/ 16 P

Aufgabe 1.

Instruktionen:

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.

3	7	10	15 ‖	8	6	9	5	2	13
1	2	3	4	5	6	7	8	9	10

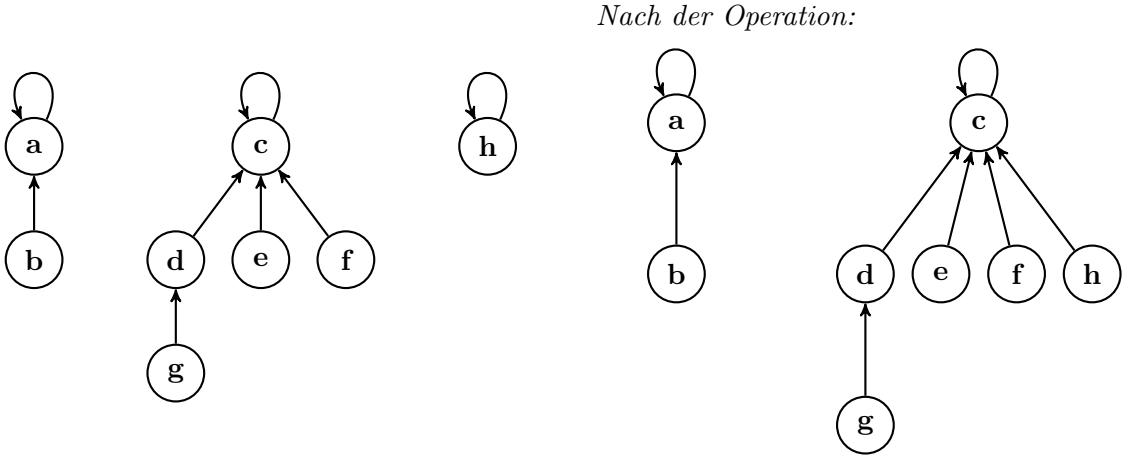
3	7	8	10	15 ‖	6	9	5	2	13
1	2	3	4	5	6	7	8	9	10

3	6	7	8	10	15 ‖	9	5	2	13
1	2	3	4	5	6	7	8	9	10

/ 1 P b) Fügen Sie die Schlüssel 20, 24, 31 und 7 in dieser Reihenfolge in die untenstehende Hashtabelle ein. Benutzen Sie offenes Hashing mit der Hashfunktion $h(k) = k \bmod 11$. Lösen Sie Kollisionen mittels quadratischem Sondieren auf. Im Falle einer Kollision soll die Sondierung *zunächst nach links und erst danach nach rechts* erfolgen.

11		24		15	5	7	31	19	9	20
0	1	2	3	4	5	6	7	8	9	10

/ 1 P c) Führen Sie auf der folgenden Union-Find-Datenstruktur $\textsc{Union}(h, \textsc{Find}(g))$ mithilfe des Verfahrens "Vereinigung nach Höhe" aus und zeichnen Sie die resultierende Datenstruktur nach der Operation.



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

1	3	6	4	9	7	10	13	8
1	2	3	4	5	6	7	8	9

3	4	6	8	9	7	10	13
1	2	3	4	5	6	7	8

/ 1 P e) Geben Sie eine kürzeste Folge von Anfragen an, welche ausgehend von der Liste

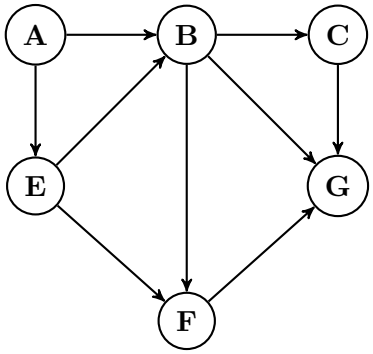
$$A \to B \to C \to D \to E \to F$$

unter Anwendung der Move-to-Front-Regel folgende Liste ergibt:

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

Lösung: Die Schlüssel E und F müssen nicht angefragt werden, da sie bereits vor der ersten Anfrage in der richtigen Reihenfolge am Ende der Liste stehen. Wir betrachten also nur die ersten vier Schlüssel der Liste. Mit Sicherheit muss zuletzt D angefragt werden, damit dieser Schlüssel an erster Position steht. Um die Reihenfolge der Schlüssel B und C zu vertauschen muss zudem zwingend C angefragt werden. Danach muss allerdings noch die Reihenfolge der Schlüssel A und C wiederhergestellt werden. Daher ist die kürzeste Folge von Anfragen C, A, D.

