Name:

Vorname:

Matrikelnummer:

Klausur-ID:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders 15.03.2021

Klausur Algorithmen II

Aufgabe 1.	Kleinaufgaben	11 Punkte
Aufgabe 2.	Randomisierte Algorithmen: Treaps	9 Punkte
Aufgabe 3.	Approximationsalgorithmen: Steinerbäume	9 Punkte
Aufgabe 4.	Flussalgorithmen: Mäusejagd	9 Punkte
Aufgabe 5.	FPT Algorithmen: Dominating Set	13 Punkte
Aufgabe 6.	Geometrische Algorithmen: Regenschauer	9 Punkte

Bitte beachten Sie:

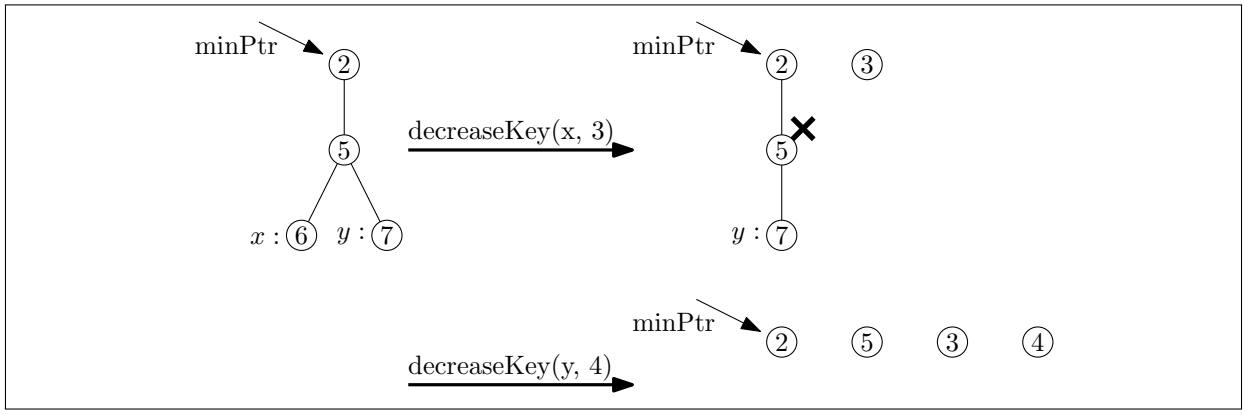
- Als Hilfsmittel ist nur **ein** DIN-A4 Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- **Schreiben** Sie auf **alle** Blätter der Klausur und Zusatzblätter Ihre **Klausur-ID**.
- Merken Sie sich Ihre **Klausur-ID** auf dem Aufkleber für den Notenaushang.
- Die Klausur enthält **16 Blätter**.
- Zum Bestehen der Klausur sind 20 Punkte hinreichend.

Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [11 Punkte]

a. Führen Sie die Operationen `decreaseKey`(x, 3) und `decreaseKey`(y, 4) auf dem unten abgebildeten Fibonacci Heap nacheinander aus und geben Sie nach jeder Operation den aktuellen Zustand an. [2 Punkte]

Lösung



b. Für ein Problem gebe es einen parallelen Algorithmus mit Ausführungszeit $T_{\text{par}}(n,p) = \frac{n^2(\log n)^2}{\sqrt{p}}$. Die Laufzeit des besten sequenziellen Algorithmus für das Problem sei $T_{\text{seq}}(n) = n^2 \log n$. Geben Sie die Effizienz und die Arbeit des parallelen Algorithmus an. [2 Punkte]

Lösung

Es ist $E = \frac{1}{\sqrt{p}\log n}$ und $W = \sqrt{p}n^2(\log n)^2$.

c. Sei $G = (V,E,\omega)$ ein gerichteter Graph mit Knotenmenge $V \subseteq \mathbb{N}^2$ und der Manhattan-Metrik als Kantengewichtsfunktion: $\omega((x_1,y_1),(x_2,y_2)) = |(x_1,y_1)-(x_2,y_2)|_1 = |x_1-x_2|+|y_1-y_2|$. Zeigen Sie, dass $f(u) = |u-t|_1$ für einen beliebigen Zielknoten $t \in V$ eine gültige Potentialfunktion für den A* Algorithmus aus der Vorlesung ist.
Anmerkung: Die Bedingung $f(t) = 0$ ist offensichtlich erfüllt und gibt keinen Punkt.
Hinweis: $|\cdot|_1$ erfüllt die Dreiecksungleichung. [2 Punkte]

Lösung

- Konsistenz: für $e = (u,v) \in E$ ist $\omega(e) + f(v) = |u-v|_1 + |v-t|_1 \geq |u-t|_1 = f(u)$.
- Nicht überschätzen: für ein $u \in V$ sei $(u = v_1, v_2, \ldots, v_p = t)$ ein kürzester u-t-Pfad in G. Dann gilt

$$\mu(u,t) = \sum_{i=1}^{p-1} \omega(v_i, v_{i+1}) = \sum_{i=1}^{p-1} |v_i - v_{i+1}|_1 \geq \left| \sum_{i=1}^{p-1} v_i - v_{i+1} \right|_1 = |u-t|_1 = f(u).$$

- Zuletzt ist $f(t) = |t-t|_1 = 0$.

d. Geben Sie das Suffix-Array und das LCP-Array für die Zeichenkette `falafaf$` an. [2 Punkte]

