Name:

Vorname:

Matrikelnummer:

Klausur-ID:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders 24.09.2021

Klausur Algorithmen II

Aufgabe 1.	Kleinaufgaben	12 Punkte
Aufgabe 2.	Randomisierte Algorithmen: Hashtabellen	8 Punkte
Aufgabe 3.	Approximationsalgorithmen: Scheduling	12 Punkte
Aufgabe 4.	Flussalgorithmen: Steinbruchlogistik	9 Punkte
Aufgabe 5.	FPT Algorithmen: Feedback Vertex Set	11 Punkte
Aufgabe 6.	Geometrische Algorithmen: Interval Point Cover	8 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 **18 Blätter**.
- Zum Bestehen der Klausur sind 20 Punkte hinreichend.

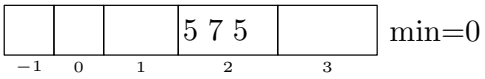
Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [12 Punkte]

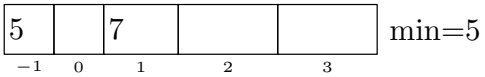
a. Gegeben sei ein leerer Radix-Heap mit $K = 3$, aus dem zuletzt das Element 0 entfernt wurde. Führen Sie nacheinander die Operationen `insert(5)`, `insert(7)` und `insert(5)` auf dem Heap aus und geben Sie im Anschluss den Zustand des Heaps an. Führen Sie anschließend eine `deleteMin()` Operation aus und geben Sie erneut den Zustand des Heaps an. [2 Punkte]

Lösung

Einfügen:



Löschen:



b. Für ein bestimmtes Graphenproblem habe der bisher beste bekannte sequentielle Algorithmus Laufzeit $T_{\text{seq}} = \Theta(n\sqrt{m})$, wobei n die Anzahl der Knoten und m die Anzahl der Kanten ist. Als Algorithmiker haben Sie nun einen neuen parallelen Algorithmus mit Laufzeit $T_{\text{par}}(p) = \Theta\left(\frac{n^2}{\sqrt{p}\log n}\right)$ entworfen. Geben Sie den absoluten Speedup Ihres Algorithmus an. Auf welchen Graphen (in Abhängigkeit von n und m) erreicht Ihr Algorithmus den besten absoluten Speedup?
Hinweis: Überlegen Sie sich, welcher Algorithmus auf welchen Graphen der beste sequentielle Algorithmus für das Problem ist.

[3 Punkte]

Lösung

Je nach Eingabegraph hat der beste sequentielle Algorithmus Laufzeit T_{seq} oder $T_{\text{par}}(1)$. Der absolute Speedup beträgt daher

$$S(p) = \begin{cases} T_{\text{par}}(1)/T_{\text{par}}(p), & n \leq \sqrt{m}\log n \\ T_{\text{seq}}/T_{\text{par}}(p), & \text{sonst} \end{cases}$$
$$= \begin{cases} \Theta\left(\sqrt{p}\right), & n \leq \sqrt{m}\log n \\ \Theta\left(\sqrt{p}\cdot\frac{\sqrt{m}\log n}{n}\right), & \text{sonst} \end{cases}.$$

Der absolute Speedup ist auf Graphen mit $n \leq \sqrt{m}\log n$ am besten.

c. Gegeben sei ein gerichteter und gewichteter Graph $G = (V, E)$, bei dem jede Kante entweder Gewicht 1 oder 2 hat. Geben Sie einen Algorithmus an, welcher mit einer Laufzeit in $\mathcal{O}(|V| + |E|)$ den kürzesten Weg zwischen zwei Knoten berechnet. [2 Punkte]

Lösung

Ausführung des Dijkstra-Algorithmus mit Bucket Queue aus der Vorlesung hat Laufzeit $\mathcal{O}(|E| + |V|C)$, wobei C das Gewicht der schwersten Kante ist $\implies \mathcal{O}(|V| + |E|)$.

Alternativlösung:
In einem Graphen bei dem alle Kanten Gewicht 1 haben, kann der kürzeste Weg zwischen zwei Knoten mit Hilfe einer Breitensuche in einer Laufzeit von $\mathcal{O}(|V| + |E|)$ berechnet werden. Im Folgenden transformieren wir jede Kante mit Gewicht 2 in Kanten mit Gewicht 1, indem wir Hilfsknoten in den Graphen einfügen.

$\mathcal{O}(|E|)$ *zusätzlicher Speicher:* Für jede Kante $e = (u, v)$ mit Gewicht 2 fügen wir einen Hilfsknoten e' und zwei Kanten (u, e') und (e', v) mit Gewicht 1 ein. Insgesamt fügen wir maximal $|E|$ neue Knoten und $|E|$ neue Kanten ein. Für die Laufzeit der Breitensuche ergibt sich