/ 1 P f) Der folgende Graph wird ausgehend von Knoten A traversiert. Die Nachbarn eines Knoten werden in alphabetischer Reihenfolge abgearbeitet. Geben Sie die beiden Reihenfolgen an, in der eine Tiefensuche und eine Breitensuche die Knoten inspizieren.



Tiefensuche:

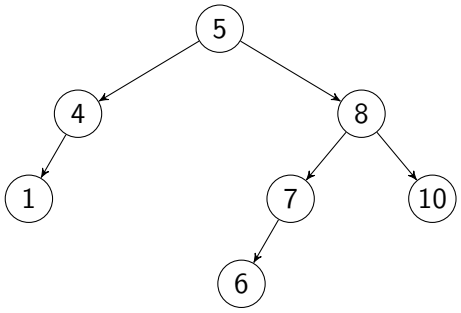
A , B , C , G , F , E .

Breitensuche:

A , B , E , C , F , G .

/ 1 P g) Zeichnen Sie den natürlichen Suchbaum T, der entsteht, wenn die Schlüssel $5, 4, 8, 7, 1, 6, 10$ in dieser Reihenfolge in einen anfangs leeren natürlichen Suchbaum eingefügt werden. Geben Sie die Preorder-, Postorder- und Inorder-Reihenfolgen der Schlüssel in T an.

natürlicher Suchbaum T:



Preorder-Reihenfolge:

$5, 4, 1, 8, 7, 6, 10$

Inorder-Reihenfolge:

$1, 4, 5, 6, 7, 8, 10$

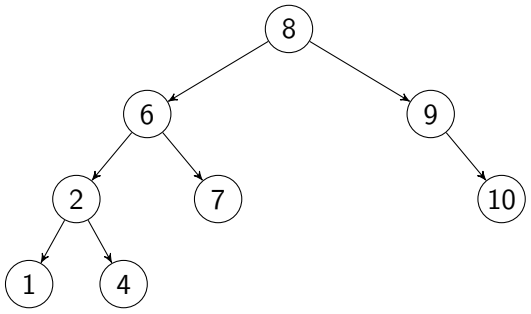
Postorder-Reihenfolge:

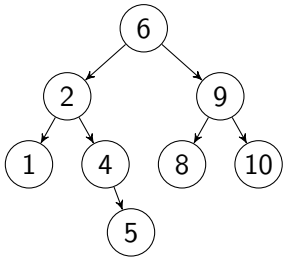
$1, 4, 6, 7, 10, 8, 5$

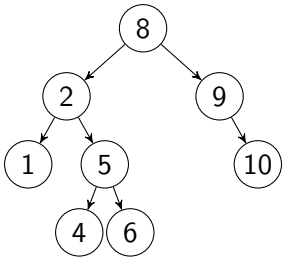
/ 2 P h) Kreuzen Sie bei jeder der folgenden Datenstrukturen an, ob der darin verwendete Baum balanciert ist (d.h. eine Höhe logarithmisch in der Anzahl der gespeicherten Schlüssel garantiert) oder nicht balanciert ist. Jede korrekte Antwort gibt 0.5 Punkte, für jede falsche Antwort werden 0.5 Punkte abgezogen. Eine fehlende Antwort gibt 0 Punkte. Insgesamt gibt die Aufgabe mindestens 0 Punkte. Sie müssen Ihre Antworten nicht begründen.

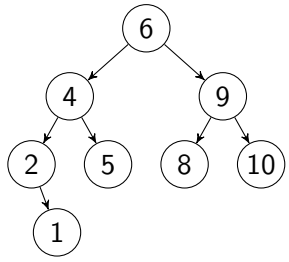
B-Baum	☒ BALANCIERT	☐ NICHT BALANCIERT
Heap	☒ BALANCIERT	☐ NICHT BALANCIERT
AVL-Baum	☒ BALANCIERT	☐ NICHT BALANCIERT
Natürlicher Suchbaum	☐ BALANCIERT	☒ NICHT BALANCIERT

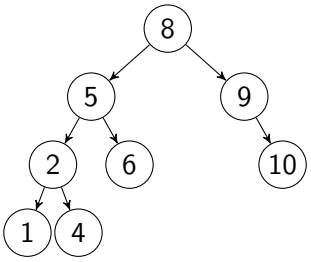
/ 1 P i) Welcher AVL-Baum entsteht, wenn im folgenden AVL-Baum zuerst Schlüssel 7 gelöscht wird und anschliessend Schlüssel 5 eingefügt wird?

