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, Übersetzungs-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 Oropax (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. Im Schleusenraum ist es **nicht** möglich, das WC zu benutzen. Es empfiehlt sich, als Prüfling des zweiten Durchgangs das WC vor dem Einlass in den Schleusenraum zu benutzen.

Programmierteil (Computerprüfung)

- **nethz Passwort:** Zur Anmeldung am Judge benötigen Sie Ihren nethz-Account. Stellen Sie sicher, dass Sie sich an Ihre Logindaten erinnern können.
- **Nicht ausschalten! Nicht abmelden! Bildschirm nicht sperren!** Der Computer darf auf keinen Fall ausgeschaltet werden, auch nicht gegen Ende der Prüfung. Es droht der Verlust aller Daten! Dasselbe gilt für das Abmelden und das Sperren des Bildschirms.
- **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.
- **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. Beachten Sie, dass dies auch die Java Collections verbietet. Einzige Ausnahmen sind die Module (z.B. `java.util.Scanner`), welche im gegebenen Template benutzt werden.

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

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: `algorismus`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

Erweiterung des AVL-Baumes

Ihre Aufgabe besteht darin, den AVL-Baum so zu erweitern, dass er folgende `rank(x)` Operation unterstützt:

`rank(`x`):` Gibt für eine gegebene Ganzzahl x die Anzahl derjenigen im AVL-Baum gespeicherten Schlüssel zurück, die kleiner sind als x oder gleich.

Der Grossteil der Implementierung eines AVL-Baumes ist bereits in der Vorlage vorhanden (Einlesen der Eingabe, Einfügen eines neuen Elementes, Ausgabe).

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

`parent`: Zeiger zum Elternknoten von v im Baum (oder `null` falls v die Wurzel des Baumes ist).

`leftChild`: Zeiger zum linken Kindknoten von v (`null` falls kein solcher Kindknoten existiert).

`rightChild`: Zeiger zum rechten Kindknoten von v (`null` falls kein solcher Kindknoten existiert).

`value`: der ganzzahlige Wert von v.

`balanceFactor`: die Balance 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).

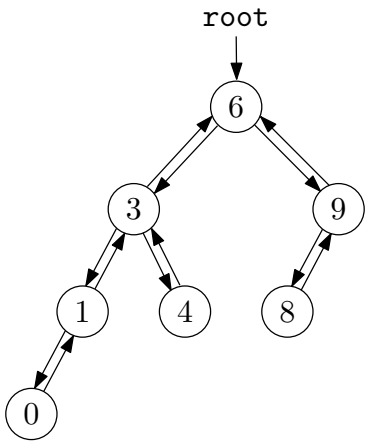
Beachten Sie, dass es zu jedem Zeiger `leftChild`, `rightChild` oder `parent`, der von einem Knoten v auf einen anderen Knoten u zeigt, einen entsprechenden Zeiger von u auf v gibt. Die AVL-Baum-Implementierung in der Vorlage enthält ausserdem einen Zeiger `root`, der auf den aktuellen Wurzelknoten des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

Um diese Aufgabe zu lösen, müssen sie die `rank(`x`)` Operation implementieren. Ausserdem müssen Sie den gegebenen Java-Code so verändern (z.B. zusätzliche Felder hinzufügen und andere bereits existierende Funktionen ergänzen), dass die `rank(`x`)` Operation in $O(\log n)$ Zeit unterstützt wird, wobei n die Anzahl Knoten im AVL-Baum bezeichnet. Die asymptotische Laufzeit der Einfügeoperation darf sich dabei nicht ändern.

Die einzufügenden Zahlen sind paarweise verschiedene Ganzzahlen zwischen 0 und 1 000 000.

(Weiter geht's auf der nächsten Seite)

Beispiel Das folgende Bild zeigt die Struktur eines AVL-Baumes mit `rank`(5)=4 und `rank`(8)=6.



Anforderung Sie können bis zu 20 Punkte am Judge bekommen, wenn Ihr Programm jede Einfügeoperation und jede `rank(`x`)` Operation in $O(\log n)$ Zeit durchführt (mit vernünftigen versteckten Konstanten), wobei n die Anzahl Knoten im AVL-Baum ist. Für weniger effiziente Lösungen können bis zu 10 Punkte bekommen.