$$\mathcal{O}((|V| + |E|) + (|E| + |E|)) = \mathcal{O}(|V| + |E|).$$

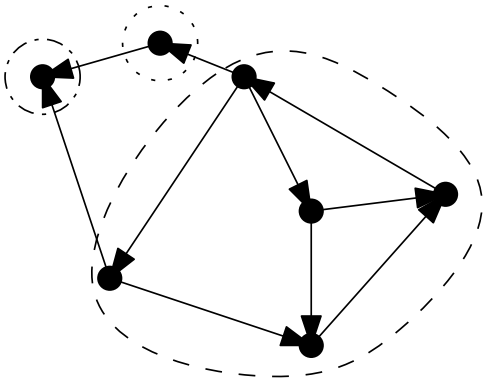
$\mathcal{O}(|V|)$ *zusätzlicher Speicher:* Für jeden Knoten u fügen wir einen Hilfsknoten u' ein. Für jede Kante (u, v) mit Gewicht 2 fügen wir zwei neue Kanten (u, u') und (u', v) mit Gewicht 1 ein und löschen die Kante (u, v) aus dem Graphen. Insgesamt fügen wir $|V|$ neue Knoten und maximal $|V|$ neue Kanten hinzu (die neue Kante ist (u, u'), (u, v) wird durch (u', v) ersetzt). Für die Laufzeit der Breitensuche ergibt sich

$$\mathcal{O}((|V| + |V|) + (|E| + |V|)) = \mathcal{O}(|V| + |E|).$$

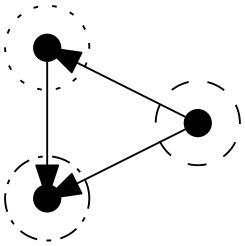
d. Markieren Sie im nachfolgenden Graphen alle starken Zusammenhangskomponenten (SCCs) und geben Sie den Schrumpfgraphen an. [2 Punkte]

Lösung

Starke Zusammenhangskomponenten:



Schrumpfgraph:



e. Geben Sie das Suffix-Array, das inverse Suffix-Array und das LCP-Array für die Zeichenkette `vimvivim$` an. [3 Punkte]

Lösung

Indices	0	1	2	3	4	5	6	7	8
Text	v	i	m	v	i	v	i	m	$
Suffix-Array	8	6	1	4	7	2	5	0	3
Inverses Suffix-Array	7	2	5	8	3	6	1	4	0
LCP-Array	-1	0	2	1	0	1	0	3	2

Lösungsvorschlag

Aufgabe 2. Randomisierte Algorithmen: Hashtabellen [8 Punkte]

a. Gegeben sei eine Cuckoo Hashtabelle der Größe 10 mit den beiden Hashfunktionen $h_1(x) = x \bmod 10$ und $h_2(x) = \lfloor \frac{x}{10} \rfloor \bmod 10$. Neue Elemente werden immer erst mit h_1 eingefügt. Die Hashtabelle enthält bereits die Elemente 42 und 64. Fügen Sie nacheinander die Elemente 107, 7, 27 in die Hashtabelle ein und geben Sie nach jeder Einfügeoperation den aktuellen Zustand der Tabelle an. [2 Punkte]

Lösung

	0	1	2	3	4	5	6	7	8	9
0.			42		64					
1.			42		64			107		
2.	107		42		64			7		
3.	7		27		42		64	107		

Bemerkungen: Lösungen, die ein neues Element x in $h_2(x)$ einfügen, wenn $h_1(x)$ schon belegt ist, sind auch richtig.

b. Geben Sie ein weiteres Element an, das bei Einfügung einen Neubau der Hashtabelle erzwingt. Begründen Sie Ihre Antwort kurz. [1 Punkt]

Lösung

Zum Beispiel 60, denn h_1 und h_2 bilden $\{7, 27, 42, 60, 64, 107\}$ auf $\{0, 2, 4, 6, 7\}$ ab. Also existiert nach dem Schubfachprinzip keine gültige Zuordnung der Elemente auf die Einträge der Hashtabelle.

c. Wir betrachten nun eine Hashtabelle der Größe m mit Verkettung, die wie folgt funktioniert: Die Hashtabelle benutzt zwei unabhängig zufällige Hashfunktionen h_1 und h_2. Jeder Eintrag speichert eine verkettete Liste. Ein neues Element x wird immer in die kürzere Liste der Einträge $h_1(x)$ oder $h_2(x)$ eingefügt. Sollten beide Listen gleich viele Elemente enthalten, wird es an eine beliebige der beiden Listen angehängt. Eine Einfügeoperation verursacht eine *Kollision*, falls beide Listen nichtleer sind.
Zeigen Sie, dass die erwartete Anzahl an Kollisionen beim Einfügen von n Elementen in eine leere Hashtabelle der Größe $m = n\sqrt{n}$ nach oben mit $\frac{1}{3}$ beschränkt ist.
Hinweis: Finden Sie eine geeignete Abschätzung für die Wahrscheinlichkeit, dass das i-te Element eine Kollision verursacht.
Hinweis: $\sum_{i=1}^{n-1} i^2 = \frac{1}{6}(n-1)(2n-1)n$. [3 Punkte]

