Theorieaufgabe T1. **/ 16 P**

Hinweise:

1) In dieser Aufgabe sollen Sie **nur die Ergebnisse** angeben. Diese können Sie direkt bei den Aufgaben notieren.
2) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

/ 1 P a) Gegeben sei das folgende Array, das nach einem Quicksort-Aufteilungsschritt entstanden ist. Welcher Schlüssel wurde als Pivotelement verwendet? Markieren Sie *alle* möglichen Kandidaten.

1	4	3	**5**	7	6	**8**	10	9
1	2	3	4	5	6	7	8	9

Lösung: Für ein Pivotelement gilt, dass nach dem Aufteilungsschritt alle Elemente links davon kleiner sind und alle Elemente rechts davon grösser. Im gegeben Array gilt das für Schlüssel 1, 5 und 8.

/ 1 P b) Das folgende Array enthält die Elemente eines in üblicher Form gespeicherten Min-Heaps. Entfernen Sie das minimale Element aus dem Heap, stellen Sie die Heap-Bedingung wieder her, und geben Sie das resultierende Array an.

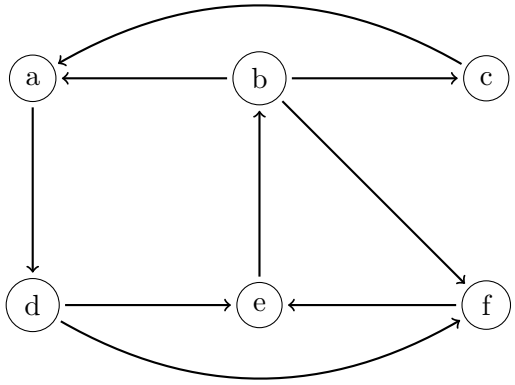
2	5	3	7	6	13	4	9	8
1	2	3	4	5	6	7	8	9

3	5	4	7	6	13	8	9
1	2	3	4	5	6	7	8

/ 1 P c) Fügen Sie die Schlüssel 27, 6, 9, 5 in dieser Reihenfolge in die untenstehende Hashtabelle ein. Benutzen Sie offenes Hashing mit der Hashfunktion $h(k) = k \bmod 11$. Lösen Sie Kollisionen mittels quadratischem Sondieren auf. Im Falle einer Kollision soll die Sondierung zunächst nach links und erst danach nach rechts erfolgen.

	12		**5**	4	**27**	17	**6**	**9**	20	
0	1	2	3	4	5	6	7	8	9	10

/ 1 P d) Gegeben Sei der folgende Graph $G = (V, E)$. Geben Sie eine Menge $E' \subset E$ kleinstmöglicher Kardinalität an, sodass $G' = (V, E \backslash E')$ topologisch sortiert werden kann. Geben Sie ausserdem eine topologische Sortierung für G' an.



$E' = \{$ (e, b) $\}$

topologische Sortierung für G':

b, c, a, d, f, e

/ 2 P e) Kreuzen Sie an, ob die folgenden Aussagen über den gerichteten Graphen G mit der folgenden Adjazenzmatrix wahr oder falsch sind. Jede korrekte Antwort gibt 0,5 Punkte, für jede falsche Antwort werden 0,5 Punkte abgezogen. Eine fehlende Antwort gibt 0 Punkte. Insgesamt gibt die Aufgabe mindestens 0 Punkte. Sie müssen Ihre Antworten nicht begründen.

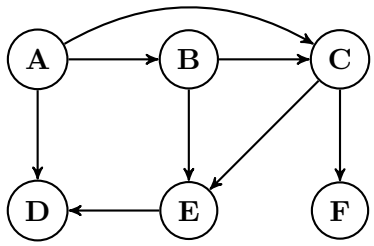
Adjazenzmatrix von G:

