Instruktionen zur Prüfung Algorithmen & Datenstrukturen

Ablauf

- **Dauer und Gewichtung:** Die Dauer der Prüfung beträgt 4 Stunden. In dieser Zeit sollen die Theorie- und die Programmieraufgaben gelöst werden. Beide Teile sind auf je 2 Stunden Dauer ausgelegt und zählen gleich viel (jeweils 40 Punkte). 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 Eingang des Prüfungsraumes deponiert werden. Es dürfen nur erlaubte Hilfsmittel plus Essen und Getränke un vernünftiger Ausprägung mit an den Platz genommen werden. Beachten Sie, dass wir Ihr Gepäck nicht beaufsichtigen können, Sie deponieren es also auf eigenes Risiko. 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.

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. Beachten Sie, dass der Judge lediglich die Ausgabe Ihres Programmes bewertet. Wir empfehlen deshalb, keine zusätzlichen Zeilen (etwa mit `System.out.println()`) auszugeben.
- **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 Verwendung von Java Collections verbietet. Einzige Ausnahmen sind die Module (z.B. `java.util.Scanner`), welche im gegebenen Template benutzt werden.

  Beachten Sie, dass diese Restriktion nicht beim Einsenden automatisiert durch den Judge geprüft, sondern *erst nach* Prüfungsende überprüft und gegebenenfalls sanktioniert 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: `flipflop`

Einreichung: siehe Abschnitt 3 der technischen Anleitung

ALGO-Turm

Gegeben sei eine Kiste mit $n \leq 5000$ *ALGO-Steinen*, welche von 1 bis n durchnummeriert sind (siehe untenstehendes Beispiel). Der i-te ALGO-Stein ist ein Quader mit Grundfläche $\ell_i \times b_i$ und Höhe h_i, mit $\ell_i, b_i \in \{1, 2, \ldots, 10000\}$ und $h_i \in \{0, 1, ..., 1000\}$.

Aus diesen Steinen soll nun ein *möglichst hoher* Turm gebaut werden. Dazu dürfen die vorhandenen Steine aufeinandergestapelt werden, solange dabei *keine Überhänge* entstehen. Formaler: Der i-te Stein darf genau dann auf den j-ten Stein aufgesetzt werden, wenn mindestens eine der folgenden zwei Bedingungen gilt:

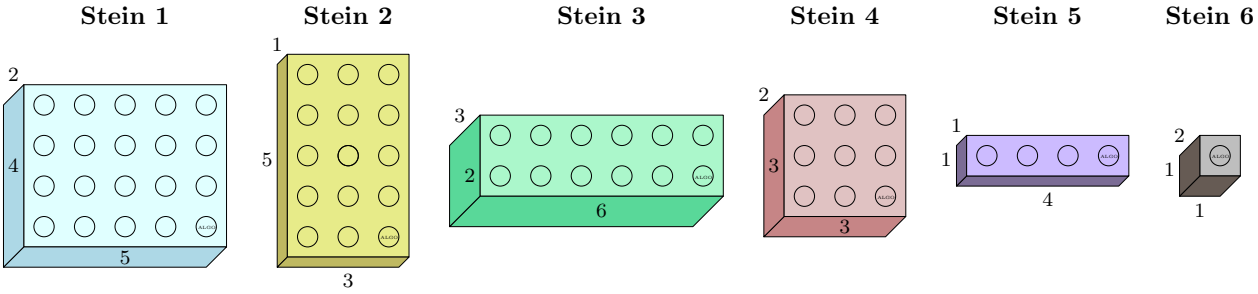
- $\ell_j \geq \ell_i$ und $b_j \geq b_i$; oder
- $\ell_j \geq b_i$ und $b_j \geq \ell_i$.

Intuitiv gesprochen entspricht die zweite Bedingung einer Drehung des i-ten Steins um 90° Grad.

Ein *ALGO-Turm* ist eine Liste $\langle t_1, t_2, ..., t_k \rangle$ von paarweise unterschiedlichen Indizes, sodass für alle $1 \leq i < k$ der Stein t_{i+1} auf den Stein t_i aufgesetzt werden darf. Die Höhe H eines solchen Turms ergibt sich als Summe der Höhen aller verwendeten Steine, das heisst $H := \sum_{i=1}^{k} h_{t_i}$.