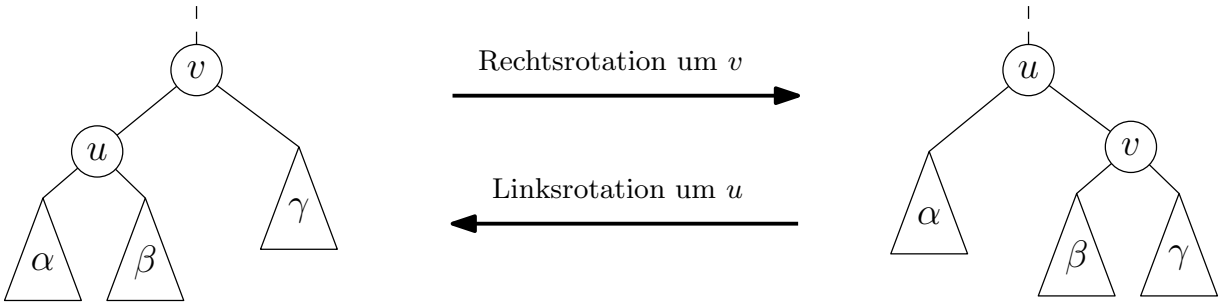
Instruktionen Wir stellen für diese Aufgabe eine Programmvorlage im Eclipse-Workspace zur Verfügung; die Programmvorlage implementiert bereits den Grossteil der Funktionen, ausser den Änderungen die benötigt werden um die `rank(`x`)` Operation zu unterstützen.

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.

Laden Sie ausschliesslich Ihr `Main.java` auf den Judge.

Bemerkungen Beachten Sie, dass Ihre Lösung auch dann noch funktionieren muss, wenn Rotationen auftreten. Die Rotationen sind bereits im Template-Code vorhanden, müssen aber gegebenenfalls verändert werden.

Das folgende Bild zeigt eine generische Rechtsrotation (bzw. Linksrotation) um den Knoten v (bzw. u).



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 Anzahl m der Operationen, die ausgeführt werden sollen. Die nächsten m Zeilen enthalten je einen Buchstaben C und eine Ganzzahl x, getrennt durch ein Leerzeichen. Jeder Buchstabe ist entweder "I" oder "R". Wenn C gleich "I" ist, dann soll x in den AVL-Baum eingefügt werden. Wenn C gleich "R" ist, dann soll eine `rank(`x`)` Operation ausgeführt werden.

Ausgabe Die Ausgabe enthält eine Zeile pro `rank(`x`)` Operation. Die i-te Zeile der Ausgabe enthält eine einzige Ganzzahl, die das Resultat der i-ten `rank(`x`)` Operation ist.

Beispiel-Eingabe:

```
1
10
I 6
I 3
I 9
R 10
I 4
I 8
I 1
I 0
R 5
R 8
```

Beispiel-Ausgabe:

```
3
4
6
```

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: `algorismus`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

Kunstgalerie

Eine Diebesbande raubt eine Kunstgalerie aus. Die Diebe besitzen einen Lastwagen mit Volumen V cm^3 und möchten ihn mit wertvollen Skulpturen füllen. Die Gallerie besitzt n Skulpturen und bewacht jede einzelne mit einem eigenen Alarmsystem. Die Diebe benötigen für die i-te Skulptur $t_i \geq 0$ Minuten um ihr Alarmsystem auszuschalten. Die i-te Skulptur bringt auf dem Schwarzmarkt $p_i > 0$ Franken ein und benötigt $v_i > 0$ cm^3 Platz im Lastwagen um sie zu transportiern.

Die Diebe haben nur T Minuten Zeit bis zum Eintreffen der Polizei. Entwickeln Sie einen Algorithmus, der die maximale Menge an Geld M berechnet, das die Diebesbande aus diesem Raubüberfall auf dem Schwarzmarkt erzielen kann.

Ihr Programm soll also folgendes berechnen:

$$M = \max_{X \subseteq \mathcal{I}} \sum_{i \in X} p_i,$$

wobei $\mathcal{I} = \{X \subseteq \{1, \ldots, n\} \ : \ \sum_{i \in X} v_i \leq V \text{ und } \sum_{i \in X} t_i \leq T\}$ alle Mengen von Skulpturen enthält, deren Gesamtvolumen höchstens V ist und die in insgesamt höchstens T Minuten gestohlen werden können.

