



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  
Andreas Bärtschi

Beispiellösung zur Prüfung  
**Datenstrukturen und Algorithmen**  
D-INFK

10. August 2016



**Aufgabe 1.**

/ 15 P

*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.

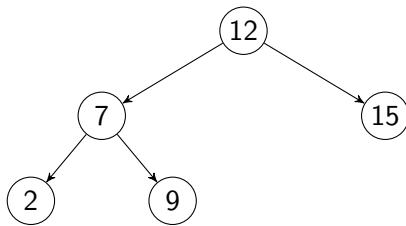
/ 1 P

- a) Fügen Sie die Schlüssel 27, 14 und 16 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 (nach *links*, d.h.  $h'(k)$  soll abgezogen werden).

<b>16</b>		<b>14</b>	<b>3</b>	<b>27</b>	<b>5</b>			<b>19</b>		
0	1	2	3	4	5	6	7	8	9	10

/ 1 P

- b) Sie sollen *einen* ganzzahligen Schlüssel in den folgenden AVL-Baum einfügen, sodass es zu einer *Doppelrotation* kommt. Alle im AVL-Baum gespeicherten Schlüssel müssen paarweise verschieden sein. Geben Sie *alle möglichen* ganzzahligen Kandidaten an.



*Lösung:* Schlüssel 8, 10 und 11 führen zu einer Doppelrotation.

/ 1 P

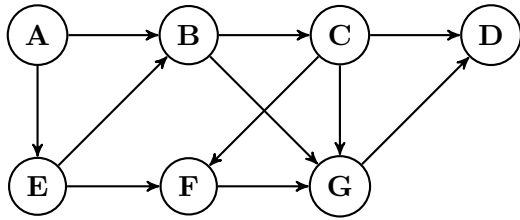
- c) Gegeben sei das folgende Array, das nach einem Quicksort-Aufteilungsschritt entstanden ist. Welcher Schlüssel wurde als Pivotelement verwendet? Markieren Sie *alle* möglichen Kandidaten.

<b>2</b>	<b>1</b>	<b>3</b>	<b>5</b>	<b>4</b>	<b>6</b>	<b>9</b>	<b>8</b>	<b>7</b>
1	2	3	4	5	6	7	8	9

*Lösung:* Für ein Pivotelement gilt, dass nach dem Aufteilungsschritt alle Elemente links davon kleiner sind und alle Elemente rechts davon grösser. Im gegebenen Array gilt das für die Schlüssel 3 und 6.

/ 1 P

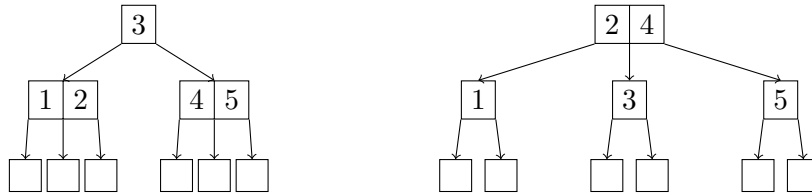
d) Geben Sie eine topologische Sortierung des untenstehenden Graphen an.



Die einzige topologische Sortierung ist  
A, E, B, C, F, G, D.

/ 1 P

e) Zeichnen Sie alle 2, 3-Bäume (B-Bäume mit bis zu 3 Kindern je Knoten) für die Schlüsselmenge 1, 2, 3, 4, 5.



/ 1 P

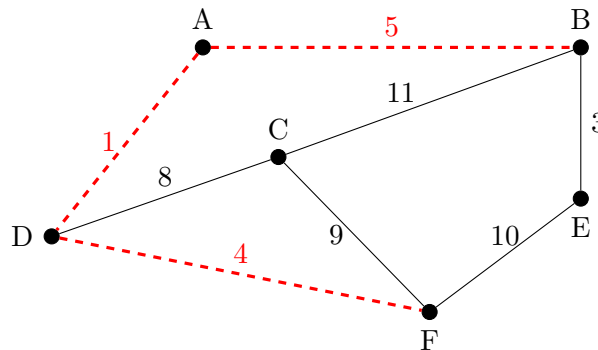
f) Beweisen oder widerlegen Sie: Sei  $G = (V, E)$  ein ungerichteter Graph,  $s, t \in V$  zwei Knoten und  $T = (V, E')$  mit  $E' \subseteq E$  ein minimaler Spannbaum von  $G$ . Der Pfad zwischen  $s$  und  $t$  in  $T$  ist ein kürzester Pfad in  $G$ .

