Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Tobias Pröger

Beispiellösung zur Prüfung
Datenstrukturen und Algorithmen
D-INFK

7. August 2014

Name, Vorname: _____

Stud.-Nummer: _____

Ich bestätige mit meiner Unterschrift, dass ich diese Prüfung unter regulären Bedingungen ablegen konnte und dass ich die untenstehenden Hinweise gelesen und verstanden habe.

Unterschrift: _____

Hinweise:

- Ausser einem Wörterbuch dürfen Sie keine Hilfsmittel verwenden.
- Bitte schreiben Sie Ihre Studierenden-Nummer auf **jedes** Blatt.
- Melden Sie sich bitte **sofort**, wenn Sie sich während der Prüfung in irgendeiner Weise bei der Arbeit gestört fühlen.
- Bitte verwenden Sie für jede Aufgabe ein neues Blatt. Pro Aufgabe kann nur eine Lösung angegeben werden. Ungültige Lösungsversuche müssen klar durchgestrichen werden.
- Bitte schreiben Sie **lesbar** mit blauer oder schwarzer Tinte. Wir werden nur bewerten, was wir lesen können.
- Sie dürfen alle Algorithmen und Datenstrukturen aus der Vorlesung verwenden, ohne sie noch einmal zu beschreiben. Wenn Sie sie modifizieren, reicht es, die Modifikationen zu beschreiben.
- Die Prüfung dauert 180 Minuten.

Viel Erfolg!

Stud.-Nummer: _____

Aufgabe	1	2	3	4	**Σ**
Mögl. Punkte	16	13	6	10	45
$\sum$ Punkte					

Aufgabe 1.

Hinweise:

1) In dieser Aufgabe sollen Sie **nur die Ergebnisse** angeben. Diese können Sie direkt bei den Aufgaben notieren.
2) Sofern Sie die Notationen, Algorithmen und Datenstrukturen aus der Vorlesung "Datenstrukturen & Algorithmen" 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.
3) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

1 P a) Führen Sie auf dem folgenden Array zwei Iterationen des Sortieralgorithmus *Sortieren durch Einfügen* aus. Das zu sortierende Array ist durch vorherige Iterationen bereits bis zum Doppelstrich sortiert worden.

2	5	7	15 ‖	9	11	8	3	1	12	14	20
1	2	3	4	5	6	7	8	9	10	11	12

2	5	7	9	15 ‖	11	8	3	1	12	14	20
1	2	3	4	5	6	7	8	9	10	11	12

2	5	7	9	11	15 ‖	8	3	1	12	14	20
1	2	3	4	5	6	7	8	9	10	11	12

1 P b) Fügen Sie die Schlüssel 9, 11, 17, 25, 31, 20 in dieser Reihenfolge in die untenstehende Hashtabelle ein. Benutzen Sie Double Hashing mit der Hashfunktion $h(k) = k \bmod 11$, und benutzen Sie $h'(k) = 1 + (k \bmod 9)$ zur Sondierung.

Lösung:

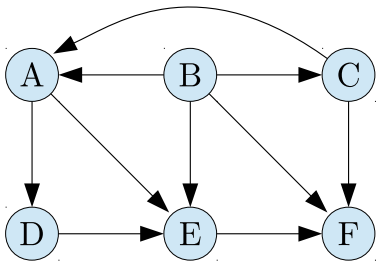
Bei Sondierung nach links:

11			25	31		17		20	9	
0	1	2	3	4	5	6	7	8	9	10

Bei Sondierung nach rechts:

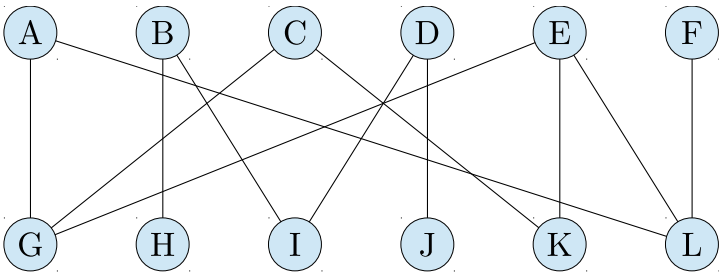
11	20		25			17		31	9	
0	1	2	3	4	5	6	7	8	9	10

1 P c) Geben Sie eine topologische Sortierung des untenstehenden Graphen an.



Topologische Sortierung: B, C, A, D, E, F

1 P d) Geben Sie eine *möglichst kleine* Teilmenge der Knoten des folgenden Graphen an, die mit dem Satz von Hall beweist, dass der Graph kein perfektes Matching besitzt.



