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
Thomas Tschager

Beispiellösung zur Prüfung
Datenstrukturen und Algorithmen
D-INFK

21. Januar 2015

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	14	13	11	10	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) Führen Sie auf dem folgenden Array zwei Iterationen des Sortieralgorithmus *Sortieren durch Auswahl* aus. Das zu sortierende Array ist durch vorherige Iterationen bereits bis zum Doppelstrich sortiert worden.

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

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

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

1 P b) Fügen Sie die Schlüssel 8, 10, 15, 9, 17 in dieser Reihenfolge in die untenstehende Hashtabelle ein. Benutzen Sie offenes Hashing mit der Hashfunktion $h(k) = k \bmod 7$ und lösen Sie Kollisionen mittels quadratischem Sondieren auf.

Lösung: Bei quadratischem Sondieren können folgende Funktionen als Sondierungsfunktionen verwendet werden: $s(j,k) = (-1)^j \cdot \lceil j/2 \rceil^2$ oder $s(j,k) = (-1)^{j+1} \cdot \lceil j/2 \rceil^2$. Die Position in der Hashtabelle wird mit $h(k) - s(j,k) \bmod 7$ berechnet. Mit diesen Wahlen erhalten wir die Hashtabellen

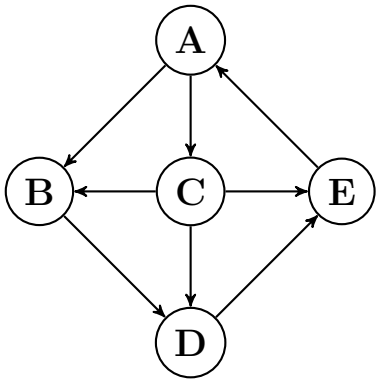
	8	15	10	17		9
0	1	2	3	4	5	6

falls $s(j,k) = (-1)^j \cdot \lceil j/2 \rceil^2$ verwendet wird, bzw.

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

falls $s(j,k) = (-1)^{j+1} \cdot \lceil j/2 \rceil^2$ verwendet wird.

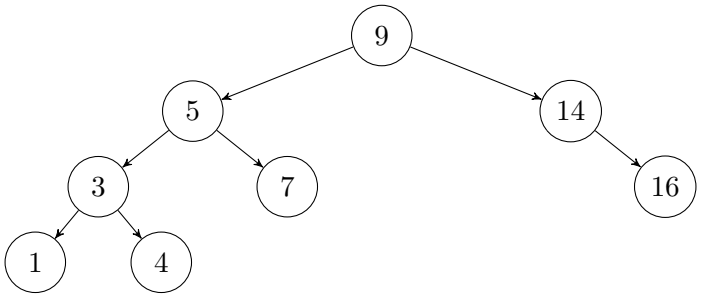
1 P c) Geben Sie jeweils eine Reihenfolge an, in der die Knoten des folgenden Graphen von einer Breitensuche (BFS) bzw. Tiefensuche (DFS) mit Startknoten A besucht werden, wenn die Nachbarn eines Knotens in alphabetischer Reihenfolge abgearbeitet werden.



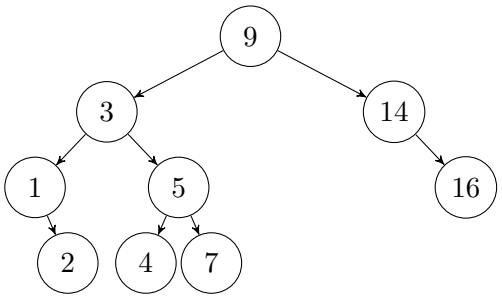
BFS: A, B, C, D, E

DFS: A, B, D, E, C

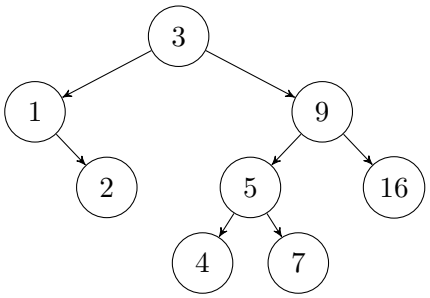
1 P d) Fügen Sie in den folgenden AVL-Baum *zuerst* den Schlüssel 2 ein und entfernen Sie *danach* den Schlüssel 14.