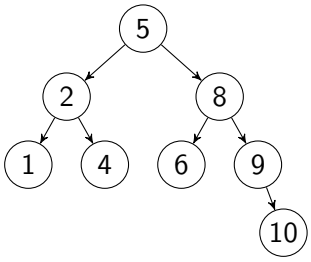


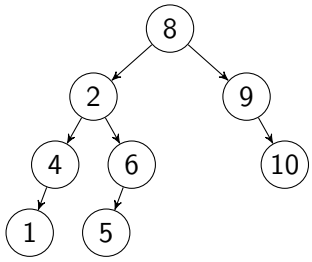
☐


■


☐


☐


☐


☐


/ 3 P j) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 5 \cdot T(n/8) + 8 & n > 1 \\ 1 & n = 1 \end{cases}$$

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

Hinweise:

(1) Sie können annehmen, dass n eine Potenz von 8 ist.
Benutzen Sie also $n = 8^k$ oder $k = \log_8(n)$.

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

Herleitung (falls benötigt):

Da wir annehmen dürfen, dass n eine Potenz von 8 ist, gilt $n = 8^k$ für ein $k \in \mathbb{N}$. Wir teleskopieren, um auf eine Formel für $T(n)$ zu kommen:

$$\begin{aligned} T(n) &= 5 \cdot T(n/8) + 8 \\ &= 5 \cdot \big(5 \cdot T(n/8^2) + 8\big) + 8 \\ &= 5 \cdot \Big(5 \cdot \big(5 \cdot T(n/8^3) + 8\big) + 8\Big) + 8 = ... \\ &= 5^k \cdot 1 + 8 \cdot \sum_{i=1}^{k-1} 5^i \\ &= 5^k + 8 \cdot \frac{5^k - 1}{4} \\ &= 5^k + 2 \cdot 5^k - 2 \\ &= 3 \cdot 5^k - 2 \end{aligned}$$

Geschlossene und vereinfachte Formel:

$$T(n) = T(8^k) = 3 \cdot 5^k - 2$$

Induktionsbeweis:

Wir beweisen nun unsere Annahme durch vollständige Induktion über k.

Induktionsverankerung ($k = 0$): Es gilt $T(8^0) = T(1) = 1 = 3 \cdot 5^0 - 2$.

Induktionsannahme: Für ein $k \in \mathbb{N}_0$ sei $T(8^k) = 3 \cdot 5^k - 2$.

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

$$T(8^{k+1}) = 5 \cdot T(8^k) + 8 \overset{\text{Ind.-Ann.}}{=} 5 \cdot (3 \cdot 5^k - 2) + 8 = 3 \cdot 5^{k+1} - 10 + 8 = 3 \cdot 5^{k+1} - 2.$$

Induktionsbeweis (Fortsetzung):

—

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

Laufzeit in möglichst knapper Θ-Notation:

$$\Theta(n \log(n))$$

Lösung: Die äussere Schleife wird $\Theta(n)$ Mal ausgeführt. Die innere Schleife (Zeilen 3 und 4) wird $\Theta(\log(n))$ 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 Θ-Notation an. Sie müssen Ihre Antwort nicht begründen.

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

Laufzeit in möglichst knapper Θ-Notation:

$$\Theta(n^2)$$

Lösung: Die äussere Schleife wird höchstens n Mal durchlaufen und die innere Schleife höchstens $2n$ Mal (wegen $i \leq n$). Die Laufzeit ist also mit $\mathcal{O}(n^2)$ nach oben beschränkt. Andererseits wird die äussere Schleife mindestens $\lfloor n/5 \rfloor$ Mal durchlaufen, und für $i > n/2$ wird die innere Schleife mindestens $n/2$ Mal durchlaufen. Die Laufzeit ist damit auch in $\Omega(n^2)$. Daher erhalten wir eine Laufzeit von $\Theta(n^2)$.