Lösung

Die Wahrscheinlichkeit, dass das $i+1$-te Element kollidiert, ist $p_i = \left(\frac{\#\text{ belegte Einträge}}{n\sqrt{n}}\right)^2$. Da $\#\text{ belegte Einträge} \leq i$, folgt $p_i \leq \left(\frac{i}{n\sqrt{n}}\right)^2$, also

$$\begin{aligned}\mathbb{E}[\textit{Anzahl Kollisionen}] &= \sum_{i=0}^{n-1} \mathbb{E}[i+1\textit{-te Element kollidiert}] = \sum_{i=0}^{n-1} p_i \\ &\leq \sum_{i=0}^{n-1} \left(\frac{i}{n\sqrt{n}}\right)^2 = \frac{(n-1)(2n-1)n}{6n^3} \leq \frac{2n^3}{6n^3} = \frac{1}{3}.\end{aligned}$$

d. Wir verallgemeinern die Hashtabelle aus Teilaufgabe c wie folgt: Statt zwei Hashfunktionen nutzen wir nun k unabhängig zufällige Hashfunktionen $h_1, \ldots, h_k$. Ein neues Element x wird immer an die kürzeste Liste der Einträge $h_1(x), \ldots, h_k(x)$ angehängt. Sollte es mehrere kürzeste Listen geben, wird es an eine beliebige der kürzesten Listen angehängt. Eine Kollision liegt vor, wenn ein Element eingefügt wird und alle k Listen bereits nichtleer sind.
Zeigen Sie, dass die erwartete Anzahl an Kollisionen beim Einfügen von n Elemente in eine leere Hashtabelle der Größe $m = n\sqrt[k]{n}$ nach oben mit 1 beschränkt ist. [2 Punkte]

Lösung

Analog zu Teilaufgabe c erhält man $p_i \leq \left(\frac{i}{n\sqrt[k]{n}}\right)^k$ und damit

$$\begin{aligned}\mathbb{E}[\textit{Anzahl Kollisionen}] &= \sum_{i=0}^{n-1} \mathbb{E}[i+1\textit{-te Element kollidiert}] = \sum_{i=0}^{n-1} p_i \\ &\leq \sum_{i=0}^{n-1} \left(\frac{i}{n\sqrt[k]{n}}\right)^k \leq \frac{(n-1)^{k+1}}{n^{k+1}} \leq 1.\end{aligned}$$

Lösungsvorschlag

Aufgabe 3. Approximationsalgorithmen: Scheduling [12 Punkte]

Im Folgenden betrachten wir das *Scheduling Problem* für eine Instanz $\mathscr{P} := (t_1, \ldots, t_n \mid m)$: Gegeben seien n unabhängige Rechenjobs mit Nummern $1, \ldots, n$ und Ausführungszeiten $t_1, \ldots, t_n$. Für die Ausführungszeiten der Rechenjobs nehmen wir an, dass diese in absteigender Reihenfolge sortiert sind (d.h. $t_1 \geq \ldots \geq t_n$). Außerdem gibt es m identische Maschinen mit Nummern $1, \ldots, m$. Gesucht ist eine Aufteilung $\mu : \{1, \ldots, n\} \to \{1, \ldots, m\}$ der Rechenjobs auf die Maschinen, welche die MAKESPAN

$$C_{\max} := \max_{i \in \{1,\ldots,m\}} C_i \quad \text{mit} \quad C_i := \sum_{\substack{j=1 \\ \mu(j)=i}}^{n} t_j$$

minimiert. Im Folgenden bezeichnen wir mit C_{OPT} die MAKESPAN einer optimalen Aufteilung für eine Scheduling Instanz $\mathscr{P} = (t_1, \ldots, t_n \mid m)$.

a. Begründen Sie, dass für eine beliebige Scheduling Instanz $\mathscr{P} = (t_1, \ldots, t_n \mid m)$ gilt:

1. $\max_{j=1}^{n} t_j \leq C_{\text{OPT}}$
2. $\frac{1}{m} \sum_{j=1}^{n} t_j \leq C_{\text{OPT}}$

[2 Punkte]

Lösung

1. Die optimale MAKESPAN ist mindestens solange wie die längste Ausführungszeit eines Rechenjobs.
2. Angenommen $C_{\text{OPT}} < \frac{1}{m} \sum_{j=1}^{n} t_j$

$$\Longrightarrow \sum_{j=1}^{n} t_j = \sum_{i=1}^{m} C_i \leq \sum_{j=1}^{m} C_{\text{OPT}} = m \cdot C_{\text{OPT}} < \sum_{j=1}^{n} t_j$$

Zum Berechnen einer Aufteilung μ für eine beliebige Scheduling Instanz $\mathscr{P} = (t_1, \ldots, t_n \mid m)$, verwenden wir nun den *Longest Processing Time* Algorithmus (LPT Algorithmus). Der Algorithmus ist im folgenden Pseudocode dargestellt.