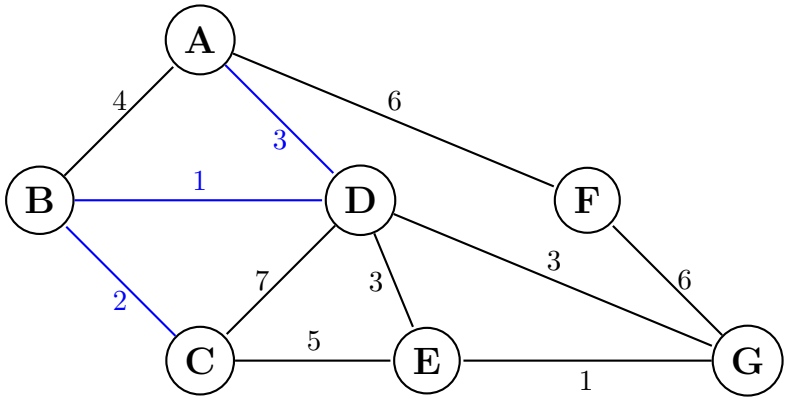
Nach Einfügen von 2:



Nach Löschen von 14:

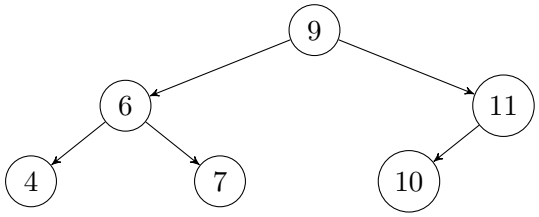


1 P e) Markieren Sie in der untenstehenden Abbildung die ersten *drei* Kanten, die der Algorithmus von Jarník, Prim und Dijkstra *ausgehend von Knoten A* in den minimalen Spannbaum aufnimmt.



1 P f) Zeichnen Sie denjenigen binären Suchbaum, bei dem in Postorder-Reihenfolge die Schlüssel 4, 7, 6, 10, 11, 9 angetroffen werden.

Lösung: Man erhält einen binären Suchbaum für Schlüssel mit gegebener Postorder-Reihenfolge, indem man die Schlüssel in umgekehrter Reihenfolge in einen leeren Baum einfügt.



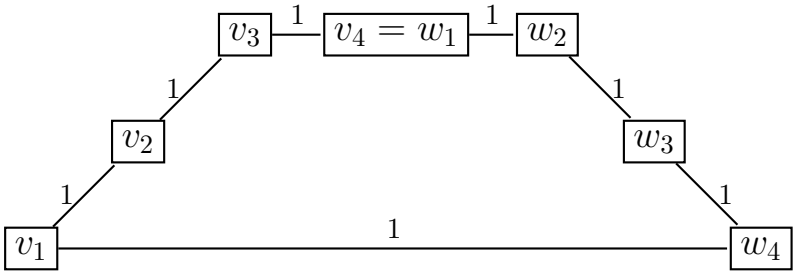
2 P g) Gegeben sei ein gewichteter Graph G. Zeigen oder widerlegen Sie die folgenden Aussagen:

1. Wenn $P = \langle v_1, \ldots, v_k \rangle$ und $Q = \langle w_1, \ldots, w_l \rangle$ kürzeste Pfade mit $v_k = w_1$ sind, dann ist $\langle v_1, \ldots, v_k = w_1, \ldots, w_l \rangle$ ein kürzester Pfad von v_1 nach w_l.

2. Sei P ein kürzester Pfad von u nach v und sei w ein innerer Knoten von P. Der Teilpfad von P von u nach w ist ein kürzester Pfad von u nach w in G. Ebenso ist der Teilpfad von w nach v ein kürzester Pfad in G.