Anforderung Sie können bis zu 20 Punkte am Judge bekommen, wenn Ihr Programm die Aufgabe vollständig löst in Zeit $\mathcal{O}(n \cdot V \cdot T)$ (mit vernünftigen versteckten Konstanten).

Mit Lösungen die folgende Spezialfälle lösen, ist es möglich Teilpunkte zu bekommen:

Spezialfall 1 Sie können bis zu 5 Punkte bekommen (von total 20 möglichen Punkten), wenn Ihr Programm Instanzen mit $n \leq 20$ korrekt löst.

Spezialfall 2 Sie können bis zu 10 Punkte bekommen (von total 20 möglichen Punkten), wenn Ihr Programm die Zeitkomponente ignoriert, d.h. Instanzen mit $T = 1$ und $t_i = 0$ für alle $i = 1, \ldots, n$ korrekt löst.

Instruktionen Wir stellen Ihnen für diese Aufgabe eine Programmvorlage im Eclipse-Workspace zur Verfügung, das Sie beim Einlesen der Eingabe und dem ausgeben der Ausgabe unterstützt.

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.

Laden Sie ausschliesslich Ihr `Main.java` auf den Judge.

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 n, V und T, getrennt durch Leerzeichen. Die nächsten n Zeilen beschreiben jeweils eine Skulptur: die i-te dieser Zeilen enthält die drei Ganzzahlen v_i, t_i und p_i.

Ausgabe Die Ausgabe enthält eine separate Zeile für jeden Test. Die i-te Zeile enthält eine Ganzzahl für die maximale Geldsumme M die die Diebe im i-ten Test erzielen können.

Beispiel-Eingabe:

```
1
6 12 20
10 19 100
1 2 30
7 11 55
4 1 16
2 9 20
3 7 43
```

Beispiel-Ausgabe (in diesem Beispiel wird die zweite, die dritte und die sechste Skulptur gestohlen):

```
128
```

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

/ 16 P

Theorieaufgabe T1.

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 eine Menge von Frauen $\mathcal{F} = \{f_1, f_2, f_3, f_4\}$ sowie eine Menge von Männern $\mathcal{M} = \{m_1, m_2, m_3, m_4\}$. Die folgenden beiden Tabellen zeigen jeweils ihre Präferenzen, die von links nach rechts absteigend sortiert sind. Führen Sie den Algorithmus von Gale und Shapley aus, und nehmen Sie an, dass die Frauen die Heiratsanträge machen.

f_1	m_2	m_1	m_3	m_4
f_2	m_1	m_4	m_2	m_3
f_3	m_4	m_2	m_1	m_3
f_4	m_2	m_3	m_4	m_1

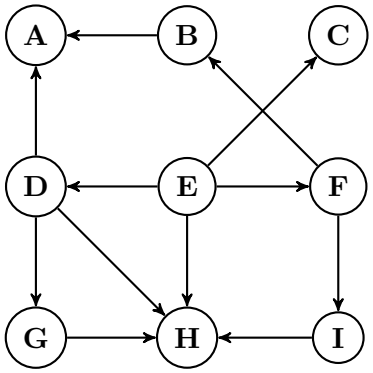
m_1	f_3	f_4	f_2	f_1
m_2	f_3	f_2	f_1	f_4
m_3	f_1	f_2	f_3	f_4
m_4	f_2	f_3	f_1	f_4

Zuordnung:

f_1		f_2		f_3		f_4	

/ 1 P

b) Führen Sie im folgenden Graph eine Breitensuche und eine Tiefensuche aus. Starten Sie dafür in Knoten E und geben sie die Reihenfolgen an, in denen die Knoten besucht werden. Nehmen Sie an, dass die beiden Traversierungen die jeweiligen Nachbarknoten in alphabetischer Reihenfolge abarbeiten.



