Instruktionen zur Prüfung Algorithmen & Datenstrukturen

Ablauf

- **Dauer und Gewichtung:** Die Dauer der Prüfung beträgt 4 Stunden. In dieser Zeit müssen die Theorie- und die Programmieraufgaben gelöst werden. Beide Teile sind auf je 2 Stunden Dauer ausgelegt und zählen gleich viel. Trotzdem können Sie individuell mehr oder weniger Zeit pro Teil verwenden. Wir empfehlen Ihnen dringend, mit dem Programmierteil zu beginnen, und nach spätestens 2 Stunden mit dem Theorieteil fortzufahren.
- **Frühe Abgabe:** Aufgrund der Prüfungsstruktur ist eine frühzeitige Abgabe nicht möglich.

Prüfungsbedingungen

- **Störungen:** Melden Sie sich bitte **sofort**, wenn Sie sich während der Prüfung in irgendeiner Weise bei der Arbeit gestört fühlen.
- **Disziplinarordnung:** Betrugsversuche unterstehen den Strafnormen der Disziplinarordnung der ETH und können zur Exmatrikulation führen.
- **Fragen:** Während der Prüfung werden inhaltliche Fragen ausschliesslich über den Judge beantwortet (siehe Punkt 4 der technischen Anleitung). Das gilt auch für die Theorieaufgaben.

Logistik

- **Erlaubte Hilfsmittel:** Ausser Stiften, Wörterbüchern und einer einfachen Uhr sind keine Hilfsmittel erlaubt (keine kommunikationsfähigen, programmierbaren und/oder speicherfähigen Geräte). Vor Beginn der Prüfung sind alle nicht erlaubten Gegenstände wegzuräumen. Jegliche elektronischen Geräte, insbesondere Natels, Smartwatches und Passwort-Sticks sind auszuschalten, dürfen auf keinen Fall benutzt werden und müssen ins Gepäck.
- **Gepäck und Jacken** müssen am Rand der Räume gelagert werden und dürfen nicht mit an den Arbeitsplatz genommen werden. Wir können Ihr Gepäck nicht beaufsichtigen, Sie deponieren es also auf eigenes Risiko, und wir empfehlen Ihnen deshalb keine Wertsachen zur Prüfung mitzubringen. Beschriften Sie Ihre Gegenstände, um Verwechslungen vorzubeugen.
- **Legi-Kontrolle:** Legen Sie Ihre Legi gut sichtbar vor sich auf den Tisch. Bei der Kontrolle nach Prüfungsbeginn werden wir Sie auch bitten, die Submissions-Webseite auf dem Judge zu öffnen. Führen Sie daher gleich nach Beginn die Schritte der technischen Anleitung durch.
- **Essen und Getränke** am Arbeitsplatz sind in vernünftiger Ausprägung erlaubt. Dabei dürfen aber andere Prüflinge nicht durch Gerüche oder Geräusche gestört werden. Erlaubt sind nur trockene (keine flüssigen oder fettigen) Speisen, und Getränke müssen in wiederverschliessbaren Verpackungen aufbewahrt und verschlossen werden, wenn nicht getrunken wird. Bitte sehr vorsichtig sein und hinterher aufräumen.
- **Schallschutz:** Nur Ohropax (oder ein vergleichbares Konkurrenzprodukt) ist erlaubt, kein Kapselgehörschutz, keine Kopfhörer.
- **Toilette:** Wenn Sie während der Prüfung die Toilette benutzen möchten, melden Sie sich per Handzeichen. Eine Aufsichtsperson begleitet Sie.