Lösung:

1. Diese Aussage ist falsch. Ein Gegenbeispiel mit $k = 4$ und $l = 4$:



2. Die Aussage ist wahr. Angenommen der Teilpfad von P von u nach w wäre kein kürzester Pfad. Dann existierte ein kürzerer Pfad von u nach w. Dieser Pfad bildet zusammen mit dem Teilpfad von P von w nach v einen kürzeren Pfad von u nach v. Dies ist ein Widerspruch, da P ein kürzester Pfad von u nach w ist.

1 P h) 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.

- $\binom{n}{4}$
- $\log^2(n)$
- $n \cdot \sqrt{n}$
- $n!$
- $\log(n^5)$
- 7^{13}
- $\log(n^n)$
- $\sqrt{6^n}$

Lösung: Es gilt $\log(n^5) = 5\log(n) \in \Theta(\log n)$ und damit $\log(n^5) \in \mathcal{O}(\log^2(n))$. Weiterhin gilt $\sqrt{6^n} < n!$ für $n \geq 5$ und damit $\sqrt{6^n} \in \mathcal{O}(n!)$. Außerdem gilt $\binom{n}{4} \in \Theta(n^4)$. Die einzige gültige Reihenfolge ist daher

$$7^{13}, \log(n^5), \log^2(n), \log(n^n), n \cdot \sqrt{n}, \binom{n}{4}, \sqrt{6^n}, n!.$$

3 P i) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 15 + 4T(n/4) & n > 1 \\ 1 & n = 1 \end{cases}$$

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

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

$$\begin{aligned} T(n) &= 15 + 4T(n/4) \\ &= 15 + 4\big(15 + 4T(n/4^2)\big) = 15 + 4 \cdot 15 + 4^2 T(n/4^2) \\ &= 15 + 4 \cdot 15 + 4^2\big(15 + 4T(n/4^3)\big) = 15 + 4 \cdot 15 + 4^2 \cdot 15 + 4^3 T(n/4^3) \\ &= \ldots \overset{!}{=} \left(15 \sum_{i=0}^{k-1} 4^i\right) + 4^k \cdot T(1) = 15 \cdot \frac{4^k - 1}{4 - 1} + 4^k \cdot 1 = 5(4^k - 1) + 4^k = 6n - 5. \end{aligned}$$

Wir beweisen nun unsere Annahme durch vollständige Induktion über k.

Induktionsverankerung ($k = 0$): Es gilt $T(4^0) = T(1) = 1 = 6 \cdot 4^0 - 5$.

Induktionsannahme: Für ein $k \in \mathbb{N}_0$ sei $T(4^k) = 6 \cdot 4^k - 5$.

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

$$\begin{aligned} T(4^{k+1}) = 15 + 4 \cdot T(4^k) &\overset{\text{Ind.-Ann.}}{=} 15 + 4 \cdot (6 \cdot 4^k - 5) \\ &= 6 \cdot 4^{k+1} + 15 - 20 = 6 \cdot 4^{k+1} - 5 = 6n - 5. \end{aligned}$$

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

Lösung: Die äussere Schleife wird $\Theta(n)$ Mal ausgeführt. Die erste innere Schleife (Zeilen 2 und 3) wird $\Theta(\log n)$ Mal durchlaufen und die zweite innere Schleife (Zeilen 5 und 6) $\Theta(\sqrt{n})$ Mal. Insgesamt ist die Laufzeit daher $\Theta(n\sqrt{n})$.