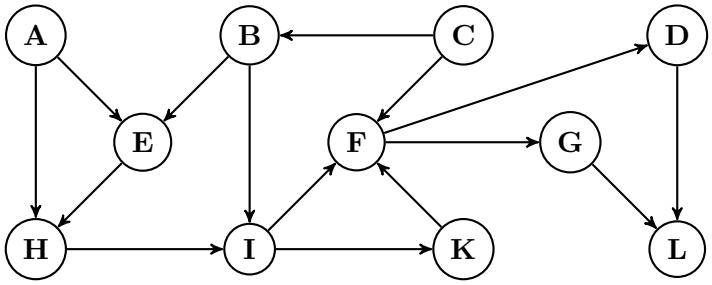
Breitensuche:

____, ____, ____, ____, ____, ____, ____, ____, ____.

Tiefensuche:

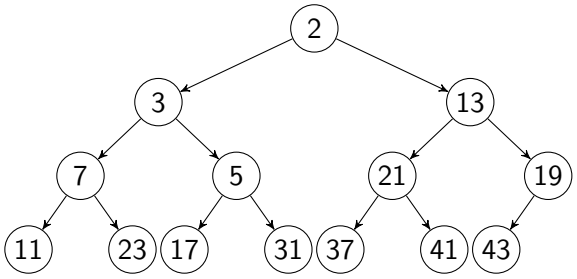
____, ____, ____, ____, ____, ____, ____, ____, ____.

/ 1 P c) Wie viele verschiedene topologische Sortierungen hat der untenstehende Graph?

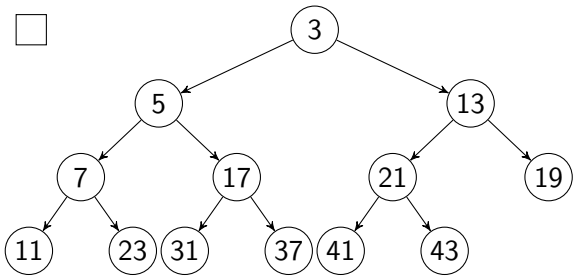


Anzahl topologische Sortierungen:

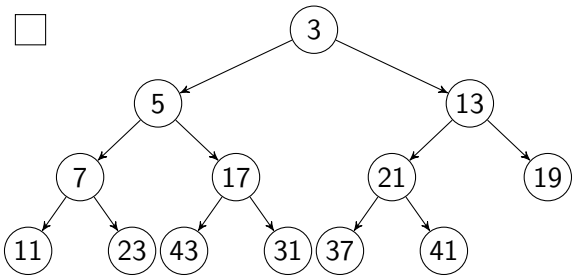
/ 1 P d) Führen Sie eine Extract-Min Operation einschliesslich Wiederherstellung der Heap-Bedingungen auf folgendem Min-Heap aus. Wie sieht der Heap nach der Operation aus? Kreuzen Sie die richtige Antwort an. Sie müssen Ihre Antwort nicht begründen.



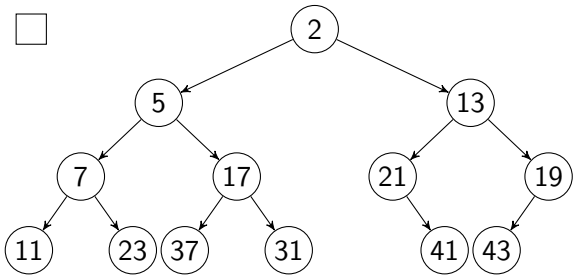
☐



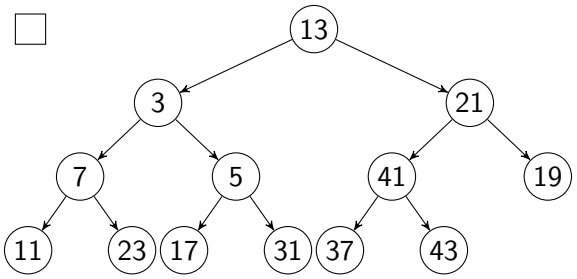
☐



☐



☐



/ 1 P

e) Führen Sie auf dem folgenden Array zwei Iterationen des Sortieralgorithmus *Natürliches 2-Wege-Mergesort* aus.