Programmierteil (Computerprüfung)

- **nethz Passwort:** Zur Anmeldung am Judge benötigen Sie Ihren nethz-Account. Stellen Sie sicher, dass Sie sich an Ihre Logindaten erinnern können.
- **Beste Abgabe zählt:** Genau wie bei den Programmieraufgaben im Semester können Sie die Programmieraufgaben so oft einschicken, wie Sie wollen. Pro Aufgabe zählt die beste Einsendung. Es zählt nur, was auf dem Judge eingereicht wurde.
- **Nicht ausschalten!** Der Computer darf auf keinen Fall ausgeschaltet werden, auch nicht gegen Ende der Prüfung. Es droht der Verlust aller Daten!
- **Bei Problemen:** Falls ein Systemfehler auftritt, melden Sie sich bitte sofort durch Handzeichen und klicken Sie die entsprechende Meldung nicht weg.
- **Java-Bibliotheken:** Sie dürfen nur die Objekte, Funktionen etc. von `java.lang.*` benutzen, und keine anderen Pakete. Das Paket `java.lang` enthält alle Objekte, die standardmässig verfügbar sind, zum Beispiel Math, Integer, String, System, Double, Boolean, etc., und Sie dürfen diese frei verwenden (z.B. `Math.max()` oder `Integer.MAX_VALUE` sind erlaubt).

  Hingegen ist das Importieren oder Benutzen anderer Pakete verboten, z.B. dürfen Sie nicht `java.util.*`, `java.io.*` etc. benutzen. Beachten Sie, dass dies auch die Java Collections verbietet. Die Module, welche im gegebenen Template benutzt werden, um die Eingabe zu lesen, sind die einzige erlaubte Ausnahme.

  Beachten Sie auch, dass diese Restriktion nicht beim Einsenden durch den Judge geprüft wird, aber nach der Prüfung überprüft und gegebenenfalls bestraft wird.

  Das dient nicht dazu, Ihre Aufgabe schwerer zu machen – im Gegenteil (da Sie alle verbotenen Dinge nicht kennen müssen). Es dient lediglich dazu, die Lösungen auf diejenigen Dinge zu beschränken, welche Sie im Rahmen dieser Vorlesung gesehen und verwendet haben.

Theorieteil (Papierprüfung)

- Bitte schreiben Sie Ihre Studierenden-Nummer auf **jedes** Blatt.
- Bitte verwenden Sie für jede Aufgabe ein neues Blatt oder schreiben Sie Ihre Lösung direkt auf das Aufgabenblatt (insbesondere bei Aufgabe T1).
- Eigenes Papier darf nicht verwendet werden. Wir stellen genügend Papier zur Verfügung.
- Bitte schreiben Sie **lesbar** mit **blauer oder schwarzer**, nicht-löschbarer Tinte. Wir werden insbesondere nichts bewerten, was wir nicht lesen können. Die Benutzung von Bleistiften ist nicht erlaubt.
- Pro Aufgabe kann nur eine Lösung angegeben werden. Ungültige Lösungsversuche müssen klar durchgestrichen werden. Formulieren Sie Ihre Lösungen nachvollziehbar.
- Sofern Sie die Notationen, Algorithmen und Datenstrukturen aus der Vorlesung "Algorithmen & Datenstrukturen" verwenden, sind Erklärungen oder Begründungen nicht notwendig. Falls Sie jedoch andere Methoden benutzen, müssen Sie diese **kurz** soweit erklären, dass Ihre Ergebnisse verständlich und nachvollziehbar sind.
- Legen Sie am Ende der Prüfung alle Blätter ausser demjenigen mit Ihrem Namenssticker in das Kuvert und verschliessen Sie es.

/ 20 P

Programmieraufgabe P1.

Passwort für Einschreibung: `algorithmus`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

Rotationen im AVL-Baum implementieren

In dieser Aufgabe sollen Sie einfache Rotationen für Einfügeoperationen in AVL-Bäume implementieren. Der Grossteil der Implementierung eines AVL-Baumes ist bereits in der Vorlage vorhanden (Einlesen der Eingabe, Einfügen eines neuen Elementes, Ausgabe).

Der Baum wird als eine Menge von `Node`-Objekten gespeichert. Jedes `Node`-Objekt v hat fünf Felder:

<parent, leftChild, rightChild, value, balanceFactor>

Einen Zeiger `parent` zum Elternknoten von v im Baum (oder `null` falls v die Wurzel des Baumes ist), zwei Zeiger `leftChild` und `rightChild` zum linken beziehungsweise zum rechten Kindknoten von v (`null` falls kein solcher Kindknoten existiert), eine Ganzzahl `value` von v und die Balance `balanceFactor` von v (d.h. die Höhe des vom rechten Kindknoten von v aufgespannten Teilbaumes minus die Höhe des vom linken Kindknoten von v aufgespannten Teilbaumes).

