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
Thomas Tschager

Beispiellösung zur Prüfung
Datenstrukturen und Algorithmen
D-INFK

27. Januar 2016

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, falls nicht explizit danach gefragt wird. 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	17	11	8	12	48
$\sum$ Punkte					

Aufgabe 1.

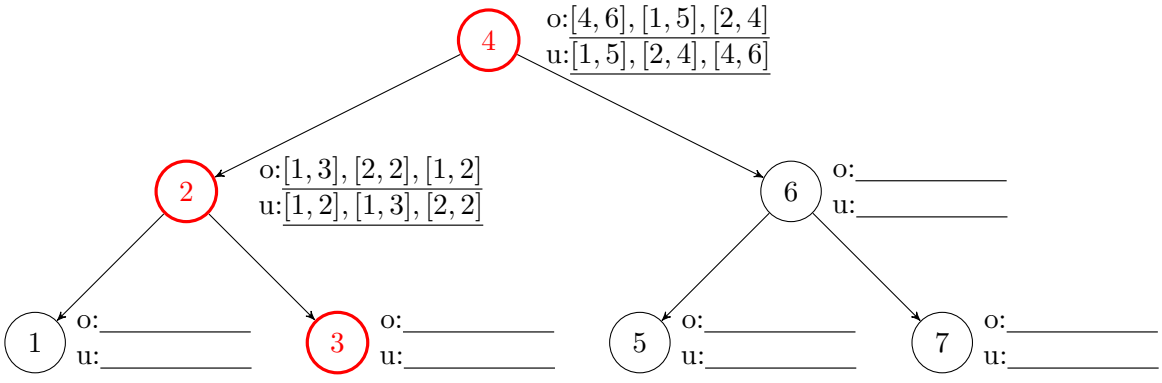
Hinweise:

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) Fügen Sie die Schlüssel 18, 4, 17, 7, 15, 29 in dieser Reihenfolge in die untenstehende Hashtabelle ein. Benutzen Sie Double Hashing mit der Hashfunktion $h(k) = k \bmod 11$ und benutzen Sie $h'(k) = 1 + (k \mod 9)$ zur Sondierung (nach links).

	29			4		17	18	15		7
0	1	2	3	4	5	6	7	8	9	10

1 P b) Fügen Sie die abgeschlossenen Intervalle $[1,3]$,$[1,5]$,$[4,6]$,$[2,4]$, $[2,2]$ und $[1,2]$ in den untenstehenden Intervallbaum ein. Markieren Sie ausserdem alle Knoten, die von einer Aufspiessanfrage für $x = 2.5$ besucht werden.

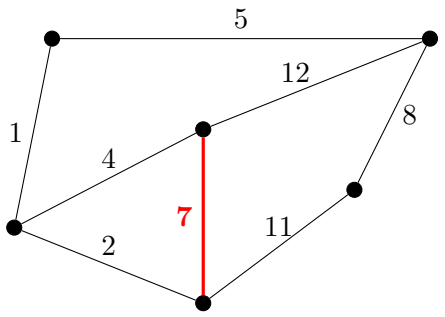


1 P c) Geben Sie an, wie viele Schlüsselvergleiche benötigt werden, wenn unter Verwendung der Move-to-Front Regel auf die folgende Liste eine Abfrage der Elemente C, B, D und B (in dieser Reihenfolge) erfolgt:

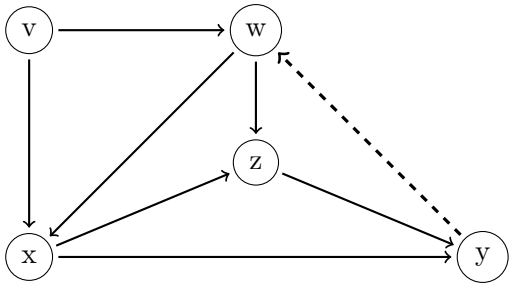
$$A \to B \to C \to D \to E$$

Lösung: Es werden $3 + 3 + 4 + 2 = 12$ Vergleiche benötigt.

1 P d) Markieren Sie im folgenden Graphen die erste Kante, die der Algorithmus von Kruskal betrachtet und ***nicht*** in den minimalen Spannbaum aufnimmt.