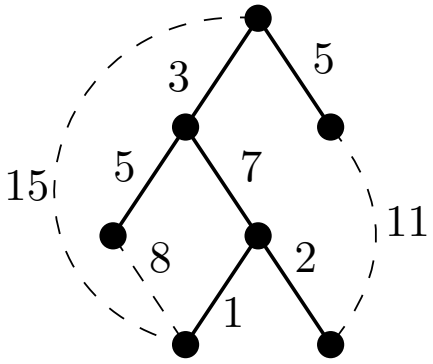
9	32	19	23	47	27	72	51	64	14	2	6
1	2	3	4	5	6	7	8	9	10	11	12

1	2	3	4	5	6	7	8	9	10	11	12

1	2	3	4	5	6	7	8	9	10	11	12

/ 2 P

f) Betrachten Sie jeweils den untenstehenden Graph, in dem ein Minimaler Spannbaum bereits eingezeichnet ist.



i) Was ist der höchste Wert auf den die Kante mit Gewicht 3 erhöht werden kann, so dass sie nicht mehr in jedem Minimalen Spannbaum dieses Graphen vorkommt?

ii) Was ist der tiefte Wert auf den die Kante mit Gewicht 11 geändert werden kann, so dass sie in mindestens einem Minimalen Spannbaum dieses Graphen vorkommt?

/ 1 P

g) Finden Sie heraus, ob folgende Postorder und Preorder zum gleichen natürlichen Suchbaum gehören.

Postorder: 4, 3, 8, 17, 12, 10, 21, 27, 23, 20, 7

Preorder: 7, 3, 4, 20, 8, 10, 12, 17, 23, 21, 27

Kreuzen Sie die richtige Antwort an. Sie müssen Ihre Antwort nicht begründen.

☐ Ja, gleicher Baum

☐ Nein, verschiedene Bäume

/ 2 P

h) Kreuzen Sie an, ob die folgenden Aussagen 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.

Für jeden AVL-Baum gibt es eine Einfügereihenfolge seiner Schlüssel, die zu diesem Baum führt, *ohne* dass eine Rotation stattfindet.	☐ Wahr	☐ Falsch
Es gibt einen AVL-Baum, bei dem *alle* inneren Knoten nicht perfekt balanciert sind (d.h. einen Balancierungsfaktor ungleich Null haben).	☐ Wahr	☐ Falsch
Kein natürlicher Suchbaum der Höhe h hat mehr als 2^{h-1} Knoten. (Die Höhe eines Suchbaumes ist die Anzahl Knoten auf dem längstem Weg von der Wurzel zu einem Blatt.)	☐ Wahr	☐ Falsch
Sei $T(n)$ die Anzahl verschiedener natürlicher Suchbäume, die genau die Schlüssel $\{1, 2, 3, \ldots, n\}$ verwalten. Dann gilt $T(0) = 1$, $T(1) = 1$ und $T(n) = \sum_{k=1}^{n}(T(k-1) \cdot T(n-k))$ für $n \geq 2$.	☐ Wahr	☐ Falsch

/ 1 P

i) Geben Sie für das folgende Codefragment die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ in Θ-Notation an. Halten Sie sich so knapp wie möglich. Die Funktion f läuft in $\Theta(1)$ Zeit. Sie müssen Ihre Antwort nicht begründen.

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

Asymptotische Laufzeit in möglichst knapper Θ-Notation:

/ 1 P

j) Geben Sie für das folgende Codefragment die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ in Θ-Notation an. Halten Sie sich so knapp wie möglich. Die Funktion f läuft in $\Theta(1)$ Zeit. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = 1; i*i ≤ n; i = i+1 ) {
2         for(int j = n; j ≥ 1; j = j/4 ) {
3                 for(int k = 1; k ≤ j; k = k+1 ) {
4                         f();
5                 }
6         }
7 }
```

Asymptotische Laufzeit in möglichst knapper Θ-Notation:

/ 1 P

k) 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 $f \leq \mathcal{O}(g)$.

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

$$n^{\frac{8}{7}},\ 24^n,\ \frac{\sqrt{n}}{\log^2 n},\ \sqrt{n!},\ \log^{100} n,\ n^{\frac{7}{8}},\ \log n!,\ 5^{2n}$$

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

/ 2 P

l) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} n + 2 \cdot T(n/5) & n > 1 \\ 1 & n = 1 \end{cases}$$

Sie können annehmen, dass n eine Potenz von 5 ist. Benutzen Sie also $n = 5^k$ oder $k = \log_5(n)$. Zeigen Sie mit vollständiger Induktion, dass

$$T(n) = T(5^k) = \frac{5^{k+1} - 2^{k+1}}{3}$$

die dazugehörige geschlossene Form ist.

Induktionsbeweis:

/ 15 P

Theorieaufgabe T2.

Zürich führt ab Sommer 2047 ein Road Pricing ein. Dabei wird jeder Strasse eine Gebühr zugeordnet. Es ist bekannt, dass jeweils am Morgen alle Autos von s nach t fahren möchten. Mit dem neuen Pricing werden selbstverständlich alle Autos entlang eines günstigsten Wegs fahren. Um sich für das neue Verkehrsaufkommen zu wappnen, möchte die Stadt Zürich herausfinden, welche Kreuzungen überhaupt befahren werden für ein gegebenes Pricing.

Das Strassennetz von Zürich ist gegeben durch n Kreuzungen und m Strassen. Jede Strasse e ist gegeben als ein Tupel (u, v, g_{uv}), definiert durch die beiden Kreuzungen u und v welche e verbindet, und durch die Gebühr von g_{uv} Franken welche *bei jeder Benützung* erhoben wird ($g_{uv} \in \mathbb{N}$).

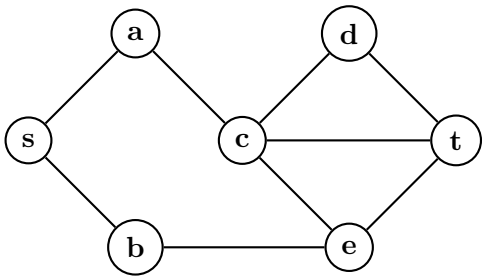
Modellieren Sie die nachfolgenden drei Fragestellungen jeweils als graphentheoretisches Problem. Geben Sie dazu an, welche Knoten und Kanten definiert werden und welche Gewichte den Kanten zugeordnet werden. Nennen Sie einen möglichst effizienten Algorithmus zur Lösung des jeweiligen graphentheoretischen Problems und bestimmen Sie seine Laufzeit in Abhängigkeit von n und m.

Sie dürfen jeweils davon ausgehen, dass mindestens ein Weg *von s nach t* existiert (für alle anderen Paare von Kreuzungen u, v gibt es hingegen nicht notwendigerweise einen Weg von u nach v).

/ 5 P a) Nehmen Sie in dieser Teilaufgabe an, dass jede Strasse, gegeben durch $e = (u, v, g_{uv})$, in *beide Richtungen* befahren werden darf. Nehmen Sie ausserdem an, dass die Gebühr für jede Strasse gleich 1 ist.

Für *jede* Kreuzung soll bestimmt werden, ob sie auf einem *günstigsten* Weg von s nach t liegt, oder nicht.

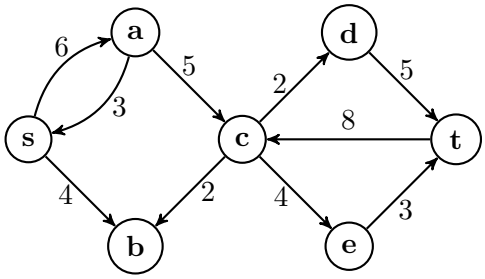
Beispiel. JA: s, a, b, c, e, t NEIN: d



/ 5 P b) Nehmen Sie in dieser Teilaufgabe an, dass jede Strasse, gegeben durch $e = (u, v, g_{uv})$, nur *von u nach v* befahren werden darf.

Für *jede* Kreuzung soll bestimmt werden, ob sie auf einem *günstigsten* Weg von s nach t liegt, oder nicht.

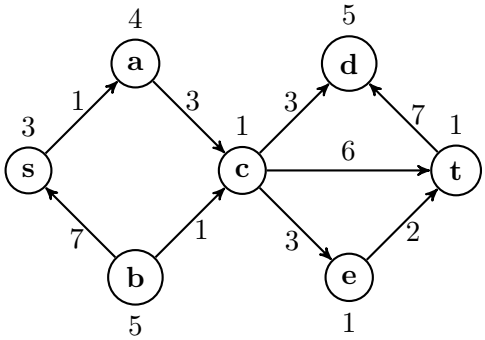
Beispiel. JA: s, a, c, d, e, t NEIN: b



/ 5 P c) Zürich beschliesst, zusätzlich zu den Strassen auch das Befahren einer Kreuzung gebührenpflichtig zu machen. Für jede Benützung einer Kreuzung v werden b_v Franken verrechnet. Nehmen Sie an, dass jede Strasse, gegeben durch $e = (u, v, g_{uv})$, nur *von u nach v* befahren werden darf.