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 = n; i > 0; i -= 1) {
2         for(int j = 0; j < i; j += 1) {
3                 ;
4         }
5 }
```

Lösung: Die äussere Schleife wird n Mal ausgeführt und die innere Schleife in jedem Durchlauf höchstens n Mal. Somit ist die Laufzeit durch $\mathcal{O}(n^2)$ nach oben beschränkt. Für $i \geq n/2$ wird die innere Schleife mindestens $n/2$ Mal durchlaufen. Insgesamt ist die Laufzeit daher $\Theta(n^2)$.

Aufgabe 2.

Eine Firma hat den Auftrag bekommen, ein Kabel der Länge mindestens L zu produzieren. Da es im Lager viele bereits früher produzierte Kabelstücke gibt, soll das gewünschte Kabel aus diesen Teilstücken zusammengesetzt werden. Konkret gibt es im Lager n Kabelstücke. Kabelstück i hat Länge l_i. Werden zwei Kabelstücke mit Längen l_i und l_j zusammengesetzt, entsteht ein Kabel der Länge $l_i + l_j$. Um das entstehende Kabel nicht unnötig lang zu machen, soll es unter allen Möglichkeiten, ein Kabel der Länge $\geq L$ herzustellen, minimale Länge haben. Sie dürfen annehmen, dass der Lagerbestand ausreichend gross ist, d.h., dass die Summe der Längen aller Kabelstücke mindestens L beträgt.

9 P a) Geben Sie einen Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die minimale Länge eines Kabels berechnet, sodass dieses Kabel aus geeigneten Kabelstücken aus $\{1, \ldots, n\}$ zusammengesetzt werden kann und mindestens Länge L hat.

Lösung:

Definition der DP-Tabelle: Wir verwenden eine $(n+1) \times (1 + \sum_{i=1}^{n} l_i)$-Tabelle T mit Einträgen aus $\{\text{true}, \text{false}\}$. Für $i \in \{0, \ldots, n\}$ und $l \in \{0, \ldots, \sum_{i=1}^{n} l_i\}$ habe $T[i, l]$ genau dann den Wert true, wenn es eine Möglichkeit gibt, mit einer Auswahl geeigneter Kabelstücke aus $\{1, \ldots, i\}$ ein Kabel der Länge l zu erzeugen.

Berechnung eines Eintrags: Wir unterscheiden drei Fälle.

1. Fall: $i = 0$. Dies entspricht der Initialisierung, wenn keine Kabelstücke benutzt werden dürfen. Die einzig erreichbare Kabellänge ist 0, also setzen wir $T[0, 0] \leftarrow$ true und $T[0, l] \leftarrow$ false für jedes $l \in \{1, \ldots, \sum_{i=1}^{n} l_i\}$.

2. Fall: $i \in \{1, \ldots, n\}$, $l \in \{0, \ldots, l_i - 1\}$. Jedes zusammengesetzte Kabel mit Länge $l < l_i$ benutzt das Kabelstück i nicht, weil dieses zu lang ist. Dann liegen alle benutzbaren Kabelstücke aus $\{1, \ldots, i\}$ in $\{1, \ldots, i-1\}$, und kann ein Kabel der Länge l aus geeigneten Kabelstücken aus $\{1, \ldots, i\}$ zusammengesetzt werden, dann kann dies auch mit geeigneten Kabelstücken aus $\{1, \ldots, i-1\}$ geschehen. Deshalb setzen wir $T[i, l] \leftarrow T[i-1, l]$ für jedes $l \in \{0, \ldots, l_i - 1\}$.

3. Fall: $i \in \{1, \ldots, n\}$, $l \in \{l_i, \ldots, \sum_{i=1}^{n} l_i\}$. Mit den Kabelstücken $\{1, \ldots, i\}$ können wir genau dann ein Kabel mit Gesamtlänge l herstellen, wenn

