Eidgenössische
Technische Hochschule
Zürich

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

Institut für Theoretische Informatik
Peter Widmayer
Tobias Pröger

Beispiellösung zur Prüfung
Datenstrukturen und Algorithmen
D-INFK

8. August 2013

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	5	**Σ**
Mögl. Punkte	8	10	13	9	10	50
$\sum$ 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 einen Aufteilungsschritt (in-situ, d.h. ohne Hilfsarray) des Sortieralgorithmus *Quicksort* durch. Benutzen Sie als Pivot das am rechten Ende stehende Element im Array.

9	5	19	11	7	15	1	0	2	17	4	8
1	2	3	4	5	6	7	8	9	10	11	12

Wenn das Pivot bis zum Ende am rechten Rand belassen und dann in die Mitte getauscht wird:

4	5	2	0	7	1	8	11	19	17	9	15
1	2	3	4	5	6	7	8	9	10	11	12

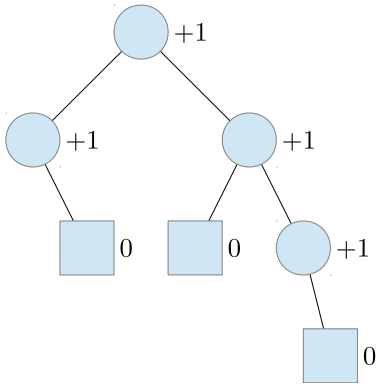
Wenn das Pivot herausgenommen und erst am Ende wieder eingefügt wird:

4	5	2	0	7	1	8	15	11	19	17	9
1	2	3	4	5	6	7	8	9	10	11	12

Wenn immer jeweils mit dem Pivot vertauscht wird:

4	5	2	0	7	1	8	15	11	17	19	9
1	2	3	4	5	6	7	8	9	10	11	12

1 P (b) Zeichnen Sie einen AVL-Baum mit 7 Knoten, bei dem *jeder* innere Knoten einen Balancierungsfaktor ungleich 0 besitzt.



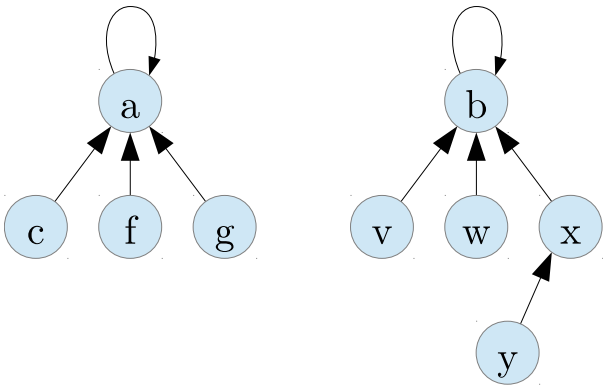
Der Einfachheit halber wurde in der obigen Abbildung auf eine Einzeichnung der Schlüssel verzichtet. Rechts neben jedem Knoten ist der jeweilige Balancierungsfaktor dieses Knotens angegeben.

3 P (c) 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.

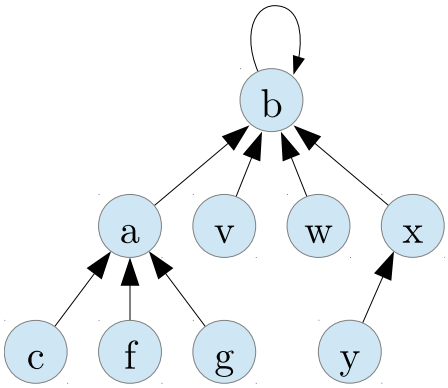
Eine Inorder-Traversierung eines binären Suchbaums erzeugt eine sortierte Liste der gespeicherten Schlüssel.	☒ Wahr	☐ Falsch
Hat eine Folge von m Operationen im schlimmsten Fall Gesamtkosten $\mathcal{O}(m)$, dann hat jede einzelne dieser Operationen im schlimmsten Fall Kosten $\mathcal{O}(1)$.	☐ Wahr	☒ Falsch
Sei $G = (V, E)$ ein Graph. Jeder Teilgraph von G mit $\lvert V\rvert - 1$ Kanten ist ein Spannbaum von G.	☐ Wahr	☒ Falsch
In einem Fibonacci-Heap mit n Schlüsseln ist die Laufzeit zur Extraktion des Minimums im schlimmsten Fall $\mathcal{O}(\log n)$.	☐ Wahr	☒ Falsch
Sortieren durch Auswahl ist ein stabiles Sortierverfahren.	☐ Wahr	☒ Falsch
Auf einem komplett vorsortierten Array ist die Laufzeit von Sortieren durch Auswahl linear.	☐ Wahr	☒ Falsch