1 P e) 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.



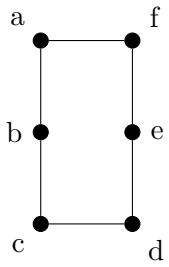
$E' =$ (y, w)

topologische Sortierung für G':

v, w, x, z, y

1 P f) Zeichnen Sie einen zusammenhängenden Graphen mit sechs Knoten, der genau zwei perfekte Matchings besitzt.

Lösung:



Zwei perfekte Matchings sind
$(a, b), (c, d), (e, f)$ und $(a, f), (b, c), (e, d)$.

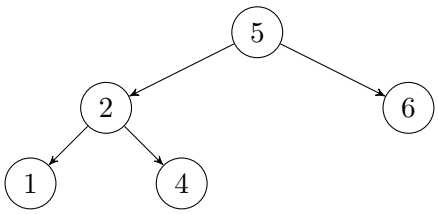
1 P g) Das folgende Array enthält die Elemente eines in üblicher Form gespeicherten Min-Heaps. Entfernen Sie das minimale Element aus dem Heap, stellen Sie die Heapbedingung wiederher, und geben Sie das resultierende Array an.

2	8	10	9	12	15	11	13	14
1	2	3	4	5	6	7	8	9

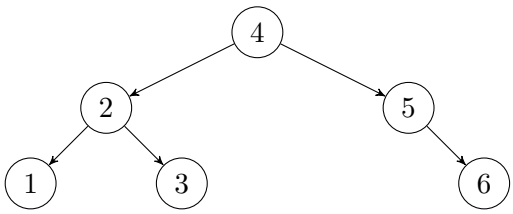
8	9	10	13	12	15	11	14	
1	2	3	4	5	6	7	8	9

1 P h) Zeichnen Sie einen AVL-Baum mit fünf Knoten und den darin gespeicherten fünf Schlüsseln, sodass durch das Einfügen eines sechsten Schlüssels eine links-rechts Doppelrotation notwendig wird. Geben Sie an, welcher Schlüssel dazu eingefügt werden muss und zeichnen Sie den dadurch entstehenden, neuen AVL-Baum.

Vor dem Einfügen des 6. Schlüssels:



Nach dem Einfügen des 6. Schlüssels:



Einzufügender Schlüssel: 3

3 P i) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 4T(n/4) + n + 6 & n > 1 \\ 1 & n = 1. \end{cases}$$

Geben Sie eine geschlossene (d.h. nicht-rekursive) und *möglichst einfache* Formel für $T(n)$ an und beweisen Sie diese mit vollständiger Induktion. Sie können annehmen, dass n eine Potenz von 4 ist.

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

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

$$\begin{aligned} T(4^k) &= 4\ T(4^{k-1}) + 4^k + 6 \\ &= 4 \cdot \Big(4\ T(4^{k-2}) + 4^{k-1} + 6\Big) + 4^k + 6 \\ &= 4 \cdot \Big(4 \cdot \Big(4\ T(4^{k-3}) + 4^{k-2} + 6\Big) + 4^{k-1} + 6\Big) + 4^k + 6 \\ &= 4^k\ T(4^0) + k \cdot 4^k + 6 \cdot (4^0 + 4^1 + \cdots + 4^{k-1}) \\ &= 4^k + k \cdot 4^k + 6 \cdot \frac{4^k - 1}{4 - 1} \\ &= 4^k + k \cdot 4^k + 2 \cdot 4^k - 1 \\ &= n + n \log_4(n) + 2n - 2 \\ &= 3n + n \log_4(n) - 2 \end{aligned}$$

Wir beweisen nun unsere Annahme durch vollständige Induktion über k.

Induktionsverankerung ($k = 0$): Es gilt $T(4^0) = T(1) = 1 = 3 \cdot 1 + 0 - 2$.

Induktionsannahme: Für ein $k \in \mathbb{N}_0$ sei $T(4^k) = 3 \cdot 4^k + k \cdot 4^k - 2$.