	A	B	C	D
A	0	1	0	0
B	0	0	1	1
C	0	0	0	0
D	0	0	1	0

G ist kreisfrei.	☒ Wahr	☐ Falsch
Die einzige topologische Sortierung der Knoten in G ist A, C, B, D.	☐ Wahr	☒ Falsch
Tiefen- und Breitensuche ausgehend von Knoten A besuchen die Knoten von G in derselben Reihenfolge (Nachbarknoten werden in alphabetischer Reihenfolge bearbeitet).	☒ Wahr	☐ Falsch
Die reflexive, transitive Hülle von G enthält gleiche viele Kanten wie G selbst.	☐ Wahr	☒ Falsch

/ 1 P

f) Geben Sie an, in welcher Reihenfolge die Knoten des folgenden Graphen bei einer Tiefensuche und bei einer Breitensuche *ausgehend von Startknoten A* besucht werden. Die Nachbarknoten werden jeweils in alphabetischer Reihenfolge abgearbeitet.



Tiefensuche:

A,B,C,E,D,F

Breitensuche:

A,B,C,D,E,F

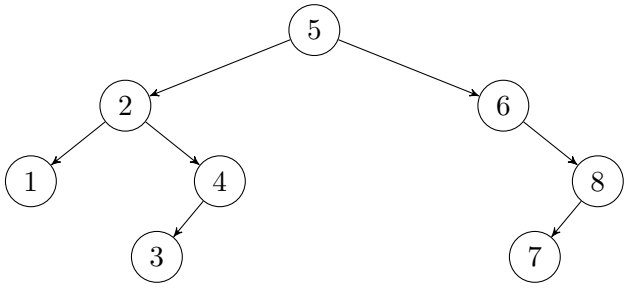
/ 1 P

g) Wieviele Kanten kann ein gerichteter Graph mit n Knoten höchstens haben, wenn er eine topologische Sortierung besitzt? Geben Sie eine geschlossene Formel an. Sie müssen Ihre Antwort nicht begründen.

Lösung: Um die maximale Anzahl an Kanten zu berechnen, zählen wir, wie viele ausgehende Kanten jeder Knoten haben kann. Es gibt mindestens einen Knoten, der keine eingehende Kanten besitzt, nämlich den ersten Knoten der topologischen Sortierung. Er kann höchstens $n-1$ ausgehende Kanten (zu allen anderen Knoten) besitzen. Der nächste Knoten in der topologischen Sortierung kann noch höchstens $n-2$ ausgehende Kanten besitzen (zu allen nachfolgenden Knoten in der topologischen Sortierung). Allgemein kann der i-te Knoten der topologischen Sortierung höchstens $i-1$ ausgehende Kanten haben (zu allen folgenden Knoten in der topologischen Sortierung). Daher hat ein gerichteter Graph mit n Knoten und einer topologischen Sortierung höchstens $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \binom{n}{2}$ Kanten.

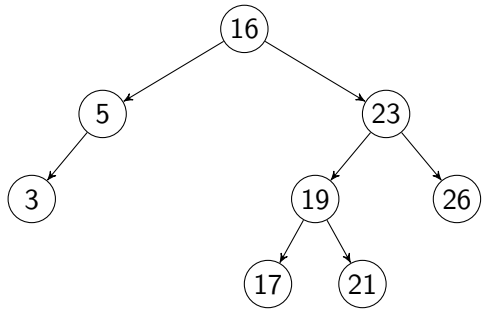
/ 1 P

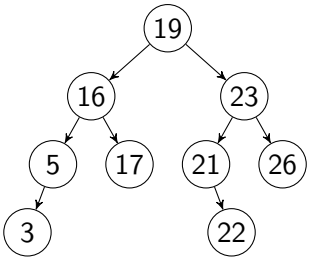
h) Zeichnen Sie den binären Suchbaum mit Schlüsselmenge $\{1, 2, \ldots, 8\}$, bei dem die Preorder-Reihenfolge der Schlüssel mit $5, 2, 1, 4, 3$ beginnt und die Postorder-Reihenfolge der Schlüssel mit $7, 8, 6, 5$ endet.