Beachten Sie, dass es zu jedem Zeiger `leftChild`, `rightChild` oder `parent`, der von einem Knoten v auf einen anderen Knoten u zeigt, einen entsprechenden Zeiger von u auf v gibt. Die AVL-Baum-Implementierung in der Vorlage enthält ausserdem einen Zeiger `root`, der auf den aktuellen Wurzelknoten des Baumes zeigt (`root` ist `null` wenn der Baum leer ist).

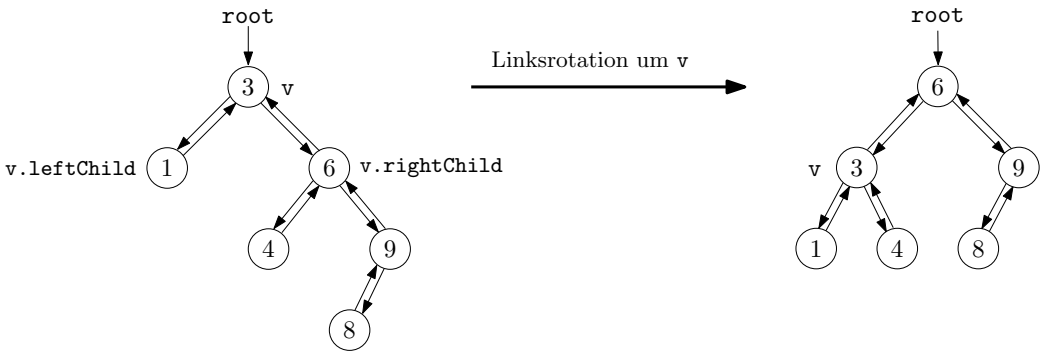
Vervollständigen Sie den Code der Funktionen `rotateLeft(Node v)` und `rotateRight(Node v)`, welche eine einfache Linksrotation beziehungsweise eine einfache Rechtsrotation um v ausführen. Dabei ist v der tiefste Knoten im AVL-Baum, welcher infolge der Einfügeoperation nicht balanciert ist.

Es genügt, die Kommentare `TODO` in der Vorlage durch Ihren Java-Code zu ersetzen. Wir empfehlen, den restlichen Code der Vorlage nicht zu verändern, obwohl das nicht verboten ist. Sorgen Sie lediglich dafür, dass die Zeiger `leftChild`, `rightChild` und `parent`, welche sich durch die Rotation ändern, korrekt aktualisiert werden. Unter Umständen muss auch der Zeiger `root` aktualisiert werden. Ihre Lösung soll *keine* Balance `balanceFactor` ändern, da diese bereits in der Vorlage aktualisiert wird. Beachten Sie, dass die Funktionen `rotateLeft(Node v)` und `rotateRight(Node v)` die Anzahl der ausgeführten Links- und Rechtsrotationen speichern. Die Anzahl der Rotationen werden von Ihrem Programm ausgegeben und werden vom Judge benötigt, um die Korrektheit Ihrer Lösung zu verifizieren. Ändern Sie diese Zähler nicht.

Die einzufügenden Zahlen sind eindeutige Ganzzahlen zwischen 0 und 1 000 000. Für die volle Punktzahl sollte eine Rotation in konstanter Zeit durchgeführt werden können.

Beispiel Das folgende Bild zeigt das Resultat einer Linksrotation in einem vorläufig nicht balancierten Baum. Beachten Sie, dass `v.parent` gleich `null` ist und dass der Zeiger `root` bei dieser

Rotation aktualisiert wird.