Lösung

Indices	0	1	2	3	4	5	6	7
Text	f	a	l	a	f	a	f	$
Suffix-Array	7	5	3	1	6	4	0	2
LCP-Array	-1	0	2	1	0	1	2	0

e. Es ist Domino-Day! Verschiedene Teams haben in mehreren mühseligen Monaten tausende von Dominosteinen aufgestellt, um sie am Domino-Day zum Fall zu bringen. Leider haben sich die Teams nicht auf einen gemeinsamen Start-Dominostein geeinigt, sodass kein einzelner umgestoßener Stein alle anderen mit umwirft. In dieser Aufgabe nehmen wir an, dass ein Dominostein nur in eine Richtung umfallen kann. Die Anordnung der Dominosteine wird durch einen gerichteten Graph $G = (V, E)$ repräsentiert, wobei V die Menge der Dominosteine darstellt. Falls ein Dominostein $u \in V$ durch sein Fallen einen anderen Dominostein $v \in V$ umwirft, existiert in unserem Graph G eine Kante $(u, v) \in E$. Geben Sie einen Algorithmus an, welcher in einer Laufzeit von $\mathcal{O}(|V| + |E|)$ die minimale Anzahl an Dominosteinen ermittelt, welche durch händisches Anstoßen alle anderen Steine zu Fall bringt. [3 Punkte]

Lösung

Alternative 1:

1. Finde alle *SCC* ($\mathcal{O}(|V| + |E|)$)
2. Kontrahiere alle *SCC* und erhalte $G' = (V', E')$ ($\mathcal{O}(|V| + |E|)$)
3. Zähle alle Knoten $v \in V'$ mit $\mathit{indegree}(v) = 0$ ($\mathcal{O}(|V| + |E|)$)

Alternative 2:

1. Finde alle *SCC* und speichere für jeden Knoten $v \in V$ die *SCC* ID in einem Array `SCC` der Größe $|V|$ ($\mathcal{O}(|V| + |E|)$)
2. Iteriere über alle Kanten $e = (x, y) \in E$: Falls x und y in unterschiedlichen SCC, inkrementiere eine Zählervariable $C[\texttt{SCC}[y]]$ ($\mathcal{O}(|V| + |E|)$)
3. Zähle alle Einträge i in C mit $C[i] = 0$ (Achtung: i muss eine valide *SCC* ID sein)

Lösungsvorschlag

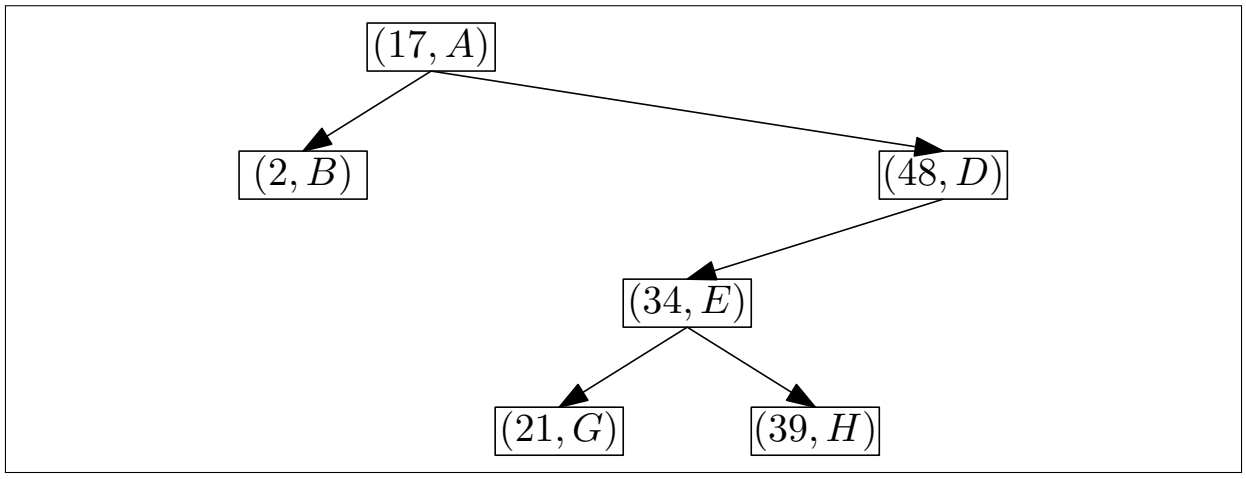
Aufgabe 2. Randomisierte Algorithmen: Treaps [9 Punkte]

Ein Treap ist ein binärer Suchbaum, in dem jeder Knoten x neben einem Schlüssel $k(x)$ auch eine Priorität $p(x)$ besitzt. Zur Einfachheit nehmen wir an, dass alle Schlüssel und Prioritäten eindeutig sind, d. h. es existieren keine $x \neq y$ mit $k(x) = k(y)$ oder $p(x) = p(y)$. Die Datenstruktur besitzt folgende Eigenschaften:

1. Jeder Knoten hat maximal Ausgangsgrad 2.
2. *Suchbaumeigenschaft:* Wenn x ein Knoten mit linkem Teilbaum $T_l = (V_l, E_l)$ bzw. rechtem Teilbaum $T_r = (V_r, E_r)$ ist, dann gilt $k(x_l) < k(x)$ für alle $x_l \in V_l$ und $k(x) < k(x_r)$ für alle $x_r \in V_r$.
3. *Heapeigenschaft:* Wenn x ein Knoten ist, dann gilt $p(x) < p(x')$ für alle Kinder x' von x. Wir sagen, dass x eine höhere Priorität als ein Knoten y hat, wenn $p(x) < p(y)$.

a. Geben Sie einen gültigen Treap mit folgenden Elementen an, wobei die Zahl jeweils den Schlüssel und der Buchstabe die Priorität (lexikographische Ordnung) angibt: $(2,B)$, $(17,A)$, $(21,G)$, $(34,E)$, $(39,H)$, $(48,D)$. [2 Punkte]

Lösung



b. Für einen Knoten v definieren wir seinen Rang $r(v)$ als dessen Position in der nach Schlüssel sortierten Liste aller Knoten. Seien x und y zwei Knoten mit Rängen $r(x)$ und $r(y)$. Begründen Sie, dass y genau dann in einem Teilbaum mit Wurzel x liegt, wenn x die höchste Priorität unter allen Knoten in $M = \{v \mid \min\{r(x), r(y)\} \leq r(v) \leq \max\{r(x), r(y)\}\}$ hat. [3 Punkte]