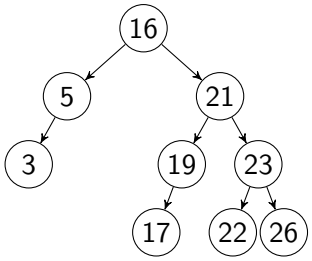


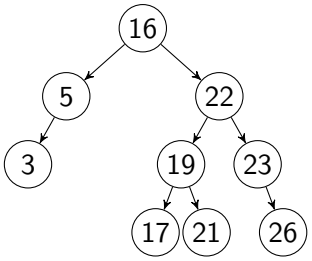
/ 1 P

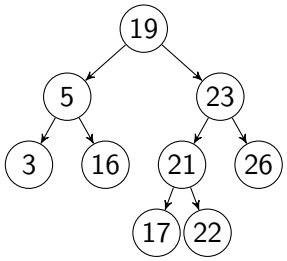
i) Welcher AVL-Baum entsteht, wenn im folgenden AVL-Baum der Schlüssel 22 eingefügt und danach die AVL-Bedingung wiederhergestellt wird?

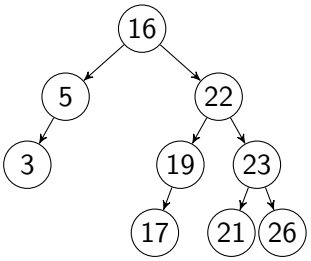


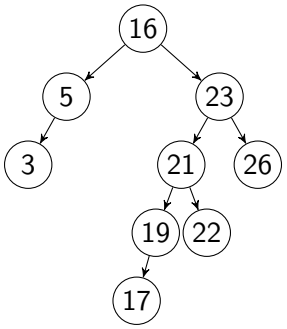
☐ 

■ 

☐ 

☐ 

☐ 

☐ 

/ 3 P j) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 4 \cdot T(n/4) + 9 & n > 1 \\ 5 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (nicht-rekursive) und *möglichst einfache* Formel für $T(n)$ an und beweisen Sie diese mit vollständiger Induktion. Sie können annehmen, dass n eine Potenz von 4 ist. Benutzen Sie also $n = 4^k$ oder $k = \log_4(n)$.

Hinweis: Für $q \neq 1$ gilt: $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

Herleitung (falls benötigt): Lösung: Da wir annehmen dürfen, dass n eine Potenz von 4 ist, gilt $n = 4^k$ für ein $k \in \mathbb{N}$. Wir teleskopieren, um auf eine Formel für $T(n)$ zu kommen:

$$\begin{aligned} T(n) &= 4 \cdot T(n/4) + 9 \\ &= 4 \cdot \big(4 \cdot T(n/4^2) + 9\big) + 9 \\ &= 4 \cdot \Big(4 \cdot \big(4 \cdot T(n/4^3) + 9\big) + 9\Big) + 9 = \dots \\ &= 4^k \cdot T(1) + 9 \cdot \sum_{i=0}^{k-1} 4^i \\ &= 4^k \cdot 5 + 9 \cdot \frac{4^k - 1}{4 - 1} \\ &= 4^k \cdot 5 + 3 \cdot 4^k - 3 \\ &= 8 \cdot 4^k - 3 \end{aligned}$$

Geschlossene und vereinfachte Formel:

$$T(n) = T(4^k) = 8 \cdot 4^k - 3$$

Induktionsbeweis:

Wir beweisen nun unsere Annahme durch vollständige Induktion über k.

Induktionsverankerung ($k = 0$): Es gilt $T(4^0) = T(1) = 5 = 8 \cdot 4^0 - 3$.

Induktionsannahme: Für ein $k \in \mathbb{N}_0$ sei $T(4^k) = 8 \cdot 4^k - 3$.

Induktionsschritt ($k \to k + 1$):

$$T(4^{k+1}) = 4 \cdot T(4^k) + 9 \overset{\text{Ind.-Ann.}}{=} 4 \cdot (8 \cdot 4^k - 3) + 9 = 8 \cdot 4 \cdot 4^k - 12 + 9 = 8 \cdot 4^{k+1} - 3.$$