Algorithmus 1 LPT ($\mathscr{P} = (t_1, \ldots, t_n \mid m)$)

$\mu(j) \leftarrow \perp$ für alle $j \in \{1, \ldots, n\}$
for $j \in \{1, \ldots, n\}$ in absteigender Reihenfolge der Ausführungszeiten t_j **do** ▷ $t_1 \geq \ldots \geq t_n$
  $i \leftarrow \arg\min_{i=1}^{m} C_i$ ▷ Maschine mit geringster Auslastung
  $\mu(j) \leftarrow i$ ▷ Zuweisung von Rechenjob j zu Maschine i
return μ

b. Sei $C_{\max}$ die MAKESPAN für eine Aufteilung μ, welche durch den LPT Algorithmus für eine Scheduling Instanz $\mathscr{P} = (t_1, \ldots, t_n \mid m)$ berechnet wurde. Des Weiteren sei i die Maschine mit maximaler Ausführungszeit, d.h. $C_{\max} = C_i$ und es sei t_j die Ausführungszeit des Rechenjobs, den der LPT Algorithmus als letztes zu Maschine i hinzugefügt hat. Zeigen Sie, dass

$$C_{\max} - t_j \leq C_{\text{OPT}}.$$

Hinweis: Verwenden Sie 2) aus Teilaufgabe a. [3 Punkte]

Lösung

Zum Zeitpunkt an dem der LPT Algorithmus den Rechenjob j zu Maschine i hinzugefügt hat, muss i die Maschine mit der geringsten Auslastung gewesen sein.

$$\Longrightarrow \left(\sum_{\mu(k)=i} t_k \right) - t_j \leq \frac{1}{m} \sum_{k=1}^{j-1} t_k$$

Hieraus ergibt sich

$$C_{\max} - t_j = \left(\sum_{\mu(k)=i} t_k \right) - t_j \leq \frac{1}{m} \sum_{k=1}^{j-1} t_k \leq \frac{1}{m} \sum_{j=1}^{n} t_j \overset{a.}{\leq} C_{\text{OPT}}$$

c. Zeigen Sie, dass für eine Scheduling Instanz $\mathscr{P} = (t_1, \ldots, t_n \mid m)$ gilt:

$$2t_{m+1} \leq C_{\text{OPT}}.$$

Anmerkung: Die Ausführungszeiten der n Rechenjobs sind in absteigender Reihenfolge sortiert (d.h. $t_1 \geq \ldots \geq t_n$). [3 Punkte]

Lösung

Die optimale Lösung für die ersten m Rechenjobs ist es, diese jeweils einzeln auf die ersten m Maschinen zu verteilen. Daher muss der $m+1$-te Rechenjob zu einer Maschine zugewiesen werden auf der bereits ein Rechenjob läuft. Sei t_j die Ausführungszeit dieses Rechenjobs. Da die Rechenjobs in absteigender Reihenfolge sortiert sind, gilt: $t_{m+1} \leq t_j$. Hieraus folgt:

$$C_{\text{OPT}} \geq t_j + t_{m+1} \geq 2t_{m+1}$$

d. Zeigen Sie, dass der LPT Algorithmus eine $\frac{3}{2}$-Approximation für eine beliebige Scheduling Instanz $\mathscr{P} = (t_1, \ldots, t_n \mid m)$ berechnet.
Hinweis: Benutzen Sie Teilaufgabe b und c. [4 Punkte]

Lösung

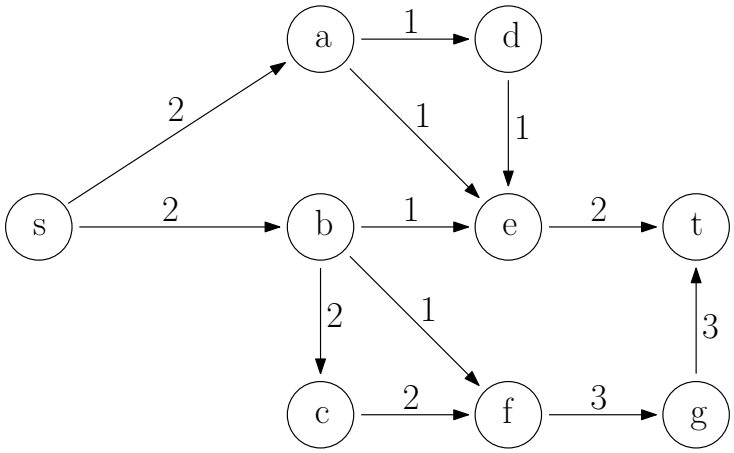
Sei i die Maschine mit der längsten Ausführungszeit, d. h. $C_{\max} = C_i$ und es sei t_j die Ausführungszeit des Rechenjob den der LPT Algorithmus als letztes zu Maschine i hinzugefügt hat. Wir nehmen im Folgenden an, dass zu Maschine i mindestens zwei Rechenjobs zugewiesen worden sind, ansonsten wäre $C_{\max}$ optimal. Daher gilt $t_j \leq t_{m+1}$, da die Ausführungszeiten in absteigender Reihenfolge sortiert sind. Es folgt:

$$C_{\max} = (C_{\max} - t_j) + t_j \overset{b.}{\leq} C_{\text{OPT}} + t_j \overset{t_j \leq t_{m+1}}{\leq} C_{\text{OPT}} + t_{m+1} \overset{d.}{\leq} \frac{3}{2} \cdot C_{\text{OPT}}$$