Lösung

Ohne Einschränkung sei $r(x) < r(y)$. Wir unterscheiden drei Fälle:

- y liegt in einem Teilbaum mit Wurzel x: sei $T_r = (V_r, E_r)$ der rechte Teilbaum von x. Da $r(x) < r(y)$ folgt $y \in V_r$, insbesondere aber $M \setminus \{x\} \subseteq V_r$. Also hat x auch die höchste Priorität unter allen Elemente in M.
- x liegt in einem Teilbaum mit Wurzel y: dann hat y eine höhere Priorität als x, x hat also nicht die höchste Priorität unter allen Elementen in M.
- Es gibt einen Knoten z mit Teilbäumen $T_l = (V_l, E_l)$ und $T_r = (V_r, E_r)$, und $x \in V_l$, $y \in V_r$. Dann ist $z \in X$, denn $k(x) < k(z) < k(y)$, und z hat eine höhere Priorität als x. Also hat x nicht die höchste Priorität unter allen Elementen in M.

Ab jetzt sind die Prioritäten der Elemente nicht Teil der Eingabe, sondern werden beim Einfügen eines Elements zufällig nach der stetigen Gleichverteilung $\mathscr{U}(0,1)$ gewählt. Dadurch ist die Wahrscheinlichkeit, dass zwei Elemente dieselbe Priorität haben, gleich 0.

c. Seien x und y Knoten mit Rängen $r(x)$ und $r(y)$. Geben Sie die Wahrscheinlichkeit p_{xy} an, dass y in einem Teilbaum mit Wurzel x liegt. [1 Punkt]

Lösung

Da die Elemente in M aus Teilaufgabe b nur durch ihren Schlüssel und unabhängig von ihrer Priorität ausgewählt werden, ist die Wahrscheinlichkeit, dass ein bestimmtes Element die kleinste Priorität besitzt gerade $p_{xy} = \frac{1}{|r(x)-r(y)|+1}$.

d. Zeigen Sie, dass die erwartete Tiefe eines Knotens in $\mathcal{O}(\log n)$ liegt, wobei n die Anzahl der Elemente im Treap ist.
Anmerkung: Die Tiefe eines Knotens v in einem Baum mit Wurzel w ist die Länge des Pfades von w nach v.
Hinweis: $H_n = \sum_{i=1}^{n} 1/i = \sum_{i=1}^{n} \frac{1}{n-i+1} \in \mathcal{O}(\log n)$. [3 Punkte]

Lösung

Sei x ein beliebiger Knoten. Für $y \neq x$ definieren wir die Indikatorvariable I_{yx} mit $\mathbb{E}[I_{yx}] = p_{yx}$ die angibt, ob x in einem Teilbaum mit Wurzel y liegt. Die erwartete Tiefe von x ist gerade die erwartete Anzahl solcher Knoten, also

$$\begin{aligned}
\mathbb{E}[\text{Tiefe}(x)] &= \sum_{y \in V \setminus \{x\}} \mathbb{E}[I_{yx}] = \sum_{y \in V \setminus \{x\}} p_{yx} \\
&= \sum_{j=1}^{r(x)-1} \frac{1}{r(x)-j+1} + \sum_{j=r(x)+1}^{n} \frac{1}{j-r(x)+1} \\
&= \sum_{k=2}^{r(x)} \frac{1}{k} + \sum_{k=2}^{n-r(x)+1} \frac{1}{k} \\
&= H_{r(x)} - 1 + H_{n-r(x)+1} - 1 \\
&\in \mathcal{O}(\log r(x) + \log(n - r(x) + 1)) \subseteq \mathcal{O}(\log n).
\end{aligned}$$

Lösungsvorschlag

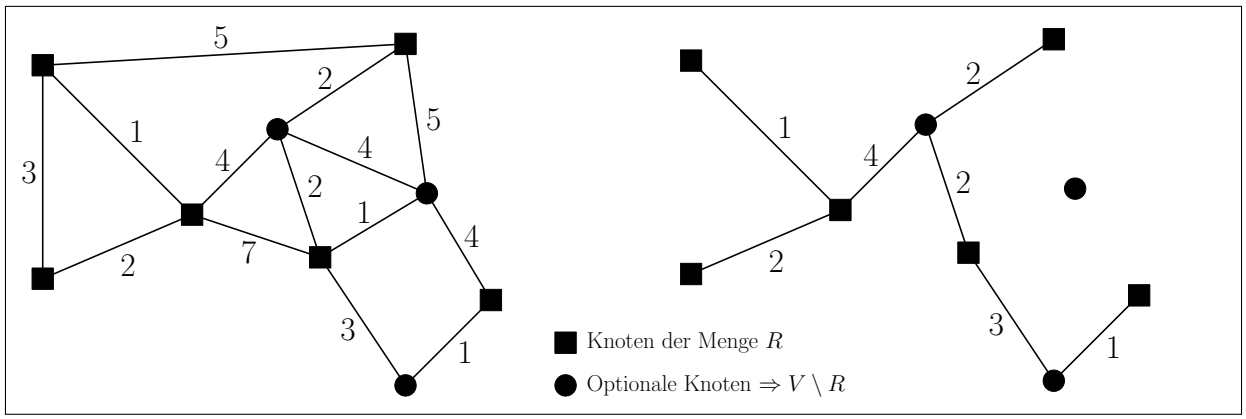
Aufgabe 3. Approximationsalgorithmen: Steinerbäume [9 Punkte]

Gegeben sei ein ungerichteter zusammenhängender gewichteter Graph $G = (V, E, \omega)$ mit Kantengewichtsfunktion $\omega : E \to \mathbb{R}_{>0}$ und eine Teilmenge an Knoten $R \subseteq V$. Wir definieren das *Steinerbaumproblem* für eine Probleminstanz (G, R) wie folgt: Finde eine Teilmenge $T \subseteq E$, welche alle Knoten in R verbindet, mit minimalem Gesamtkantengewicht $\omega(T) \coloneqq \sum_{e \in T} \omega(e)$. Eine Lösung $T \subseteq E$ ist *gültig*, falls Sie alle Knoten in R verbindet.