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

$$\begin{aligned} T(4^{k+1}) &\overset{\text{Rek.}}{=} 4 \cdot T(4^k) + 4^{k+1} + 6 \\ &\overset{\text{Ind.-Ann.}}{=} 4 \cdot (3 \cdot 4^k + k \cdot 4^k - 2) + 4^{k+1} + 6 \\ &= 3 \cdot 4 \cdot 4^k + 4k \cdot 4^k - 8 + 4^{k+1} + 6 \\ &= 3 \cdot 4^{k+1} + (k+1) \cdot 4^{k+1} - 2 \\ &= 3n + n \log_4(n) - 2. \end{aligned}$$

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·i)
2         for(int j = n; j >= 0; j = j−1)
3                 ;
```

Lösung: Die Schleife in Schritt 1 wird $\log(n)$ Mal durchlaufen. Die Schleife in Schritt 2 wird n Mal durchlaufen. Die Gesamtlaufzeit ist daher in $\Theta(n \log n)$.

1 P k) 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 = i+2) {
2         for(int j = 1; 2*j <= i; j = j + 1)
3                 ;
4         int k = n;
5         while(k > 2)
6                 k = k / 2;
7 }
```

Lösung: Die Schleife in Schritt 1 wird $\Theta(n)$ Mal durchlaufen. Die erste innere Schleife in Schritt 2 wird ebenfalls $\Theta(n)$ Mal durchlaufen. Die zweite innere Schleife (Schritt 5 und 6) wird $\Theta(\log n)$ Mal durchlaufen. Dadurch ergibt sich eine Gesamtlaufzeit in $\Theta(n^2)$.

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

Lösung: Die Schleife in Schritt 1 wird $\Theta(\sqrt{(n)})$ Mal durchlaufen. Die Schleife in Schritt 2 wird $\Theta(n)$ Mal durchlaufen. Die Gesamtlaufzeit ist daher in $\Theta(n\sqrt{n})$.

1 P m) Geben Sie für die untenstehenden Funktionen eine **Reihenfolge** an, sodass 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.

$$\log(n^{4n}),\ \binom{n}{5},\ \sqrt{16^n},\ \frac{n}{\log(n)},\ n+n^3,\ n!,\ 1052^{24}$$

Lösung: Es sind $1052^{24} \in \mathcal{O}(1)$ und $\log(n^{4n}) \in \mathcal{O}(n\log(n))$ sowie $\binom{n}{5} \in \Theta(n^5)$. Die einzig mögliche Reihenfolge ist

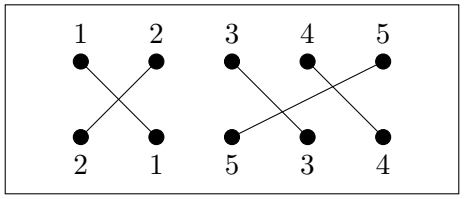
$$1052^{24},\ \frac{n}{\log(n)},\ \log(n^{4n}),\ n+n^3,\ \binom{n}{5},\ \sqrt{16^n},\ n!$$

2 P n) 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.

Ein natürlicher Suchbaum ist balanciert.	☐ Wahr	☒ Falsch
Sei $G = (V, E)$ ein zusammenhängender Graph. Jeder Spannbaum von G hat genau $\lvert V\rvert - 1$ Kanten.	☒ Wahr	☐ Falsch
Sortieren durch Auswahl und Sortieren durch Einfügen sind zwei in-situ *Sortierverfahren.*	☒ Wahr	☐ Falsch
Werden in einen leeren binären Suchbaum ausschliesslich Schlüssel eingefügt (und keine entfernt), so entspricht die Preorder-Reihenfolge genau der Einfüge-Reihenfolge.	☐ Wahr	☒ Falsch

Aufgabe 2.

Problemdefinition. Auf einer Platine werden $2n$ Bauelemente in zwei untereinanderliegenden Reihen mit je n Bauelementen angeordnet. Jedes Bauelement der oberen Reihe wird genau einem Bauelement der unteren Reihe zugeordnet und umgekehrt. Die Paare von Bauelementen sollen mit geradlinigen Leiterbahnen verbunden werden. Geben Sie einen Algorithmus an, der berechnet, wie viele Bauelemente höchstens verbunden werden können, wenn sich zwei Leiterbahnen nicht überkreuzen dürfen.



Beispiel: Im oben dargestellten Beispiel sind 10 Bauteile auf einer Platine angeordnet. Es sollen jeweils die Bauelemente mit gleicher Nummerierung verbunden werden. In diesem Beispiel gibt es zwei Lösungen, wobei jeweils drei Paare verbunden werden können:

(i) Bauelemente mit den Nummern $1, 3$ und 4. (ii) Bauelemente mit den Nummern $2, 3$ und 4.