Lösungsvorschlag

Aufgabe 4. Flussalgorithmen: Steinbruchlogistik [9 Punkte]

a. Betrachten Sie das unten abgebildete Flussnetzwerk. Die Kanten sind mit ihren Kapazitäten beschriftet. Berechnen Sie mit dem *Edmonds Karp* Algorithmus den maximalen Fluss von der Quelle s zur Senke t. Geben Sie dabei für jeden Schritt den augmentierenden Pfad in Form einer Knotenliste und den Fluss den Sie über diesen Pfad schieben an. Geben Sie zusätzlich den Wert des maximalen Flusses nach der Ausführung an.
Anmerkung: Der Edmonds Karp Algorithmus augmentiert den Fluss immer über den kürzesten Pfad von der Quelle s zur Senke t im Residualgraphen (die Länge eines Pfades ist die Anzahl an Knoten in dem Pfad). [3 Punkte]



Lösung

Der Edmond Karp Algorithmus augmentiert immer auf dem kürzesten Pfad.

1. $p_1 = s \to b \to e \to t, f(p_1) = 1$
2. $p_2 = s \to a \to e \to t, f(p_2) = 1$
3. $p_3 = s \to b \to f \to g \to t, f(p_3) = 1$
4. $p_4 = s \to a \to d \to e \to b \to c \to f \to g \to t, f(p_4) = 1$

Schritt 1 und 2 können auch vertauscht werden. Der maximale Fluss hat den Wert 4.

Aus einem Steinbruch sollen Steine zu einer Fabrik transportiert werden. Für den Transport stehen dem Steinbruch x LKW zur Verfügung. Jeder LKW soll pro Tag **genau einmal** zu der Fabrik fahren und jeder LKW soll **mit genau dem gleichen Gewicht** an Steinen beladen werden. Das Straßennetzwerk kann durch einen gerichteten und gewichteten Graphen $G = (V, E, \omega)$ mit Kantengewichtsfunktion $\omega : E \to \mathbb{N}_+$ dargestellt werden. Da das Straßennetzwerk in einem schlechten Zustand ist, kann über jede Straße $e \in E$ **nur maximal** $\omega(e)$ Kilogramm an Steinen pro Tag transportiert werden. Der Steinbruch befindet sich am Knoten s und die Fabrik am Knoten t.

b. Geben Sie einen Algorithmus an, welcher mit Hilfe eines maximalen Flusses in einem Flussnetzwerk entscheidet, ob jeder der x LKW mit k Kilogramm an Steinen pro Tag beladen werden kann. Begründen Sie die Korrektheit Ihres Algorithmus. [3 Punkte]

Lösung

Wir transformieren den Graphen $G = (V, E, \omega)$ in ein Flussnetzwerk $\mathcal{N} = (V, E, c, s, t)$ mit Kapazitätsfunktion $c(e) = \lfloor \frac{\omega(e)}{k} \rfloor$ für alle Kanten $e \in E$. Die Kapazität einer Kante gibt nun an, wie viele LKW, welche mit k Kilogramm an Steinen beladen sind, die entsprechende Straße pro Tag befahren können. Ein maximaler Fluss zwischen s und t in $\mathcal{N}$ ist äquivalent zu dem minimalen (s,t)-Schnitt. Das Gewicht des minimalen (s,t)-Schnittes stellt die maximale Anzahl an LKW dar, welche an einem Tag von dem Steinbruch zur Fabrik fahren können. Ist dieses Gewicht größer oder gleich x, dann ist es möglich, dass alle LKW mit k Kilogramm pro Tag beladen werden können. Andernfalls nicht.

c. Der Steinbruch aus Teilaufgabe b kann pro Tag maximal $N \in \mathbb{N}_{\geq 0}$ Kilogramm an Steinen produzieren. Um die Lagerhaltung der Steine zu vermeiden, soll die Produktionsmenge so angepasst werden, dass alle produzierten Steine pro Tag zu der Fabrik transportiert werden können. Wie in Teilaufgabe b stehen dem Steinbruch x LKW für den Transport der Steine zur Verfügung, die alle mit genau dem gleichen Gewicht beladen werden sollen. Geben Sie einen Algorithmus an, welcher das maximale Gewicht in Kilogramm an Steinen bestimmt, die der Steinbruch pro Tag ohne Lagerhaltung produzieren kann. Die Laufzeit Ihres Algorithmus soll in $\mathcal{O}(\log(N) T_{\text{flow}})$ liegen, wobei T_{flow} die Laufzeit eines Flussalgorithmus ist.
Anmerkung: Es sind nur ganzzahlige Produktionsmengen möglich. [3 Punkte]