Anmerkung: Falls $R = V$, dann ist die optimale Lösung für das Steinerbaumproblem ein minimaler Spannbaum. Falls $R \subset V$, dann können Kanten, welche Knoten aus $V \setminus R$ verbinden (auch *Steinerknoten* genannt), optional in der Lösungsmenge enthalten sein. Dadurch wird das Problem NP-vollständig.

a. Gegeben sei der unten abgebildete ungerichtete gewichtete Graph $G = (V, E, \omega)$. Die Zahl neben einer Kante $e \in E$ ist ihr jeweiliges Gewicht $\omega(e)$. Die rechteckigen Knoten repräsentieren die Menge R. Die kreisförmigen Knoten sind optionale Knoten. Markieren Sie in der unten abgebildeten Lösungsskizze alle Kanten die in einer optimalen Lösung für die Steinerbauminstanz (G, R) enthalten sind. [2 Punkte]

Lösung



Wir definieren den metrischen Abschluss $G_R = (R, E_R, d)$ von G mit $E_R := \{\{u,v\} \mid u,v \in R\}$, welcher nur die Knoten aus R enhält sowie die Kantengewichtsfunktion $d : E_R \to \mathbb{R}_{>0}$, wobei $d(u,v)$ die Länge des kürzesten Pfades zwischen u und v in G ist. Für eine Teilkantenmenge $T \subseteq E_R$ definieren wir $d(T) := \sum_{\{u,v\} \in T} d(u,v)$.

b. Geben Sie einen Algorithmus an, welcher aus einem minimalen Spannbaum T_{MST} von G_R, einen Baum T berechnet mit $\omega(T) \leq d(T_{MST})$, welcher eine gültige Lösung für die Steinerbauminstanz (G,R) ist. [2 Punkte]

Lösung

Für jede Kante $\{u,v\} \in T_{MST}$ fügen wir die Kanten des kürzesten Pfades zwischen u und v in G zu T' hinzu. Es gilt: $\omega(T') \leq d(T_{MST})$. Ein minimaler Spannbaum T von $G' = (V,T')$ ergibt eine gültige Lösung für die Steinerbauminstanz (G,R) mit $\omega(T) \leq \omega(T') \leq d(T_{MST})$.

c. Wir definieren folgenden Algorithmus:

1. Berechne minimalen Spannbaum T_{MST} von G_R.
2. Berechne mit Teilaufgabe b eine gültige Lösung T für die Steinbauminstanz (G,R) mit $\omega(T) \leq d(T_{MST})$.

Zeigen Sie, dass T eine 2-Approximation für die Steinerbauminstanz (G,R) ist.
Hinweis: Verwenden Sie einen ähnlichen Beweis wie zur Berechnung der 2-Approximation für das metrische *Handlungsreisendenproblem* aus der Vorlesung. [5 Punkte]

Lösung

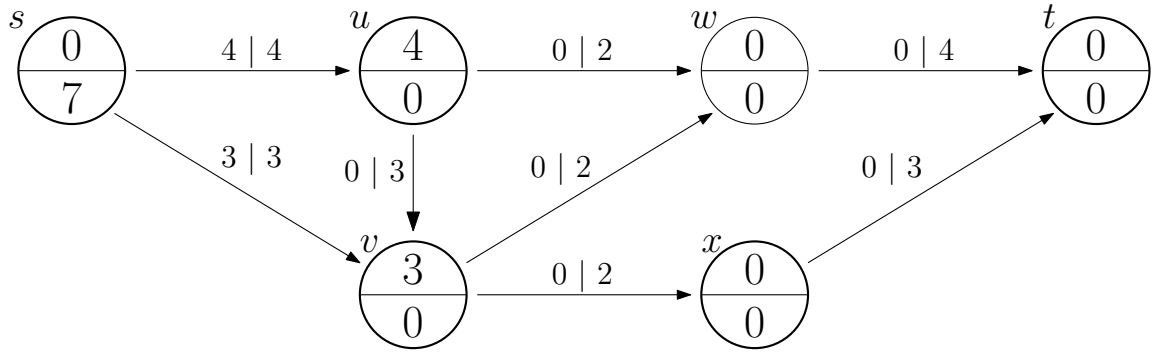
Sei T_{OPT} die optimale Lösung für die Steinerbauminstanz (G,R). Wir verdoppeln alle Kanten von T_{OPT} und erhalten T'_{OPT} mit $\omega(T'_{OPT}) = 2\omega(T_{OPT})$. Anschließend berechnen wir eine Eulertour T_{Euler} die jede Kante aus T'_{OPT} genau einmal besucht. Es gilt: $\omega(T_{Euler}) = 2\omega(T_{OPT})$. Eine solche Tour existiert, da die Knoten des Graphen $G = (V, T'_{OPT})$ alle geraden Grad besitzen. Anschließend entfernen wir aus T_{Euler} alle Knoten die nicht in R enthalten sind und erhalten T'_{Euler}. Für diese Tour gilt $d(T'_{Euler}) \leq \omega(T_{Euler})$, da die Gewichte der Kanten in G_R gleich dem kürzesten Pfad zwischen zwei Knoten entsprechen. T'_{Euler} ist eine Eulertour in G_R. Es gilt:

$$\omega(T) \overset{b}{\leq} d(T_{MST}) \leq d(T'_{Euler}) \leq \omega(T_{Euler}) \leq 2\omega(T_{OPT})$$