Bewertung Sie können bis zu 20 Punkte am Judge bekommen. Ihr Programm sollte jede Rotation in konstanter Zeit durchführen. Reichen Sie nur Ihr `Main.java` ein. Sie können bis zu 9 Punkte erhalten, wenn Sie nur die Linksrotation korrekt implementieren. Ebenso können Sie bis zu 9 Punkte erhalten, wenn Sie nur die Rechtsrotation korrekt implementieren.

Instruktionen Wir stellen für diese Aufgabe eine Programmvorlage im Eclipse-Workspace zur Verfügung; die Programmvorlage implementiert bereits den Grossteil der Funktionen. Ihre Aufgabe besteht darin, den Code der Funktionen `rotateLeft(Node v)` und `rotateRight(Node v)` zu vervollständigen.

Wie üblich enthält die Vorlage Testdaten, damit Sie lokal testen können, und enthält ein Programm `Judge.java`, das Ihr `Main.java` mit allen verfügbaren Testdaten testet – öffnen und starten sie dazu einfach `Judge.java` im Projekt. Die lokalen Testdaten unterscheiden sich von den Testdaten, welche der Online-Judge verwendet.

Die Ein- und Ausgabe werden von der Vorlage verarbeitet – Sie sollten den Rest dieses Texts nicht benötigen.

Eingabe Die erste Zeile der Eingabe enthält einzig die Anzahl der Tests.

Jeder Test besteht aus zwei Zeilen. Die erste Zeile enthält n, die Anzahl der Ganzzahlen, die in einen anfangs leeren AVL-Baum eingefügt werden sollen. Die zweite Zeile enthält n durch Leerzeichen getrennte Ganzzahlen, die eingefügt werden sollen.

Ausgabe Nachdem alle n Zahlen eingefügt worden sind, sollen zwei Ganzzahlen in einer Zeile, durch Leerzeichen getrennt, ausgegeben werden. Die erste Zahl entspricht die Anzahl der durchgeführten Linksrotationen und die zweite Zahl die Anzahl der durchgeführten Rechtsrotationen.

Beispiel-Eingabe (wenn 8 eingefügt wird, muss die oben dargestellte Rotation durchgeführt werden):

```
1
9
3 1 6 9 4 8 10 11 12
```

Beispiel-Ausgabe:

```
2 0
```

Platz für Ihre Notizen. Diese werden nicht bewertet. Nur was auf dem Judge eingereicht wird, zählt für diese Aufgabe.

/ 20 P

Programmieraufgabe P2.

Passwort für Einschreibung: `algorithmus`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

Dyno

Dyno, der dynamisch programmierende Dinosaurier aus Abbildung 1 will eine vollkommen ebene Wüste durchqueren. Die Wüste ist L Meter lang und in L Segmente mit Indizes von 0 bis $L-1$ aufgeteilt. Jedes Segment kann entweder *leer* sein oder einen von C *Kakteen*, welche in der Wüste wachsen, beinhalten. Dyno kann in einem Segment i entweder gehen oder um ein fixes, gegebenes D nach vorne springen. Er kann von Segment i zu Segment $i+1$ gehen, wenn Segment $i+1$ leer ist. Er kann von Segment i zu Segment $i+D$ springen, wenn Segment $i+D$ leer ist, auch wenn zwischen Segment i und Segment $i+D$ Kakteen sind. Dyno startet bei Segment 0, das immer leer ist. Ihre Aufgabe ist es, Dyno zu helfen, möglichst viele Segmente der Wüste zu überwinden, d.h. das Segment mit dem grösstmöglichen Index zu erreichen.

Für die Eingabe gilt $10 \leq L \leq 1\,000\,000$, $0 \leq C \leq 1\,000\,000$, and $2 \leq D \leq 10$. Beachten Sie, dass approximative (fast optimale) Lösungen und "Greedy-Algorithmen" *keine* Punkte erhalten werden – nur optimale Lösungen werden akzeptiert. Wir empfehlen Ihnen, ein eindimensionales dynamisches Programm zu entwickeln.