Lösung

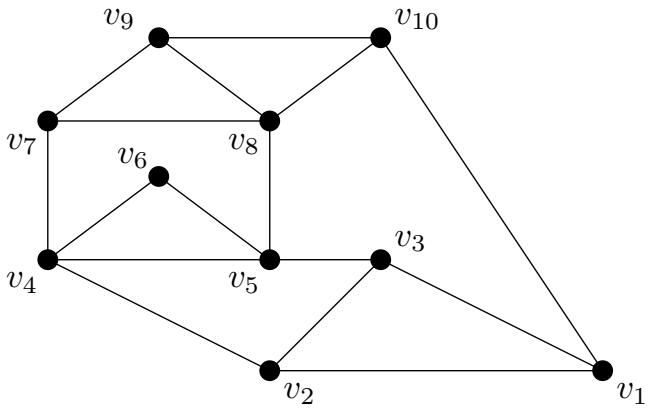
Wir führen eine binäre Suche nach der maximalen Produktionmenge X in Kilogramm aus und nutzen Teilaufgabe b, um zu entscheiden, ob diese Produktionsmenge pro Tag von dem Steinbruch zu der Fabrik transpotiert werden kann. Für die binäre Suche initialisieren wir $L = 0$ und $R = N$. In jedem Schritt testen wir mit Teilaufgabe b, ob es möglich ist, jeden LKW mit genau $k = \frac{M}{x}$ Kilogramm zu beladen, wobei $M = \frac{L+R}{2}$. Falls es möglich ist, setzen wir $L = M$. Andernfalls $R = M - 1$. Die binäre Suche wird ausgeführt solange $L + 1 < R$. Am Ende geben wir L als Lösung aus.

Lösungsvorschlag

Aufgabe 5. FPT Algorithmen: Feedback Vertex Set [11 Punkte]

Für einen ungerichteten Graphen G mit Knoten $V(G)$ und Kanten $E(G)$ bezeichnen wir eine Knotenteilmenge $X \subseteq V(G)$ als *Feedback Vertex Set* falls $G-X$ ein Wald ist (dabei bezeichnet $G-X$ den von $V(G) \setminus X$ induzierten Subgraphen von G)
Das Problem `Feedback Vertex Set` entscheidet, gegeben einen ungerichteten Graphen G und eine Zahl $k \in \mathbb{N}_0$, ob in G ein Feedback Vertex Set X der Größe maximal k existiert ($|X| \leq k$).

a. Geben Sie für den folgenden Graphen ein Feedback Vertex Set der Größe 3 an. [2 Punkte]



Lösung

Zum Beispiel $X = \{v_1, v_5, v_9\}$ oder $X = \{v_1, v_4, v_8\}$.

b. Sei G ein ungerichteter Graph und X ein Feedback Vertex Set in G. Zeigen Sie, dass

$$\sum_{v \in X} d(v) > |E(G)| - |V(G)|.$$

Hinweis: Verwenden Sie, dass $G-X$ ein Wald ist, also $|V(G-X)| > |E(G-X)|$ erfüllt. [2 Punkte]

Lösung

Es gilt

$$\sum_{v \in X} d(v) \overset{(i)}{\geq} |E(G)| - |E(G-X)| \overset{(ii)}{>} |E(G)| - |V(G)|,$$

denn (i) jede Kante in $E(G) \setminus E(G-X)$ ist inzident zu einem Knoten in X und (ii) $|V(G)| \geq |V(G-X)| > |E(G-X)|$, da $G-X$ ein Wald ist.

Im Folgenden bezeichnen wir mit $v_1, \ldots, v_n$ die nach absteigendem Grad sortierten Knoten einer Probleminstanz (G,k), d.h. $d(v_1) \geq \cdots \geq d(v_n)$. Die $5k$ Knoten mit dem höchsten Grad in G bezeichnen wir mit $V_{5k} := \{v_1, \ldots, v_{5k}\}$.

c. Sei (G,k) mit $|V(G)| \geq 5k$ eine Probleminstanz mit Feedback Vertex Set X, $|X| \leq k$. Angenommen, X enthalte keinen Knoten aus V_{5k}. Zeigen Sie mit Teilaufgabe b, dass dann die Ungleichung

$$\sum_{v \in V(G)} d(v) = \sum_{i=1}^{5k} d(v_i) + \sum_{i>5k} d(v_i) > 6|E(G)| - 6|V(G)|$$

erfüllt ist. [2 Punkte]

Lösung

Da jeder Knoten in V_{5k} einen Grad mindestens so hoch wie der höchste Grad in X hat, folgt $\sum_{i=1}^{5k} d(v_i) > 5|E(G)| - 5|V(G)|$ mit Teilaufgabe b. Außerdem ist $\sum_{i>5k} d(v_i) > |E(G)| - |V(G)|$, denn nach Annahme ist $X \subseteq V(G) \setminus V_{5k}$.

Für das `Feedback Vertex Set` Problem steht Ihnen eine Reduktionsregel $\mathscr{R}$ zur Verfügung, die eine Probleminstanz (G,k) in polynomieller Zeit auf eine Probleminstanz (G',k') reduziert, wobei G' keinen Knoten von Grad ≤ 2 enthält und $k' \leq k$. In G' gibt es genau dann ein Feedback Vertex Set der Größe k', wenn es in G ein Feedback Vertex Set der Größe k gibt. Eine Probleminstanz, die bereits mit $\mathscr{R}$ reduziert wurde, bezeichnen wir als *reduzierte Probleminstanz*.