Lösung: Die Menge $X = \{H, I, J\}$ hat Kardinalität 3, aber die Nachbarschaft $\Gamma(X) = \{B, D\}$ enthält lediglich 2 Elemente. Nach dem Satz von Hall kann der obige Graph also kein perfektes Matching haben.

1 P e) Geben Sie an, wieviele Schlüsselvergleiche benötigt werden, wenn in der folgenden selbstanordnenden Liste mit der Move-to-Front-Regel auf die Elemente 'L', 'A', 'M', 'A', 'H', 'A', 'A', 'R' in dieser Reihenfolge zugegriffen wird.

$$\text{A} \longrightarrow \text{L} \longrightarrow \text{G} \longrightarrow \text{O} \longrightarrow \text{R} \longrightarrow \text{I} \longrightarrow \text{T} \longrightarrow \text{H} \longrightarrow \text{M} \longrightarrow \text{U} \longrightarrow \text{S}$$

Lösung:

Zugriff auf 'L': 2 Vergleiche. Neue Struktur der Liste:

$$\text{L} \longrightarrow \text{A} \longrightarrow \text{G} \longrightarrow \text{O} \longrightarrow \text{R} \longrightarrow \text{I} \longrightarrow \text{T} \longrightarrow \text{H} \longrightarrow \text{M} \longrightarrow \text{U} \longrightarrow \text{S}$$

Zugriff auf 'A': 2 Vergleiche. Neue Struktur der Liste:

$$\text{A} \longrightarrow \text{L} \longrightarrow \text{G} \longrightarrow \text{O} \longrightarrow \text{R} \longrightarrow \text{I} \longrightarrow \text{T} \longrightarrow \text{H} \longrightarrow \text{M} \longrightarrow \text{U} \longrightarrow \text{S}$$

Zugriff auf 'M': 9 Vergleiche. Neue Struktur der Liste:

$$\text{M} \longrightarrow \text{A} \longrightarrow \text{L} \longrightarrow \text{G} \longrightarrow \text{O} \longrightarrow \text{R} \longrightarrow \text{I} \longrightarrow \text{T} \longrightarrow \text{H} \longrightarrow \text{U} \longrightarrow \text{S}$$

Zugriff auf 'A': 2 Vergleiche. Neue Struktur der Liste:

$$\text{A} \longrightarrow \text{M} \longrightarrow \text{L} \longrightarrow \text{G} \longrightarrow \text{O} \longrightarrow \text{R} \longrightarrow \text{I} \longrightarrow \text{T} \longrightarrow \text{H} \longrightarrow \text{U} \longrightarrow \text{S}$$

Zugriff auf 'H': 9 Vergleiche. Neue Struktur der Liste:

$$\text{H} \longrightarrow \text{A} \longrightarrow \text{M} \longrightarrow \text{L} \longrightarrow \text{G} \longrightarrow \text{O} \longrightarrow \text{R} \longrightarrow \text{I} \longrightarrow \text{T} \longrightarrow \text{U} \longrightarrow \text{S}$$

Zugriff auf 'A': 2 Vergleiche. Neue Struktur der Liste:

$$\text{A} \longrightarrow \text{H} \longrightarrow \text{M} \longrightarrow \text{L} \longrightarrow \text{G} \longrightarrow \text{O} \longrightarrow \text{R} \longrightarrow \text{I} \longrightarrow \text{T} \longrightarrow \text{U} \longrightarrow \text{S}$$

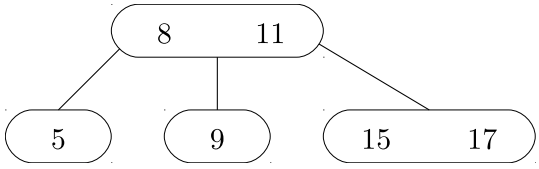
Zugriff auf 'A': 1 Vergleich. Neue Struktur der Liste:

$$\text{A} \longrightarrow \text{H} \longrightarrow \text{M} \longrightarrow \text{L} \longrightarrow \text{G} \longrightarrow \text{O} \longrightarrow \text{R} \longrightarrow \text{I} \longrightarrow \text{T} \longrightarrow \text{U} \longrightarrow \text{S}$$

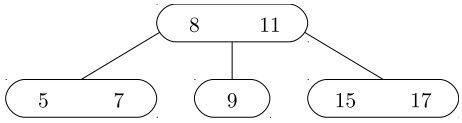
Zugriff auf 'R': 7 Vergleiche.