/ 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.

$$\frac{n^2}{\log(n)},\ \log(n^2),\ \sqrt{6^n},\ 2^{15},\ n\sqrt{n},\ \binom{n}{6},\ n\log(n)$$

Lösung: Es gilt $\log(n^2) = 2\log(n) \in \mathcal{O}(\log(n))$. Weiters gilt $n^{3/2} = n\sqrt{n}$. Wegen $\lim_{n\to\infty} \frac{n\sqrt{n}}{n^2/\log n} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0$ erhalten wir $n\sqrt{n} \in \mathcal{O}(\frac{n^2}{\log n})$. Die einzige korrekte Reihenfolge lautet

$$2^{15},\ \log(n^2),\ n\log(n),\ n^{3/2},\ \frac{n^2}{\log(n)},\ \binom{n}{6},\ \sqrt{6^n}$$

Aufgabe 2. **/ 10 P**

Ein *Palindrom* ist eine Zeichenfolge, die sich von vorne wie von hinten gleich liest, also z.B. das Wort `RENTNER`. Formal ist ein Palindrom eine Zeichenfolge der Länge 0 oder 1, oder eine Zeichenfolge $\langle a_1, \ldots, a_l \rangle$, wobei $a_1 = a_l$ gilt und $\langle a_2, \ldots, a_{l-1} \rangle$ ein Palindrom ist.

Gegeben sei eine Zeichenfolge $Z = \langle a_1, \ldots, a_n \rangle$ der Länge n. Wir sagen, dass Z ein Palindrom P enthält, wenn P eine (nicht notwendigerweise zusammenhängende) Teilfolge von Z ist. Gesucht ist das längste in Z enthaltene Palindrom.

Beispiel: Die Zeichenfolge $Z = \langle \mathtt{A}, \mathtt{N}, \mathtt{A}, \mathtt{N}, \mathtt{A}, \mathtt{S} \rangle$ enthält das Palindrom der Länge 0, sowie die Palindrome `A`, `N`, `S`, `AA`, `NN`, `AAA`, `ANA`, `NAN`, `ANNA` sowie `ANANA` (einige dieser Palindrome kommen mehrfach vor). Das längste in Z enthaltene Palindrom ist also `ANANA`.

Entwerfen Sie für das obige Problem einen Algorithmus, der nach dem Prinzip der *dynamischen Programmierung* arbeitet. Gehen Sie dabei auf die folgenden Aspekte ein.

1) Was ist die Bedeutung eines Tabelleneintrags, und welche Grösse hat die DP-Tabelle?
2) Wie kann die Tabelle initialisiert werden, und wie berechnet sich ein Tabelleneintrag aus früher berechneten Einträgen?
3) In welcher Reihenfolge müssen die Einträge berechnet werden?
4) Wie lässt sich die Lösung aus der DP-Tabelle auslesen?

Geben Sie auch die Laufzeit Ihrer Lösung an, und begründen Sie Ihre Antwort.

Grösse der DP-Tabelle / Anzahl Einträge: *Wir verwenden eine Tabelle der Grösse $n \times n$.*

Bedeutung eines Tabelleneintrags:

$DP[i, j]$: *Länge eines längsten in $\langle a_i, \ldots, a_j \rangle$ enthaltenen Palindroms.*

Berechnung eines Eintrags:

- *Zeichenfolgen der Länge 1 sind immer ein Palindrom, also setzen wir $DP[i, i] = 1$ für alle $i \in \{1, \ldots, n\}$.*
- *Das längste in einer Zeichenfolge der Länge 2 enthaltene Palindrom hat genau dann Länge 2, wenn die beiden Zeichen übereinstimmen, und Länge 1 sonst. Für $i \in \{1, \ldots, n-1\}$ setzen wir also $DP[i, i+1] = 2$, falls $a_i = a_{i+1}$ gilt, und $DP[i, i+1] = 1$ ansonsten.*
- *Für allgemeine i und j mit $1 \leq i < i+2 \leq j \leq n$ unterscheiden wir zwei Fälle:*
  - *Falls $a_i = a_j$ gilt, dann ist $DP[i, j] = 2 + DP[i+1, j-1]$: Ein längstes in $\langle a_i, \ldots, a_j \rangle$ enthaltenes Palindrom benutzt den ersten und den letzten Buchstaben sowie ein längstes in $\langle a_{i+1}, \ldots, a_{j-1} \rangle$ enthaltenes Palindrom.*
  - *Falls $a_i \neq a_j$ gilt, dann ist $DP[i, j] = \max\{DP[i, j-1], DP[i+1, j]\}$: Das längste in $\langle a_i, \ldots, a_j \rangle$ enthaltene Palindrom ist entweder in $\langle a_i, \ldots, a_{j-1} \rangle$ oder in $\langle a_{i+1}, \ldots, a_j \rangle$ enthalten.*

Berechnungsreihenfolge:

Wir berechnen zunächst die Einträge $DP[i, i]$ für $i = 1, \ldots, n$, danach $DP[i, i+1]$ für $i = 1, \ldots, n-1$, danach $DP[i, i+2]$ für $i = 1, \ldots, n-2$, usw., bis schliesslich $DP[1, n]$ berechnet wurde.