*Lösung:* Die Aussage ist falsch. Ein Gegenbeispiel ist der folgende Graph  $G$  (links) und ein minimaler Spannbaum  $T$  von  $G$  (rechts). Der Pfad zwischen  $s$  und  $t$  in  $T$  ist nicht ein kürzester Pfad in  $G$ .



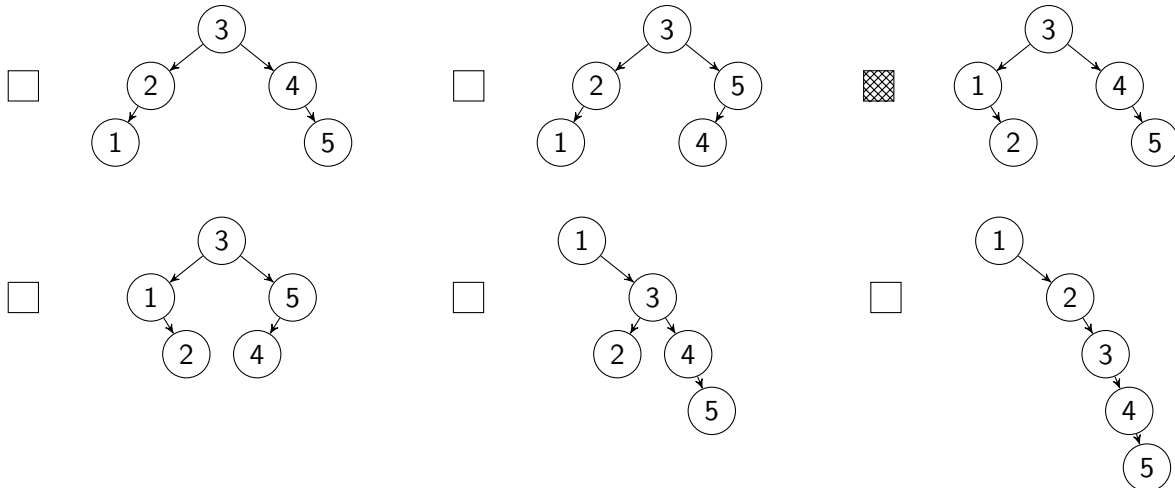
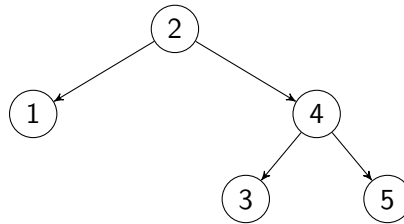
/ 1 P

g) 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

- h) Welcher Splay-Baum entsteht, wenn im folgenden Splay-Baum Schlüssel 1 und Schlüssel 3 in dieser Reihenfolge angefragt werden?



/ 1 P

- i) Gegeben seien ein Array  $A[1..n]$  und die folgende Java-Implementation des Algorithmus *Sortieren durch Einfügen*. Der Aufruf von `swap(A, i, j)` vertauscht die Elemente  $A[i]$  und  $A[j]$ . Um ein Array  $A[1..n]$  aufsteigend zu sortieren wird die Funktion mit den Parametern  $l = 1$  und  $r = n$  aufgerufen. Vervollständigen Sie die Implementation (Zeilen 2, 3 und 4).

```

1 public void insertionSort(int[] A, int l, int r) {
2     for (int i=l; i<=r; i++)
3         for (int j=i-1; j>=l && A[j]>A[j+1]; j--)
4             swap(A, j, j+1);
5 }
  
```

/ 3 P

j) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 3 \cdot T(n/7) + 4 & n > 1 \\ 3 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (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 7 ist. Benutzen Sie also  $n = 7^k$  oder  $k = \log_7(n)$ .