Beispiel Die folgende Abbildung zeigt ein Beispiel mit $L = 15$, $D = 4$ und $C = 6$. In einer optimalen Lösung erreicht Dyno das Segment 10 (indem er von 0 bis 1 geht, dann von 1 bis 5 und von 5 bis 9 springt und schliesslich von 9 bis 10 geht). Die dazugehörige Eingabe finden Sie weiter unten.

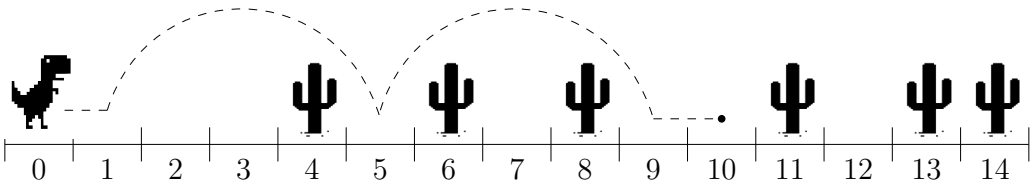


Abbildung 1: Beispiel. Die gestrichelte Linie zeigt die einzige optimale Lösung.[1]

[1] Dyno ist nicht im Massstab dargestellt.

Bewertung Sie können bis zu 20 Punkte am Judge bekommen. Für die volle Punktzahl sollte Ihr Programm eine Laufzeit in $\mathcal{O}(L \log C)$ haben. Langsamere Lösungen können weniger Punkte erhalten. Reichen Sie nur Ihr `Main.java` ein.

Instruktionen Wir stellen für diese Aufgabe eine Programmvorlage im Eclipse-Workspace zur Verfügung, die Ihnen hilft, die Eingabe zu lesen und die Ausgabe zu schreiben.

Wie üblich enthält die Vorlage Testdaten, damit Sie lokal testen können, und enthält ein Programm `Judge.java`, das Ihr `Main.java` mit allen verfügbaren Testdaten testet – öffnen und starten sie dazu einfach `Judge.java` im Projekt. Die lokalen Testdaten unterscheiden sich von den Testdaten, welche der Online-Judge verwendet, und sind im Allgemeinen kleiner.

Die Ein- und Ausgabe werden von der Vorlage verarbeitet – Sie sollten den Rest dieses Texts nicht benötigen.

Eingabe Die erste Zeile der Eingabe enthält einzig die Anzahl der Tests.

Die erste Zeile jedes Tests enthält die Ganzzahlen L, D und C, durch Leerzeichen getrennt. Die zweite Zeile jedes Tests enthält die Standorte der C Kakteen als n durch Leerzeichen getrennte Ganzzahlen. Die Standorte der Kakteen sind nach aufsteigend sortierter Segmentnummer gegeben.

Ausgabe Die Ausgabe besteht aus einer Zeile für jeden Test, welche den grössten Index eines für Dyno erreichbaren Segmentes enthält.

Beispieleingabe (für das Beispiel oben):

```
1
15 4 6
4 6 8 11 13 14
```

Beispielausgabe:

```
10
```

Platz für Ihre Notizen. Diese werden nicht bewertet. Nur was auf dem Judge eingereicht wird, zählt für diese Aufgabe.

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

/ 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.

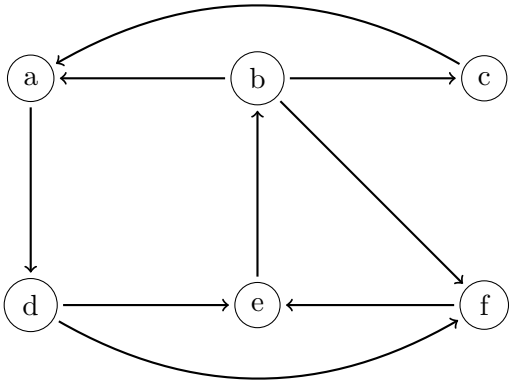
2	5	3	7	6	13	4	9	8
1	2	3	4	5	6	7	8	9

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			4		17			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.



$E' = \{$_____$\}$

topologische Sortierung für G':

_____

/ 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.