1 P (d) Führen Sie auf der folgenden Union-Find-Datenstruktur zuächst $\textsc{Union}(a, c)$ und danach $\textsc{Union}(\textsc{Find}(f), b)$ aus. Benutzen Sie das Verfahren "Vereinigung nach Höhe", und zeichnen Sie die nach diesen zwei Operationen resultierende Datenstruktur.

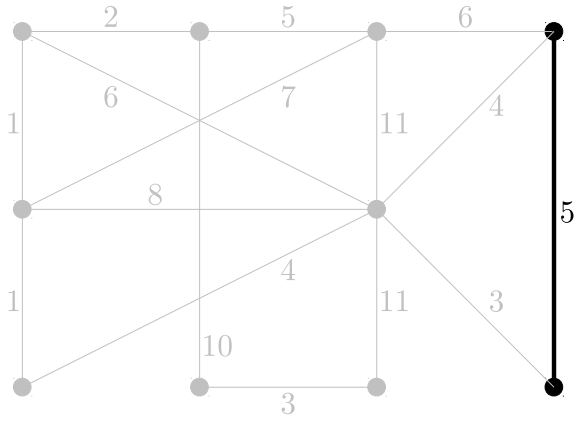
Nach UNION(a, c):



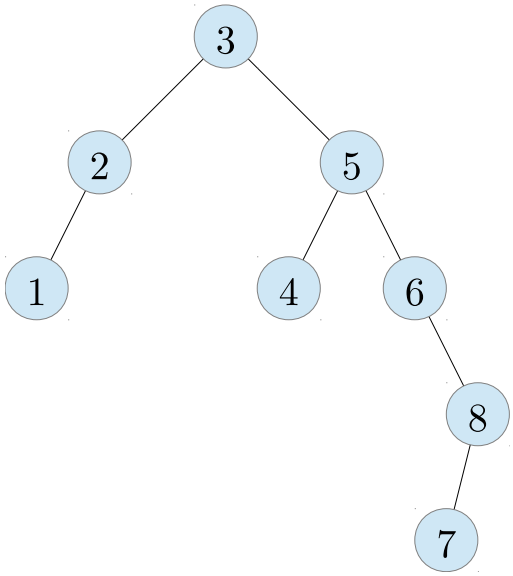
Nach UNION(FIND$(f), b)$:



1 P (e) Markieren Sie im untenstehenden gewichteten Graphen die erste Kante, die der Algorithmus von Kruskal *nicht* in den minimalen Spannbaum aufnimmt.



1 P (f) Zeichnen Sie den binären Suchbaum zur Schlüsselmenge $\{1, \ldots, 8\}$, dessen Preorder-Reihenfolge mit $3, 2, 1, 5, 4$ beginnt, und dessen Postorder-Reihenfolge mit $7, 8, 6, 5, 3$ endet.



Aufgabe 2.

1 P (a) 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 $f \in \mathcal{O}(g)$.

- $\frac{n^2}{\log n}$
- $\binom{n}{2}$
- $n \log n$
- $n\sqrt{n}$
- $2^{\sqrt{n}}$
- $\log(n^2)$

Lösung: Es gilt $\log(n^2) = 2\log(n) = \Theta(\log n)$. Wegen $\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0$ ist $n \log n \in o(n\sqrt{n})$ und damit auch insbesondere $n \log n \in \mathcal{O}(n\sqrt{n})$. 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 weiterhin $n\sqrt{n} \in \mathcal{O}(\frac{n^2}{\log n})$. Die einzige gültige Reihenfolge ist daher:

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

3 P (b) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 6 + 7T(n/3) & n > 1 \\ 6 & n = 1 \end{cases}$$

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 3 ist, gilt $n = 3^k$ für ein $n \in \mathbb{N}$. Wir teleskopieren, um auf eine Formel für $T(n)$ zu kommen:

$$\begin{aligned} T(n) &= 6 + 7T(n/3) \\ &= 6 + 7(6 + 7T(n/3^2)) = 6 + 7 \cdot 6 + 7^2 T(n/3^2) \\ &= 6 + 7 \cdot 6 + 7^2(6 + 7T(n/3^3)) = 6 + 7 \cdot 6 + 7^2 \cdot 6 + 7^3 T(n/3^3) \\ &= \ldots = 6 \sum_{i=0}^{k-1} 7^i + 7^k \cdot 6 = 6 \sum_{i=0}^{k} 7^i = 6 \cdot \frac{7^{k+1} - 1}{7 - 1} = 7^{k+1} - 1. \end{aligned}$$

Wir beweisen nun unsere Annahme durch vollständige Induktion über k.

Induktionsverankerung ($k = 0$): Es gilt $T(3^0) = T(1) = 6 = 7^{0+1} - 1$.

Induktionsannahme: Für ein $k \in \mathbb{N}_0$ sei $T(3^k) = 7^{k+1} - 1$.