*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 7 ist, gilt  $n = 7^k$  für ein  $k \in \mathbb{N}$ . Wir teleskopieren, um auf eine Formel für  $T(n)$  zu kommen:

$$\begin{aligned} T(n) &= 3 \cdot T(n/7) + 4 \\ &= 3 \cdot (3 \cdot T(n/7^2) + 4) + 4 \\ &= 3 \cdot (3 \cdot (3 \cdot T(n/7^3) + 4) + 4) + 4 = \dots \\ &= 3^k \cdot 3 + 4 \cdot \sum_{i=1}^{k-1} 3^i \\ &= 3^k \cdot 3 + 4 \cdot \frac{3^k - 1}{2} \\ &= 3^k \cdot 3 + 2 \cdot 3^k - 2 \\ &= 5 \cdot 3^k - 2 \end{aligned}$$

Wir beweisen nun unsere Annahme durch vollständige Induktion über  $k$ .

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

*Induktionsannahme:* Für ein  $k \in \mathbb{N}_0$  sei  $T(7^k) = 5 \cdot 3^k - 2$ .

*Induktionsschritt* ( $k \rightarrow k + 1$ ):

$$T(7^{k+1}) = 3 \cdot T(7^k) + 4 \stackrel{\text{Ind.-Ann.}}{=} 3 \cdot (5 \cdot 3^k - 2) + 4 = 5 \cdot 3^{k+1} - 6 + 4 = 5 \cdot 3^{k+1} - 2.$$

/ 1 P

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

---

```

1 for ( int i = n; i >= 1; i = i / 2 ) {
2     int j = n;
3     while ( j >= 1 ) {
4         j = j - 20;
5     }
6 }
```

---

*Lösung:* Die äussere Schleife wird  $\Theta(\log n)$  Mal ausgeführt. Die innere Schleife (Zeilen 3 und 4) wird  $\Theta(n)$  Mal durchlaufen. Insgesamt ist die Laufzeit daher in  $\Theta(n \log(n))$ .

/ 1 P

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

---

```

1 int f( int n ) {
2     if ( n <= 1 ) return 1;
3     for ( int i = 1; i · i <= n; i = i + 2 ) {
4         ;
5     }
6     return f( n/4 ) + 1;
7 }
```

---

*Lösung:* Der Aufwand für die Zeilen 1 bis 6 ohne den rekursiven Aufruf in Schritt 5 ist in  $\Theta(\sqrt{n})$ , d.h. er beträgt höchstens  $c\sqrt{n}$  für eine geeignet gewählte Konstante  $c > 0$ . Die Gesamtlaufzeit (mit Berücksichtigung der rekursiven Aufrufe) ist dann durch

$$c\sqrt{n} + c \cdot \frac{\sqrt{n}}{2} + c \cdot \frac{\sqrt{n}}{2^2} + \dots = c\sqrt{n} \left( 1 + \frac{1}{2} + \frac{1}{2^2} + \dots \right) \leq 2c\sqrt{n}$$

nach oben beschränkt und beträgt damit ebenfalls  $\Theta(\sqrt{n})$ .

/ 1 P

- m) 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.

$$4^{n/2}, \sqrt{n}, \binom{n}{4}, 10^{64}, \log(n!), n^n, \log(n^{16})$$

*Lösung:* Es gilt  $\log(n^{16}) = 16 \log(n) \in \mathcal{O}(\log(n))$ . Ausserdem beobachten wir  $\log(n!) = \log\left(\prod_{i=1}^n i\right) = \sum_{i=1}^n \log(i) \leq n \log(n)$  und  $4^{n/2} = (4^{1/2})^n = 2^n$ . Die einzige korrekte Reihenfolge lautet

