Karlsruher Institut für Technologie

Lösung zum Übungsblatt 1

Abgabe bis 6. November 2024

Aufgabe 1: Verschiedenes

3 · 2 = 6 Punkte

Teilaufgabe (a) Gibt es einen Unterschied zwischen $\Theta(2^n)$ und $2^{\Theta(n)}$? Wenn ja, welches der beiden wächst asymptotisch schneller?

Lösung 1a:

Ja, denn $2^{\Theta(n)}$ kann beispielsweise für $2^{2n} = 4^n$ stehen, was asymptotisch echt schneller wächst als 2^n. Umgekehrt beinhaltet $2^{\Theta(n)}$ auch $2^{n/2} = (\sqrt{2})^n \approx 1{,}41^n$, was langsamer wächst als 2^n. Man kann also nicht sagen, dass $2^{\Theta(n)}$ langsamer oder schneller wächst als $\Theta(2^n)$. Allerdings ist $2^{\Theta(n)}$ ungenauer.

Teilaufgabe (b) Sei T ein ungewurzelter Binärbaum, d.h. T enthält nur Knoten mit Grad 3 (innere Knoten) und mit Grad 1 (Blätter). Sei n_3 die Anzahl der inneren Knoten und n_1 die Anzahl der Blätter. Zeige, dass $n_3 = n_1 - 2$ gilt.

Lösung 1b:

Da T ein Baum ist gilt $m = n - 1 = n_1 + n_3 - 1$. Außerdem ist die Summe der Knotengrade $2m$, also $2m = n_1 + 3n_3$. Ersetzt man m in der zweiten Gleichung durch die erste, so erhält man

$$\begin{aligned} 2(n_1 + n_3 - 1) &= n_1 + 3n_3, \\ 2n_1 + 2n_3 - 2 &= n_1 + 3n_3, \\ n_1 - 2 &= n_3. \end{aligned}$$

Teilaufgabe (c) Gib zwei parametrisierte Probleme an, für die es genau dann einen FPT-Algorithmus gibt, wenn P = NP.

Lösung 1c:

Angenommen ein Problem P mit Parameter k ist für konstantes k NP-vollständig. Dann würde ein FPT-Algorithmus für dieses parametrisierte Problem P = NP implizieren. Umgekehrt würde P = NP implizieren, dass P in polynomieller Zeit (und damit insbesondere in FPT Zeit) lösbar ist. Man muss also nur parametrisierte Probleme finden, die für einen konstanten Parameter schon NP-schwer sind. Hier ein paar Beispiele:

- k-Sat (jede Klausel enthält k Literale) mit dem Parameter k ist für $k = 3$ NP-vollständig.
- k-Färbung in Graphen mit dem Parameter k ist für $k = 3$ schon NP-vollständig.
- Hamilton Pfad in Graphen mit dem Maximalgrad k als Parameter ist für $k = 3$ schon NP-vollständig.

Aufgabe 2: Verzweigungsvektor

2 + 3 = 5 Punkte

Diese Aufgabe baut auf der zweiten Vorlesung auf. Gib für die folgenden Rekurrenzen möglichst scharfe[1] obere Schranken der Form x^n an und beweise deren Korrektheit mittels Induktion.

$$f(n) = \begin{cases} 3 \cdot f(n-1), & n > 5 \\ 1, & \text{sonst} \end{cases} \qquad g(n) = \begin{cases} g(n-1) + g(n-2) + g(n-3), & n > 17 \\ 1, & \text{sonst} \end{cases}$$

Lösung 2:

Für die erste Funktion behaupten wir eine obere Schranke von $f(n) \leq 3^n$. Die Korrektheit lässt sich per Induktion zeigen. Im Basisfall gilt $f(n) = 1 \leq 3^n$ für $n \leq 5$. Für den Induktionsschritt betrachten wir $f(n+1)$. Es gilt

$$f(n+1) = 3 \cdot f(n),$$

da per Induktionsannahme $f(n) \leq 3^n$ gilt, erhalten wir somit

$$\leq 3 \cdot 3^n = 3^{n+1}.$$

Somit haben wir die obere Schranke für alle n gezeigt.

Für die zweite Funktion nehmen wir an, dass $g(n) = r^n$. Durch Einsetzen in die Rekurrenz erhalten wir dann

$$r^n = r^{n-1} + r^{n-2} + r^{n-3}.$$

Wir können durch r^{n-3} teilen und erhalten

$$r^3 = r^2 + r + 1$$

und somit

$$r^3 - r^2 - r - 1 = 0.$$

[1] 3-4 Nachkommastellen sollen hier genügen

Durch Approximation der größten positiven Nullstelle des Polynoms erhalten wir $r = 1.8393$. Es verbleibt die Korrektheit zu beweisen.