- das Kabelstück i benutzt wird und mit den Kabelstücken $\{1, \ldots, i-1\}$ ein Kabel mit Gesamtlänge $l - l_i$ hergestellt werden kann, oder wenn
- das Kabelstück i nicht benutzt wird und mit den Kabelstücken $\{1, \ldots, i-1\}$ ein Kabel mit Gesamtlänge l hergestellt werden kann.

Folglich wird $T[i, l] \leftarrow T[i-1, l] \vee T[i-1, l - l_i]$ gesetzt.

Berechnungsreihenfolge: Wir berechnen die Einträge $T[i, l]$ für aufsteigende $i \in \{0, \ldots, n\}$. Für gleiche i berechnen wir die Einträge für aufsteigende $l \in \{0, \ldots, \sum_{i=1}^{n} l_i\}$.

Auslesen der Lösung: Wir setzen $l \leftarrow L$. Falls $T[n, l]$ nicht 'true' ist, inkrementieren wir l um 1, bis dies der Fall ist. Die gesuchte Länge ist l.

2 P b) Beschreiben Sie detailliert, wie aus der DP-Tabelle abgelesen werden kann, welches Kabelstück in einer optimalen Lösung benutzt wird.

Lösung: Sei l die vom Algorithmus aus a) berechnete minimale Kabellänge. Wir setzen $i \leftarrow n$ und wiederholen die folgenden Schritte, bis $i = 0$ ist. Betrachte den Eintrag $T[i, l]$.

- Ist $l \geq l_i$ und $T[i, l] = T[i-1, l-l_i]$, dann gibt es eine optimale Lösung, die das Kabelstück i benutzt, also geben wir i aus. Wir setzen $l \leftarrow l - l_i$ und $i \leftarrow i - 1$ und fahren analog fort.
- Ansonsten kann das Kabelstück i nicht verwendet werden, also wird nichts ausgegeben und l bleibt unverändert. Wir setzen aber $i \leftarrow i - 1$ und fahren analog fort.

2 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 Zeit zur Berechnung eines Eintrags ist $\Theta(1)$. Da die Tabelle $\Theta(n \sum_{i=1}^{n} l_i)$ Einträge besitzt, beträgt die Laufzeit des Algorithmus in a) $\Theta(n \sum_{i=1}^{n} l_i)$. Der Algorithmus in b) braucht zum Prüfen der zwei Fälle nur konstante Zeit, und da i von n bis 1 (bzw. 0) läuft, ist die Laufzeit $\Theta(n)$. Als Gesamtlaufzeit ergibt sich also $\Theta((n \sum_{i=1}^{n} l_i) + n) = \Theta(n \sum_{i=1}^{n} l_i)$. Diese ist pseudopolynomiell, weil $\sum_{i=1}^{n} l_i$ Teil der Eingabe ist, aber durch nur $\lceil \log \left(\sum_{i=1}^{n} l_i + 1 \right) \rceil$ Bits dargestellt werden kann.

Aufgabe 3.

In einem Krankenhaus soll bestimmt werden, welcher Arzt an welchem Tag arbeitet. Diese Aufgabe befasst sich mit der Arbeitszeitplanung zu Ferientagen. Sei T die Menge aller Ferientage. Es gibt k Ferienzeiten $\{1, \ldots, k\}$, die aus einem oder mehreren zusammenhängenden Tagen $T_j \subseteq T$ bestehen (für $j \in \{1, \ldots, k\}$). Jeder Tag in T kommt in genau einer Ferienzeit vor, d.h. insbesondere ist $T_i \cap T_j = \emptyset$ für $i \neq j$. Im Krankenhaus arbeiten n Ärzte. Jeder Arzt i gibt eine Menge von Tagen $S_i \subseteq T$ an, an denen er anwesend sein kann. Die Aufgabe besteht nun darin, einen Einsatzplan zu entwerfen, der die Ferientage T so unter den Ärzten verteilt, dass an jedem Tag genau ein Arzt anwesend ist. Ausserdem soll kein Arzt an mehr als $C \in \mathbb{N}$ Ferientagen insgesamt arbeiten müssen. Die Aufgabe besteht nun darin, effizient zu entscheiden, ob für ein gegebenes C ein geeigneter Einsatzplan existiert, und falls ja, effient zu berechnen, wie dieser aussieht.