$$10^{64}, \log(n^{16}), \sqrt{n}, \log(n!), \binom{n}{4}, 4^{n/2}, n^n$$





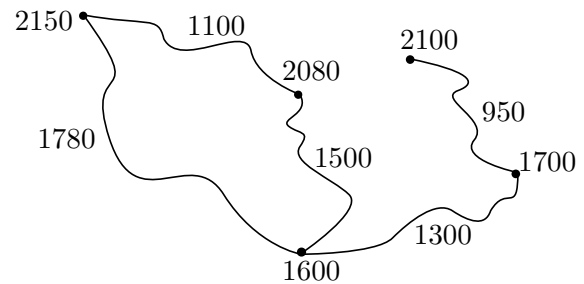
**Aufgabe 2.**

/ 12 P

In einem Skigebiet gibt es  $n$  Stationen, die mit  $m$  Pisten verbunden sind. Die Stationen  $i \in \{1, \dots, n\}$  sind nach ihrer Höhe  $h_i \in \mathbb{N}$  über dem Meeresspiegel absteigend sortiert, das heisst aus  $i < j$  folgt  $h_i \geq h_j$ . Die Pisten sind als Paare  $(i, j)$  für Stationen  $i, j \in \{1, \dots, n\}$  mit  $i \neq j$  gegeben, wobei für jede Piste  $h_i > h_j$  gilt. Weiters sind die Längen der Pisten  $l_1, l_2, \dots, l_m$  gegeben mit  $l_i \in \mathbb{N}$  für alle  $i \in \{1, \dots, m\}$ , die maximale Höhe im Skigebiet  $H = \max_{i \in \{1, \dots, n\}} h_i$  und die Gesamtlänge aller Pisten im Skigebiet  $L = \sum_{i=1}^m l_i$ .

Eine Abfahrt ist eine Abfolge von Pisten, die Sie ohne Unterbrechung befahren, d.h. mit Ausnahme der letzten Piste gilt, dass die Endstation einer Piste zugleich die Anfangsstation der nächsten Piste ist. Die Länge einer Abfahrt ist die Summe der Längen der befahrenen Pisten. Berechnen Sie eine längste Abfahrt im Skigebiet.

*Beispiel:* Es gibt  $n = 5$  Stationen (Punkte) und  $m = 5$  Pisten (Kurvenstücke). Die Höhen der Stationen und die Längen der Pisten sind im Bild vermerkt. Die Gesamtlänge aller Pisten ist  $L = 6630$  Meter und die maximale Meereshöhe  $H = 2150$  Meter. Die längste Abfahrt hat eine Länge von 2600 Meter.



/ 7 P

a) Geben Sie einen möglichst effizienten Algorithmus an, der nach dem Prinzip der dynamischen Programmierung arbeitet und die Länge einer längsten Abfahrt berechnet. Definieren Sie die Grösse der DP-Tabelle und die Bedeutung eines Eintrags und gehen Sie 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 aus früher berechneten Einträgen?
- 3) In welcher Reihenfolge können die Einträge berechnet werden?
- 4) Wie kann aus der DP-Tabelle die Länge einer längsten Abfahrt ausgelesen werden?

**Wählen Sie als erstes die Dimension Ihrer DP-Tabelle:**

- (I) **eindimensional** → Benutzen Sie das Schema auf Seite 8.
- (II) **zweidimensional**
- (III) **andere Dimension.**

Beantworten Sie die Teilaufgaben b) und c) auf Seite 9.

/ 2 P

b) Beschreiben Sie detailliert, wie aus der DP-Tabelle ihres dynamischen Programms abgelesen werden kann, aus welchen Pisten eine mögliche längste Abfahrt zusammengesetzt ist.

/ 3 P

c) Geben Sie die Laufzeit des in a) und b) entwickelten dynamischen Programms an und begründen Sie Ihre Antwort. Ist die Laufzeit polynomiell? Begründen Sie Ihre Antwort.

(I) Für Algorithmen mit einer eindimensionalen DP-Tabelle:

**Grösse der DP-Tabelle / Anzahl Einträge:**

- $n$                         $2^n$                         $L + 1$                        andere:  
  $m$                         $2^m$                         $H + 1$                       \_\_\_\_\_

**Bedeutung eines Tabelleneintrags:**

Bedeutung des Eintrags an Position  $i$ :

$DP[i]$ : Länge einer längsten Abfahrt, die in Station  $i$  endet.