Im Basisfall gilt $g(n) = 1 < 1.8393 \leq 1.8393^n$ für $1 \leq n \leq 16$. Wir nehmen nun an, dass $g(n) \leq 1.8393^n$ für alle Eingaben bis n gilt und betrachten $g(n+1)$. Es gilt

$$\begin{aligned} g(n+1) &= g(n) + g(n-1) + g(n-2) \\ &= 1.8393^n + 1.8393^{n-1} + 1.8393^{n-2} \\ &= 1.8393^{n-2} \cdot (1.8393^2 + 1.8393 + 1) \end{aligned}$$

hier können wir verwenden, dass $1.8393^2 + 1.8393 + 1 \leq 1.8393^3$ gilt, und erhalten

$$\leq 1.8393^{n-2} \cdot 1.8393^3 = 1.8393^{n+1}$$

Somit gilt $g(n) \leq 1.8393^n$ für alle n.

Aufgabe 3: Dominating Set auf speziellen Graphen 4 + 5 + 3 Punkte

Sei $G = (V, E)$ ein Graph. Eine *dominierende Menge* ist eine Teilmenge $X \subseteq V$, sodass für jeden Knoten $v \in V$ ein Knoten aus $N[v] = \{v\} \cup N(v)$ in X enthalten ist. Das Problem Dominating Set besteht darin, eine möglichst kleine Dominierende Menge in einem gegebenen Graphen zu berechnen.

Teilaufgabe (a) Gib einen polynomiellen Algorithmus an, der Dominating Set auf Bäumen löst. Gelingt es dir eine lineare Abhängigkeit der Laufzeit von der Graphgröße zu erhalten?

Lösung 3a:

Wir lösen das Problem mithilfe eines DPs. Dazu nehmen wir an, dass der Baum gewurzelt ist. Für jeden Knoten $v \in V$, sei T_v der Teilbaum, der aus v und allen seinen Nachfolgern besteht. Sei V_v die Knotenmenge von T_v. Wir nennen einen Knoten *abgedeckt* (bezüglich einer Knotenmenge), wenn der Knoten oder mindestens einer seiner Nachbarn in der Knotenmenge enthalten ist.

Angenommen wir haben bereits Knoten in T_v gewählt und möchten zusätzlich Knoten aus $V \setminus V_v$ wählen um ein möglichst kleines Dominating Set zu erhalten. Dies ist nur möglich wenn die Nachbarn von v in T_v abgedeckt sind. Es können somit folgende Fälle auftreten: (1) v ist ausgewählt, (2) v ist nicht ausgewählt, aber durch ein Kind abgedeckt, oder (3) v ist nicht ausgewählt und nicht abgedeckt.

Für das DP definieren wir also

- $X[v, 1]$ als die Größe des kleinsten Dominating Set von T_v, bei dem v ausgewählt ist,
- $X[v, 2]$ als die Größe des kleinsten Dominating Set von T_v, bei dem v nicht ausgewählt ist, aber ein Kind von v,

- $X[v,3]$ als die Größe der kleinsten Teilmenge von V_v, bei der alle Knoten außer v abgedeckt sind und bei der weder v noch Kinder von v ausgewählt sind.

Diese Teillösungen lassen sich dann für jeden Knoten einfach berechnen. Für jedes Blatt ℓ lässt sich dann X mit $X[\ell,1] = 1$, $X[\ell,2] = \infty$, und $X[\ell,3] = 0$ initialisieren. Für einen Knoten v, bei dem X für alle Kinder bereits ausgefüllt wurde, ergeben sich die Werte für X wie folgt.

Für $X[v,1]$ muss v ausgewählt und für alle Kinder kann die günstigste Teillösung verwendet werden. Somit ergibt sich $X[v,1] = 1 + \sum_{c \in \text{child}(v)} \min_{\circ \in \{1,2,3\}} \{X[c,\circ]\}$, was sich in $O(\deg(v))$ Zeit ausrechnen lässt.