Lösungsvorschlag

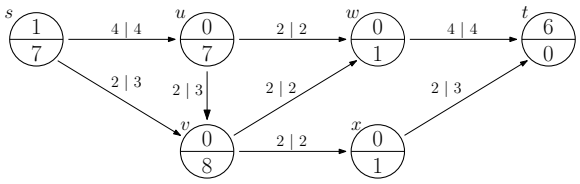
Aufgabe 4. Flussalgorithmen: Mäusejagd [9 Punkte]

a. Gegeben sei das folgende Flussnetzwerk mit Quelle s und Senke t. Es beschreibt den Zustand des preflow-push Algorithmus nach der Initialisierung. Die Knoten sind mit ihrem momentanen Überschuss im oberen und ihrem Distanzlabel im unteren Halbkreis beschriftet. Die Kanten sind jeweils mit ihrem aktuellen Flusswert (links) und Kapazität (rechts) beschriftet. Ihre Aufgabe ist es, mittels des *generic preflow-push* Algorithmus, einen maximalen Fluss f zu berechnen. Geben Sie dafür eine Folge von gültigen `push(a,b)` und `relabel(a)` Operationen an. Für eine `push(a,b)`-Operation geben Sie bitte an, wie viel Fluss Sie über die Kante (a,b) schieben und für eine `relabel(a)`-Operation geben Sie bitte an, auf welchen Wert sich das Distanzlabel von Knoten a ändert.



[3 Punkte]

Lösung



1. `relabel(u)` $\rightarrow d(u) = 1$
2. `push(u,w)` $\rightarrow f(u,w) = 2$
3. `push(u,v)` $\rightarrow f(u,v) = 2$
4. `relabel(v)` $\rightarrow d(v) = 1$
5. `push(v,w)` $\rightarrow f(v,w) = 2$
6. `push(v,x)` $\rightarrow f(v,x) = 2$
7. `relabel(w)` $\rightarrow d(w) = 1$
8. `push(w,t)` $\rightarrow f(w,t) = 4$
9. `relabel(x)` $\rightarrow d(x) = 1$
10. `push(x,t)` $\rightarrow f(x,t) = 2$
11. `relabel(v)` $\rightarrow d(v) = 2$
12. `push(v,u)` $\rightarrow f(v,u) = 1$
13. `relabel(u)` $\rightarrow d(u) = 3$
14. `push(u,v)` $\rightarrow f(u,v) = 1$
15. `relabel(v)` $\rightarrow d(v) = 4$
16. `push(v,u)` $\rightarrow f(v,u) = 1$
17. ...
18. `relabel(v)` $\rightarrow d(v) = 8$
19. `push(v,s)` $\rightarrow f(v,s) = 1$

Ein Greifvogel fliegt über einem Feld, um Mäuse zu jagen. Auf dem Feld befinden sich n Mäuse. Eine Maus gilt als *sicher* vor dem Greifvogel, falls sie es schafft, sich innerhalb von x Sekunden in einem der $m \leq n$ vorhandenen Löcher zu verstecken. In ein Loch passt genau eine Maus und jede Maus rennt mit konstanter Geschwindigkeit v Metern pro Sekunde. Die Distanz von Maus i zu Loch j beträgt genau d_{ij} Meter ($1 \leq i \leq n$ und $1 \leq j \leq m$).

b. Geben Sie einen Algorithmus an, welcher mit Hilfe eines maximalen Flusses in einem Flussnetzwerk $G = (V, E, c, s, t)$ mit Kapazitätsfunktion $c : E \to \mathbb{R}_+$, Quelle s und Senke t, die maximale Anzahl an Mäusen bestimmt, welche sich vor dem Greifvogel in Sicherheit bringen können. Konstruieren Sie hierfür das Flussnetzwerk G und begründen Sie die Korrektheit Ihres Algorithmus. [4 Punkte]

Lösung

Es handelt sich hierbei um ein Zuordnungsproblem, welches man mit Hilfe eines maximalen Flusses in einem bipartiten Flussnetzwerk $G = (M \cup L, E, c, s, t)$ lösen kann, wobei $M = \{1, \ldots, n\}$ die Menge der Mäuse und $L = \{1, \ldots, m\}$ die Menge der Löcher darstellt. Eine Maus i kann sich in einem Loch j vor dem Greifvogel in Sicherheit bringen, falls $\frac{d_{ij}}{v} \leq x$. Das Flussnetwerk ist wie folgt definiert:

1. Von der Quelle s fügen wir zu jeder Maus $i \in M$ eine Kante (s, i) hinzu mit Kapazität $c(s, i) = 1$.
2. Von jedem Loch $j \in L$ fügen wir zur Senke t eine Kante (j, t) hinzu mit Kapazität $c(j, t) = 1$.
3. Von jeder Maus $i \in M$ fügen wir zum Loch $j \in L$ eine Kante (i, j) hinzu, falls $\frac{d_{ij}}{v} \leq x$, mit Kapazität $c(i, j) = \infty$ (oder $c(i, j) = 1$).

Da die Kapazität von der Quelle zu jeder Maus und von jedem Loch zur Senke genau 1 ist, kann jeder Maus nur maximal ein Loch zugeordnet werden und jedem Loch maximal eine Maus. Der Wert des maximalen Flusses f stellt die maximale Anzahl an Mäusen dar, welche sich vor dem Greifvogel in Sicherheit bringen können.

c. Geben Sie einen Algorithmus an, welcher eine Zuordnung der Mäuse zu den Löchern ausgibt, um eine maximale Anzahl an Mäusen vor dem Greifvogel in Sicherheit zu bringen. [1 Punkt]

Lösung

Die Zuordnung der Mäuse zu den Löchern, um eine maximale Anzahl an Mäusen vor dem Greifvogel in Sicherheit zu bringen, entspricht einem maximalen bipartiten Matching in dem bipartiten Graphen aus Teilaufgabe b. Die Kardinalität des maximalen bipartiten Matching ist gleich dem maximalen Fluss f in G. Wir ordnen Maus i zu dem Loch j zu, falls $f(i, j) = 1$.

d. Wie kann das Flussnetzwerk G aus Teilaufgabe b angepasst werden, falls sich in jedem Loch j genau l_j Mäuse vor dem Greifvogel in Sicherheit bringen können? [1 Punkt]

Lösung

Wir passen die Kapazitäten der Kanten von jedem Loch $j \in L$ zur Senke t an: $c(j, t) = l_j$.