**Initialisierung:** Alle Einträge werden mit 0 initialisiert, d.h.  $DP[i] \leftarrow 0$  für alle  $i = 1, \dots, n$

**Berechnung eines Eintrags:**

$$DP[i] = \max_{\text{Pisten } (j,i)} (DP[j] + \text{Länge der Piste } (j, i))$$

**Berechnungsreihenfolge:** Für die Berechnung eines Eintrags  $DP[i]$  werden nur Einträge  $DP[j]$  mit  $j < i$  berücksichtigt, da jede Piste von einer höher liegenden Station zu einer tiefer liegenden Station führt und die Stationen nach Höhe absteigend sortiert sind. Daher werden die Einträge für aufsteigende  $i$  von 1 bis  $n$  berechnet.

**Auslesen der Länge einer längsten Abfahrt:** Nachdem alle Einträge berechnet wurden, kann die Länge einer längsten Abfahrt ausgelesen werden, indem der maximale Tabelleneintrag gesucht wird:  $\max_{i \in \{1, \dots, n\}} (DP[i])$ .

*Teilaufgaben b) und c)*

- b) Beschreiben Sie detailliert, wie aus der DP-Tabelle ihres dynamischen Programms abgelesen werden kann, aus welchen Pisten eine mögliche längste Abfahrt zusammengesetzt ist.

Wir finden zunächst eine Station  $i$  mit  $DP[i] = \max_{i \in \{1, \dots, n\}}(DP[i])$ . Wir geben  $i$  aus und iterieren wir über alle Pisten  $(j, i)$ ; sobald für eine Station  $j$  gilt, dass  $DP[i] = DP[j] + \text{Länge der Piste } (j, i)$  gilt, brechen wir ab, setzen  $i \leftarrow j$  und wiederholen die Prozedur. Sobald eine Station  $i$  ohne eingehende Pisten  $(j, i)$  erreicht ist, brechen wir ab. Die Pisten einer längsten Abfahrt wurden nun in umgekehrter Reihenfolge ausgegeben.

- c) Geben Sie die Laufzeit des in a) und b) entwickelten dynamischen Programms an und begründen Sie Ihre Antwort. Ist die Laufzeit polynomiell? Begründen Sie Ihre Antwort.

Unsere Tabelle hat  $n$  Einträge und für jeden Eintrag müssen maximal  $m$  Pisten überprüft werden. Falls für jede Station eine Liste der eingehenden Pisten gespeichert ist und nicht in jedem Schritt alle  $n$  Pisten überprüft werden müssen, ist die Gesamtlaufzeit in  $\mathcal{O}(n + m)$  (also polynomiell). Die Rückverfolgung zur Berechnung einer längsten Abfahrt in Teilaufgabe b) kann beschleunigt werden, wenn bei der Berechnung der Tabelle zusätzlich gespeichert wird, welche Piste  $(j, i)$  für die Berechnung eines Eintrags  $DP[i]$  verwendet wird.



**Aufgabe 3.**

/ 7 P
-------

In einem Krankenhaus sollen Blutkonserven an Patienten verteilt werden. Bei der Zuteilung der Konserven an die Patienten muss die Blutgruppenkompatibilität beachtet werden. In der nebenstehenden Tabelle ist die Kompatibilität der Blutgruppen zusammengefasst. Beispielsweise kann ein Patient mit Blutgruppe A Konserven der Blutgruppen 0 oder A erhalten, während ein Patient mit Blutgruppe 0 nur Konserven der Blutgruppe 0 erhalten kann.

Patient	Konserve			
	0	A	B	AB
AB	✓	✓	✓	✓
A	✓	✓		
B	✓		✓	
0	✓			

Gegeben sind die Anzahl  $k_a, k_b, k_{ab}$  und  $k_0$  der verfügbaren Konserven für die Blutgruppen A, B, AB und 0 und die Anzahl  $b_a, b_b, b_{ab}$  und  $b_0$  der Patienten mit Blutgruppe A, B, AB und 0, die je eine Konserve benötigen. Entwickeln Sie einen Algorithmus, der berechnet, ob eine Zuteilung existiert, sodass jeder Patient eine Konserve mit einer kompatiblen Blutgruppe erhält.

