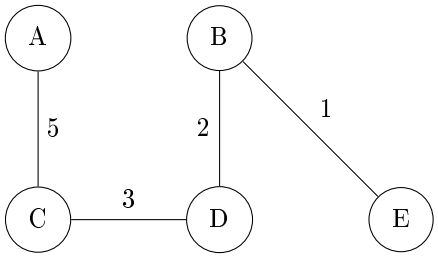


In diesem Graphen liefert uns der Dijkstra Algorithmus vom Startknoten A aus die Distanz 2 zum Knoten B. Offensichtlich ist aber die kürzeste Distanz 1. Damit ist die Aussage widerlegt. □

b) Der Algorithmus von Prim füllt die Tabelle wie folgt aus:

#Iteration	A	B	C	D	E
1	$\underline{0}$	∞	∞	∞	∞
2		6	$\underline{5}$	∞	∞
3		6		$\underline{3}$	∞
4		$\underline{2}$			4
5					$\underline{1}$

Hierbei gibt eine unterstrichene Zahl an in welcher Iteration (zugehöriger Zeilenkopf) welcher Knoten (zugehöriger Spaltenkopf) durch `extractMin(Q)` gewählt wurde. Wir erhalten den folgenden minimalen Spannbaum:



c) Die Tabellen werden durch den Algorithmus von Floyd wie folgt gefüllt:

(1)	A	B	C	D
A	0	5	∞	∞
B	∞	0	1	7
C	3	∞	0	2
D	∞	4	∞	0

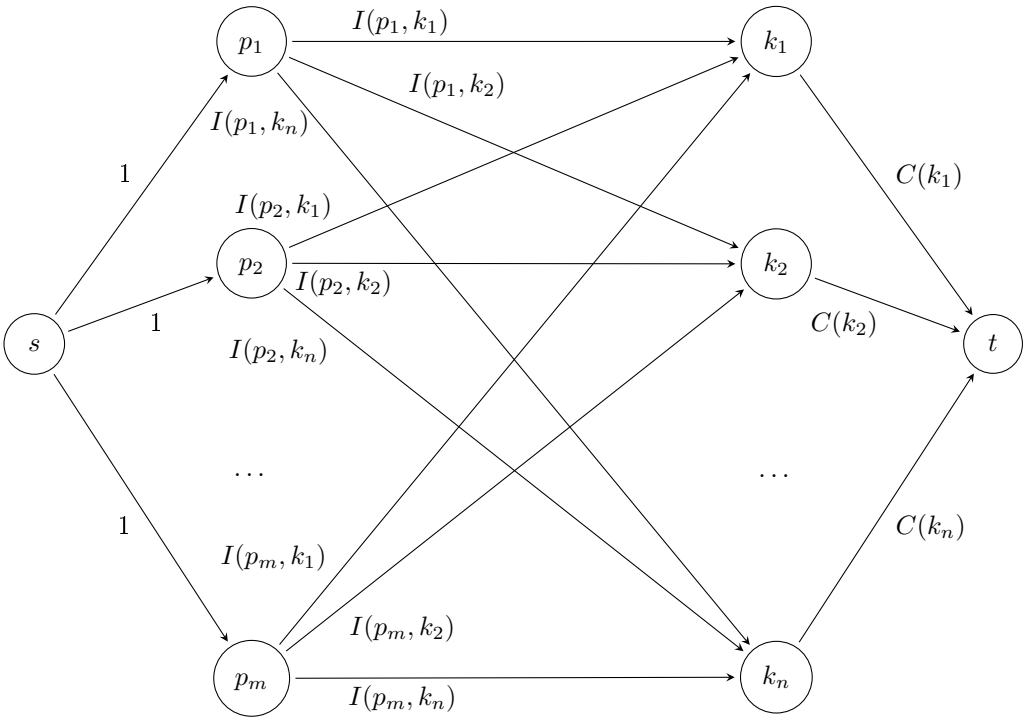
(2)	A	B	C	D
A	0	5	∞	∞
B	∞	0	1	7
C	3	8	0	2
D	∞	4	∞	0