Zusammen ergeben sich also $2 + 2 + 9 + 2 + 9 + 2 + 1 + 7 = 34$ Vergleiche.

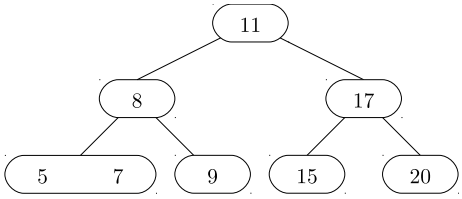
1 P f) Fügen Sie in den untenstehenden 2-3-Baum (B-Baum der Ordnung 3) zuerst den Schlüssel 7 und in den entstehenden Baum den Schlüssel 20 ein. Führen Sie auch die zugehörigen Strukturänderungen durch.



Nach Einfügen von 7:



Nach Einfügen von 20:



1 P g) 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.

- 2^n
- $\binom{n}{2}$
- $n \cdot (\log n)^5$
- 15^7
- $3^{n/2}$
- $n!$
- $\frac{n}{(\log n)^2}$

Lösung: Es gilt $\binom{n}{2} = \frac{n(n-1)}{2} \in \Theta(n^2)$. Weiterhin gilt $3^{n/2} = (3^{1/2})^n = (\sqrt{3})^n$, und wegen $\sqrt{3} < 2$ ist $(\sqrt{3})^n \in \mathcal{O}(2^n)$. Die einzige gültige Reihenfolge ist daher

$$15^7, \frac{n}{(\log n)^2}, n \cdot (\log n)^5, \binom{n}{2}, (\sqrt{3})^n, 2^n, n!$$

3 P h) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 12 + 7T(n/7) & n > 1 \\ 3 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (d.h. nicht-rekursive) und *möglichst einfache* Formel für $T(n)$ an und beweisen Sie diese mit vollständiger Induktion.

Lösung: Da wir annehmen dürfen, dass n eine Potenz von 7 ist, gilt $n = 7^k$ für ein $n \in \mathbb{N}$. Wir teleskopieren, um auf eine Formel für $T(n)$ zu kommen:

$$\begin{aligned} T(n) &= 12 + 7T(n/7) \\ &= 12 + 7(12 + 7T(n/7^2)) = 12 + 7 \cdot 12 + 7^2 T(n/7^2) \\ &= 12 + 7 \cdot 12 + 7^2(12 + 7T(n/7^3)) = 12 + 7 \cdot 12 + 7^2 \cdot 12 + 7^3 T(n/7^3) \\ &= \ldots \stackrel{!}{=} \left(12 \sum_{i=0}^{k-1} 7^i \right) + 7^k \cdot T(1) = 12 \cdot \frac{7^k - 1}{7 - 1} + 7^k \cdot 3 = 2(7^k - 1) + 3 \cdot 7^k = 5 \cdot 7^k - 2. \end{aligned}$$

Wir beweisen nun unsere Annahme durch vollständige Induktion über k.

Induktionsverankerung ($k = 0$): Es gilt $T(7^0) = T(1) = 3 = 5 \cdot 7^0 - 2$.

Induktionsannahme: Für ein $k \in \mathbb{N}_0$ sei $T(7^k) = 5 \cdot 7^k - 2$.

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

$$\begin{aligned} T(7^{k+1}) = 12 + 7 \cdot T(7^k) &\stackrel{\text{Ind.-Ann.}}{=} 12 + 7 \cdot (5 \cdot 7^k - 2) \\ &= 5 \cdot 7^{k+1} + 12 - 14 = 5 \cdot 7^{k+1} - 2. \end{aligned}$$