Auslesen des längsten enthaltenen Palindroms:

Wir beginnen die Berechnung des längsten enthaltenen Palindroms im Eintrag $DP[1, n]$. Dieser enthält eine Zahl l, die die Länge eines längsten enthaltenen Palindroms angibt. Wir führen nun den Berechnungsschritt rückwärts aus: Ist $a_1 = a_n$, dann ist dies auch das erste und das letzte Zeichen des Palindroms, und wir setzen die Rekonstruktion im Eintrag $DP[2, n-1]$ fort. Ist dagegen $a_1 \neq a_n$, dann betrachten wir die Einträge $DP[1, n-1]$ und $DP[2, n]$, und setzen die Rekonstruktion bei dem Eintrag fort, der einen grösseren Wert besitzt. Auf diese Art wird die Rekonstruktion fortgesetzt, bis wir das mittlere bzw. die beiden mittleren Zeichen des Palindroms rekonstruiert haben.

Laufzeit (mit Begründung):

Die Berechnung eines Eintrags $DP[i, j]$ erfolgt in konstanter Zeit, da nur konstant viele Fälle unterschieden werden und für jeden Fall nur konstant viele Werte anderer Tabelleneinträge betrachtet werden. Da die Tabelle n^2 viele Einträge besitzt, können alle Einträge in Zeit $\Theta(n^2)$ berechnet werden.

Bei der Rekonstruktion fällt pro Schritt ebenfalls nur konstante Zeit an. Danach wird die Rekonstruktion in einem Eintrag fortgesetzt, dessen Zeilen- oder Spaltennummer (oder sogar beide) jeweils um 1 grösser sind. Damit endet die Rekonstruktion nach spätestens $2n$ Schritten, und die Laufzeit ist in $\Theta(n)$.

Insgesamt fällt also Zeit $\Theta(n^2) + \Theta(n) = \Theta(n^2)$ an.

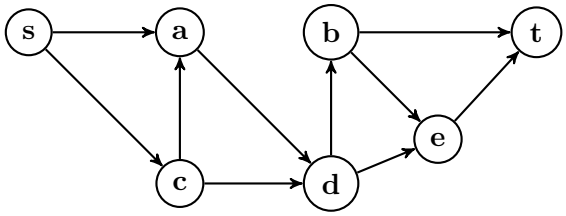
/ 7 P

Aufgabe 3.

Zwei Computer sollen über ein Netz zuverlässig kommunizieren. Die Computer sind nicht unbedingt direkt miteinander verbunden, sondern schicken im Allgemeinen Pakete über verschiedene Zwischenstationen durchs Netz. Es soll sichergestellt werden, dass ein Computer dem anderen Computer auch bei einem Ausfall einer einzigen, beliebigen Verbindung im Netz Pakete schicken kann.

Das Netz ist durch eine Menge V von Knoten und eine Menge E von (gerichteten) Verbindungen zwischen je zwei Knoten gegeben. Weiterhin sind zwei Knoten s und t in V gegeben, die miteinander kommunizieren sollen.

/ 3 P a) Wir wollen entscheiden, ob s auch dann noch Pakete an t schicken kann, wenn eine einzige, beliebige Verbindung in E unterbrochen wird. Modellieren Sie dazu das Problem als Flussproblem. Beschreiben Sie dazu die Konstruktion eines geeigneten Netzes $N_a = (V_a, E_a, c_a)$ und geben Sie an, welche Kapazitäten c_a die Kanten besitzen sollen. Wie kann aus dem Wert eines maximalen Flusses abgelesen werden, ob die Kommunikation von s nach t möglich ist, wenn eine einzige, beliebige Verbindung in E unterbrochen wird?



Beispiel: Im links dargestellten Netz kann der Knoten s auch dann noch Pakete an den Knoten t schicken, wenn eine einzige, beliebige Kante entfernt wird.