7 P a) Geben Sie einen möglichst effizienten Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die maximale Anzahl von möglichen Leiterbahnen, welche sich paarweise nicht kreuzen, berechnet. Gehen Sie in Ihrer Lösung auf die folgenden Aspekte ein:

1) Was ist die Bedeutung eines Tabelleneintrags und welche Grösse hat die DP-Tabelle?
2) Wie berechnet sich ein Tabelleneintrag?
3) Beschreiben Sie, in welcher Reihenfolge die Berechnungsschritte ausgeführt werden.
4) Wie kann aus der DP-Tabelle ausgelesen werden, wie viele Bauelemente höchstens verbunden werden können?

Hinweis: Der triviale Algorithmus, der einfach alle überhaupt möglichen Lösungen inspiziert, gibt wegen mangelnder Effizienz **keine** Punkte.

Lösung: Wir bezeichnen die Nummerierung der unteren n Bauelemente als Zahlenfolge $X = X[1], \ldots, X[n]$. Eine Menge von Bauelement-Paaren, die durch gerade und sich nicht kreuzende Leiterbahnen verbunden werden kann, entspricht einer aufsteigenden Teilfolge von X. Gesucht ist daher eine längste aufsteigende Teilfolge von X.

Definition der DP-Tabelle: Wir verwenden eine eindimensionale Tabelle T mit n Einträgen. Der Eintrag $T[i]$ enthält das kleinste Element $X[j]$ für $j \in \{1, \ldots, i\}$ sodass $X[j]$ am Ende einer aufsteigenden Teilfolge der Länge i steht.

Berechnung eines Eintrags: Alle Einträge der Tabelle werden mit ∞ initialisiert. Im i-ten Berechnungsschritt wird der grösste Index $j < i$ gesucht, sodass $T[j] < X[i]$. Da die Einträge in T zu jedem Zeitpunkt eine monotone Folge bilden, kann dieser Eintrag effizient durch eine binäre Suche gefunden werden. Falls kein solcher Index existiert, setzten wir $j = 0$. Nun wird der Eintrag $T[j+1]$ aktualisiert:

$$T[j+1] = \min(T[j+1], X[i]).$$

Berechnungsreihenfolge: Die Einträge der Folge X werden mit aufsteigendem $i = 1, \ldots, n$ betrachtet. In jedem Berechnungsschritt wird genau ein Eintrag der Tabelle geändert.

Auslesen der Lösung: Es gibt genau dann (mindestens) eine aufsteigende Teilfolge der Länge i, wenn $T[i] \neq \infty$. Daher wird der grösste Index k mit $T[k] \neq \infty$ gesucht.

2 P b) Beschreiben Sie detailliert, wie der Algorithmus modifiziert werden muss, damit zusätzlich eine möglichst grosse Menge von sich paarweise nicht überkreuzenden Leiterbahnen ausgegeben wird.

Lösung: Um zusätzlich eine möglichst grosse Menge von Bauelement-Paaren, die kreuzungsfrei verbunden werden können, zu berechnen, wird zusätzlich für jedes Element der Folge X ein Vorgänger gespeichert. Dazu kann eine zusätzliche Tabelle P mit n Einträgen verwendet werden. Wird ein Eintrag $T[j+1] = X[i]$ gesetzt, muss nun zusätzlich $P[i] = T[j]$ gesetzt werden. Durch Rückverfolgung kann nun eine längste aufsteigende Teilfolge in Laufzeit $O(k)$, wobei k die Länge einer längsten aufsteigenden Teilfolge ist, berechnet werden, indem ausgehend von $T[k]$ die $k-1$ Vorgänger $P[T[k]], P[P[T[k]]], \ldots$ ausgegeben werden.

2 P c) Geben Sie die Laufzeit der in a) und b) entwickelten Verfahren an und begründen Sie Ihre Antwort.

Lösung: Jeder Berechnungsschritt in Teilaufgabe a) benötigt wegen der binären Suche $\mathcal{O}(\log(n))$ Zeit. Daher ist die Gesamtlaufzeit für Teilaufgabe a) $\mathcal{O}(n \log(n))$. In Teilaufgabe b) wird in jedem Berechnungsschritt in konstanter Zeit die zusätzliche Tabelle P aktualisiert. Die Rückverfolgung benötigt $\mathcal{O}(n)$ Zeit, wobei $k \in \mathcal{O}(n)$ die Länge einer längsten aufsteigenden Teilfolge ist. Die Gesamtlaufzeit für beide Teilaufgaben liegt daher in $\mathcal{O}(n \log(n))$.