4 P a) 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 die Kanten besitzen sollen. Wie kann aus dem Wert eines maximalen Flusses abgelesen werden, ob ein geeigneter Einsatzplan existiert oder nicht?

Lösung: Für jeden Arzt i erzeugen wir einen Knoten v_i^A, für jeden Ferientag $t \in T$ erzeugen wir einen Knoten v_t. Zusätzlich werden eine Quelle q und eine Senke s erzeugt. Das Netz N besitzt die folgenden Kanten:

- Für jeden Arzt $i \in \{1, \ldots, n\}$ eine Kante (q, v_i^A) mit Kapazität C,
- Für jeden Arzt $i \in \{1, \ldots, n\}$ und jeden Tag $t \in S_i$, an dem der Arzt anwesend sein könnte, eine Kante (v_i^A, v_t) mit Kapazität 1,
- Für jeden Ferientag $t \in T$ eine Kante (v_t, s) mit Kapazität 1.

Wir erzeugen also ein Netz $N = (V, E, c)$ mit

$$V := \{v_i^A \mid i \in \{1, \ldots, n\}\} \cup \{v_t \mid t \in T\} \cup \{q\} \cup \{s\} \tag{1}$$

$$E := \{(s, v_i^A) \mid i \in \{1, \ldots, n\}\} \cup \{(v_i^A, v_t) \mid i \in \{1, \ldots, n\} \wedge t \in S_i\} \cup \{(v_t, s) \mid t \in T\} \tag{2}$$

$$c((s, v_i^A)) := C \text{ für alle } i \in \{1, \ldots, n\} \tag{3}$$

$$c((v_i^A, v_t)) := 1 \text{ für alle } i \in \{1, \ldots, n\}, t \in S_i \tag{4}$$

$$c((v_t, s)) := 1 \text{ für alle } t \in T \tag{5}$$

Ein Einsatzplan wie beschrieben existiert genau dann, wenn es im obigen Netz einen Fluss mit Wert $|T|$ gibt.

2 P b) Nennen Sie einen Algorithmus, der das Problem aus a) möglichst effizient löst, und geben Sie die Laufzeit im schlimmsten Fall in Abhängigkeit von der Anzahl der Ärzte n und der Anzahl der Ferientage $m = |T|$ an. Begründen Sie Ihre Antwort.

Lösung: Das Netz besitzt $2 + n + m$ viele Knoten und $n + \sum_{i=1}^{n} |S_i| + m$ viele Kanten. Da der Fluss maximal Wert m hat, ist der beste (in der Vorlesung vorgestellte) Algorithmus derjenige von Ford und Fulkerson. Er löst das Problem in Zeit $\mathcal{O}((n + \sum_{i=1}^{n} |S_i| + m) \cdot m)$.

3 P c) Nehmen Sie an, das Flussproblem aus a) wurde bereits gelöst, d.h. zu einem maximalen Fluss ϕ

kennen Sie den Fluss ϕ_e auf jeder Kante e. Nehmen Sie weiter an, ein geeigneter Einsatzplan wie oben beschrieben existierte tatsächlich. Beschreiben Sie detailliert einen Algorithmus, der aus den ϕ_e einen solchen Einsatzplan berechnet. Welche Laufzeit hat Ihr Verfahren, wenn auf jedes ϕ_e in Zeit $\Theta(1)$ zugegriffen werden kann?

Lösung: Wir iterieren über alle Kanten $e = (v_i^A, v_t)$ des Netzes, die von einem Arzt $i \in \{1, \ldots, n\}$ zu einem Ferientag $t \in T$ verlaufen. Der Arzt i muss am Tag t genau dann anwesend sein, wenn $\phi_e = 1$ ist. Die Laufzeit ist linear in der Anzahl der Kanten, beträgt also $\Theta(\sum_{i=1}^{n} |S_i|)$.