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

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

1 P (c) 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*n; i += 10) {
2         for(int j = 1; j*j <= n; j ++)
3                 ;
4 }
```

Lösung: Die innere Schleife wird genau $\lceil\sqrt{n}\rceil$ Mal durchlaufen. Aufgrund der äusseren Schleife wird sie genau $n^2/10$ Mal aufgerufen, und die Gesamtlaufzeit beträgt $\Theta(n^2\sqrt{n})$.

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

Lösung: Die innere Schleife wird genau n Mal durchlaufen. Die äussere Schleife wird aufgrund der Halbierung des Werts von i genau $\log_2(n)$ Mal durchlaufen. Damit beträgt die Laufzeit $\Theta(n \log n)$.

4 P (e) Wir betrachten einen Dezimalzähler, der mit 0 initialisiert wird und nur eine einzige Operation ERHÖHE unterstützt, die den Wert des Zählers um 1 erhöht. Die Kosten für diese Operation entsprechen genau der Anzahl der Stellen, die verändert werden. Hat der Zähler z.B. den Wert 18 und wird ERHÖHE aufgerufen, dann ist der neue Wert des Zählers 19, und die Kosten für die Operation betragen 1 (nur die 8 wurde verändert). Hat der Zähler den Wert 19999 und wird ERHÖHE aufgerufen, dann ist der Wert des Zählers 20000, und die Operation hatte Kosten 5.

Zeigen Sie nun mittels amortisierter Analyse, dass die Kosten der Operation ERHÖHE amortisiert $\mathcal{O}(1)$ sind. Definieren Sie dazu eine geeignete Kontostandsfunktion Φ, und geben Sie jeweils die realen und die amortisierten Kosten an, wenn die letzten $k \geq 0$ Stellen des Zählers den Wert 9 haben.

Lösung: Wir definieren die Kontostandsfunktion

$$\Phi_i := \text{Anzahl der Vorkommen von 9 im Zähler nach } i\text{-maligem Aufruf von ERHÖHE.}$$

Mit dieser Definition ist $\Phi_0 = 0$.

Haben die letzten k Stellen des Zählers den Wert 9 und wird die Operation ERHÖHE aufgerufen, dann betragen die realen Kosten genau $a_i = k+1$ (die k Stellen am Ende werden auf 0 gesetzt,

und die davorliegende Stelle um 1 erhöht). Der Wert der Kontostandsfunktion fällt dann um k oder $k-1$ (denn der Zähler hat nach Aufruf von ERHÖHE k Neunen am Ende weniger, dafür kann aber die davorliegende Stelle neu zur Neun geworden sein). Wir haben damit

$$\Phi_i = \begin{cases} \Phi_{i-1} - k + 1 & \text{falls die } (k+1)\text{-letzte Stelle nach ERHÖHE Wert 9 hat} \\ \Phi_{i-1} - k & \text{ansonsten} \end{cases}$$

Folglich ist nun $\Phi_i - \Phi_{i-1} \leq 1 - k$. Nach Definition betragen die amortisierten Kosten genau

$$a_i + \Phi_i - \Phi_{i-1} \leq k + 1 + 1 - k = 2,$$

sie sind also konstant.

Aufgabe 3.

Ein *Palindrom* ist eine Zeichenkette, die sich von vorne wie von hinten gleich liest, also z.B. das Wort `RENTNER`. Formal ist ein Palindrom eine Zeichenkette $\langle a_1, \ldots, a_n \rangle$, wobei entweder $n = 1$ gilt, oder aber es sind $a_1 = a_n$ und $\langle a_2, \ldots, a_{n-1} \rangle$ ein Palindrom. Ein Array $A[1 \ldots n]$ speichere eine Zeichenkette der Länge n. Ein Teilarray $A[i \ldots j]$, $1 \leq i \leq j \leq n$, heisst *Palindrom in* A, falls $\langle A[i], \ldots, A[j] \rangle$ ein Palindrom ist.