1 P i) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für den folgenden Algorithmus (so knapp wie möglich) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = 1; i <= n; i += 3) {
2         for(int j = 1; j <= 2*i; j ++)
3                 ;
4 }
```

Lösung: Wir schätzen zunächst die Laufzeit nach oben ab. Die äussere Schleife wird höchstens n Mal durchlaufen, und wegen $i \leq n$ wird die innere Schleife höchstens $2n$ Mal durchlaufen. Die Laufzeit ist also in $\mathcal{O}(n^2)$. Andererseits wird die äussere Schleife mindestens $\lfloor n/3 \rfloor$ Mal durchlaufen, und für $i > n/2$ wird die innere Schleife mindestens n Mal durchlaufen. Die Laufzeit ist damit auch in $\Omega(n^2)$. Insgesamt erhalten wir also eine Laufzeit von $\Theta(n^2)$.

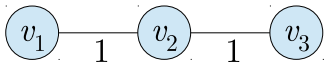
1 P j) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für den folgenden Algorithmus (so knapp wie möglich) in Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

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

Lösung: Die äussere Schleife wird genau $n - 1 \in \Theta(n)$ Mal durchlaufen, die innere genau $\lfloor \log_3(n) \rfloor \in \Theta(\log n)$ Mal. Wir erhalten also eine Gesamtlaufzeit in $\Theta(n \log n)$.

1 P k) Zeigen oder widerlegen Sie: Der minimale Spannbaum eines ungerichteten gewichteten Graphen $G = (V, E, w)$ ist genau dann eindeutig, wenn G keine zwei Kanten mit dem gleichen Gewicht besitzt.

Lösung: Die Aussage ist falsch. Zwar ist der minimale Spannbaum eindeutig bestimmt, wenn keine zwei Kanten das gleiche Gewicht besitzen, die umgekehrte Richtung gilt allerdings nicht. Ein Gegenbeispiel ist z.B. der folgende Graph, der ausschliesslich aus Kanten mit Gewicht 1 besteht, und in dem der minimale Spannbaum dem Graphen selbst entspricht (d.h., der insbesondere eindeutig bestimmt ist).



3 P l) Gegeben ist ein AVL-Baum der Höhe h (zur Erinnerung: Ein AVL-Baum der Höhe 1 besteht aus genau einem Knoten). Wir definieren $\phi := (1+\sqrt{5})/2$. Sei $N(h)$ die minimale Knotenanzahl eines AVL-Baums mit der Höhe h. Zeigen Sie durch allgemeine Induktion über h, dass jeder AVL-Baum der Höhe h mindestens ϕ^{h-1} viele Knoten hat, d.h., dass $N(h) \geq \phi^{h-1}$ gilt. Betrachten Sie dazu im Induktionsschritt einen AVL-Baum mit der Höhe h und minimaler Knotenanzahl, und skizzieren Sie die Struktur eines solchen Baums.

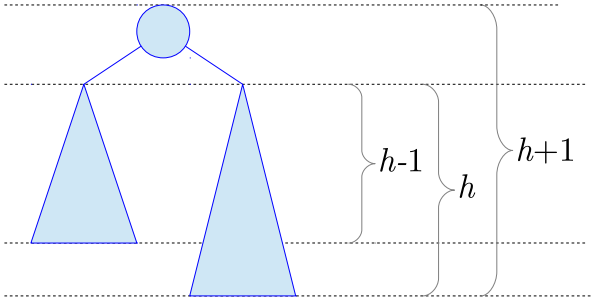
Lösung:

Induktionsverankerung I ($h = 1$): Ein AVL-Baum der Höhe 1 besteht aus genau einem Knoten, hat folglich also mindestens $N(1) = 1 \geq \phi^{1-1} = 1$ Knoten.

Induktionsverankerung II ($h = 2$): Ein AVL-Baum der Höhe 2 besitzt neben der Wurzel mindestens einen weiteren Knoten, hat folglich also mindestens $N(2) = 2 \geq \phi^{2-1} = \phi$ Knoten.

Induktionsannahme: Angenommen, jeder AVL-Baum der Höhe höchstens h hat mindestens ϕ^{h-1} viele Knoten, d.h. es sei $N(h') \geq \phi^{h'-1}$ für alle $h' \in \{1, \ldots, h\}$.

Induktionsschritt ($h \to h+1$): Ein AVL-Baum der Höhe h mit einer minimalen Anzahl von Knoten besteht aus einer Wurzel mit einem Teilbaum der Höhe $h-1$ und einem Teilbaum der Höhe h (hätten beide Teilbäume Höhe h, dann wäre die Knotenanzahl sicherlich nicht minimal). Ein solcher Baum besitzt die folgende Struktur:



Wir erhalten also

$$\begin{aligned} N(h+1) \geq 1 + N(h-1) + N(h) &\overset{\text{Ind.-Ann.}}{\geq} 1 + \phi^{h-2} + \phi^{h-1} = 1 + \phi^{h-2}(1+\phi) \\ &= 1 + \phi^{h-2}\phi^2 = 1 + \phi^h \geq \phi^h, \end{aligned}$$

was die Aussage beweist.

Aufgabe 2.

Ein Student möchte heute und morgen jeweils eine Portion Bratkartoffeln kochen und hat dazu n Kartoffeln $\{1, \ldots, n\}$ mit einem Gesamtgewicht von $G \in \mathbb{N}$ Gramm zur Verfügung. Die Kartoffel i wiegt $g_i \in \mathbb{N}$ Gramm. Es sollen nun *alle* n Kartoffeln so auf die zwei Portionen A und B verteilt werden, dass diese annähernd gleich schwer sind. Da die Portionen für unterschiedliche Tage bestimmt sind, muss eine Kartoffel entweder komplett für die Portion A oder komplett für die Portion B verwendet werden (jede Kartoffel muss entweder sofort benutzt werden oder *vollständig* für morgen aufbewahrt werden). Konkret suchen wir also zwei Mengen A und B von Kartoffeln mit $A \cap B = \emptyset$, $A \cup B = \{1, \ldots, n\}$, deren Gewichtsdifferenz minimal ist.

8 P a) Geben Sie einen Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die kleinstmögliche Gewichtsdifferenz zweier Mengen A und B (wie oben beschrieben) berechnet.

Lösung:

Definition der DP-Tabelle: Wir verwenden eine $(n+1) \times (\lfloor G/2 \rfloor + 1)$-Tabelle T mit Einträgen, die entweder "true" oder "false" sind. Für $0 \leq i \leq n$ und $0 \leq g \leq \lfloor G/2 \rfloor$ sei genau dann $T[i, g] =$ true, wenn es eine Menge $K \subseteq \{1, \ldots, i\}$ der ersten i Kartoffeln gibt, deren Gesamtgewicht exakt g beträgt, d.h. die $\sum_{k \in K} g_k = g$ erfüllt.

Berechnung eines Eintrags: Wir unterscheiden drei Fälle.

- $T[i, 0] =$ true für jedes $i \in \{0, \ldots, n\}$, denn die Menge $\{1, \ldots, i\}$ enthält die leere Menge $K = \emptyset$ mit Gewicht 0.
- $T[0, g] =$ false für jedes $g \in \{1, \ldots, \lfloor G/2 \rfloor\}$, denn mit der leeren Menge kann kein Gewicht $g > 0$ erzielt werden.
- Für alle $i \in \{1, \ldots, n\}$ und $g \in \{1, \ldots, \lfloor G/2 \rfloor\}$ setzen wir

$$T[i, g] = \begin{cases} T[i-1, g] & \text{falls } g_i > g \\ T[i-1, g] \vee T[i-1, g-g_i] & \text{sonst} \end{cases}$$

  Für $g_i > g$ wiegt die Kartoffel i mehr als das zulässige Gesamtgewicht der Auswahl, folglich kann sie niemals in dieser enthalten sein und es gilt $T[i, g] = T[i-1, g]$. Ansonsten kann die Kartoffel i in der Auswahl entweder enthalten sein, oder nicht, d.h., mit den Kartoffeln $\{1, \ldots, i-1\}$ muss entweder das Gewicht $g - g_i$ erzielt werden können (falls die Kartoffel benutzt wird), oder das Gewicht g (falls die Kartoffel nicht benutzt wird).

Berechnungsreihenfolge: Wir berechnen die Einträge $T[i, g]$ für aufsteigende Werte von i, und für gleiche Werte von i für aufsteigende Werte von g.