Ihre Aufgabe ist es nun, die Höhe eines höchstmöglichen ALGO-Turms zu berechnen, welcher mit den vorhandenen ALGO-Steinen gebaut werden kann. Dabei dürfen Sie annehmen, dass alle Grundflächen paarweise unterschiedlich sind, und dass die Steine nach absteigender Grundfläche sortiert sind. In anderen Worten gilt für $1 \leq i < n$, dass $\ell_i \cdot b_i > \ell_{i+1} \cdot b_{i+1}$.

Beispiel



Die oben abgebildeten $n = 6$ ALGO-Steine haben die folgenden Dimensionen:

Stein 1 $\ell_1 = 5$, $b_1 = 4$, $h_1 = 2$;

Stein 2 $\ell_2 = 3$, $b_2 = 5$, $h_2 = 1$;

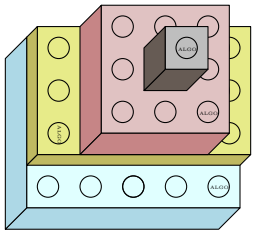
Stein 3 $\ell_3 = 6$, $b_3 = 2$, $h_3 = 3$;

Stein 4 $\ell_4 = 3$, $b_4 = 3$, $h_4 = 2$;

Stein 5 $\ell_5 = 4$, $b_5 = 1$, $h_5 = 1$;

Stein 6 $\ell_6 = 1$, $b_6 = 1$, $h_6 = 2$;

Der höchste ALGO-Turm, welcher mit diesen Steinen gebaut werden kann, ist demzufolge $\langle 1, 2, 4, 6 \rangle$, siehe auch die untenstehende Abbildung. Er hat Höhe $h_1 + h_2 + h_4 + h_6 = 2 + 1 + 2 + 2 = 7$.



Hierbei musste Stein 2 (in gelb) *gedreht* werden, um ihn auf Stein 1 (in blau) aufsetzen zu können.

Anforderung

Insgesamt gibt es für diese Aufgabe maximal 20 Punkte auf dem Judge. Als Richtlinie gilt, dass eine korrekte Lösung mit einer Laufzeit in $O(n^2)$ die volle Punktzahl erreichen kann. Ansonsten können Sie Teilpunkte erhalten für das Lösen der folgenden Teilaufgaben:

Teilaufgabe 1: Es gibt bis zu 8 Punkte für das korrekte Lösen von Instanzen mit $n \leq 200$, in welchen die höchstmögliche Höhe auch erreicht werden kann, indem nur ungedrehte Steine verwendet werden (in Bezug auf ihre Ausrichtung gemäss Eingabe).

Teilaufgabe 2: Es gibt bis zu 8 (weitere) Punkte für das korrekte Lösen von Instanzen mit $n \leq 200$. In dieser Teilaufgabe kann es für das Erreichen der höchstmöglichen Höhe nötig sein, Steine zu verwenden, welche dafür zuvor gedreht werden müssen.

Greedy-Ansätze führen – wie im zweiten Test der Beispiel-Eingabe nachvollziehbar – im Allgemeinen nicht zu einer korrekten Lösung.

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 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.

(Fortsetzung auf nächster Seite)

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 4 Zeilen: Die erste Zeile enthält die Anzahl ALGO-Steine n;
die zweite Zeile enthält als Ganzzahlen die Längen der Steine $\ell_1, \ell_2, \ldots, \ell_n$;
die dritte Zeile enthält als Ganzzahlen die Breiten der Steine $b_1, b_2, \ldots, b_n$;
die vierte Zeile enthält als Ganzzahlen die Höhen der Steine $h_1, h_2, \ldots, h_n$.

Ganzzahlen auf der gleichen Zeile sind jeweils durch Leerzeichen getrennt.

Ausgabe Geben Sie für jeden Test eine einzelne Zeile aus, welche die Höhe eines höchstmöglichen ALGO-Turms enthält, welcher mit den verfügbaren ALGO-Steinen gebaut werden kann.