Beispiel: Das Array [`L`, `A`, `R`, `A`] enthält die Palindrome `A`, `R`, `L` sowie `ARA` (das Palindrom `A` kommt doppelt vor). Das Array [`A`, `N`, `N`, `A`] enthält die Palindrome `A`, `N`, `NN` sowie `ANNA` (die Palindrome `A` und `N` kommen doppelt vor).

9 P (a) Sei A ein Array, das eine Zeichenkette der Länge n speichert. Entwerfen Sie einen Algorithmus nach dem Prinzip der *dynamischen Programmierung*, der alle Paare (i, j) ausgibt, für die $\langle A[i], \ldots, A[j] \rangle$ ein Palindrom ist.

Lösung:

Definition der DP-Tabelle: Wir verwenden eine $n \times n$-Tabelle T mit Einträgen, die entweder 0 oder 1 sind. Für $1 \leq i \leq j \leq n$ sei genau dann $T[i, j] = 1$, wenn $\langle A[i], \ldots, A[j] \rangle$ ein Palindrom ist.

Berechnung eines Eintrags: Wir unterscheiden drei Fälle.

1. *Fall:* $1 \leq i = j \leq n$. $A[i]$ ist ein Palindrom der Länge 1, also setzen wir $T[i, i] = 1$ für alle i, $1 \leq i \leq n$.
2. *Fall:* $1 \leq i \leq n$, $j = i + 1 \leq n$. Dann betrachten wir Palindrome der Länge 2, und wir setzen $T[i, i + 1] = 1$ genau dann, wenn $A[i] = A[i + 1]$ gilt.
3. *Fall:* $1 \leq i \leq n$, $i + 1 < j \leq n$. Wir betrachten nun die Zeichenkette $\langle A[i], \ldots, A[j] \rangle$. Diese ist nach Definition genau dann ein Palindrom, wenn $A[i] = A[j]$ gilt und zusätzlich $\langle A[i + 1], \ldots, A[j - 1] \rangle$ ein Palindrom ist. Wir setzen also genau dann $T[i, j] = 1$, wenn $A[i] = A[j]$ gilt und $T[i + 1, j - 1] = 1$ ist.

Berechnungsreihenfolge: Wir berechnen die Einträge $T[i, j]$ mit wachsender Differenz von $j - i$, starten also bei $T[i, i]$ für $1 \leq i \leq n$. Danach fahren wir fort mit der Berechnung von $T[i, i + 1]$ für $1 \leq i \leq n - 1$, danach mit $T[i, i + 2]$ für $1 \leq i \leq n - 2$, usw. Schlussendlich wird $T[1, n]$ berechnet.

Auslesen der Lösung: Wir iterieren über alle Einträge $T[i, j]$, $1 \leq i \leq j \leq n$, und geben genau dann (i, j) aus, wenn $T[i, j] = 1$ ist.

1 P (b) Geben Sie die Laufzeit Ihrer Lösung an.

Lösung: Die Tabelle hat n^2 Einträge. Die Berechnung jedes Eintrags geht in Zeit $\mathcal{O}(1)$. Zum Auslesen der Lösung wird jeder Eintrag der Tabelle erneut betrachtet, und auch dort fällt nur Zeit $\mathcal{O}(1)$ pro Eintrag an. Die Gesamtlaufzeit beträgt daher $\mathcal{O}(n^2)$.

3 P (c) Angenommen, der Algorithmus aus (a) hat die DP-Tabelle bereits berechnet. Beschreiben Sie detailliert, wie Sie aus der DP-Tabelle ein längstes Palindrom in A ablesen können. Geben Sie auch die maximal benötigte Laufzeit an.

Lösung: Wir betrachten die Einträge der Tabelle in umgekehrter Berechnungsreihenfolge, d.h. wir starten mit $T[1, n]$, betrachten danach $T[1, n-1]$ und $T[2, n]$, usw. Sobald wir einen Eintrag $T[i, j]$ mit Wert 1 finden, terminiert unser Verfahren, und wir geben $\langle A[i], \ldots, A[j] \rangle$ aus.

Wie vorher fällt bei der Betrachtung eines Eintrags nur Zeit $\mathcal{O}(1)$ an. Da es n^2 Einträge gibt und das ausgegebene Palindrom maximal n Zeichen lang ist, ergibt sich wie vorher eine Gesamtlaufzeit von $\mathcal{O}(n^2)$.

Aufgabe 4.