Aufgabe 3.

6 P a) Gegeben seien n Assistenten und m Übungsgruppen, die zu unterschiedlichen Zeiten stattfinden. Jeder Assistent gibt an, welche Übungsgruppen er übernehmen kann. Ein Assistent darf höchstens eine Übungsgruppe betreuen und für jede Gruppe wird ein Assistent benötigt. Entwerfen Sie einen Algorithmus, der berechnet, wie viele Übungsgruppen höchstens betreut werden können.

Beispiel: Es soll eine Zuteilung für $n = m = 3$ Assistenten und Übungsgruppen gefunden werden (siehe Tabelle). In diesem Beispiel können alle Gruppen betreut werden: Assistent 1 übernimmt Gruppe 1, Assistent 2 übernimmt Gruppe 3 und Assistent 3 übernimmt Gruppe 2.

Assistent	*mögliche Gruppen*
Assistent 1	Gruppe 1 oder 3
Assistent 2	Gruppe 3
Assistent 3	Gruppe 1 oder 2

Modellieren Sie das o.g. Problem als Flussproblem. Beschreiben Sie dazu die Konstruktion eines geeigneten Netzes $N = (V, E, c)$ mit der Knotenmenge V sowie der Kantenmenge E, und geben Sie an, welche Kapazitäten c die Kanten besitzen sollen. Nennen Sie einen effizienten Algorithmus zur Berechnung eines maximalen Flusses in N. Welche Laufzeit hat dieser Algorithmus in Abhängigkeit von n und m? Wie kann am Ergebnis abgelesen werden, wie viele Übungsgruppen betreut werden können?

Lösung: Wir bezeichnen die Menge der Assistenten mit A und die Menge der Übungsgruppen mit G. Die Knoten des Netzes N sind $V = A \cup G \cup \{s, t\}$, wobei s und t die Quelle und die Senke des Netzes sind. Die Quelle s wird durch gerichtete Kanten (s, a) mit jedem Assistenten $a \in A$ verbunden. Jede Übungsgruppe $g \in G$ wird durch eine gerichtete Kante (g, t) mit der Senke t verbunden. Schliesslich fügen wir eine gerichtete Kante (a, g) genau dann in das Netzwerk ein, wenn Assistent a die Übungsgruppe g übernehmen kann. Jede Kante unseres Netzwerks hat Kapazität 1.

Wir berechnen einen maximalen Fluss von s nach t in N. Der Wert eines solchen maximalen Flusses ist durch $\min(n, m)$ nach oben begrenzt. Daher ist der Algorithmus von Ford und Fulkerson in diesem Fall effizient. Die Laufzeit ist in $\mathcal{O}(\min(n, m) \cdot n \cdot m)$, da das Netzwerk $n + m + 2$ Knoten und $\mathcal{O}(n \cdot m)$ Kanten hat. Der Wert des maximalen Flusses von s nach t in N gibt an, wie viele Übungsgruppen betreut werden können.

2 P b) Nun müssen Räume für die Übungsgruppen reserviert werden. Es stehen k Räume zur Verfügung. Für jede Übungsgruppe ist gegeben, welche Räume infrage kommen. Berechnen Sie eine Zuteilung möglichst vieler Assistenten zu Übungsgruppen und Räumen, sodass kein Raum doppelt belegt ist.

Beispiel: Gesucht ist eine Zuteilung für $n = 3$ Assistenten, $m = 3$ Übungsgruppen und $k = 4$ Räume:

Assistent	*mögliche Gruppen*
Assistent 1	Gruppe 1 und 3
Assistent 2	Gruppe 2
Assistent 3	Gruppe 2

Übungsgruppe	*verfügbare Räume*
Gruppe 1	Raum 1
Gruppe 2	Raum 1
Gruppe 3	Raum 2, 3 oder 4

Es können nicht alle Gruppen angeboten werden: Beispielsweise kann Assistent 1 Gruppe 3 in Raum 1 betreuen und Assistent 2 Gruppe 3 in Raum 4. Gruppe 1 kann in diesem Fall nicht zugeteilt werden, da Raum 1 bereits von Gruppe 3 belegt ist.