Auslesen der Lösung: Für jedes $g \in \{0, \ldots, \lfloor G/2 \rfloor\}$ ist genau dann $T[n, g] =$ true, wenn die gegebenen Kartoffeln eine Teilmenge mit Gewicht g besitzen. Zur Ermittlung der grösstmöglichen Gewichtsdifferenz suchen wir das grösste g_{max} mit $T[n, g_{max}] =$ true. Dies ist das maximal mögliche Gewicht einer Portion mit Gewicht höchstens $\lfloor G/2 \rfloor$. Die minimale Gewichtsdifferenz beträgt dann $(G - g_{max}) - g_{max} = G - 2g_{max}$ Gramm.

2 P b) Beschreiben Sie detailliert, wie aus der DP-Tabelle abgelesen werden kann, welche Kartoffel an welchem Tag benutzt wird.

Lösung: Sei g_{max} wie in a) der grösste Wert mit $T[n, g_{max}] = \text{true}$. Wir setzen $i \leftarrow n$ und $g \leftarrow g_{max}$ und verfahren wie folgt. Ist $i = 0$, dann sind wir fertig. Ansonsten prüfen wir, ob $T[i, g] = T[i-1, g]$ gilt. Falls ja, dann setzen wir $i \leftarrow i - 1$ und fahren wie beschrieben fort. Ansonsten gilt $T[i, g] = T[i-1, g-g_i]$, wir geben i aus, setzen $i \leftarrow i - 1$ und $g \leftarrow g - g_i$, und fahren wie beschrieben fort. Die auf diese Art ausgegebenen Kartoffeln werden heute, die übrigen am nächsten Tag benutzt (oder umgekehrt).

3 P c) Geben Sie die Laufzeit des in a) und b) entwickelten Verfahrens an und begründen Sie Ihre Antwort. Ist die Laufzeit polynomiell?

Lösung: Die Tabelle in a) hat eine Grösse von $\Theta(nG)$, jeder Eintrag kann in konstanter Zeit aus früher berechneten Einträgen berechnet werden. Damit beträgt die Gesamtlaufzeit $\Theta(nG)$, was lediglich pseudopolynomiell ist.

Der Algorithmus in b) terminiert nach $\Theta(n)$ Schritten (wenn g_{max} bekannt ist, was wir annehmen). Jeder Schritt ist in konstanter Zeit ausführbar. Die Laufzeit beträgt daher $\Theta(n)$.

Aufgabe 3.

Im Devisenhandel bezeichnet *Arbitrage* das Ausnutzen von Preisunterschieden, um durch mehrfaches Wechseln von Währungen einen Gewinn zu erzielen. Am 2. Juni 2009 zum Beispiel konnte 1 US-Dollar in 95.729 Yen gewechselt werden, 1 Yen in 0.00638 Britische Pfund und 1 Britisches Pfund in 1.65133 US-Dollar. Hätte ein Händler also 1 US-Dollar in Yen gewechselt, den erhaltenen Betrag in Britische Pfund und schliesslich diesen Betrag dann zurück in US-Dollar getauscht, dann hätte er $95.729 \cdot 0.00638 \cdot 1.65133 \approx 1.0086$ US-Dollar erhalten, was einem Gewinn von 0.86% entspricht.

3 P a) Gegeben seien n Währungen $\{1, \ldots, n\}$ und eine $(n \times n)$-Wechselkursmatrix $R \in (\mathbb{Q}^+)^2$. Für zwei Währungen $i, j \in \{1, \ldots, n\}$ kann eine Einheit der Währung i in $R(i, j) > 0$ Einheiten der Währung j getauscht werden. Es soll entschieden werden, ob ein Arbitrage-Geschäft möglich ist, d.h., ob es eine Folge von k verschiedenen Währungen $W_1, \ldots, W_k \in \{1, \ldots, n\}$ mit $R(W_1, W_2) \cdot R(W_2, W_3) \cdots R(W_{k-1}, W_k) \cdot R(W_k, W_1) > 1$ gibt.

Modellieren Sie das Problem als Graphproblem. Konstruieren Sie aus der obigen Eingabe einen gerichteten, gewichteten Graphen $G = (V, E, w)$, der *genau dann* einen Kreis mit negativem Gewicht besitzt, wenn ein Arbitrage-Geschäft möglich ist. Begründen Sie, dass G genau dann einen Kreis negativer Länge enthält, wenn ein Arbitrage-Geschäft möglich ist.