Gegeben sei ein $n \times n$-Gitter, d.h. ein ungerichteter Graph mit n Zeilen und n Spalten, der Kanten zwischen je zwei horizontal sowie vertikal benachbarten Knoten enthält. Beim *Fluchtwegeproblem* sind $m \leq n^2$ Startknoten $s_1, \ldots, s_m$ gegeben. Wir möchten entscheiden, ob es m kantendisjunkte Wege von den Startpunkten s_i zu je einem Randpunkt des Gitters gibt. Beachten Sie, dass je zwei Wege zwar einen oder mehrere Knoten, aber keine gemeinsamen Kanten besitzen dürfen.

4 P (a) Modellieren Sie das obige Entscheidungsproblem als Flussproblem. Beachten Sie, dass wir bei Flussproblemen *gerichtete* Netzwerke betrachten, das Gitter selbst aber ungerichtet ist. Erläutern Sie daher genau, wie ein geeignetes Netzwerk $N = (V, E, c)$ konstruiert werden kann, d.h. welche Knoten V und welche Kanten E definiert werden, und welche Kapazitäten den Kanten zugewiesen werden müssen. Überlegen Sie, wie Sie aus dem Wert eines maximalen Flusses schlussfolgern können, ob m kantendisjunkte Fluchtwege existieren oder nicht.

Lösung: Sei $G = (V_G, E_G)$ das ungerichtete Gitter, das als Eingabe gegeben ist. Die Knotenmenge V des Netzwerks N enthält neben allen Knoten aus G zusätzlich noch eine Quelle s und eine Senke t. Die Kantenmenge von N enthält nun die folgenden Kanten:

- Für jeden Startknoten s_i enthält N die Kante (s, s_i).
- Für jede (ungerichtete) Kante $\{v, w\} \in E_G$ des Gitters enthält N die zwei gerichteten Kanten (v, w) und (w, v).
- Für jeden Randpunkt r_j enthält N die Kante (r_j, t).

Wir konstruieren also ein Netzwerk $N = (V, E, c)$ mit

$$
\begin{aligned}
V &:= V_G \cup \{s\} \cup \{t\} \\
E &:= \{(s, s_i) \mid i \in \{1, ..., m\}\} \cup \{(v, w), (w, v) \mid \{v, w\} \in E_G\} \cup \\
&\qquad \{(r_j, t) \mid r_j \text{ ist Randpunkt von } G\} \\
c((s, s_i)) &:= 1 \; \forall i \in \{1, ..., m\} \\
c((v, w)) &:= 1 \; \forall \{v, w\} \in E_G \\
c((r_j, t)) &:= 2 \; \forall \text{ Randpunkte } r_j
\end{aligned}
$$

Man beachte, dass ein Randpunkt doppelt benutzt werden kann. Einerseits kann er der letzte Knoten eines Fluchtwegs sein, der im Inneren des Gitters startet. Andererseits kann es aber auch einen Fluchtweg geben, der nur aus genau diesem Randpunkt besteht. Aus diesem Grund muss als Kapazität der Kanten (r_j, t) genau 2 gewählt werden (und nicht 1). Es gibt nun in G genau dann m kantendisjunkte Fluchtwege, falls es im obigen Netzwerk N einen Fluss mit Wert mindestens m gibt.

2 P (b) Nennen Sie einen Flussalgorithmus, der das Problem aus (a) möglichst effizient löst. Geben Sie die Laufzeit der Lösung aus (a) in Abhängigkeit von n und m an.

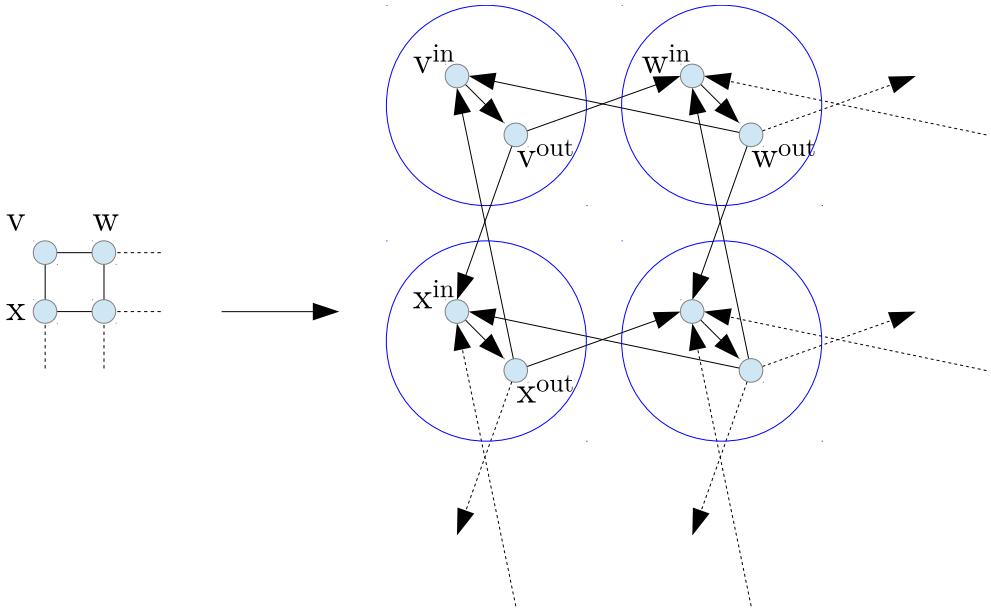
Lösung: Wir analysieren zunächst die Grösse des Netzwerks in Abhängigkeit von n. Das Gitter G hat $|V_G| = n^2$ viele Knoten, also ist $|V| = 2 + |V_G| = \Theta(n^2)$. Ausserdem hat G genau $|E_G| = 2n(n-1) \leq 2n^2$ viele Kanten und $2n + 2(n-2) = 4n - 4$ viele Randpunkte. Da jede Kante von G in N doppelt vorkommt, ist $|E| = m + 2|E_G| + 4n - 4 \leq n^2 + 2n^2 + 4n = \Theta(n^2)$. Der Wert eines Flusses ist sowohl durch m (die Anzahl der Startknoten) also auch durch $4n - 4$

nach oben beschränkt, da es nur so viele Randpunkte hat (zwar können zwei verschiedene Fluchtwege im gleichen Randpunkt enden, dann muss aber mindestens einer von ihnen einen weiteren Randpunkt besuchen).