Modellieren Sie dieses erweiterte Zuteilungsproblem als Flussproblem und definieren Sie ein neues Netzwerk $N' = (V', E', c')$. Beschreiben Sie die Knotenmenge V', die Kantenmenge E', sowie die Kapazitäten c'. Wie kann am Ergebnis abgelesen werden, wie viele Übungsgruppen betreut werden können?

Lösung: Wir bezeichnen die Menge der Räume mit R und definieren ein Netzwerk N' mit Knotenmenge $V' = A \cup G \cup R \cup \{s, t\}$, wobei s die Quelle und t die Senke des Netzwerkes sind. Die Quelle s wird durch gerichtete Kanten (s, a) mit jedem Assistenten $a \in A$ verbunden und jeder Raum $r \in R$ durch eine gerichtete Kante (r, t) mit der Senke t. Weiters verbinden wir einen Assistenten $a \in A$ mit einer Gruppe $g \in G$ genau dann durch eine gerichtete Kante (a, g), wenn Assistent a Übungsgruppe g übernehmen kann. Eine Übungsgruppe $g \in G$ wird mit einem Raum $r \in R$ genau dann durch eine gerichtete Kante (g, r) verbunden, wenn Raum r für Übungsgruppe g verfügbar ist. Wie in Teilaufgabe a) sind die Kapazitäten aller Kanten 1 und der Wert eines maximalen Flusses von s nach t gibt an, wie viele Übungsgruppen betreut werden können.

Aufgabe 4.

Eine Menge von n sich nicht schneidenden Liniensegmenten repräsentiert einen Querschnitt von n Dächern. Jedes Dach sei als Liniensegment durch seine beiden Endpunkte mit Koordinaten (x_l, y_l) und (x_r, y_r) gegeben. Alle x-Koordinaten und alle y-Koordinaten sind paarweise verschieden. Es regnet überall gleichmässig. Der Regen fällt senkrecht von oben auf gewisse Dächer (diejenigen die nicht von anderen Dächern geschützt sind) und rinnt dann jedes dieser Dächer hinunter bis zum unteren Ende. Von dort tropft er weiter senkrecht nach unten auf ein nächstes Dach oder auf den Boden. Je nach Anordnung der Dächer werden manche der Dächer etwas nass, andere bleiben ganz und gar trocken. Die ganz trockenen Dächer sollen ermittelt werden.

9 P a) Alle Dächer sind nach rechts geneigt, d.h. für jedes Dach gilt $y_r < y_l$. Der rechte Rand ist also der untere Rand für jedes Dach. Entwerfen Sie einen möglichst effizienten Scanline-Algorithmus, der berechnet, welche Dächer trocken bleiben. Gehen Sie in Ihrer Lösung insbesondere auf die folgenden Aspekte ein:

1) In welche Richtung verläuft die Scanline, und was sind die Haltepunkte?

2) Welche Objekte muss die Scanline-Datenstruktur verwalten, und was ist eine angemessene Datenstruktur?

3) Was passiert, wenn die Scanline auf einen neuen Haltepunkt trifft?

4) Wie kann die Menge der trockenen Dächer bestimmt werden?

5) Welche Laufzeit in Abhängigkeit von n hat Ihr Algorithmus? Begründen Sie Ihre Antwort.

Lösung:

1) Wir verwenden eine vertikale Scanline, die von links nach rechts verläuft. Für jedes Liniensegment definieren wir zwei Haltepunkte x_l und x_r (die Endpunkte der Liniensegmente).

2) Wir können als Scanline-Datenstruktur einen AVL-Baum verwenden, der Liniensegmente nach y-Koordinate sortiert speichert. Das ist möglich, da die Liniensegmente sich nicht schneiden und sich die relative Reihenfolge zweier Dächer nicht ändert. Die Geradengleichung eines Liniensegments wird als Schlüssel gespeichert und bei einem Schlüsselvergleich an der aktuellen x-Koordinate der Scanline ausgewertet. Für jedes Liniensegment speichern wir ausserdem, ob das Dach "nass" oder "vorerst trocken" ist. Zu Beginn des Algorithmus ist jedes Dach "vorerst trocken".