/ 1 P

k) Geben Sie für das folgende Codefragment in Θ-Notation (so knapp wie möglich) an, wie oft die Funktion f in Abhängigkeit von $n \in \mathbb{N}$ asymptotisch gesehen aufgerufen wird. Die Funktion f ruft sich nicht selbst wieder auf. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = 1; i <= n; i = i+5) {
2         for(int j = i; j >= 1; j = j/2)
3                 f();
4         for(int k = i*i; k >= 1; k = k–1)
5                 f();
6 }
```

Anzahl der Funktionsaufrufe in möglichst knapper Θ-Notation:

$\Theta(n^3)$

Lösung: Die äussere Schleife wird $\mathcal{O}(n)$ Mal durchlaufen. Die erste innere Schleife wird $\Theta(\log i)$ Mal durchlaufen. Die zweite innere Schleife dekrementiert den Wert von k in jedem Schritt um 1 bis $k < 1$ gilt, wird also $\Theta(i^2)$ Mal durchlaufen. Da i für mindestens $n/2$ Werte mindestens $n/2$ beträgt und i niemals grösser als n wird, wird f asymptotisch $\Theta(n \cdot (\log n + n^2))) = \Theta(n^3)$ Mal aufgerufen.

/ 1 P

l) Geben Sie für das folgende Codefragment in Θ-Notation (so knapp wie möglich) an, wie oft die Funktion f in Abhängigkeit von $n \in \mathbb{N}$ asymptotisch gesehen aufgerufen wird. Die Funktion f ruft sich nicht selbst wieder auf. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = n; i >= 1; i = i/2) {
2         f();
3         for(int j = 1; j <= n*n*n; j = 3*j)
4                 f();
5 }
```

Anzahl der Funktionsaufrufe in möglichst knapper Θ-Notation:

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

Lösung: Die äussere Schleife wird $\Theta(\log(n))$ Mal ausgeführt und die innere Schleife in jedem Durchlauf $\Theta(\log_3(n^3)) = \Theta(3\log(n)) = \Theta(\log(n))$ Mal. Daher liegt die Anzahl der Aufrufe von f in $\Theta(\log(n)^2)$.

/ 1 P

m) Geben Sie für die untenstehenden Funktionen eine **Reihenfolge** an, so dass folgendes gilt: Wenn eine Funktion f links von einer Funktion g steht, dann gilt $f \in \mathcal{O}(g)$.

Beispiel: Die drei Funktionen n^3, n^7, n^9 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in \mathcal{O}(n^7)$ und $n^7 \in \mathcal{O}(n^9)$ gilt.

$$\log(\sqrt{n}),\ (\sqrt{n})^{\sqrt{n}},\ \sqrt{\log n},\ \log(10),\ n^2,\ n\sqrt{n},\ \frac{\sqrt{n}}{\log n}$$

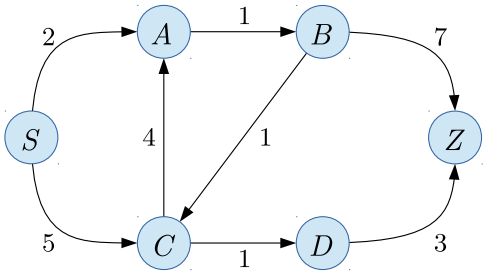
Lösung:

$$\log(10),\ \sqrt{\log n},\ \log(\sqrt{n}),\ \frac{\sqrt{n}}{\log n},\ n\sqrt{n},\ n^2,\ (\sqrt{n})^{\sqrt{n}}$$

/ 12 P

Theorieaufgabe T2.

Sie haben ein Elektroauto und wollen von S nach Z fahren. Das Navigationssystem Ihres Autos hat eine Strassenkarte, in der alle Orte verzeichnet sind. Wenn es eine direkte Strassenverbindung von einem Ort i zu einem Ort j gibt, auf der keine weiteren Orte liegen, dann nennen wir j einen *direkt benachbarten Ort von* i (in der Abbildung rechts ist etwa B direkt benachbart von A, aber nicht von S). Von einem Ort i gelangt man zu einem direkt benachbarten Ort j in Zeit $l_{ij} \geq 0$.