*Beispiel:* Die Anzahl der Konserven und der Patienten sind in der Tabelle rechts gegeben. Bei folgender Zuteilung erhält jeder Patient eine kompatible Konserve: Konserven der Blutgruppe 0 werden an Patienten mit Blutgruppe 0 (2 Konserven) und Blutgruppe B (1 Konserve) verteilt, Konserven der Blutgruppe A an Patienten mit Blutgruppe A (3 Konserven) und Blutgruppe AB (1 Konserve), und die Konserven der restlichen Blutgruppen jeweils an Patienten mit derselben Blutgruppe.

Blutgruppe	0	A	B	AB
Bedarf	2	3	2	4
Lager	3	4	1	3

/ 4 P
-------

a) Modellieren Sie das oben genannte 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. Wie kann aus dem Wert eines maximalen Flusses abgelesen werden, ob eine Zuteilung existiert?

/ 3 P
-------

b) Nehmen Sie an, das Flussproblem aus a) wurde bereits gelöst, d.h. zu einem maximalen Fluss  $\varphi$  in  $N$  kennen Sie den Fluss  $\varphi_e$  auf jeder Kante  $e \in E$ . Nehmen Sie weiter an, eine Zuteilung existiert, sodass jeder Patient eine Konserve mit einer kompatiblen Blutgruppe erhält. Beschreiben Sie detailliert einen Algorithmus, der aus den  $\varphi_e$  eine solche Zuteilung berechnet. Welche Laufzeit hat Ihr Algorithmus, wenn auf jedes  $\varphi_e$  in konstanter Zeit zugegriffen werden kann?

*Teilaufgabe a)*

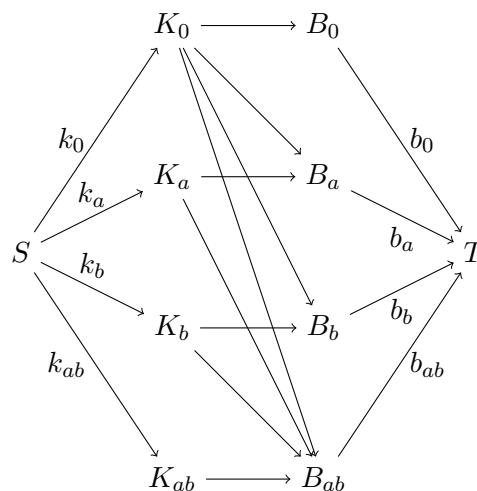
**Definition des Netzes  $N$  (wenn möglich in Worten und nicht formal):**

**Knotenmenge  $V$ :** Die Knotenmenge umfasst eine Quelle  $S$ , eine Senke  $T$ , sowie für jede Blutgruppe  $i \in \{a, b, ab, 0\}$  je einen Knoten für die verfügbaren Konserven  $K_i$  und einen Knoten für den Bedarf an Konserven dieser Gruppen  $B_i$ .

**Kantenmenge  $E$ :** Die Quelle  $S$  wird mit einer Kante  $(S, K_i)$  für  $i \in \{a, b, ab, 0\}$  mit den Knoten für die verfügbaren Konserven für jede Blutgruppe verbunden. Die Knoten für den Bedarf werden mit einer Kante  $(B_j, T)$  für  $j \in \{a, b, ab, 0\}$  mit der Senke  $T$  verbunden. Weiters gibt es eine Kante  $(K_i, B_j)$  für  $i, j \in \{a, b, ab, 0\}$ , falls Blutgruppe  $i$  an einen Patienten mit Blutgruppe  $j$  zugeteilt werden kann.

**Kapazitäten  $c$ :** Die Kanten  $(S, K_i)$  für  $i \in \{a, b, ab, 0\}$  haben Gewicht  $k_i$  und die Kanten  $(B_j, T)$  für  $j \in \{a, b, ab, 0\}$  haben Gewicht  $b_j$ . Die übrigen Kanten haben Gewicht  $\infty$ .