3) Falls die Scanline auf einen Haltepunkt x_l eines Liniensegmentes trifft, d.h. auf das linke Ende, wird das Liniensegment in die Datenstruktur eingefügt. Trifft die Scanline auf das rechte Ende x_r eines Liniensegmentes d, so wird das direkt darunterliegende Liniensegment mithilfe des AVL-Baums ermittelt. Das Liniensegment d wird aus der Datenstruktur entfernt und das darunterliegende Liniensegment auf "nass" gesetzt, wenn d als "nass" gespeichert wurde. Zudem wird nach jeder Einfüge- und Lösch-Operation das Liniensegment mit der höchsten y-Koordinate ermittelt und auf "nass" gesetzt.

4) Sobald das letzte Liniensegment aus der Datenstruktur entfernt wurde, ist der Zustand jedes Liniensegmentes endgültig ermittelt worden: Jedes Liniensegment, das als "vorerst trocken" gekennzeichnet ist, wird ausgegeben.

5) Die Haltepunkte müssen zuerst in Zeit $\mathcal{O}(n \cdot \log(n))$ sortiert werden. Es werden $\mathcal{O}(n)$ Haltepunkte besucht. Ein Liniensegment kann in Zeit $\mathcal{O}(\log n)$ zur Datenstruktur hinzugefügt, beziehungsweise aus der Datenstruktur gelöscht werden. In der selben Zeit kann auch ein Nachfolger und das oberste Dach ermittelt werden. Somit ist auch die Gesamtlaufzeit in $\mathcal{O}(n \log n)$.

3 P b) Nun betrachten wir den Fall, bei dem Dächer beliebig geneigt sein können. Für jedes gegebene Dach gilt also $y_l \neq y_r$. Beschreiben Sie, wie Ihr Algorithmus modifiziert werden muss, um zu berechnen, welche Dächer trocken bleiben. Welche Laufzeit in Abhängigkeit von n hat Ihr Algorithmus? Begründen Sie Ihre Antwort.

Lösung: Anstatt des Zustandes eines Liniensegments speichern wir nun seinen direkten Nachfolger und ob es zu irgendeinem Zeitpunkt direkt vom Regen getroffen wird (also nicht durch ein anderes Dach geschützt wird). Die Richtung der Scanline, die Haltepunkte, sowie die Datenstruktur sind wie in Teilaufgabe a) definiert. Wir unterscheiden jedoch nicht mehr nur rechte und linke Haltepunkte, sondern zusätzlich obere und untere Haltepunkte (gemäß der Neigung der Dächer). Wie in Teilaufgabe a) wird das Liniensegment beim linken Haltepunkt in die Datenstruktur eingefügt und beim rechten Haltepunkt wieder entfernt. Bei einem unteren Haltepunkt wird zusätzlich das direkt darunterliegende Liniensegment als direkter Nachfolger gespeichert. Wiederum wird nach jeder Operation das oberste Liniensegment ermittelt und gekennzeichnet.

Sobald das letzte Liniensegment aus der Datenstruktur entfernt wurde, können wir die trockenen Dächer mithilfe der Nachfolger-Beziehungen der Liniensegmente ermittelt werden. Alle obersten Liniensegmente wurden bereits als "nass" gekennzeichnet. Für jedes übrige Liniensegment gilt, das es genau dann "nass" ist, wenn mindestens ein Vorgänger ebenfalls "nass" ist. Daher kennzeichnen wir die Nachfolger eines als "nass" gekennzeichneten Daches werden ebenfalls als "nass" und wiederholen diesen Vorgang für die neu als "nass" gekennzeichneten Dächer. Der Zustand aller Liniensegmente in Zeit $\mathcal{O}(n)$ berechnet werden, da jedes Dach höchstens einen Nachfolger hat und die Nachfolger eines Daches nur betrachtet werden, wenn es selbst als "nass" gekennzeichnet wird. Die Gesamtlaufzeit des Algorithmus entspricht der Laufzeit in Teilaufgabe *a*).

Beispiele: In den folgenden Abbildungen wird Regenwasser durch gestrichelte Linien symbolisiert. Die linke Abbildung betrifft Teilaufgabe a). Die Dächer d_4 und d_5 bleiben trocken. In der rechten Abbildung für Teilaufgabe b) bleibt lediglich d_4 trocken.