d. Sei (G,k) mit $|V(G)| \geq 5k$ eine reduzierte Probleminstanz, d. h. jeder Knoten in G hat mindestens Grad 3. Sei weiter X mit $|X| \leq k$ ein Feedback Vertex Set in G. Zeigen Sie, dass X mindestens einen Knoten aus V_{5k} enthält.
Hinweis: Führen Sie einen Widerspruchsbeweis mit Teilaufgabe c.
Hinweis: $\sum_{v \in V(G)} d(v) = 2|E(G)|$. [2 Punkte]

Lösung

Da jeder Knoten mindestens Grad 3 hat, gilt $2|E(G)| = \sum_{v \in V(G)} d(v) \geq 3|V(G)|$ $(\star)$ Zusammen mit Teilaufgabe c folgt

$$2|E(G)| = \sum_{v \in V(G)} d(v) > 6|E(G)| - 6|V(G)| \implies 6|V(G)| > 4|E(G)| \overset{(\star)}{\geq} 6|V(G)|,$$

ein Widerspruch.

e. Geben Sie einen FPT Algorithmus für das `Feedback Vertex Set` Problem an. Begründen Sie, wieso Ihr Algorithmus fixed parameter tractable ist.
Hinweis: Verwenden Sie die Reduktionsregel $\mathscr{R}$ und das Ergebnis von Teilaufgabe d. [3 Punkte]

Lösung

Sei (G,k) die zu lösende Probleminstanz. Wir wenden zunächst Reduktionsregel $\mathscr{R}$ an und erhalten die reduzierte Probleminstanz (G',k'). Falls $k' = 0$ und G' kein Wald ist, geben wir `Nein` zurück. Falls $k' \geq 0$ und G' ein Wald ist, geben wir `Ja` zurück. Ansonsten verzweigen wir: falls es in G' ein Feedback Vertex Set der Größe k' gibt, dann enthält es nach Teilaufgabe d mindestens einen Knoten aus $V_{5k'}$ (oder der Graph hat weniger als $5k$ Knoten). Das nutzen wir, indem wir jeden Knoten $v \in V_{5k'}$ probeweise in unser Feedback Vertex Set aufnehmen und rekursiv die Probleminstanz $(G'-v,k'-1)$ lösen.
Das ist ein fixed parameter tractable Algorithmus nach dem Prinzip des beschränkten Suchbaums. Der Verzweigungsgrad beträgt maximal $5k$ und die Tiefe ist maximal k. Insgesamt ergibt sich Laufzeit $\mathscr{O}\left((5k)^k n^{\mathscr{O}(1)}\right)$. Also ist das Problem fixed parameter tractable.

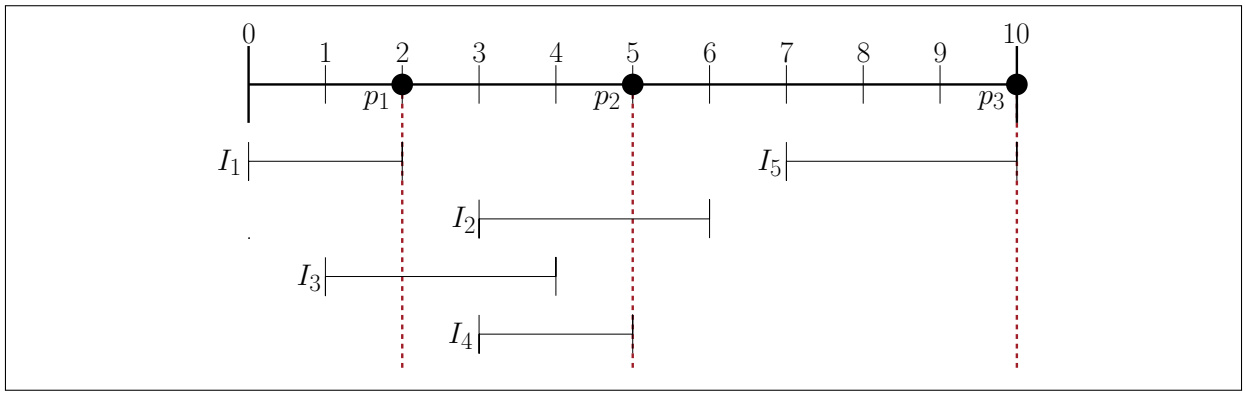
Lösungsvorschlag

Aufgabe 6. Geometrische Algorithmen: Interval Point Cover [8 Punkte]

Gegeben sei eine Menge an Intervallen $I = \{I_1, \ldots, I_n\}$. Jedes Intervall $I_i \in I$ ist von der Form $I_i = [l_i, r_i]$ mit $l_i \leq r_i$ und $l_i, r_i \in \mathbb{N}_{\geq 0}$. Das *Interval Point Cover Problem* ist wie folgt definiert: Finden Sie eine minimale Anzahl an Punkten $P = \{p_1, \ldots, p_k\}$ mit $p_i \in \mathbb{N}_{\geq 0}$, sodass jedes Intervall mindestens einen Punkt aus P enthält. Formaler, $\forall [l, r] \in I : \exists p \in P : l \leq p \leq r$ und $|P|$ minimal.

a. Zeichnen Sie im folgenden Beispiel eine optimale Lösung für das Interval Point Cover Problem ein. [1 Punkt]