Schematische Darstellung / Zeichnung des Netzes:



Eine Zuteilung existiert, falls der Wert des maximalen Flusses den Wert  $b_0 + b_a + b_b + b_{ab}$  hat.

*Teilaufgaben b)*

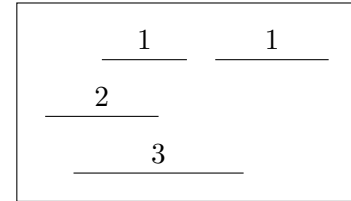
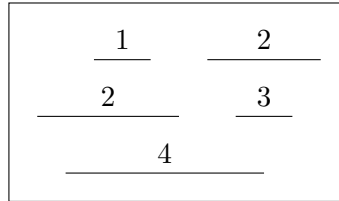
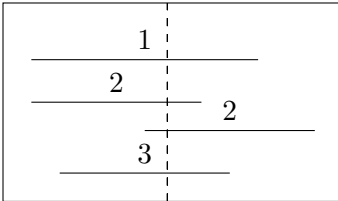
- b) Nehmen Sie an, das Flussproblem aus a) wurde bereits gelöst, d.h. zu einem maximalen Fluss  $\varphi$  in  $N$  kennen Sie den Fluss  $\varphi_e$  auf jeder Kante  $e \in E$ . Nehmen Sie weiter an, eine Zuteilung existiert, sodass jeder Patient eine Konserve mit einer kompatiblen Blutgruppe erhält. Beschreiben Sie detailliert einen Algorithmus, der aus den  $\varphi_e$  eine solche Zuteilung berechnet. Welche Laufzeit hat Ihr Algorithmus, wenn auf jedes  $\varphi_e$  in konstanter Zeit zugegriffen werden kann?

Wir betrachten alle Kanten  $(K_i, B_j)$  für  $i, j \in \{a, b, ab, 0\}$ . Der Fluss auf jeder einer Kante  $(K_i, B_j)$  entspricht der Zuteilung von Konserven der Blutgruppe  $i$  an Patienten mit Blutgruppe  $j$ . Da es nur eine konstante Anzahl von Kanten gibt, ist die Laufzeit des Algorithmus in  $\Theta(1)$ .

**Aufgabe 4.**

/ 10 P

Gegeben sei eine Menge von  $n$  horizontalen, sich nicht schneidenden Liniensegmenten in der Ebene. Jedes Liniensegment ist durch die Koordinaten seiner beiden Endpunkte  $(x_l, y)$  und  $(x_r, y)$  gegeben. Eine Nummerierung der Liniensegmente mit Zahlen aus  $\{1, \dots, m\}$  ist *zulässig*, wenn für jede vertikale Gerade gilt, dass die von ihr geschnittenen Liniensegmente von oben nach unten aufsteigend nummeriert sind. Gesucht ist die kleinste Zahl  $m \leq n$ , für die eine zulässige Nummerierung der Liniensegmente mit Zahlen aus  $\{1, \dots, m\}$  existiert.



*Beispiele:* *Links:* Die Nummerierung der Liniensegmente ist nicht zulässig, da die von der dargestellten vertikalen Gerade (gestrichelte Linie) geschnittenen Liniensegmente nicht aufsteigend nummeriert sind. *Mitte:* Die Nummerierung ist zulässig, es gibt jedoch auch eine zulässige Nummerierung mit  $m = 3$ . *Rechts:* Die Nummerierung ist zulässig und es gibt keine zulässige Nummerierung mit  $m < 3$ .

/ 8 P

a) 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 läuft 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 gesuchte, kleinste Zahl  $m$ , für die eine Nummerierung der Liniensegmente mit den geforderten Eigenschaften existiert, bestimmt werden?
- 5) Welche Laufzeit in Abhängigkeit von  $n$  hat Ihr Algorithmus? Begründen Sie Ihre Antwort.

/ 2 P

b) Beschreiben Sie, wie Sie Ihren Algorithmus erweitern können, damit er zusätzlich eine zulässige Nummerierung der Liniensegmente mit den Zahlen aus  $\{1, \dots, m\}$  berechnet.