(3)	A	B	C	D
A	0	5	6	12
B	∞	0	1	7
C	3	8	0	2
D	∞	4	5	0

(4)	A	B	C	D
A	0	5	6	8
B	4	0	1	3
C	3	8	0	2
D	8	4	5	0

(5)	A	B	C	D
A	0	5	6	8
B	4	0	1	3
C	3	6	0	2
D	8	4	5	0

d) Wir modellieren das Problem als Flussnetzwerk wie folgt: Die Zustandsmenge V ist $\{s,t\} \cup K \cup P$ mit Quelle s und Senke t. Die Kantenmenge E definieren wir folgendermaßen: Von der Quelle geht jeweils eine Kante mit Kapazität 1 zu jedem Zustand aus P. Von einer Person p_i zu einem Kurs k_j existiert eine Kante mit Kapazität 1 genau dann, wenn $I(p_i, k_j) = 1$ gilt. Von jedem Zustand $k_i \in K$ geht genau eine Kante mit Kapazität $C(k_i)$ zur Senke. Ansonsten gibt es keine Kanten. Die Ford-Fulkerson Methode implementiert durch den Edmonds-Karp Algorithmus liefert uns einen maximalen Fluss in diesem Netzwerk in $\mathcal{O}(|V| \cdot |E|^2)$ Schritten. Da $|V| = n + m + 2$ und $|E| \in \mathcal{O}(|V|^2)$, gilt $|V| \cdot |E|^2 \in \mathcal{O}((n+m)^5)$. Aus dem maximalen Fluss lesen wir die Verteilung auf die Kurse so ab, dass jede Kante zwischen einer Person und einem Kurs, über die dieser Fluss läuft, einer Zuordnung dieser Person zu diesem Kurs entspricht. Zur Illustration der Modellierung dient folgendes Schaubild:



Aufgabe 6 (Dynamische Programmierung): (6 + 10 = 16 Punkte)

a) Gegeben sei ein Array `a` der Länge `n`, wobei die Einträge im Array `a[0]`, ..., `a[n-1]` ganzzahlig sind. Bestimmen Sie eine **Rekursionsgleichung** $L(i)$, die die **Länge der längsten aufsteigenden Teilsequenz** des Arrays `a` bestimmt, die im i-ten Arrayelement endet, wobei $0 \leq i < n$. Wie kann man mit dieser Rekursionsgleichung die Länge der längsten aufsteigenden Teilsequenz des Arrays `a` bestimmen?

Beispiel: Die längsten aufsteigenden Teilsequenzen des Arrays `[6, 14, 1, 9, 5, 13]` sind `(6, 9, 13)`, `(1, 9, 13)` und `(1, 5, 13)`, wobei $L(0) = L(2) = 1$, $L(1) = L(3) = L(4) = 2$ und $L(5) = 3$.

b) Bestimmen Sie die **längste gemeinsame Teilsequenz** der Sequenzen `DOSTAL` und `EDISAFL`. Benutzen Sie hierfür den in der Vorlesung vorgestellten Algorithmus mit dynamischer Programmierung und füllen Sie die folgende Tabelle aus. Beschreiben Sie wie man anhand der Tabelle die längste gemeinsame Teilsequenz der gegebenen Wörter und die Länge dieser Teilsequenz bestimmen kann.

	∅	E	D	I	S	A	F	L
∅								
D								
O								
S								
T								
A								
L								

Lösung:

a) Sei $M(i) = \{j \mid 0 \leq j < n,\ j < i,\ \texttt{a[j]} < \texttt{a[i]}\}$, dann ist:

$$L(i) = \begin{cases} 1, & \text{falls } M(i) = \emptyset \\ max(\{L(j) \mid j \in M(i)\}) + 1, & \text{sonst} \end{cases}$$

Die Länge der längsten aufsteigenden Teilsequenz des Arrays `a` ist dann $max(\{L(i) \mid 0 \leq i < n\})$.