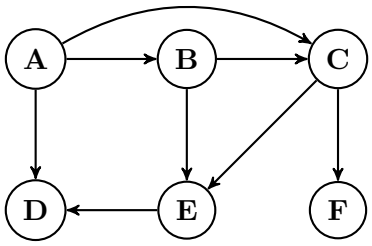
Adjazenzmatrix von G:

	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:

_____

Breitensuche:

_____

/ 1 P

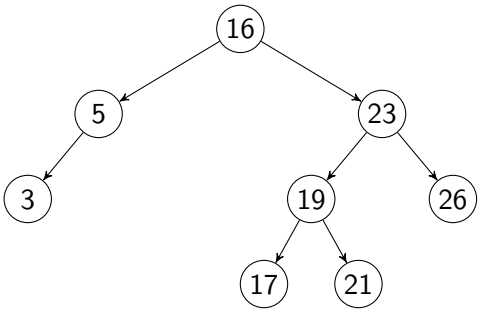
g) Wieviele Kanten kann ein gerichteter Graph mit n Knoten höchstens haben, wenn er eine topologische Sortierung besitzt? Geben Sie eine geschlossene Formel an. Sie müssen Ihre Antwort nicht begründen.

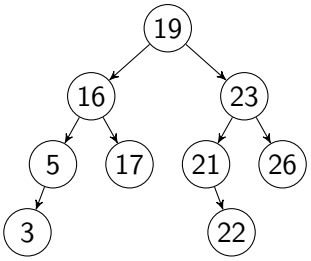
/ 1 P

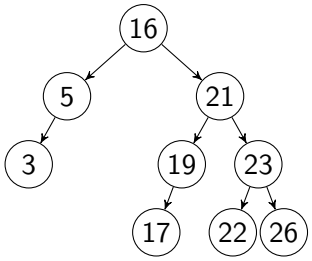
h) Zeichnen Sie den binären Suchbaum mit Schlüsselmenge $\{1, 2, \ldots, 8\}$, bei dem die Preorder-Reihenfolge der Schlüssel mit $5, 2, 1, 4, 3$ beginnt und die Postorder-Reihenfolge der Schlüssel mit $7, 8, 6, 5$ endet.

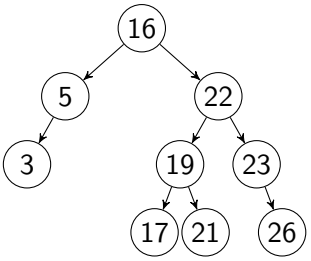
/ 1 P

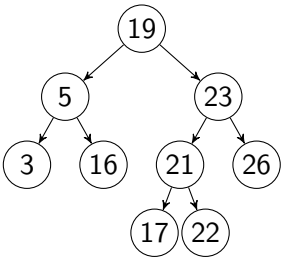
i) Welcher AVL-Baum entsteht, wenn im folgenden AVL-Baum der Schlüssel 22 eingefügt und danach die AVL-Bedingung wiederhergestellt wird?

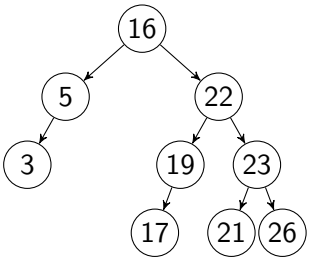


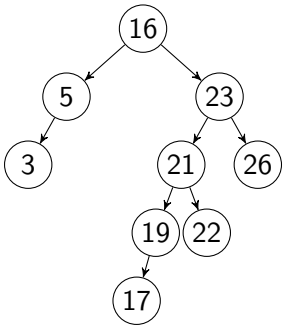
☐ 

☐ 

☐ 

☐ 

☐ 

☐ 

/ 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):

Geschlossene und vereinfachte Formel:

$T(n) = T(4^k) =$

Induktionsbeweis:

Induktionsbeweis (Fortsetzung):

/ 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:

/ 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:

/ 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.