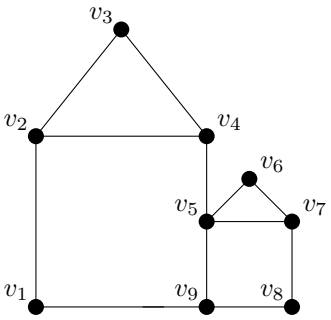
Lösungsvorschlag

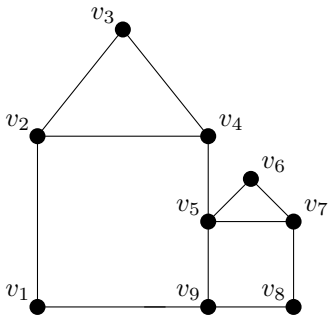
Aufgabe 5. FPT Algorithmen: Dominating Set [13 Punkte]

Gegeben sei ein ungerichteter Graph $G = (V,E)$. Eine Menge $D \subseteq V$ ist eine *dominierende Menge*, falls jeder Knoten in $V \setminus D$ zu einem Knoten in D adjazent ist. Das Ziel des Problems `Dominating Set` ist es, zu entscheiden, ob eine dominierende Menge D mit Kardinalität k existiert ($|D| = k$).

a. Markieren Sie im nachfolgenden Graphen eine dominierende Menge D mit kleinstmöglicher Kardinalität. [1 Punkt]

Zwei Kopien. Sollten Sie mehr als eine Kopie verwenden, machen Sie deutlich welche bewertet werden soll. Ansonsten wird die Teilaufgabe mit 0 Punkten bewertet.





Lösung

Zum Beispiel $D = \{v_2, v_7, v_9\}$.

Für allgemeine Graphen ist `Dominating Set` vermutlich nicht *fixed parameter tractable*. **Darum betrachten wir in der restlichen Aufgabe nur Graphen, die keine Kreise der Länge** 3 **und** 4 **enthalten.**

b. Sei $G = (V,E)$ ein ungerichteter Graph, der keine Kreise der Länge 3 und 4 enthält und $u \in V$ ein Knoten mit zwei Nachbarn v und w. Zeigen Sie, dass v und w nicht adjazent sind und u der einzige gemeinsame Nachbar von v und w ist. [2 Punkte]

Lösung

Die Nachbarn v und w können nicht adjazent sein, denn andernfalls wäre (u,v,w,u) ein Kreis der Länge 3. Wenn es noch einen weiteren gemeinsamen Nachbarn x von v und w gäbe, dann wäre (u,v,x,w,u) ein Kreis der Länge 4.

Der nachfolgende Reduktionsalgorithmus reduziert eine Probleminstanz (G,k) auf einen Kern oder gibt aus, dass es in G keine dominierende Menge D mit Kardinalität k gibt (Rückgabewert `false`). Dazu färbt der Algorithmus jeden Knoten *rot*, *schwarz* oder *weiß*, wobei zu Beginn alle Knoten schwarz gefärbt werden.

- Rote Knoten (Menge R) sind solche, die in die dominierende Menge D aufgenommen werden.
- Weiße Knoten (Menge W) sind solche, die zu einem roten Knoten adjazent sind.
- Schwarze Knoten (Menge S) sind alle verbleibenden Knoten.

Algorithmus 1 `reduce`$(G=(V,E),k)$

Färbe alle Knoten schwarz $(S \leftarrow V)$
while $\exists u \in S \cup W$: u hat mehr als $k-|R|$ schwarze Nachbarn **do**
  Färbe u rot $(R \leftarrow R \cup \{u\}, S \leftarrow S \setminus \{u\}, W \leftarrow W \setminus \{u\})$
  Färbe alle schwarzen Nachbarn v von u weiß $(W \leftarrow W \cup \{v\}, S \leftarrow S \setminus \{v\})$
  Lösche weiße Knoten w ohne schwarzen Nachbarn aus G $(V \leftarrow V \setminus \{w\}, W \leftarrow W \setminus \{w\})$
  if $|R| > k$ **then**
    return `false`
return $(G, k-|R|)$

c. Zeigen Sie für die Korrektheit des Reduktionsalgorithmus folgende Invariante: Wenn es eine dominierende Menge D mit Kardinalität k gibt, dann gilt zu jedem Zeitpunkt: $R \subseteq D$.
Hinweis: Verwenden Sie Teilaufgabe b. [3 Punkte]

Lösung

Sei D eine dominierende Menge mit Kardinalität k und $v \in S \cup W$ ein schwarzer oder weißer Knoten mit mehr als $k-|R|$ schwarzen Nachbarn. Wir zeigen, dass wenn $R \subseteq D$, dann auch $R \cup \{v\} \subseteq D$ (am Anfang gilt trivialerweise $\emptyset \subseteq D$).
Da D eine dominierende Menge ist, gibt es eine kleinste Teilmenge $D' \subseteq D$, die alle Knoten in S dominiert. Nach Definition von S ist $D' \cap R = \emptyset$. Da nach Voraussetzung $R \subseteq D$, folgt $|D'| = k-|R|$.
Angenommen, $v \notin D'$. Dann muss jeder schwarze Nachbar von v entweder in D' enthalten sein oder von einem Knoten in D' dominiert werden. Da die Nachbarn von v nicht adjazent sind und keine zwei Nachbarn von v durch den selben Knoten dominiert werden können (jeweils Teilaufgabe b), folgt daraus $|D'| > k-|R|$, ein Widerspruch.