Für *jede* Kreuzung soll bestimmt werden, ob sie auf einem *günstigsten* Weg von s nach t liegt, oder nicht.

Beispiel. JA: s, a, c, e, t NEIN: b, d



Teilaufgabe a)

- Graphentheoretische Modellierung inklusive Kantengewichte (in Worten):

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

- Möglichst effizienter Algorithmus zur Klassifikation der n Kreuzungen:

- Laufzeit in Abhängigkeit von n, m (in möglichst knapper Θ-Notation) inklusive Begründung:

Teilaufgabe b)

- Graphentheoretische Modellierung inklusive Kantengewichte (in Worten):

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

- Möglichst effizienter Algorithmus zur Klassifikation der n Kreuzungen:

- Laufzeit in Abhängigkeit von n, m (in möglichst knapper Θ-Notation) inklusive Begründung:

Teilaufgabe c)

- Graphentheoretische Modellierung inklusive Kantengewichte (in Worten):

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

- Möglichst effizienter Algorithmus zur Klassifikation der n Kreuzungen:

- Laufzeit in Abhängigkeit von n, m (in möglichst knapper Θ-Notation) inklusive Begründung:

Theorieaufgabe T3. **/ 10 P**

In einem weit entfernten Alpenland möchte das Militär seine veraltete Kommunikationsstruktur austauschen. Um das Parlament von diesem teuren Vorhaben zu überzeugen, möchte die Führung einige metrische Angaben über das aktuelle Netzwerk vorlegen. Leider fehlen zur Berechnung dieser Angaben die nötigen IT-Spezialisten. Deshalb werden Sie als Berater angeheuert.

Das Netzwerk ist gegeben als *gerichteter, gewichteter Graph* $G = (V, E, w)$, wobei $w\colon E \to \mathbb{R}$ die Kantengewichtsfunktion ist. Das Netzwerk hat $n = |V|$ Knoten und $m = |E|$ gerichtete Kanten. Für beliebige Knoten $u \neq v$ bezeichnet $d(u, v)$ die Länge eines kürzesten Pfads *von u nach v*. Gesucht sind möglichst effiziente Algorithmen zur Berechnung des *Minimums* über alle Knotenpaare, $\min_{u \neq v} d(u, v)$, und des *Maximums* über alle Knotenpaare, $\max_{u \neq v} d(u, v)$.

Sie erhalten das Netzwerk G als *Adjazenzliste*.

/ 3 P a) Zusätzlich zum Netzwerk erhalten Sie zwei Hinweise: (i) es gibt $m = \Theta(n^{1.5})$ Kanten und (ii) alle Kanten $(u, v) \in E$ haben nicht-negatives Gewicht $w(u, v) \geq 0$.

Beschreiben Sie einen möglichst effizienten Algorithmus, der den Wert $\min\limits_{u \neq v} d(u, v)$ berechnet. Bestimmen Sie die Laufzeit in Abhängigkeit von n.

/ 3 P b) Sie erhalten die selben Hinweise wie in Teilaufgabe a): (i) es gibt $m = \Theta(n^{1.5})$ Kanten und (ii) alle Kanten $(u, v) \in E$ haben nicht-negatives Gewicht $w(u, v) \geq 0$.

Beschreiben Sie einen möglichst effizienten Algorithmus, der den Wert $\max\limits_{u \neq v} d(u, v)$ berechnet. Bestimmen Sie die Laufzeit in Abhängigkeit von n.

weiter auf nächster Seite!

Die Skeptische Partei bezweifelt die präsentierten Ergebnisse. Deren Parlamentsfraktion vermutet, dass das Netzwerk auch negative Kantengewichte enthält.

/ 4 P

c) Nun erhalten Sie also *nur noch* den ersten Hinweis: (i) $m = \Theta(n^{1.5})$.

Beschreiben Sie einen möglichst effizienten Algorithmus, welcher
– abbricht, falls das Netzwerk einen Kreis mit negativem Gewicht enthält,
– andernfalls die Werte $\min_{u \neq v} d(u, v)$ und $\max_{u \neq v} d(u, v)$ berechnet.

Bestimmen Sie die Laufzeit Ihres neuen Algorithmus in Abhängigkeit von n.

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