**Scanline-Richtung (genau eine Möglichkeit ankreuzen):**

- Die Scanline ist eine *vertikale Gerade*; sie bewegt sich  
 von links nach rechts /  von rechts nach links.
- Die Scanline ist eine *horizontale Gerade*; sie bewegt sich  
 von oben nach unten /  von unten nach oben.
- Die Scanline ist eine *Halbgerade*; sie rotiert  
 im Uhrzeigersinn /  im Gegenuhrzeigersinn um den Punkt \_\_\_\_\_.
- andere Richtung: \_\_\_\_\_  
\_\_\_\_\_

**Haltepunkte:**

Menge der Haltepunkte: Die Start- und Endpunkte aller Liniensegmente

sortiert nach: ihren  $x$ -Koordinaten.

**Scanline-Datenstruktur:**

Ein AVL-Baum, der die an der aktuellen Scanline-Position geschnittenen Liniensegmente enthält, sortiert nach deren  $y$ -Koordinaten.

**Operationen bei einem Haltepunkt:** Ist der Haltepunkt der linke Endpunkt eines Liniensegmentes, so wird das Liniensegment in die Scanline eingefügt. Anschliessend wird das direkt oberhalb und unterhalb liegende Liniensegment ermittelt (falls es existiert). Die relative Position des neu eingefügten Liniensegmentes wird gespeichert (bspw. Liniensegment  $i$  liegt oberhalb Liniensegment  $j$ , etc.). Trifft die Scanline auf den rechten Eckpunkt eines Liniensegmentes, so wird wiederum seine relative Position ermittelt und gespeichert. Anschliessend wird das Liniensegment aus der Datenstruktur entfernt.

**Kleinste Zahl  $m$  berechnen:** Nachdem alle Haltepunkte abgearbeitet wurden, liegen für jedes Liniensegment höchstens vier relative Positionen vor. Wir bauen einen Graph  $G$ , in dem jeder Knoten einem Liniensegment entspricht und eine gerichtete Kante  $(i, j)$  die relative Position “Liniensegment  $i$  liegt oberhalb Liniensegment  $j$ ” repräsentiert. Der gerichtete Graph ist azyklisch und hat  $\mathcal{O}(n)$  Kanten. Wir entfernen nun alle Quellen (Knoten ohne eingehende Kanten) und deren ausgehende Kanten aus  $G$ . Diesen Vorgang wiederholen auf dem resultierenden Graph bis der Graph keine Knoten mehr enthält. Die Anzahl der Wiederholungen entspricht der Zahl  $m$  mit  $1 \leq m \leq n$ . Bei einer sorgfältigen Implementierung ist die Laufzeit für diese Berechnung in  $\mathcal{O}(n)$ , da der Graph nur  $\mathcal{O}(n)$  Knoten und Kanten enthält.

**Laufzeit des Algorithmus:** Die Haltepunkte können in  $\mathcal{O}(n \log(n))$  Zeit sortiert werden und in derselben Zeit können alle Operationen beim Abarbeiten der Haltepunkte durchgeführt werden. Die Berechnung der kleinsten Zahl  $m$  benötigt nur  $\mathcal{O}(n)$  Zeit. Die Gesamtlaufzeit ist daher in  $\mathcal{O}(n \log(n))$ .

### *Teilaufgabe b)*

- b) Beschreiben Sie, wie Sie Ihren Algorithmus erweitern können, damit er zusätzlich eine zulässige Nummerierung der Liniensegmente mit den Zahlen aus  $\{1, \dots, m\}$  berechnet.

Wir verwenden wiederum den Graph  $G$  wie in Teilaufgabe a) beschrieben und entfernen in  $m$  Runden alle Knoten und Kanten im Graph. Dabei werden die Liniensegmente der in Runde  $i$  entfernten Knoten mit der Zahl  $i$  nummeriert. Nach  $m$  Runden wurde allen Liniensegmenten eine Zahl zugewiesen und diese Nummerierung der Liniensegmente ist zulässig.