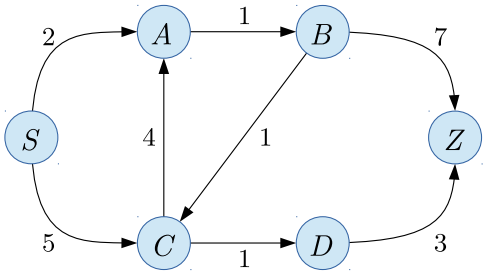
$$\log(\sqrt{n}),\ (\sqrt{n})^{\sqrt{n}},\ \sqrt{\log n},\ \log(10),\ n^2,\ n\sqrt{n},\ \frac{\sqrt{n}}{\log n}$$

Lösung: _____, _____, _____, _____, _____, _____, _____.

/ 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 $l_{ij} \geq 0$.



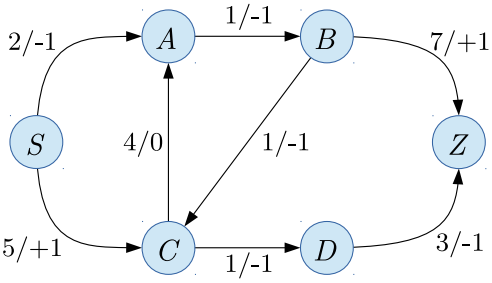
Modellieren Sie die nachfolgenden drei Fragestellungen jeweils als graphentheoretisches Problem. Geben Sie dazu an, welche Knoten und Kanten definiert werden und welche Gewichte den Kanten zugeordnet werden. Erläutern Sie in Teil b) und c), wie die Lösung der Fragestellung von einem geeigneten Weg abgelesen werden kann. Nennen Sie einen effizienten Algorithmus zur Lösung des graphentheoretischen Problems 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 auf dem Weg dorthin 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 wie zuvor 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):

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

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

- Laufzeit (in möglichst knapper Θ-Notation):

Teilaufgabe b)

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

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

- Ablesen der Lösung:

- Möglichst effizienter Algorithmus zur Berechnung der kürzesten Wege:

- Laufzeit (in möglichst knapper Θ-Notation):

Teilaufgabe c)

- Modellierung als graphentheoretisches Problem:

- Möglichst effizienter Algorithmus zur Wegberechnung:

- Laufzeit (in möglichst knapper Θ-Notation):

/ 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 $[\texttt{B}, \texttt{A}, \texttt{B}, \texttt{A}]$ kann in zwei Subarrays $[\texttt{B}]$ und $[\texttt{A}, \texttt{B}, \texttt{A}]$ 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 $[\texttt{B}, \texttt{A}, \texttt{B}]$ und $[\texttt{A}]$. Das Array $[\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}]$ kann in vier Subarrays $[\texttt{A}]$, $[\texttt{B}]$, $[\texttt{C}]$ und $[\texttt{D}]$ zerlegt werden und es gibt keine Zerlegung in weniger Subarrays. Das Array $[\texttt{A}, \texttt{B}, \texttt{A}]$ 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. Falls Sie mehrere DP-Tabellen verwenden, beschreiben Sie deren Abhängigkeit.

- ☐ **(I) eindimensional** → Benutzen Sie das Schema auf Seite 24.
- ☐ **(II) zweidimensional** → Benutzen Sie das Schema auf Seite 25.
- ☐ **(III) andere Dimension** → Beschreiben Sie Ihr dynamisches Programm auf Seite 26.

Beantworten Sie die Teilaufgabe b) auf Seite 27.

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

Grösse der DP-Tabelle / Anzahl Einträge:

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

Bedeutung eines Tabelleneintrags:

Bedeutung des Eintrags an Position i:

$DP[i]$: _____

_____

Initialisierung:

Berechnung eines Eintrags:

$DP[i] =$

Berechnungsreihenfolge:

Auslesen der Lösung:

(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:

$DP[i, j]$: _____

_____

Initialisierung:

Berechnung eines Eintrags:

$DP[i, j] =$

Berechnungsreihenfolge:

Auslesen der Lösung:

(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:

Extraplatz. Bitte kennzeichnen Sie deutlich zu welcher Aufgabe Ihre Aufschrift gehört. Notizen, die Sie nicht bewerten lassen möchten, bitte durchstreichen.