d. Sei (G,k) eine Probleminstanz, für die eine dominierende Menge D der Größe k existiert. Zeigen Sie, dass der Reduktionsalgorithmus einen Kern mit maximal $\mathcal{O}(k^2)$ schwarzen Knoten berechnet. [2 Punkte]

Lösung

Da der Algorithmus alle weißen und schwarzen Knoten mit mehr als $k - |R|$ schwarzen Nachbarn rot färbt, hat jeder weiße oder schwarze Knoten maximal k schwarze Nachbarn. Da nach Voraussetzung ein Dominating Set D mit k Elementen existiert, müssen maximal k weiße oder schwarze Knoten alle anderen schwarzen Knoten dominieren. Deswegen kann es maximal $k^2 + k \in \mathcal{O}(k^2)$ schwarze Knoten geben.

e. Sei (G,k) eine Probleminstanz, für die eine dominierende Menge D der Größe k existiert. Zeigen Sie, dass der Reduktionsalgorithmus einen Kern mit maximal $\mathcal{O}(k^3)$ weißen Knoten berechnet.
Hinweis: Zeigen Sie, dass ein schwarzer Knoten zu maximal k weißen Knoten benachbart sein kann. [3 Punkte]

Lösung

Im Kern hat jeder weiße Knoten mindestens einen schwarzen Nachbarn, denn andernfalls hätte der Reduktionsalgorithmus den Knoten entfernt. Wenn wir nun zeigen, dass jeder schwarze Knoten maximal k weiße Nachbarn hat, können wir mit Teilaufgabe d die Anzahl der weißen Knoten gegen $k \cdot \mathcal{O}(k^2) = \mathcal{O}(k^3)$ nach oben abschätzen. Dazu betrachten wir einen schwarzen Knoten und dessen weißen Nachbarn. Jeder weiße Nachbar muss zu mindestens einem roten Knoten adjazent sein, denn andernfalls wäre der Nachbar nicht weiß. Nach Teilaufgabe b müssen die weißen Knoten aber zu verschiedenen roten Knoten adjazent sein. Da es aber nur maximal k viele rote Knoten geben kann (andernfalls bricht der Reduktionsalgorithmus ab), kann es auch nur k weiße Nachbarn geben.

f. Begründen Sie kurz, wieso `Dominating Set` eingeschränkt auf Graphen ohne Kreise der Längen 3 und 4 *fixed parameter tractable* ist.
Anmerkung: Der vorgestellte Reduktionsalgorithmus hat polynomielle Laufzeit.
Hinweis: Beachten Sie, dass Teilaufgaben d und e nur erfüllbare Probleminstanzen betrachten.
[2 Punkte]

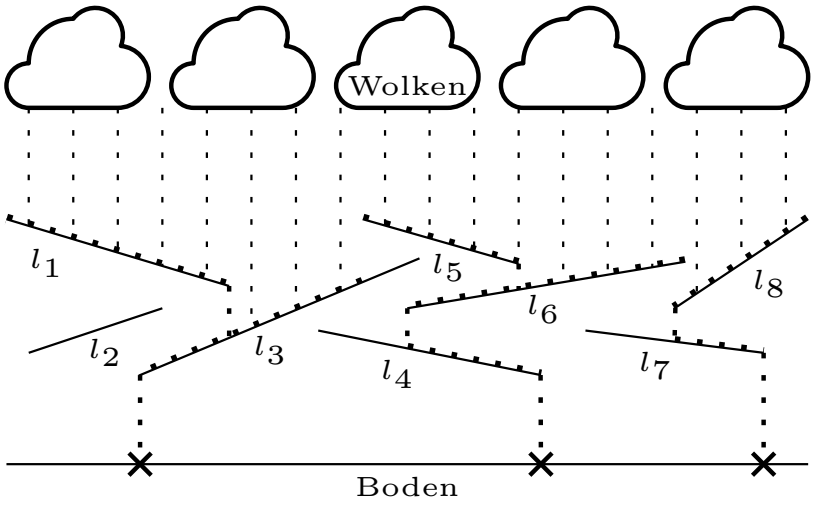
Lösung

Die Laufzeit des Reduktionsalgorithmus ist offensichtlich polynomiell in n. Wenn der Reduktionsalgorithmus `false` zurück gibt, kann es nach Teilaufgabe c kein Dominating Set mit Kardinalität k geben und wir können `Nein` ausgeben. Wenn der Kern mehr als k rote, $k^2 + k$ schwarze (Teilaufgabe d) oder $k^3 + k^2$ weiße (Teilaufgabe e) Knoten besitzt, ist die Instanz unerfüllbar. Andernfalls haben wir bei einen Kern mit $\mathcal{O}(k^3)$ Knoten, den wir mit Bruteforce lösen können. Die Laufzeit beträgt also $\mathcal{O}(p(n) + f(k))$, wobei $p(n)$ die polynomielle Laufzeit des Reduktionsalgorithmus und $f(k)$ die berechenbare Laufzeit des Bruteforce Algorithmus ist. Damit ist das Problem *fixed parameter tractable*.