Beispiel-Eingabe (der erste Test entspricht dem in der Aufgabenstellung abgebildeten Beispiel):

```
2
6
5 3 6 3 4 1
4 5 2 3 1 1
2 1 3 2 1 2
3
10 13 11
4 3 2
8 4 6
```

Beispiel-Ausgabe:

```
7
10
```

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: `flipflop`

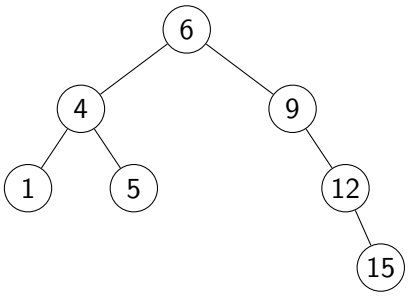
Einreichung: siehe Abschnitt 3 der technischen Anleitung

Traversierungen

In dieser Aufgabe geht es um einen binären Suchbaum T, in dem paarweise verschiedene Schlüssel aus $\{1, \ldots, 10000\}$ gespeichert werden. Gegeben sind die Anzahl Knoten $n \leq 10000$ und eine Liste aller gespeicherten Schlüssel in der Reihenfolge ihrer *Preorder = Hauptreihenfolge* (Wurzel, linker Teilbaum, rechter Teilbaum) in T. Ihre Aufgabe ist es nun, die gespeicherten Schlüssel in derjenigen Reihenfolge auszugeben, in der eine Breitensuche auf T die zugehörigen Knoten besucht – also in ihrer *Breitenordnung*.

Wie die Preorder soll auch die Breitensuche der Regel "links vor rechts" folgen: Falls u ein Knoten mit linkem Kind v und rechtem Kind w ist, dann wird in beiden Fällen v vor w besucht.

Beispiel



Betrachten Sie den obenstehenden binären Suchbaum. Eine Preorder-Traversierung besucht die Knoten der gespeicherten Schlüssel in der Reihenfolge: 6, 4, 1, 5, 9, 12, 15. Eine Breitensuche hingegen besucht die gespeicherten Schlüssel in der Reihenfolge: 6, 4, 9, 1, 5, 12, 15.

Anforderung Insgesamt gibt es für diese Aufgabe maximal 20 Punkte auf dem Judge. Als Richtlinie gilt, dass eine korrekte Lösung mit einer Laufzeit in $O(n^2)$ die volle Punktzahl erreichen kann.

Ansonsten können Sie Teilpunkte erhalten für das Lösen der folgenden Teilaufgaben:

Teilaufgabe 1: Es gibt bis zu 8 Punkte für das korrekte Lösen von Instanzen, in denen T ein *vollständiger* Binärbaum der Höhe ≤ 10 ist.

Teilaufgabe 2: Es gibt bis zu 8 (weitere) Punkte für das korrekte Lösen von Instanzen, in denen T eine Höhe ≤ 10 hat, aber nicht unbedingt vollständig ist.

Eine mögliche Lösung der gesamten Aufgabe besteht aus zwei Schritten: Der erste Schritt rekonstruiert den binären Suchbaum T aus der Preorder mithilfe der zur Verfügung gestellten `Vertex` Klasse (siehe Instruktionen weiter unten). Der zweite Schritt führt eine Breitensuche auf dem rekonstruierten Suchbaum T aus. *Anmerkung:* Dieser Ansatz bedingt keiner Kenntnis der Höhe des Baumes. Für Ihre Implemetierung dürfen Sie aber annehmen, dass die Höhe des Baumes ≤ 500 ist.

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.

Zusätzlich enthält die Vorlage eine `Vertex` Klasse, die Sie benutzen können, um Ihre Lösung zu implementieren. Grundsätzlich sind keine Änderungen an der `Vertex` Klasse erforderlich, um die Aufgabe korrekt zu lösen. Es steht Ihnen aber frei, die Klasse zu verändern, erweitern, löschen oder gänzlich zu ignorieren.

Wie üblich enthält die Vorlage Testdaten, damit Sie lokal testen können, und 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.

Jeder Test besteht aus zwei Zeilen. Die erste Zeile enthält eine Ganzzahl n, für die Anzahl Knoten in T. Die zweite Zeile besteht aus n Ganzzahlen für die gespeicherten Schlüssel in T in der Reihenfolge ihrer Preorder. Ganzzahlen auf der gleichen Zeile sind jeweils durch Leerzeichen getrennt.

Ausgabe Geben Sie für jeden Test eine einzelne Zeile aus, die alle gespeicherten Schlüssel in T in Breitenordnung enthält. Die Schlüssel müssen jeweils durch ein Leerzeichen getrennt sein.

Beispiel-Eingabe (entsprechend dem Beispiel weiter oben):

```
1
7
6 4 1 5 9 12 15
```

Beispiel-Ausgabe:

```
6 4 9 1 5 12 15
```

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.