Es gibt nun verschiedene Flussalgorithmen zur Lösung. Beispielsweise berechnet der Algorithmus von Ford und Fulkerson einen maximalen Fluss mit $\mathcal{O}(|E| \cdot \phi^*)$ Operationen, wenn ϕ^* der maximale Flusswert ist. Damit ist seine Laufzeit pseudopolynomiell, und im Allgemeinen existieren effizientere Methoden zur Flussmaximierung. Da aber hier $|E| = \Theta(n^2)$ als auch $\phi^* = \min\{m, 4n-4\} = \Theta(\min\{m, n\})$ gelten, beträgt die Laufzeit des Algorithmus von Ford und Fulkerson nur $\mathcal{O}(n^2 \min\{m, n\})$, und die Wahl dieses Verfahrens ist *in diesem speziellen Fall* optimal.

3 P (c) Wir möchten nun entscheiden, ob es eine Menge von m *knotendisjunkten* Fluchtwegen gibt, d.h. eine Menge von Fluchtwegen, von denen keine zwei einen gemeinsamen Knoten besitzen. Beschreiben Sie detailliert, wie Sie Ihre Lösung aus (a) modifizieren müssen, damit durch jeden Knoten maximal ein Fluchtweg läuft. Verändert sich die Laufzeit Ihrer Lösung, und wenn ja, wie?

Lösung: Sei $v \in V_G$ ein Knoten des Gitters und $N(v) = \{w \in V_G \mid \{v, w\} \in E_G\}$ die Menge aller Nachbarknoten von v in G. Wir erzeugen nun im Netzwerk N für v genau zwei Knoten v^{in} und v^{out}, die durch die gerichtete Kante $(v^{\text{in}}, v^{\text{out}})$ verbunden werden. Abgesehen von dieser Kante enthält v^{in} nur eingehende und v^{out} nur ausgehende Kanten. Für jeden Nachbar $w \in N(v)$ erzeugen wir jeweils die Kanten $(v^{\text{out}}, w^{\text{in}})$ und $(w^{\text{out}}, v^{\text{in}})$.



Für alle Startknoten s_i erzeugen wir ausgehend von der Quelle s eine Kante (s, s_i^{in}). Ausserdem erzeugen wir von jedem Randpunkt r_j ausgehend eine Kante (r_j^{out}, t) zur Senke t.

Im Vergleich zu (a) wird die Anzahl der Knoten nahezu verdoppelt (die Quelle und die Senke werden nicht doppelt angelegt). Pro Knoten $v \in G$ wird dabei nur eine weitere Kante angelegt, nämlich die innere Kante $(v_{\text{in}}, v_{\text{out}})$. Damit sind wie vorher $|V| = \Theta(n^2)$ und $|E| = \Theta(n^2)$, und die Laufzeit verändert sich nicht.

Aufgabe 5.

Wir betrachten eine Menge von n Kühen sowie ein Gehege aus m geradlinigen orthogonalen Zaunsegmenten. Die Position jeder Kuh i, $1 \leq i \leq n$, kann mittels eines GPS-Empfängers bestimmt werden und ist durch den Punkt $K_i \in \mathbb{Q}^2$ gegeben. Das Gehege ist ein geschlossenes einfaches Polygon mit den Eckpunkten (also den Zaunpfählen) $P_i \in \mathbb{Q}^2$, $1 \leq j \leq m$. Dabei seien P_i und P_{i+1} für $i \in \{1, \ldots, m-1\}$ und zusätzlich P_m und P_1 jeweils durch ein Zaunsegment verbunden. Weiterhin sei P_1 der am weitesten links liegende Punkt (falls es mehr als nur einen solchen Punkt gibt, dann sei P_1 unter diesen der am weitesten unten liegende Punkt). Sie dürfen davon ausgehen, dass sich keine Kuh direkt auf einem Zaunsegment befindet, d.h. kein Punkt K_i liegt auf einer Polygonkante. Wir wollen ermitteln, wie viele Kühe sich innerhalb des Geheges befinden.

9 P (a) Wir nehmen der Einfachheit halber an, dass alle Zaunsegmente parallel zur x- oder zur y-Achse verlaufen. Geben Sie einen Scanline-Algorithmus an, der als Eingabe die Positionen der Kühe K_i sowie die Positionen der Zaunpfähle P_j erhält, und die Anzahl der Kühe berechnet, die sich innerhalb des Geheges befinden.