Lösungsvorschlag

Aufgabe 6. Geometrische Algorithmen: Regenschauer [9 Punkte]

Wir betrachten den Querschnitt eines überdachten Außenbereichs mit unnötig komplexer Dachstruktur. Die Dächer seien durch eine Menge von n Liniensegmenten $\{l_1, \ldots, l_n\}$ gegeben, wobei ein Liniensegment $l_i = \overline{P_i Q_i}$ die Verbindungsstrecke zwischen zwei Punkten $P_i, Q_i \in \mathbb{R}^2$ ist. Wenn es regnet, fällt Wasser geradlinig von oben auf die Dächer und fließt ab (nach den Gesetzen der Schwerkraft). Am Rand eines Daches fließt Wasser geradlinig nach unten.
Wie im untenstehenden Beispiel angedeutet, gibt es Dächer, die direkt beregnet werden (l_1, l_3, l_5, l_6 und l_8), und solche, die vollständig von anderen Dächern überdeckt werden und daher nur indirekt (l_4 und l_7) oder gar nicht (l_2) beregnet werden.
Ziel dieser Aufgabe ist es, alle Punkte zu berechnen, an denen Wasser auf den Boden fällt (im Beispiel mit **X** markiert).



Um Spezialfälle zu vermeiden, treffen wir folgende vereinfachende Annahmen:

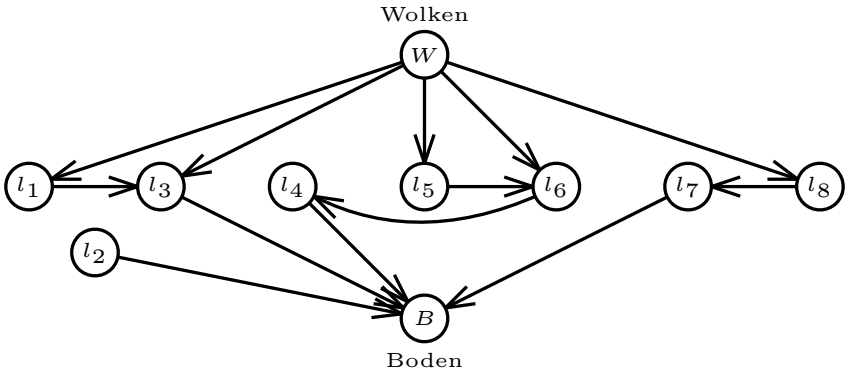
- es regnet nur auf Dächer und nicht direkt auf den Boden,
- es gibt keine horizontalen Dächer,
- an einer x Koordinate beginnt oder endet maximal ein Dach,
- und Dächer berühren oder schneiden sich nicht gegenseitig.

a. Geben Sie einen Algorithmus an, der in einer Laufzeit von $\mathcal{O}(n \log n)$ alle direkt beregneten Dächer markiert.
Hinweis: Verwenden Sie einen Sweep-Line Algorithmus, der Events nach aufsteigender x-Koordinate abarbeitet und ähnlich zum in der Vorlesung vorgestellten Line Intersection Algorithmus arbeitet. [4 Punkte]

Lösung

Das Problem kann mit einer Sweep-Line gelöst werden. Dazu benötigen wir eine sortierte Liste T, die alle Dächer enthält, die sich momentan mit der Sweep-Line schneiden. Die Dächer in T sind aufsteigend nach y-Koordinate des Schnittpunktes mit der Sweep-Line sortiert.
Der Algorithmus sortiert nun alle Endpunkte von Dächern nach aufsteigender x-Koordinate und geht sie in dieser Reihenfolge durch. Wenn ein Event ein Startpunkt ist, wird das neue Dach in T eingefügt. Wenn es ein Endpunkt ist, wird das zugehörige Dach aus T entfernt. Da jedes Dach, das zu irgendeinem Zeitpunkt das erste Element in T war direkt beregnet wird und alle x-Koordinaten der Start- und Endpunkte unterschiedlich sind, markieren wir bei jeder Änderung an T das oberste Dach.
Analyse (nicht gefordert): Die Einfüge- und Löschoperation von T benötigen jeweils logarithmische Laufzeit. Das sortieren der x-Koordinaten benötigt $\mathcal{O}(n \log n)$ Zeit. Dadurch ergibt sich insgesamt die Laufzeit $\mathcal{O}(n \log n)$.

Wir definieren den Regenschauergraphen $G = (\{W, B\} \cup \{l_1, \ldots, l_n\}, E)$, der neben den Knoten W (Wolken) und B (Boden) alle Dächer als Knoten enthält. Für jedes direkt beregnete Dach l_i fügen wir eine Kante (W, l_i) ein. Für jedes Dach l_i, von dem Wasser direkt auf den Boden fließen könnte, fügen wir eine Kante (l_i, B) ein. Wenn Wasser von Dach l_i direkt auf Dach l_j fließen könnte, fügen wir eine Kante (l_i, l_j) ein.
Für das zuvor abgebildete Beispiel ergibt sich somit folgender Regenschauergraph:



b. Geben Sie einen Algorithmus an, der in einer Laufzeit von $\mathcal{O}(n \log n)$ den Regenschauergraphen aufbaut.
Hinweis: Können Sie Ihren Algorithmus aus Teilaufgabe a anpassen? [3 Punkte]

Lösung

Wir ergänzen den Algorithmus aus Teilaufgabe a wie folgt. Wenn wir vorher ein Dach l_i als direkt beregnet markiert haben, fügen wir nun eine Kante (W, l_i) ein. Wenn wir bei einem Event ein Dach l_i in T einfügen oder rauslöschen, und das Event der Endpunkt von l_i ist, an dem das Wasser runter fließt, fügen wir eine Kante (l_i, l_j) zum darunterliegenden Dach l_j in T ein (falls ein solches existiert). Wenn l_i schon das unterste Liniensegment in T ist, fügen wir eine Kante (l_i, B) ein.

c. Geben Sie einen Algorithmus an, der mithilfe des Regenschauergraphen $G = (V, E)$ in einer Laufzeit von $\mathcal{O}(|V| + |E|)$ alle x-Koordinaten findet, an denen Wasser auf den Boden fällt. [2 Punkte]

Lösung

Wir starten eine Breitensuche (oder Tiefensuche) von W. Wenn wir einen Knoten besuchen, der adjazent zu B ist, geben wir die x-Koordinate seines unteren Endpunktes aus.