/ 3 P

a) Nachfolgend sehen Sie drei Folgen gewisser Momentaufnahmen von jeweils einem der vier Algorithmen (a) `InsertionSort` (Sortieren durch Einfügen), (b) `SelectionSort` (Sortieren durch Auswahl), (c) `QuickSort` und (d) `BubbleSort`. Geben Sie unter den Folgen jeweils den Namen des zugehörigen Algorithmus an.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

_____Sort

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

_____Sort

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

_____Sort

/ 2 P

b) Gegeben sei eine Menge von Frauen $\mathcal{F} = \{f_1, f_2, f_3, f_4\}$ sowie eine Menge von Männern $\mathcal{M} = \{m_1, m_2, m_3, m_4, m_5\}$. Die folgenden beiden Tabellen zeigen jeweils ihre Präferenzen, die von links nach rechts absteigend sortiert sind.

f_1	m_1	m_2	m_3	m_4	m_5
f_2	m_4	m_2	m_3	m_5	m_1
f_3	m_4	m_1	m_5	m_2	m_3
f_4	m_2	m_4	m_3	m_5	m_1

m_1	f_2	f_4	f_1	f_3
m_2	f_2	f_1	f_4	f_3
m_3	f_3	f_4	f_1	f_2
m_4	f_3	f_1	f_4	f_2
m_5	f_1	f_2	f_3	f_4

Führen Sie den Algorithmus von Gale und Shapley aus und beantworten Sie folgende Fragen.

(i) Welcher Mann bleibt Junggeselle, wenn die Frauen die Heiratsanträge machen?

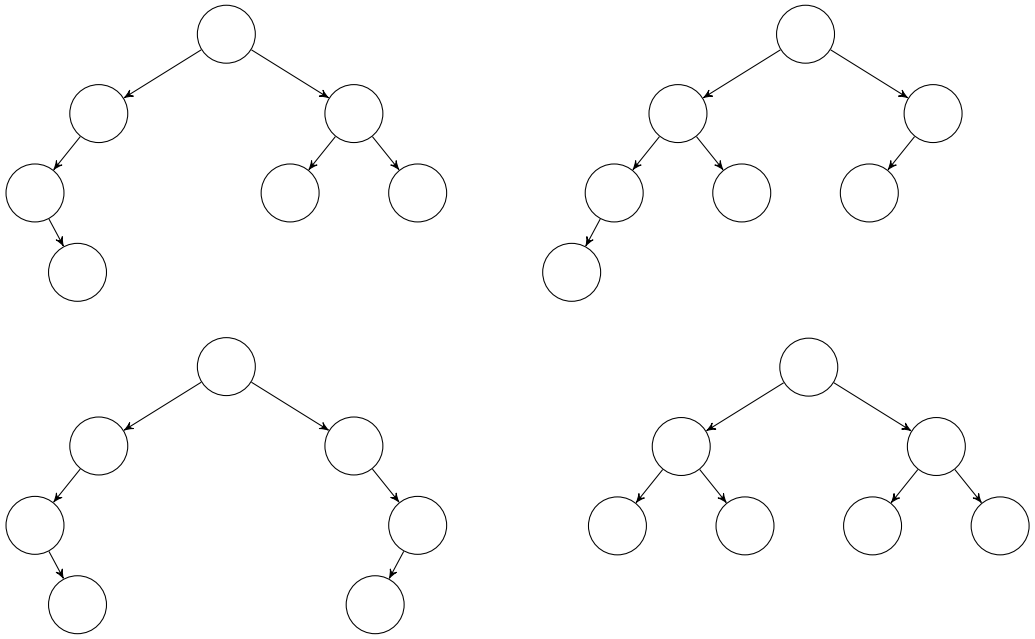
Junggeselle in (i):

(ii) Welcher Mann bleibt Junggeselle, wenn die Männer die Heiratsanträge machen?