Lösung: Wir erzeugen einen vollständigen Graphen mit der Knotenmenge $V = \{1, \ldots, n\}$. Die Kante (i, j) bekommt das Gewicht $w((i, j)) = -\ln(R(i, j))$. Angenommen, ein Arbitrage-Geschäft mit den Währungen $W_1, \ldots, W_k \in \{1, \ldots, n\}$ ist möglich. Dann gilt

$$
\begin{aligned}
& R(W_1, W_2) \cdot R(W_2, W_3) \cdots R(W_{k-1}, W_k) \cdot R(W_k, W_1) > 1 \\
& \quad \Leftrightarrow \ln(R(W_1, W_2) \cdot R(W_2, W_3) \cdots R(W_{k-1}, W_k) \cdot R(W_k, W_1)) > \ln(1) = 0 \\
& \quad \Leftrightarrow \ln(R(W_1, W_2)) + \ln(R(W_2, W_3)) + \cdots + \ln(R(W_{k-1}, W_k)) + \ln(R(W_k, W_1)) > 0 \\
& \quad \Leftrightarrow -\ln(R(W_1, W_2)) - \ln(R(W_2, W_3)) - \cdots - \ln(R(W_{k-1}, W_k)) - \ln(R(W_k, W_1)) < 0 \\
& \quad \Leftrightarrow w((W_1, W_2)) + w((W_2, W_3)) + \cdots + w((W_{k-1}, W_k)) + w((W_k, W_1)) < 0,
\end{aligned}
$$

folglich enthält G einen Zyklus mit negativem Gewicht. Da nur Äquivalenzumformungen durchgeführt wurden, gilt die Argumentation also in beide Richtungen.

3 P b) Welcher Kürzeste-Wege-Algorithmus kann benutzt werden, um in einem Graphen Kreise negativer Länge zu erkennen? An welchem Knoten kann der Algorithmus beginnen? Welche Laufzeit (in Abhängigkeit von n) erreicht der gewählte Algorithmus, wenn er auf den Graphen in a) angewendet wird?

Lösung: Der Algorithmus von Bellman und Ford ist eine sinnvolle Wahl, da er Zyklen mit negativem Gewicht erkennt. Da der Graph zusammenhängend ist und jeder Zyklus mit negativem Gewicht von jedem Knoten ausgehend erreicht werden kann, kann der Algorithmus bei jedem Knoten von G beginnen. Es gibt $|V| = n$ Knoten und $|E| = n(n-1) \in \Theta(n^2)$ viele Kanten. Der Algorithmus hat daher eine Laufzeit von $\Theta(|V||E|) = \Theta(n^3)$.

Aufgabe 4.

Gegeben sei eine zweidimensionale Landkarte, die einen Ausschnitt eines Gebirges zeigt. Auf dieser Karte sind die Positionen von n Wanderern eingetragen. Der Wanderer i befindet sich an der Position $W_i = (x_i^W, y_i^W) \in \mathbb{Q}^2$. Im Gebirge sind m Mobilfunk-Sendemasten aufgestellt. Der Sendemast j hat die Position $S_j = (x_j^S, y_j^S) \in \mathbb{Q}^2$ und eine Reichweite von $r_j \in \mathbb{Q}^+$. Konkret heisst dies, dass die Natels aller Personen, die sich innerhalb des Kreises mit Mittelpunkt S_j und Radius r_j befinden, Empfang haben. Die Aufgabe besteht nun darin, alle Wanderer zu identifizieren, deren Natel *keinen* Empfang hat.

9 P a) Entwerfen Sie einen möglichst effizienten Scanline-Algorithmus, der als Eingabe die Positionen $W_1, \ldots, W_n$ von n Wanderern, die Positionen $S_1, \ldots, S_m$ von m Sendemasten sowie die entsprechenden Reichweiten $r_1, \ldots, r_m$ erhält, und der die Menge aller Wanderer berechnet, deren Natel keinen Empfang hat.

Lösung:

Haltepunkte: Wir verwenden eine vertikale Scanline, die von links nach rechts läuft. Dabei stoppen wir, sobald die Scanline die Position eines Wanderers schneidet, oder wenn ein Kreis startet oder endet. Die Haltepunkte sind also x_i^W für $i \in \{1, \ldots, n\}$, sowie $x_j^S - r_j$ und $x_j^S + r_j$ für $j \in \{1, \ldots, m\}$.