/ 4 P b) Jetzt möchten wir entscheiden, ob der Knoten s auch dann noch Pakete an den Knoten t schicken kann, wenn ein *Knoten* aus $V \setminus \{s, t\}$ defekt ist und nicht genutzt werden kann. Modellieren Sie dazu das Problem als Flussproblem und beschreiben Sie ein geeignetes Netz $N_b = (V_b, E_b, c_b)$. Wie kann aus dem Wert eines maximalen Flusses in diesem Netz abgelesen werden, ob die Kommunikation von s nach t möglich ist, wenn nur ein Knoten entfernt wird?

Beispiel: Im oben dargestellten Netz kann der Knoten s keine Pakete an den Knoten t schicken, wenn der Knoten d nicht genutzt werden kann.

Teilaufgabe a)

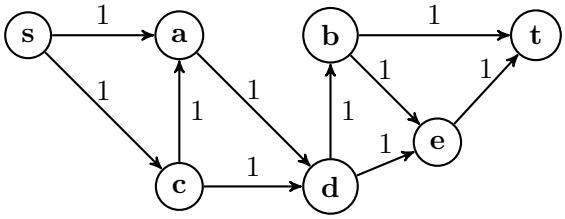
Definition des Netzes N_a (wenn möglich in Worten und nicht formal):

Knotenmenge V_a: *Wir erzeugen für jeden Computer einen Knoten.*

Kantenmenge E_a: *Für jede gerichtete Verbindung von einem Computer v zu einem Computer w erzeugen wir eine Kante (v, w).*

Kapazitäten c_a: *Alle Kanten haben Kapazität 1.*

Schematische Darstellung / Zeichnung des Netzes:



Eine Kommunikation von s nach t ist möglich, falls *der maximale Fluss von s nach t mindestens* 2 *beträgt. Nach dem Max-Flow-Min-Cut Theorem existiert genau in diesem Fall kein s-t-Schnitt mit Kapazität 0 oder 1.*

Teilaufgabe b)

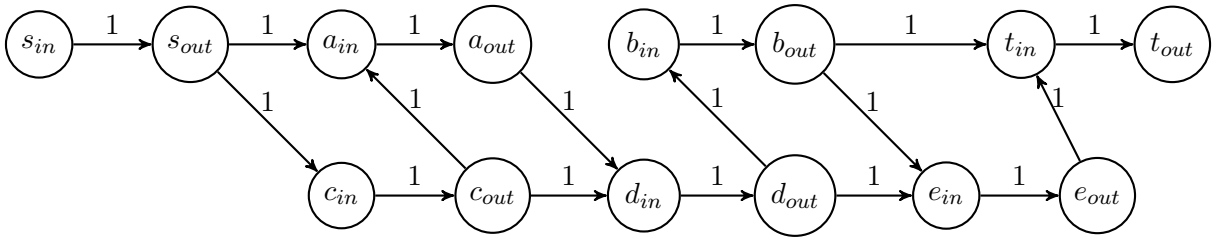
Definition des Netzes N_b (wenn möglich in Worten und nicht formal):

Knotenmenge V_b: *Wir erzeugen für jeden Computer v zwei Knoten v_{in} und v_{out}.*

Kantenmenge E_b: *Für jede gerichtete Verbindung von einem Computer v zu einem Computer w erzeugen wir eine Kante (v_{out}, w_{in}). Für jeden Computer v erzeugen wir eine Kante (v_{in}, v_{out}).*

Kapazitäten c_b: *Alle Kanten haben Kapazität 1.*

Schematische Darstellung / Zeichnung des Netzes:

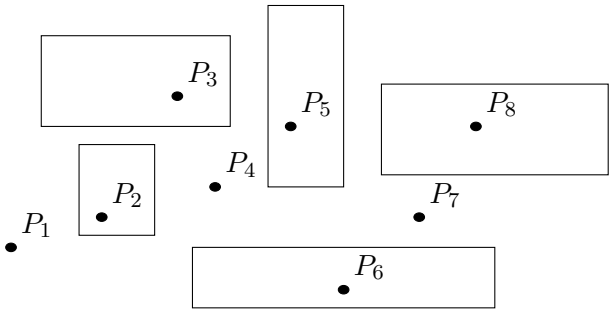


Eine Kommunikation von s nach t ist möglich, falls *der maximale Fluss von s_{out} nach t_{in} mindestens 2 beträgt. Nach dem Max-Flow-Min-Cut Theorem existiert genau in diesem Fall kein s-t-Knoten-Schnitt mit Kapazität 0 oder 1.*