Junggeselle in (ii):

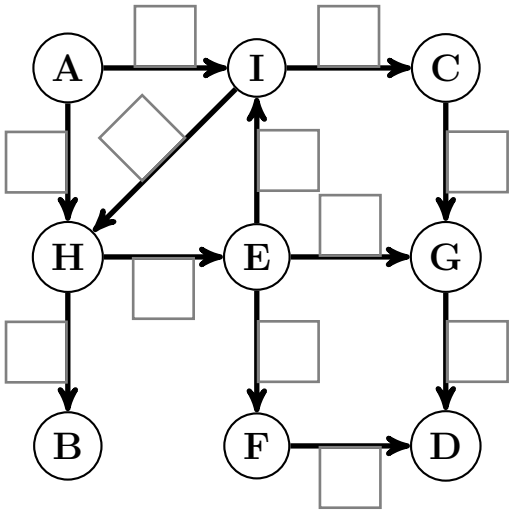
/ 1 P

c) *AVL-Bäume*: Tragen Sie die Schlüssel `1`, `2`, `3`, `4`, `5`, `6`, `7` in die Knoten von *zwei* der vier folgenden binären Suchbäume ein, sodass diese beiden Bäume gültige AVL-Bäume darstellen.



/ 2 P

d) Führen Sie im folgenden Graphen eine Tiefensuche aus. Starten Sie in Knoten A und nehmen Sie an, dass die Traversierung die jeweiligen Nachbarknoten in alphabetischer Reihenfolge besucht. Geben Sie die Tiefenordnung an und schreiben sie zu jeder Kante, ob sie eine Baumkante (B), eine Vorwärtskante (V), eine Rückwärtskante (R) oder eine Querkante (Q) ist.



Tiefenordnung:

___, ___, ___, ___, ___, ___, ___, ___, ___.

/ 1P

e) Wenden Sie auf den nachfolgenden Array von Zahlen den Algorithmus von Blum (Median der Medianе) an. Geben Sie dazu den **ersten** Median der Mediane an.

11	54	3	18	24	21	7	51	63	12	8	14	33	25	9

Erster Median der Mediane:

/ 1 P

f) Geben Sie für das folgende Codefragment die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ in **möglichst knapper Θ-Notation** an. Die Funktion f läuft in $\Theta(1)$ Zeit. Sie müssen Ihre Antwort nicht begründen.

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

Asymptotische Laufzeit:

/ 1 P

g) Geben Sie für das folgende Codefragment die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ in **möglichst knapper Θ-Notation** an. Die Funktion f läuft in $\Theta(1)$ Zeit. Sie müssen Ihre Antwort nicht begründen.

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

Asymptotische Laufzeit:

/ 1P

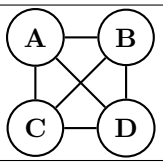
h) Führen Sie eine Extract-Min Operation einschliesslich Wiederherstellung der Heap-Bedingungen auf folgendem Min-Heap aus. Wie sieht der Heap nach der Operation aus?

12	14	50	30	15	52	63	61	35	18	99	67

Heap nach dem Löschen:

/ 2P

i) Beantworten Sie nachfolgende Fragen. Jede korrekte Antwort gibt 0,5 Punkte, für jede falsche oder fehlende Antwort werden 0,5 Punkte abgezogen. Insgesamt gibt die Aufgabe mindestens 0 Punkte. Sie müssen Ihre Antworten nicht begründen.

Wie viele Kanten hat ein ungerichteter Graph, wenn die Summe seiner Knotengrade 96 beträgt?	
Aus wie vielen Zusammenhangskomponenten besteht ein Wald mit 10 Kanten und 17 Knoten?	
Wie viele Blätter (Knoten mit Knotengrad 1) hat obiger Wald mindestens?	
Wie viele verschiedene Spannbäume gibt es im abgebildeten vollständigen Graphen mit 4 Knoten? 	

/ 2 P

j) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 72 + 9 \cdot T(\frac{n}{3}) + 4n & n > 1 \text{ und } n \text{ eine Dreierpotenz} \\ 13 & n = 1 \end{cases}$$

Sie können annehmen, dass n eine Potenz von 3 ist. Zeigen Sie mit vollständiger Induktion, dass

$$T(n) = 24n^2 - 2n - 9$$

die dazugehörige geschlossene Form ist.

Induktionsbeweis:

Theorieaufgabe T2. **/ 10 P**

Mit der Einführung von Inland-Fernbussen, Spartickets und Zonenabonnements wird es für Sie als studentische(n) Wochenaufenthalter(in) immer unklarer, ob sich ein Generalabonnement noch lohnt. Entscheidend ist, wie teuer Sie jeweils am Freitagabend eine Heimfahrt zu stehen kommt. Selbstverständlich möchten Sie dabei nicht auf Komfort verzichten, und deshalb *so selten wie möglich umsteigen*.