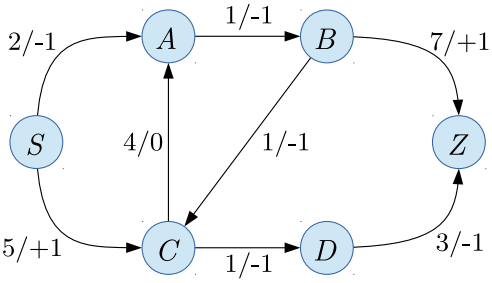
Modellieren Sie die nachfolgenden drei Fragestellungen jeweils als graphentheoretisches Problem. Geben Sie dazu an, welche Knoten und Kanten definiert werden und welche Gewichte den Kanten zugeordnet werden. Erläutern Sie in Teil b) und c), wie die Lösung der Fragestellung von einem geeigneten Weg abgelesen werden kann. Geben Sie einen effizienten Algorithmus zur Lösung des graphentheoretischen Problems an und bestimmen Sie seine Laufzeit in Abhängigkeit von der Anzahl der Orte n und der Anzahl direkter Strassenverbindungen m.

/ 2 P

a) Gesucht wird ein schnellster Weg von S nach Z. In dem oben dargestellten Strassennetz ist dies der Weg (S, A, B, C, D, Z) mit Reisezeit 8. Vereinfachend nehmen wir für diese Teilaufgabe an, dass die Batterie des Elektroautos hinreichend gross dimensioniert ist, sodass Sie sich über das Nachladen keine Gedanken machen müssen.

/ 6 P

b) Ab sofort nehmen wir an, dass die Batterie Ihres Elektroautos fünf Ladezustände $0, \ldots, 4$ annehmen kann (wobei 0 einer leeren und 4 einer komplett geladenen Batterie entspricht). Auf dem Weg von einem Ort i zu einem direkt benachbarten Ort j verändert sich der Ladezustand der Batterie um $e_{ij} \in \{-1, 0, 1\}$. Ein Wert $e_{ij} = +1$ bedeutet, dass die Batterie auf diesem Strassensegment aufgeladen wird (weil die Strasse abschüssig ist). Die Batterie kann aber niemals mehr als den Ladezustand 4 erreichen.



Befindet sich das Fahrzeug bei einem Ort i und hat die Batterie Ladezustand 0, dann darf ein Strassensegment zu einem benachbarten Ort j nur dann befahren werden, wenn dort keine Energie benötigt wird. Zu Beginn der Fahrt ist die Batterie voll geladen.

Gesucht wird nun ein schnellster Weg von S nach Z, auf dem der Ladezustand der Batterie niemals negativ wird. In dem in der vorigen Abbildung dargestellten Strassennetz ist dies der Weg (S, C, D, Z) mit Reisezeit 9. Der Weg (S, A, B, C, D, Z) mit Reisezeit 8 ist keine gültige Lösung, da die Batterie bei Ort D Ladezustand 0 besitzt und somit zur Fahrt nach Z keine Energie mehr übrig ist.

/ 4 P

c) Wie in b) nehmen wir an, dass die Batterie fünf Ladezustände besitzt und zu Fahrtbeginn voll geladen ist. Gesucht wird nun ein (nicht notwendigerweise schnellster) Weg von S nach Z, auf dem der Ladezustand der Batterie niemals negativ wird, und bei dessen Ankunft bei Z die Batterie so voll wie möglich ist. In der unteren Abbildung auf der Seite vorher ist dies (S, C, A, B, Z), bei dessen Ankunft die Batterie voll geladen ist (bei S, C und A ist der Ladezustand 4, bei B dann 3 und bei Z wieder 4).

Teilaufgabe a)

- Definition des Graphen inklusive Kantengewichten (wenn möglich in Worten und nicht formal):

  Wir definieren einen gerichteten Graphen mit n Knoten und m Kanten. Die Knoten repräsentieren die n Orte und die Kanten die m direkten Strassenverbindungen. Es führt also genau dann eine Kante von Knoten v_i zu Knoten v_j, wenn man von Ort i zu Ort j durch eine direkte Strassenverbindung gelangen kann. Das Gewicht einer Kante von v_i zu v_j entspricht der benötigten Zeit l_{ij}.