Aufgabe 4. **/ 9 P**

Gegeben sei eine Menge von n Punkten $P_1, \ldots, P_n$ mit $P_i = (x_i, y_i) \in \mathbb{Q}^2$ für alle $i \in \{1, \ldots, n\}$. Keine zwei Punkte haben dieselben x- oder y-Koordinaten. Gegeben sei ausserdem eine Menge von m achsen-parallelen Rechtecken $R_1, \ldots, R_m$ in der Ebene. Jedes Rechteck R_j mit $j \in \{1, \ldots, m\}$ ist durch seinen linken, unteren Eckpunkt $(l_j, u_j) \in \mathbb{Q}^2$ und seinen rechten, oberen Eckpunkt $(r_j, o_j) \in \mathbb{Q}^2$ gegeben. Keine zwei Rechtecke überlappen oder berühren sich, und kein Punkt P_i liegt genau auf dem Rand eines Rechtecks. Gesucht sind alle Punkte, die sich nicht innerhalb eines der m Rechtecke befinden.

Beispiel: In der Abbildung rechts sind fünf Rechtecke und acht Punkte gegeben. Die Punkte P_1, P_4 und P_7 befinden sich in keinem Rechteck.



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 verwaltet die Scanline-Datenstruktur, und was ist eine angemessene Datenstruktur?
3) Was passiert, wenn die Scanline auf einen neuen Haltepunkt trifft?
4) Wie können diejenigen Punkte bestimmt werden, die sich nicht innerhalb eines Rechteckes befinden?
5) Welche Laufzeit in Abhängigkeit von n und m hat Ihr Algorithmus? Begründen Sie Ihre Antwort.

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 n Punkte $P_1, \ldots, P_n$ und die linken und rechten Seiten (l_j, u_j, o_j) und (r_j, u_j, o_j) der m Rechtecke $R_1, \ldots, R_m$.*

sortiert nach: *ihren x-Koordinaten. Bei Punkten mit gleicher x-Koordinate wie die Seitenlinie eines Rechtecks, spielt deren Reihenfolge keine Rolle, da sie sich nach Annahme in der Aufgabe nicht berühren dürfen.*

Scanline-Datenstruktur:

Wir benutzen einen AVL-Baum, der die oberen und unteren Randkoordinaten der an der momentanen Scanline-Position geschnittenen Rechtecke enthält, sortiert nach deren y-Koordinaten. Der AVL-Baum wird so erweitert, dass wir die Grösse von Teilbäumen mitspeichern und damit auch in $\mathcal{O}(\log n)$ die Anzahl Elemente zählen können, die kleiner als ein Anfrageschlüssel sind (wie in der Vorlesung behandelt).

Operationen bei einem Haltepunkt:

Ist der Haltepunkt eine linke Seite (l_j, u_j, o_j) eines Rechtecks R_j, so fügen wir u_j und o_j in den AVL-Baum ein. Ist der Haltepunkt eine rechte Seite (r_j, u_j, o_j) eines Rechtecks R_j, so entfernen wir u_j und o_j aus dem AVL-Baum.

Ist der Haltepunkt ein Punkt $P_i = (x_i, y_i)$, so ermitteln wir die Anzahl A der Einträge kleiner als y_i im AVL-Baum. Da sich die Rechtecke nicht überlappen, folgt aus der Parität von A, ob P_i in einem Rechteck enthalten ist oder nicht. Ist A gerade, so liegt P_i in keinem Rechteck und wir speichern P_i für das Endresultat. Ist A ungerade, so liegt P_i in einem Rechteck und wir müssen nichts weiter tun.

Auslesen der Lösung:

Die Punkte, die sich nicht innerhalb eines Rechtecks befinden, werden bereits während dem Scanline-Vorgang fortlaufend ermittelt und gespeichert.

Laufzeit des Algorithmus:

Die $n + 2m$ Haltepunkte können in $\mathcal{O}((n + m)\log(n + m))$ Zeit sortiert werden. Der AVL-Baum enthält $\mathcal{O}(m)$ Elemente, daher dauert jede Operation beim Abarbeiten der Haltepunkte $\mathcal{O}(\log m)$. Insgesamt dominiert also die Laufzeit des Sortierens und die Gesamtlaufzeit ist in $\mathcal{O}((n+m)\log(n+m))$.