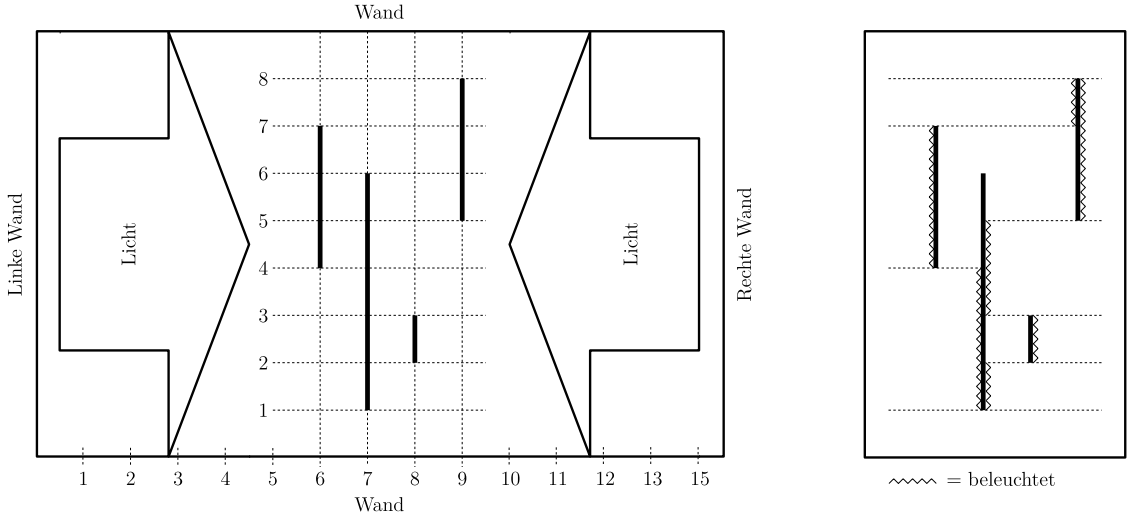
2 P d) Das bisherige Verfahren hat den Nachteil, dass es u.U. Einsatzpläne generiert, bei denen manche Ärzte sehr viel und andere Ärzte sehr wenig eingesetzt werden. Wir wollen daher einen Einsatzplan finden, der die o.g. Bedingungen erfüllt und zusätzlich jedem Arzt für jedes $j \in \{1, \ldots, k\}$ maximal einen Tag in T_j zuweist. Für das obige Beispiel heisst dies, dass ein Arzt zu Weihnachten entweder am 24.12, am 25.12, am 26.12 oder überhaupt keinen Dienst hat. Beschreiben Sie, wie das in a) konstruierte Netz modifiziert werden muss, um diese zusätzliche Anforderung zu erfüllen.

Lösung: Zunächst werden alle Kanten (v_i^A, v_t) gelöscht, die zwischen einem Arzt i und seinen verfügbaren Tagen $t \in S_i$ verlaufen. Stattdessen erzeugen wir für jeden Arzt $i \in \{1, \ldots, n\}$ und jede Ferienzeit $j \in \{1, \ldots, k\}$ einen Arzt-Ferienzeit-Knoten v_{ij}^{AF} und fügen dem Netz die folgenden Kanten hinzu:

- Für jeden Arzt $i \in \{1, \ldots, n\}$ und jede Ferienzeit $j \in \{1, \ldots, k\}$ eine Kante (v_i^A, v_{ij}^{AF}) mit Kapazität 1,
- Für jeden Arzt $i \in \{1, \ldots, n\}$, jede Ferienzeit $j \in \{1, \ldots, k\}$ und jeden Tag $t \in (S_i \cap T_j)$ der Ferienzeit j, an dem Arzt i anwesend sein kann, eine Kante (v_{ij}^{AF}, v_t) mit Kapazität 1.

Aufgabe 4.

Ein Künstler fertigt einen Grundrissplan seines Kunstwerks an, bei dem vertikale Platten mit Laserlicht von der Seite bestrahlt werden.



Es gibt n Platten mit verschiedenen Positionen und Breiten. Platte i werde dabei durch ein Tripel $P_i = (x_i, y_i, b_i)$ repräsentiert, wobei x_i der Abstand zur linken Wand, y_i der Abstand zur unten gezeichneten Wand und b_i die Breite der Platte angeben. Eine mögliche Eingabe für die Skizze oben wäre beispielsweise $P_1 = (7, 1, 5)$, $P_2 = (9, 5, 3)$, $P_3 = (8, 2, 1)$ sowie $P_4 = (6, 4, 3)$.

Die Platten werden von flächenförmigen Laserlichtquellen (an den Wänden links und rechts) bestrahlt, welche über die gesamte Breite und Höhe des Kunstwerks horizontale Lichtstrahlen aussenden. Licht, welches auf eine Platte trifft, wird vollständig absorbiert. Weil das Licht die Platten erhitzt, muss jede bestrahlte Platte gekühlt werden, und zwar proportional zum Betrag des einfallenden Lichts. Deshalb möchte der Künstler nun *für jede Platte* die gesamte bestrahlte Fläche (als Summe der von links bestrahlten Fläche und der von rechts bestrahlten Fläche) berechnen. Da die Platten vom Fussboden bis zur Decke reichen, ist die Höhe aller Platten gleich und es genügt, anstatt der bestrahlten Fläche die bestrahlte Breite der Platten zu berechnen. Daher liegt wie in der obigen Skizze ein zweidimensionales Problem vor.