- Anzahl der Knoten und Kanten in Abhängigkeit von n und m (in möglichst knapper Θ-Notation):

  Der Graph beinhaltet genau $n \in \Theta(n)$ Knoten und $m \in \Theta(m)$ Kanten.

- Möglichst effizienter Algorithmus zur Berechnung eines kürzesten Wegs von S nach Z:

  Wir können den kürzesten Weg im Graphen mit dem Algorithmus von Dijkstra berechnen, da der Graph keine Kanten mit negativen Gewichten enthält.

- Laufzeit (in möglichst knapper Θ-Notation):

  $\Theta(n \log(n) + m)$

Teilaufgabe b)

- Definition des Graphen inklusive Kantengewichten (wenn möglich in Worten und nicht formal):

  Wir definieren einen gerichteten Graphen mit fünf Knoten für jeden der n Orte. Die fünf Knoten entsprechen den fünf möglichen Ladezuständen, mit denen das Elektroauto den entsprechenden Ort erreicht. Der Knoten $v_{i,j}$ entspricht dem Ort i mit Ladezustand j. Für jede direkte Strassenverbindung von einem Ort a zu einem Ort b fügen wir Kanten zum Graph hinzu, welche den Veränderungen der Ladezustände entsprechen. Formal fügen wir für eine Strassenverbindung von a nach b Kanten von $v_{a,i}$ zu $v_{b,i+e_{ab}}$ für alle $i \in \{0, \ldots, 4\}$ mit $0 \leq i + e_{ab} \leq 4$ zum Graphen hinzu. Zusätzlich fügen wir die Kante von $v_{4,i}$ zu $v_{4,j}$ hinzu, falls $e_{ab} = +1$ gilt, denn die Batterie kann keinen höheren Ladezustand als 4 besitzen. Wir fügen auch noch einen weiteren Knoten t hinzu, der ausschliesslich von den Knoten für Zielort Z direkt erreichbar ist. Wir fügen also Kanten von $v_{z,i}$ zu t für alle $i \in \{0, \ldots, 4\}$ hinzu, wobei jede dieser Kanten Gewicht 0 hat.

- Anzahl der Knoten und Kanten in Abhängigkeit von n und m (in möglichst knapper Θ-Notation):

  Der Graph enthält $5 \cdot n + 1 \in \Theta(n)$ Knoten. Für jede direkte Strassenverbindung enthält der Graph höchstens 5 Kanten und weiters 5 zusätzliche Kanten für Knoten t, insgesamt also $\Theta(m)$ Kanten.

- Ablesen der Lösung:

  Die Lösung entspricht einem kürzesten Pfad von $v_{s,4}$ zu t.

- Möglichst effizienter Algorithmus zur Berechnung der kürzesten Wege:

  Wir verwenden wieder den Algorithmus von Dijkstra, da der Graph keine Kanten mit negativen Gewichten enthält.

- Laufzeit (in möglichst knapper Θ-Notation):

  $\Theta(n \log(n) + m)$

Teilaufgabe c)

- Modellierung als graphentheoretisches Problem:

  Wir verwenden denselben Graph wie in Aufgabe b), ignorieren jedoch die Kantengewichte und entfernen den zusätzlichen Knoten t und alle eingehenden Kanten.

  Um herauszufinden, ob das Auto den Zielort mit voller Batterie erreichen kann, müssen wir berechnen, ob Knoten $v_{z,4}$ vom Startknoten $v_{s,4}$ erreichbar ist. Falls ein Weg zwischen diesen beiden Knoten existiert, können wir das Ziel mit voller Batterie erreichen. Andernfalls prüfen wir nacheinander, ob Knoten $v_{z,3}$, $v_{z,2}$, $v_{z,1}$ oder $v_{z,0}$ vom Startknoten $v_{s,4}$ erreichbar ist. Sobald wir einen Zielknoten gefunden haben, der erreichbar ist, stellt der entsprechende Weg von $v_{s,4}$ die Lösung dar.

- Möglichst effizienter Algorithmus zur Wegberechnung:

  Wir können die Erreichbarkeit der Zielknoten mit einer einfachen Breitensuche berechnen.

- Laufzeit (in möglichst knapper Θ-Notation):

  $\Theta(m + n)$

/ 12 P

Theorieaufgabe T3.