Scanline-Datenstruktur: Da sich keine zwei Kreise schneiden, können die Kreise eindeutig nach y-Koordinate verglichen werden. Zwar haben die Kreise y-Koordinaten, die in x-Richtung variieren, die *relative* Ordnung der Kreise bleibt jedoch gleich. Wir beobachten folgendes: Sind zwei Kreise K_j, K_l mit den Mittelpunkten (x_j^S, y_j^S) und (x_l^S, y_l^S) gegeben, und ist $y_j < y_l$, dann liegt K_j unterhalb von K_l. Daher können wir als Scanline-Datenstruktur einen AVL-Baum verwenden, der in y-Richtung aufsteigend geordnet ist, und als Schlüssel die y-Koordinaten y_j der entsprechenden Kreise benutzt.

Aktualisierung: Hier unterscheiden wir drei Fälle.

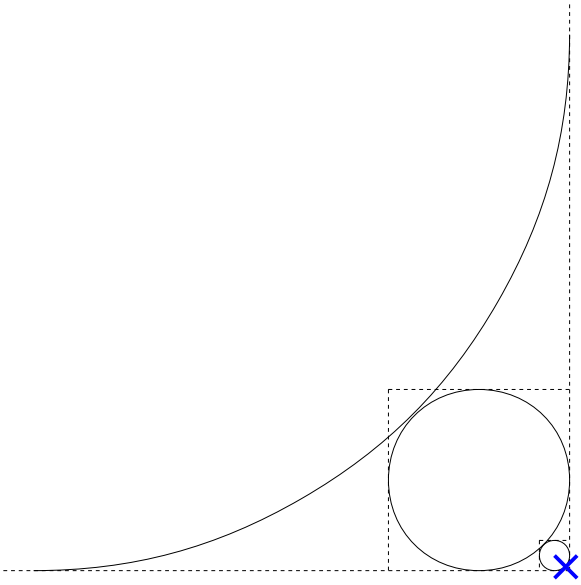
1. *Fall: Ein neuer Kreis startet.* Sei (x_j^S, y_j^S) der Mittelpunkt des entsprechenden Kreises. Dann fügen wir y_j^S in die Scanline-Datenstruktur ein.

2. *Fall: Ein Kreis endet.* Sei (x_j^S, y_j^S) der Mittelpunkt des entsprechenden Kreises. Dann entfernen wir y_j^S aus der Scanline-Datenstruktur.

3. *Fall: Ein Wanderer wird geschnitten.* Sei y_i^W die y-Koordinate des entsprechenden Wanderers. Sei K_j der in der Wurzel des AVL-Baums gespeicherte Kreis. Wir prüfen, ob (x_i^W, y_i^W) in K_j liegt. Ist dies der Fall, dann hat das Natel des Wanderers Empfang, folglich muss nichts ausgegeben werden und wir sind fertig. Ansonsten prüfen wir, ob (x_i^W, y_i^W) unterhalb oder oberhalb des Kreises K_j liegt, und fahren auf die gleiche Weise im linken bzw. rechten Teilbaum der Wurzel fort. Falls das Natel des Wanderers Empfang hat, dann finden wir auf diese Weise einen Kreis, der (x_i^W, y_i^W) enthält. Ansonsten endet unsere Suche erfolglos in einem Blatt, und wir wissen, dass das Natel des Wanderers keinen Empfang hat. In diesem Fall wird der Index i ausgegeben.

Auslesen der Lösung: Die Lösung wird direkt während der Aktualisierung der Datenstruktur ausgegeben. Für jeden Wanderer wird geprüft, ob sein Natel Empfang hat oder nicht, und der entsprechende Index wird ausgegeben falls das Natel keinen Empfang hat.

Hinweis: Ein Scanline-Algorithmus, der um jeden Kreis eine Bounding Box legt und die Kreise wie Quadrate behandelt, führt im schlimmsten Fall zu einer Laufzeit von $\Omega(nm)$ (tatsächlich sogar $\Omega(nm + (n+m)\log(n+m))$, da zusätzlich noch sortiert werden muss, und für konstante m oder n der Aufwand für das Sortieren nicht entfällt). Diese Laufzeit wird z.B. erreicht, wenn sich alle Wanderer ausserhalb aller Kreise, aber innerhalb aller Bounding Boxen befinden. Das folgende Beispiel zeigt die Position eines Wanderers (in blau eingezeichnet), der sich ausserhalb aller Kreise, aber innerhalb aller Bounding Boxen (gestrichelt eingezeichnet) befindet.



1 P b) Geben Sie die Laufzeit Ihres in a) entwickelten Algorithmus an und begründen Sie Ihre Antwort.

Lösung: Sei n die Anzahl der Wanderer und m die Anzahl der Sendemasten. Das Sortieren der Eckpunkte benötigt eine Laufzeit von $\Theta((n+m)\log(n+m))$. Einfügen und Löschen im AVL-Baum benötigen jeweils Zeit $\mathcal{O}(\log m)$ für alle $2m$ Haltepunkte der Sendemasten. Der Test, ob das Natel eines Wanderers Empfang hat, kann wie beschrieben ebenfalls in Zeit $\mathcal{O}(\log m)$ durchgeführt werden. Insgesamt erhalten wir also eine Laufzeit von $\Theta((n+m)\log(n+m))$.