Vorhanden ist eine Liste von m verschiedenen Strecken mit Preisangabe im Format s_i t_i p_i: Die i-te Strecke kann ohne Umsteigen von s_i nach t_i zum Preis von $p_i > 0$ Franken befahren werden. Im Anschluss können Sie auf eine beliebige Strecke j mit $s_j = t_i$ umsteigen.

Gesucht ist *der Preis einer günstigsten Reise* von Zürich zu Ihrem Wohnort, wobei Sie *möglichst wenig oft umsteigen möchten*. Sie dürfen annehmen, dass so eine Reise immer existiert.

Beispiel. Sie wohnen in Disentis und haben die folgenden Strecken zur Auswahl:
`Zürich Chur 30.-, Zürich Bellinzona 40.-, Zürich Erstfeld 20.-,`
`Chur Disentis 15.-, Bellinzona Disentis 10.-,`
`Erstfeld Andermatt 10.-, Andermatt Disentis 10.-`
Dann ist die gesuchte Reise Zürich—Chur—Disentis zum Preis von 45.- (die Reise via Bellinzona ist teurer; die Reise via Erstfeld–Andermatt ist zwar günstiger, benötigt aber mehr Umsteigevorgänge).

Aufgabe. Modellieren Sie die Fragestellung als graphentheoretisches Problem. Geben Sie an, welche Knoten und Kanten definiert werden und gegebenenfalls welche Gewichte den Kanten zugeordnet werden. Beschreiben Sie insbesondere, wo sich der gesuchte Preis in ihrer Modellierung wiederfindet. Entwerfen Sie einen möglichst effizienten Algorithmus zur Lösung dieses Problems und bestimmen Sie seine Laufzeit in Abhängigkeit von m.

- Graphentheoretische Modellierung inklusive allfälliger Kantengewichte (in Worten).
  Verwenden Sie Zürich als Knoten Z und den Wohnort als Knoten W:

- Der gesuchte Preis entspricht in obiger Modellierung:

- Entwurf eines möglichst effizienten Algorithmus:

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

/ 14 P

Theorieaufgabe T3.

Die folgenden (voneinander unabhängigen) Teilaufgaben drehen sich um *minimale Spanngraphen*. Gegeben sei ein ungerichteter zusammenhängender Graph $G = (V, E, c)$ in Adjazenzlistenform, mit $n := |V|$ Knoten, $m := |E|$ Kanten und Kantenkostenfunktion $c\colon E \to \mathbb{R}$.

Ein *Spanngraph* ist ein zusammenhängender Teilgraph, welcher alle Knoten enthält. Gesucht ist ein *minimaler Spanngraph*, also ein Spanngraph $G' = (V, E')$ mit $E' \subseteq E$, sodass $\sum_{e \in E'} c(e)$ minimal ist.

/ 4 P

a) Nehmen Sie in dieser Teilaufgabe an, dass alle Kantenkosten positiv sind, $\forall e \in E\colon c(e) > 0$. Beweisen Sie, dass in diesem Fall jeder minimale Spanngraph ein Baum ist. Geben Sie einen möglichst effizienten Algorithmus an, welcher einen minimalen Spanngraphen berechnet, und nennen Sie dessen Laufzeit.

Beweis:

Möglichst effizienter Algorithmus:

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

/ 5 P

b) In dieser Teilaufgabe gibt es *keine* Einschränkungen der Kantenkosten, es können also auch nicht-positive Kosten $c(e) \leq 0$ auftreten.

Entwerfen Sie einen möglichst effizienten Algorithmus, welcher einen minimalen Spanngraphen berechnet. Analysieren Sie auch dessen Laufzeit.

Algorithmenentwurf:

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

/ 5 P

c) Nehmen Sie in dieser Teilaufgabe an, dass alle Kantenkosten positiv sind, das heisst $\forall e \in E\colon c(e) > 0$. Sie haben für den Graphen G bereits einen minimalen Spannbaum T von G berechnet. Nun erhalten Sie eine zusätzliche Kante e^* mit Kosten $c(e^*) > 0$.

Entwerfen Sie einen möglichst effizienten Algorithmus, welcher einen minimalen Spannbaum des um die Kante e^* erweiterten Graphen G berechnet. Nutzen Sie dazu die Kenntnis von T möglichst gut aus. Analysieren Sie auch die Laufzeit Ihres Algorithmus.

Algorithmenentwurf:

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

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