Für jede Platte i soll die gesamte bestrahlte Breite B_i als Summe der von links bestrahlten Breite und der von rechts bestrahlten Breite berechnet werden (siehe Abbildung rechts). Für die obige Eingabe soll also $B_1 = 6$, $B_2 = 4$, $B_3 = 1$ sowie $B_4 = 3$ ausgegeben werden.

10 P Entwerfen Sie einen möglichst effizienten Scanline-Algorithmus für das obige Problem. Gehen Sie in Ihrer Lösung 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 für jede Platte die bestrahlte Breite B_i berechnet werden?

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

Hinweis: Der Einfachheit halber nehmen wir an, dass sich keine zwei Platten direkt übereinander befinden oder direkt nebeneinander starten oder enden. Beachten Sie aber, dass wie im obigen Beispiel die Platten der Eingabe nicht notwendigerweise nach x-Koordinate sortiert sind.

Lösung:

Haltepunkte: Wir verwenden eine horizontale Scanline, die von unten nach oben läuft. Diese stoppt jedesmal, wenn an ihrer Position eine Platte startet oder endet. Die Haltepunkte sind also y_i und $y_i + b_i$ für alle $i \in \{1, \ldots, n\}$.

Scanline-Datenstruktur: Wir benutzen einen AVL-Baum, der als Schlüssel die x-Koordinaten aller Platten speichert, die die Scanline derzeit schneidet. Zusätzlich enthält jeder Knoten für die dort gespeicherte Platte den Betrag ihrer beleuchteten Breite unterhalb der Scanline (als Gesamtbetrag von links und rechts).

Aktualisierung: Sei y_{prev} die Position, die die Scanline zum Zeitpunkt der letzten Aktualisierung hatte, und sei $y_{current}$ die aktuelle Position der Scanline. Der Knoten mit minimalem (bzw. maximalem) Schlüssel repräsentiert die Platte, die von links (bzw. von rechts) her beleuchtet ist. Da diese im Intervall $[y_{prev}, y_{current})$ beleuchtet ist, erhöhen wir den im Knoten mit minimalem Schlüssel gespeicherten Beleuchtungswert um $y_{current} - y_{prev}$, und genauso verfahren wir für den im Knoten mit maximalem Schlüssel gespeicherten Beleuchtungswert. Nach dieser Aktualisierung verfahren wir wie folgt:

1. *Fall: Eine neue Platte $P_i = (x_i, y_i, b_i)$ startet.* Wir fügen den Schlüssel x_i in die Scanline-Datenstruktur ein und setzen den Betrag der beleuchteten Breite auf 0.

2. *Fall: Eine Platte $P_i = (x_i, y_i, b_i)$ endet.* Wir löschen den Knoten mit Schlüssel x_i aus der Scanline-Datenstruktur und geben den dort gespeicherten Beleuchtungswert für die Platte P_i aus.

Auslesen der Lösung: Die Lösung wird direkt während der Aktualisierung der Datenstruktur ausgegeben, wenn eine Platte endet.

Laufzeit: Es gibt $2n$ Haltepunkte, und das Sortieren der Haltepunkte dauert Zeit $\mathcal{O}(n \log n)$. Im Aktualisierungsschritt müssen wir den Knoten mit minimalem bzw. maximalem Schlüssel finden, neue Schlüssel einfügen und bestehende Schlüssel löschen. Ein AVL-Baum unterstützt die Ausführung all dieser Operationen in Zeit $\mathcal{O}(\log n)$. Da die Aktualisierung an $2n$ Haltepunkten vorgenommen wird, erhalten wir eine Gesamtlaufzeit von $\mathcal{O}(n \log n)$.