Lösung



b. Geben Sie einen *Greedy* Algorithmus an, welcher das Interval Point Cover Problem in einer Laufzeit von $\mathcal{O}(n\log(n))$ mit $n = |I|$ löst. Beweisen Sie, dass Ihr Algorithmus eine optimale Lösung findet.
Hinweis: Sie können einen Induktionsbeweis verwenden. [5 Punkte]

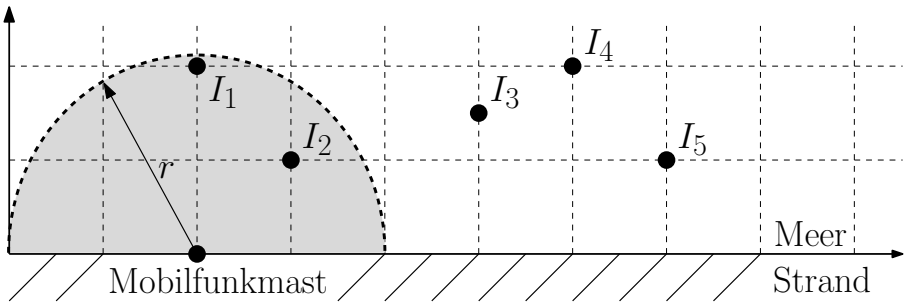
Lösung

Algorithmus 2 `IntervalPointCover` $(I = \{I_1, \ldots I_n\})$

$P \leftarrow \emptyset$
for $[l,r] \in I$ in aufsteigender Reihenfolge der Endpunkte r **do**
  if $P = \emptyset$ oder $\max(P) < l$ **then**
    $P \leftarrow P \cup \{r\}$
return P

Wir führen einen Induktionsbeweis über die in aufsteigender Reihenfolge ihrer Endpunkte sortierten Intervalle. Für $I = \{I_1\}$ liefert der Algorithmus offensichtlich die optimale Lösung. Wir nehmen nun an, dass die aktuelle Lösung P_k unseres Algorithmus eine optimale Lösung für die ersten k Intervalle ist. Zu zeigen ist, dass die Lösung immer noch optimal ist, nachdem wir das Intervall $I_{k+1} = [l,r]$ verarbeitet haben. Falls $l \leq \max(P_k)$, dann wird r nicht in P_k aufgenommen und somit ist P_k immer noch eine optimale Lösung für die ersten $k+1$ Intervalle, da $\max(P_k) \in I_{k+1}$.
Es sei nun $l > \max(P_k)$, dann wird r in P_k aufgenommen. Jedes Intervall, dessen Endpunkt in P_k aufgenommen wurde, ist disjunkt zu allen anderen Intervallen, deren Endpunkt in P_k aufgenommen wurde, da sonst der Endpunkt des einen Intervalls innerhalb des Anderen liegen müsste, was der Konstruktion des Algorithmus widerspricht. Da P_k eine optimale Lösung für die ersten k Intervalle ist, gibt es $|P_k|$ disjunkte Intervalle. Da $l > \max(P_k)$ gilt, ist I_{k+1} disjunkt zu allen Intervallen, deren Endpunkt in P_k aufgenommen wurde. Somit existieren $|P_k|+1$ disjunkte Intervalle, für die mindestens $|P_k|+1$ Punkte in P_{k+1} benötigt werden.

An einem Strandabschnitt sollen Mobilfunkmasten aufgestellt werden, um eine naheliegende Inselgruppe mit mobilen Daten zu versorgen. Die Grenze zwischen Meer und Strand verläuft geradlinig und die Mobilfunkmasten sollen genau an dieser Grenze aufgestellt werden. Die Inseln $I = \{I_1, \ldots, I_n\}$ sind so klein, dass sie als zweidimensionale Punkte modelliert werden können. Die Mobilfunkmasten versorgen alle Inseln in einem Radius r mit mobilen Daten. Sie können im Folgenden annehmen, dass jede Insel über einen Mobilfunkmasten mit mobilen Daten versorgt werden kann. Der Sachverhalt wird in der unten abgebildeten Illustration dargestellt.



c. Geben Sie einen Algorithmus an, welcher in einer Laufzeit von $\mathcal{O}(n\log(n))$ die minimale Anzahl an Mobilfunkmasten bestimmt, die am Strand aufgestellt werden müssen, um alle Inseln mit mobilen Daten zu versorgen. [2 Punkte]

Lösung

Wir zeichnen um jede Inseln einen Kreis mit Radius r. Die Schnittpunkte mit dem Strand definiert einen Intervall auf dem ein Mobilfunkmast aufgestellt werden muss, um die entsprechende Insel mit mobilen Daten zu versorgen. Somit haben wir das Problem auf das Interval Point Cover Problem reduziert und können den Algorithmus aus Teilaufgabe b verwenden, um die minimale Anzahl an Mobilfunkmasten zu bestimmen.
Sonderfall: Der Kreis um eine Insel schneidet den Strand an einem Punkt p. In diesem Fall können wir das Interval $[p, p]$ hinzufügen, da der Algorithmus aus Teilaufgabe b damit umgehen kann.