b) Die Tabelle wird vom Algorithmus wie folgt gefüllt:

	∅	E	D	I	S	A	F	L
∅	0	0	0	0	0	0	0	0
D	0	0	↖ 1	1	1	1	1	1
O	0	0	↑ 1	← 1	1	1	1	1
S	0	0	1	1	↖ 2	2	2	2
T	0	0	1	1	↑ 2	2	2	2
A	0	0	1	1	2	↖ 3	← 3	3
L	0	0	1	1	2	3	3	↖ 4

Also erhalten wir die Sequenz `DSAL` als längste gemeinsame Teilsequenz der Sequenzen `DOSTAL` und `EDISAFL`.

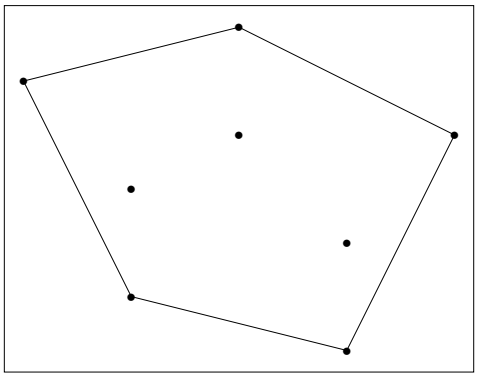
Dies lässt sich von der Tabelle wie folgt ablesen: Wenn eine Zeile einen Pfeil nach links oben enthält dann ist der Buchstabe, der den Zeilenkopf bildet, teil der längsten gemeinsamen Teilsequenz. Die Pfeile zeigen dabei an wie der folgende Algorithmus für gegebene Wörter `wordA` und `wordB` durch die erstellte Tabelle `C` läuft:

```
int i = wordA.length(); int j = wordB.length();
while (i > 0 && j > 0) {
    if (wordA.charAt(i-1) == wordB.charAt(j-1)) { i--; j--; }
    else if (C[i][j-1] >= C[i-1][j]) j--;
    else i--;
}
```

Aufgabe 7 (Geometrische Algorithmen): (2 + 2 + 3 + 5 = 12 Punkte)

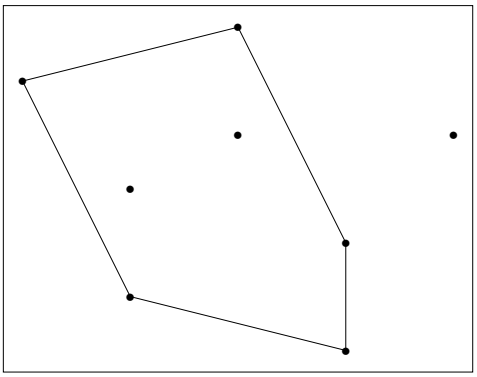
a) Geben Sie die Definition der konvexen Hülle einer endlichen Menge von Punkten an.

b) Welche der folgenden Strukturen ist eine konvexe Hülle der eingezeichneten Punktmenge? Begründen Sie ihre Antwort kurz, wenn keine konvexe Hülle vorliegt.



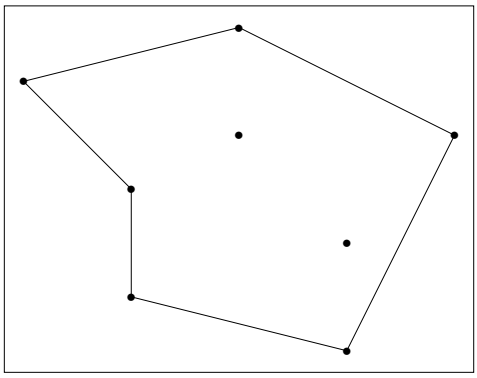
☐ Ja

☐ Nein, weil:



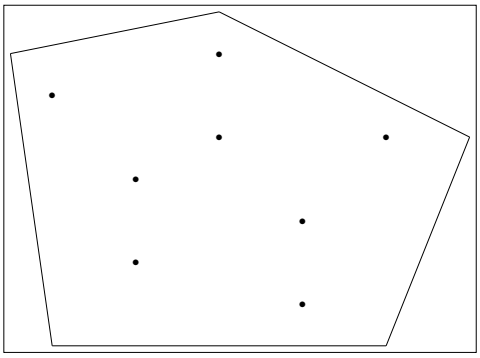
☐ Ja

☐ Nein, weil:



☐ Ja

☐ Nein, weil:



☐ Ja

☐ Nein, weil:

c) Welche Komplexität hat der Grahamsche Scan (O-Notation), wenn ein Sortierverfahren mit der Laufzeit $\mathcal{O}(n \cdot \log n)$ verwendet wird? Sie dürfen dabei die Berechnung des Polarwinkels als konstant annehmen.

d) Für den Grahamschen Scan muss festgestellt werden, ob sich ein Punkt links oder rechts von einem gegebenen Liniensegment befindet. Berechnen Sie mit der **Determinantenmethode** für das Liniensegment zwischen den Punkten $(1,2)^T$ und $(5,3)^T$, ob sich die Punkte $P_1 = (2,2)^T$ und $P_2 = (4,1)^T$ jeweils rechts oder links davon befinden.

Lösung:

a) **Alternative I:** Die konvexe Hülle einer endlichen Menge Q von Punkten ist das kleinste konvexe Polygon P, für das sich jeder Punkt in Q entweder auf dem Rand von P oder in seinem Inneren befindet.

Alternative II: Die konvexe Hülle einer endlichen Menge Q von Punkten ist das kleinste Polytop (konvexes Polygon) P mit $Q \subseteq P$.

Alternative III: Gegeben sei ein Vektorraum V. Die konvexe Hülle einer Menge $Q \subseteq V$ ist die kleinste konvexe Menge $P \subseteq V$ mit $Q \subseteq P$.

Alternative IV: Gegeben sei ein Vektorraum V. Die konvexe Hülle einer Menge $Q \subseteq V$ ist der Schnitt $\bigcup_{\substack{Q \subseteq P \subseteq V \\ P \text{ konvex}}} P$ aller konvexen Obermengen von Q in V.

b)
- Ja.
- Nein, nicht alle Punkte enthalten.
- Nein, nicht konvex.
- Nein, nicht minimal.

c) Die Komplexität ist in $\mathcal{O}(n \log n + n)$.

- Referenzpunkt finden geht in $\mathcal{O}(n)$.
- Das initiale Sortieren hat bei einem geeigneten Sortierverfahren eine Laufzeit in $\mathcal{O}(n \log n + n)$.
- Da der Polarwinkel konstant berechnet werden kann, ist die Berechnung der Richtungen in $\mathcal{O}(1)$.
- Die innere Schleife nimmt nur Punkte aus der Menge, die vorher in der äußeren hinzugefügt wurden, dies ist also in $\mathcal{O}(n)$.

Deshalb hängt die Laufzeit nur von der Komplexität des Sortierverfahrens ab und ist somit in $\mathcal{O}(n \log n + n)$.

d) Den Vektor der Strecke bekommt man, indem man vom Endpunkt den Startpunkt abzieht. Dies ergibt $\vec{a} = (4, 1)^T$. Da unsere Strecke nicht bei $(0, 0)^T$ ihren Ursprung hat, müssen auch die Punktvektoren nun normiert werden, es wird also der Startpunkt jeweils abgezogen:

$$\vec{P}_1' = (1, 0)^T \text{ und } \vec{P}_2' = (3, -1)^T$$

Nun kann die Determinantenmethode angewandt werden und es ergibt sich

$$\det(\vec{a}, \vec{P}_1') = \det \begin{pmatrix} 4 & 1 \\ 1 & 0 \end{pmatrix} = -1$$
$$\det(\vec{a}, \vec{P}_2') = \det \begin{pmatrix} 4 & 3 \\ 1 & -1 \end{pmatrix} = -7$$

Somit befinden sich beide Punkte rechts von dem Liniensegment.