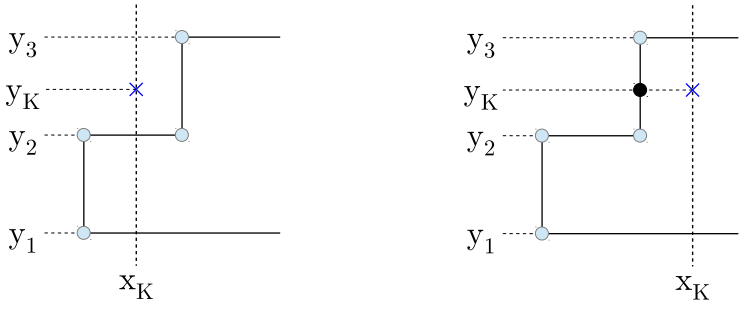
Lösung:

Haltepunkte: Wir benutzen eine vertikale Scanline, die von links nach rechts läuft. Diese hält jedes Mal, wenn sie ein vertikales Zaunsegment oder eine Kuh gefunden wird.

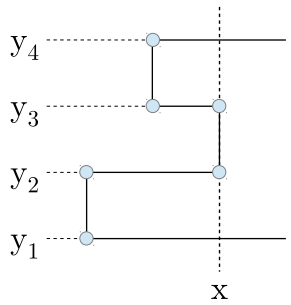
Scanline-Datenstruktur: Als Datenstruktur benutzen wir einen Intervallbaum. Dieser speichert genau die Intervalle, die der Schnitt der Scanline mit dem Gehege bildet. Dabei haben je zwei dieser Intervalle einen leeren Schnitt, d.h. es gibt im Intervallbaum keine zwei Intervalle, die direkt aneinander grenzen (diese Eigenschaft ermöglicht eine effiziente Aktualisierung). Desweiteren benutzen wir einen Zähler, der die Anzahl der Kühe im Gehege zählt, und der mit 0 initialisiert wird.

Aktualisierung: Hier unterscheiden wir drei Fälle.

1. *Fall: Der Haltepunkt repräsentiert eine Kuh.* Sei (x_K, y_K) die Position der Kuh. Dann prüfen wir mit einer Aufspiessanfrage, ob es ein Intervall gibt, das von y_K aufgespiesst wird. Falls ja, dann befindet sich die Kuh innerhalb des Geheges und wir erhöhen den Zähler um 1. Ansonsten befindet sich die Kuh ausserhalb des Geheges.

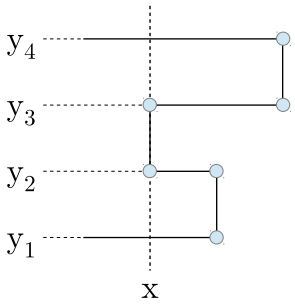


2. *Fall: Der Haltepunkt repräsentiert ein Zaunsegment links vom Gehege.* Sei $[y_u, y_o]$ das Intervall, das der Schnitt der Scanline mit dem Zaunsegment bildet. Wir können $[y_u, y_o]$ nicht direkt in den Segmentbaum einfügen, falls andere Intervalle direkt angrenzen. Die folgende Abbildung beschreibt z.B. eine Situation, in der $[y_2, y_3]$ nicht eingefügt werden kann, da $[y_1, y_2]$ und $[y_3, y_4]$ direkt angrenzen.



Wir führen zunächst eine Aufspiessanfrage nach y_u und y_o durch und ermitteln so, ob und welche Intervalle unten bzw. oben an $[y_u, y_o]$ angrenzen. Gegebenenfalls entfernen wir diese Intervalle und fügen danach das gesamte Intervall an, welches alle vorher entfernten Intervalle umfasst. In der obigen Situation werden $[y_1, y_2]$ und $[y_3, y_4]$ entfernt, und $[y_1, y_4]$ eingefügt.

3. Fall: Der Haltepunkt repräsentiert ein Zaunsegment rechts vom Gehege. Wir gehen analog zum vorigen Fall vor. Sei $[y_u, y_o]$ das Intervall, das der Schnitt der Scanline mit dem Zaunsegment bildet. Wir ermitteln zunächst mittels einer Aufspiessanfrage das zugehörige gespeicherte Intervall $[y'_u, y'_o]$, das $[y_u, y_o]$ umfasst, entfernen dieses und fügen $[y'_u, y_u]$ sowie $[y'_o, y_o]$ ein, sofern diese Intervalle nicht leer sind. In der untenstehenden Abbildung wird also das Segment $[y_1, y_4]$ entfernt und dafür $[y_1, y_2]$ sowie $[y_3, y_4]$ eingefügt.



Auslesen der Lösung: Es wird der Wert des Zählers ausgegeben.

1 P (b) Geben Sie die Laufzeit des Verfahrens aus (a) an.

Lösung: Das Sortieren der Kuh-Positionen kostet Zeit $\mathcal{O}(n \log n)$, das Sortieren der Polygon-Kanten kostet Zeit $\mathcal{O}(m \log m)$. Insgesamt gibt es $n+m$ Haltepunkte, und an jedem wird Zeit $\mathcal{O}(\log m)$ benötigt (dies entspricht genau der Laufzeit einer Aufspiessanfrage sowie der zum Einfügen und Entfernen von Intervallen benötigte Zeit). Insgesamt wird daher Zeit $\mathcal{O}(n \log n + m \log m)$ benötigt.