Gegeben sei ein Array $A[1..n]$, das eine Zeichenkette $\langle a_1, \ldots, a_n \rangle$ der Länge n speichert. Ein *Palindrom* ist eine Zeichenkette, die sich von vorne wie von hinten gleich liest, also z.B. das Wort RENTNER. Formal ist ein Palindrom eine Zeichenkette $\langle a_1, \ldots, a_n \rangle$, sodass entweder $n = 1$ gilt, oder es gilt $a_1 = a_n$ und $\langle a_2, \ldots, a_{n-1} \rangle$ ist ebenfalls ein Palindrom (für $n = 2$ fordern wir nur $a_1 = a_2$). Ein Subarray $A[i..j]$, $1 \leq i \leq j \leq n$, heisst *Palindrom in A*, falls $\langle A[i], \ldots, A[j] \rangle$ ein Palindrom ist.

Wir möchten das gegebene Array A in möglichst wenige Subarrays zerlegen, sodass jedes Zeichen $A[i]$ für $i \in \{1, \ldots, n\}$ in genau einem Subarray enthalten ist und jedes Subarray ein Palindrom in A ist. Berechnen Sie, in wie viele Subarrays A *mindestens* zerlegt werden muss.

Beispiel: Das Array [B, A, B, A] kann in zwei Subarrays [B] und [A, B, A] zerlegt werden, die beide Palindrome in A sind. Eine Zerlegung in weniger Subarrays nicht möglich, da das Array selbst kein Palindrom ist. Es gibt eine weitere Zerlegung in zwei Subarrays [B, A, B] und [A]. Das Array [A, B, C, D] kann in vier Subarrays [A], [B], [C] und [D] zerlegt werden und es gibt keine Zerlegung in weniger Subarrays. Das Array [A, B, A] ist selbst ein Palindrom und kann daher in ein einziges Subarray zerlegt werden.

/ 10 P a) Geben Sie einen möglichst effizienten Algorithmus nach dem Prinzip der dynamischen Programmierung an, der als Eingabe ein Array $A[1..n]$ erhält, und der berechnet, in wie viele Subarrays das gegebene Array mindestens zerlegt werden muss, damit jedes Subarray ein Palindrom in A ist. Erklären Sie dabei die folgenden Aspekte:

1) Was ist die Bedeutung eines Tabelleneintrags, und welche Grösse hat die DP-Tabelle?
2) Wie kann die Tabelle initialisiert werden, und wie berechnet sich ein Tabelleneintrag aus früher berechneten Einträgen?
3) In welcher Reihenfolge können die Einträge berechnet werden?
4) Wie lässt sich die Lösung aus der DP-Tabelle auslesen?

/ 2 P b) Geben Sie die asymptotischen Laufzeiten Ihres Algorithmus an und begründen Sie diese.

Wählen Sie als erstes die Dimension Ihrer DP-Tabelle:

- ☒ **(I) eindimensional** → Benutzen Sie das Schema auf Seite 13.
- ☒ **(II) zweidimensional** → Benutzen Sie das Schema auf Seite 14.
- ☐ **(III) andere Dimension** → Beschreiben Sie Ihr dynamisches Programm auf Seite 15.

Beantworten Sie die Teilaufgabe b) auf Seite 16.

Lösung: **Wir beschreiben auf Seite 13 einen $\mathcal{O}(n^2)$-Algorithmus für das Problem. Auf Seite 14 beschreiben wir einen langsameren $\mathcal{O}(n^3)$-Algorithmus.**

(I) Für Algorithmen mit einer eindimensionalen DP-Tabelle:

Grösse der DP-Tabelle / Anzahl Einträge:

☐ n ☐ n^2 ☐ 2^n ☐ $n!$ ☒ andere: $n+1$

Bedeutung eines Tabelleneintrags:

Bedeutung des Eintrags an Position i:

$DP[i]$: minimale Anzahl Subarrays, welche für eine Zerlegung von $A[1..i]$ benötigt werden.

Initialisierung:

$$DP[0] = 0$$

Berechnung eines Eintrags:

Zur Berechnung eines Eintrags $DP[i]$ müssen wir für jedes i, j mit $1 \leq i < j \leq n$ wissen, ob das Subarray $A[i..j]$ ein Palindrom in A ist. In den Übungen haben wir gesehen, wie wir diese Informationen (für alle Paare $(i, j),\ i < j$) in einer Tabelle $P[i, j]$ in $\mathcal{O}(n^2)$ vorberechnen können. Wir haben "$A[k+1..i]$ ist Palindrom in A" $\iff$ "$P[i, j] ==$ true".
Damit sind Palindromabfragen nun in konstanter Zeit möglich.

$$DP[i] = \min\big\{DP[k] + 1 \mid \text{über alle } 0 \leq k < i \text{ für welche gilt: } A[k+1..i] \text{ ist Palindrom in } A\big\}$$

Berechnungsreihenfolge:

Aufsteigende i von 1 bis n.

Auslesen der Lösung:

Nachdem alle Einträge berechnet wurden, steht die Lösung in $DP[n]$.

(II) Für Algorithmen mit einer zweidimensionalen DP-Tabelle:

Grösse der DP-Tabelle: *Anzahl Zeilen*

☒ n	☐ 2^n	☐ andere:
☐ n^2	☐ $n!$	_____

Grösse der DP-Tabelle: *Anzahl Spalten*

☒ n	☐ 2^n	☐ andere:
☐ n^2	☐ $n!$	_____

Bedeutung eines Tabelleneintrags:

Bedeutung des Eintrags in Zeile i und Spalte j:

$DP[i,j]$: minimale Anzahl Subarrays, welche für eine Zerlegung von $A[i..j]$ benötigt werden.

Initialisierung:

$DP[i,i] = 1$ für alle i.
$DP[i,j] = 0$ für alle $j \neq i$.

Berechnung eines Eintrags: Für alle $j > i$ gilt:

$$DP[i,j] = \begin{cases} 1 & \text{wenn A[i..j] ein Palindrom ist}^* \\ \min_{i \leq k < j}(DP[i,k] + DP[k+1,j]) & \text{sonst} \end{cases}$$

* $A[i..j]$ ist ein Palindrom, falls $A[i] = A[j]$ und $i = j-1$, oder $A[i] = A[j]$ und $DP[i+1, j-1] = 1$.

Berechnungsreihenfolge:

Mit aufsteigender Differenz $j - i$, bei gleicher Differenz aufsteigend nach i.

Auslesen der Lösung:

Nachdem alle Einträge berechnet wurden, steht die Lösung in $DP[1,n]$.

(III) Für Algorithmen mit einer DP-Tabelle von Dimension ≥ 3*:*

Teilaufgabe b)

b) Geben Sie die asymptotischen Laufzeiten Ihres Algorithmus an und begründen Sie diese.

Laufzeiten (in möglichst knapper Θ-Notation) mit Begründung:

Für die Lösung auf Seite 13:

Wie bereits erwähnt können wir im Voraus für jedes $1 \leq i \leq j \leq n$ berechnen, ob $A[i..j]$ ein Palindrom in A ist. Wir verweisen auf die Beispiellösung von Aufgabe 7.1 der Übungen für einen Algorithmus mit Laufzeit in $\mathcal{O}(n^2)$. Anschliessend müssen wir für jeden der $n+1$ Einträge $\mathcal{O}(n)$ vorherige Einträge der Tabelle betrachten. Daher kann die gesamte Tabelle in $\mathcal{O}(n^2)$ berechnet werden. Die Lösung kann in konstanter Zeit ausgelesen werden. Die Gesamtlaufzeit ist daher ebenfalls in $\mathcal{O}(n^2)$.

Für die Lösung auf Seite 14:

Unsere Tabelle hat $\mathcal{O}(n^2)$ Einträge und jeder Eintrag kann in $\mathcal{O}(n)$ Laufzeit berechnet werden. Das Auslesen der Lösung benötigt konstante Zeit, sobald alle Einträge berechnet wurden. Die Gesamtlaufzeit ist daher in $\mathcal{O}(n^3)$.