Für $X[v,2]$ darf v nicht ausgewählt werden, sondern muss über ein Kind $c^\star$ abgedeckt sein. Wir probieren also alle Wahlen von $c^\star \in \text{child}(v)$ aus und kombinieren die Teillösung $X[c^\star,1]$ mit den jeweils günstigeren Teillösungen der anderen Kinder bei denen jeweils die Wurzel schon abgedeckt ist. Es ergibt sich $X[v,2] = \min_{c^\star \in \text{child}(v)} \left(X[c^\star,1] + \sum_{c \in \text{child}(v) \setminus c^\star} \min\{X[c,1],X[c,2]\}\right)$. Dies lässt sich in $O(\deg(v))$ Zeit berechnen.

Für $X[v,3]$ darf weder v noch ein Kind von v gewählt werden, die Kinder von v müssen aber bereits abgedeckt sein. Es ergibt sich $X[v,3] = \sum_{c \in \text{child}(v)} X[c,2]$. Dies kann in $O(\deg(v))$ berechnet werden.

Insgesamt ergibt sich eine Laufzeit von $O(\sum_{v \in V} \deg(v)) = O(2m) = O(n)$. Beachte, dass der angegebene Algorithmus auch für gewichtetes Dominating Set funktioniert.

Teilaufgabe (b) Was ändert sich wenn wir stattdessen (für konstantes k) eine möglichst kleine k-dominierende Menge finden möchten? Für eine k-dominierende Menge muss für jeden Knoten er selbst oder ein anderer Knoten in Distanz höchstens k ausgewählt werden.

Lösung 3b:

Das DP aus Teilaufgabe (a) lässt sich zwar auch für k-dominating Set erweitern, soweit wir wissen aber nicht ohne erheblichen Aufwand und eine Verschlechterung der Laufzeit.

Stattdessen geben wir einen Greedy-Algorithmus an, der ein kleinstmögliches k-dominating Set berechnet. Wir nehmen außerdem an, dass der Baum gewurzelt ist. Außerdem merken wir uns für jeden Knoten, ob er ausgewählt, durch einen anderen Knoten abgedeckt oder noch nicht abgedeckt ist. Initial sind alle Knoten nicht abgedeckt. Wir suchen nun in jedem Schritt einen noch nicht abgedeckten Knoten v mit möglichst großer Distanz zur Wurzel. Falls v Abstand weniger als k zur Wurzel hat, fügen wir die Wurzel zur Lösung hinzu. Andernfalls fügen wir den Knoten mit Abstand k zu v auf dem Pfad von v zur Wurzel zur Lösung hinzu. Dies wiederholen wir, bis alle Knoten abgedeckt sind.

Es verbleibt zu zeigen, dass der Algorithmus ein minimales k-Dominating Set berechnet. Wir gehen induktiv vor und nehmen an, dass die bereits ausgewählten Knoten Teil einer minimalen Lösung sind. Wir müssen nun zeigen, dass der nächste ausgewählte Knoten v zusammen mit

den bereits ausgewählten Knoten ebenfalls Teil einer minimalen Lösung sind. Entsprechend des Algorithmus ist v entweder die Wurzel, oder hat Entfernung k zu einem noch nicht abgedeckten Knoten w mit maximaler Entfernung zur Wurzel. Sei X ein minimales k-Dominating Set, dass alle vor v ausgewählten Knoten enthält. Falls $v \notin X$, enthält X einen anderen Knoten v', der w abdeckt. Somit hat v' höchstens Abstand k zu w und liegt im Teilbaum unterhalb von v. Betrachte die Knotenmenge $X' = X \cup \{v\} \setminus \{v'\}$, die wir erhalten, indem wir in X v' durch v ersetzen. Falls es einen Knoten x gibt, der von X' nicht abgedeckt wird, wurde dieser in X von v' abgedeckt, aber nicht von v in X'. Also hat x Abstand mehr als k zu v, wurde von keinem vor v ausgewählten Knoten abgedeckt und liegt im Teilbaum unter v. Dies widerspricht der Wahl von w als noch nicht abgedecktem Knoten mit maximaler Entfernung zur Wurzel. Es folgt per Induktion, dass der Algorithmus in jedem Schritt Knoten wählt, die Teil einer minimalen Lösung sind. Folglich wird auch insgesamt eine minimale Lösung berechnet.

Teilaufgabe (c) Wir betrachten nun Dominating Set auf Graphen mit Maximalgrad d. Gib einen FPT-Algorithmus für den kombinierten Parameter Lösungsgröße + d an.

Lösung 3c:

Wie bei Teilaufgabe (b) markieren wir Knoten als ausgewählt, abgedeckt oder nicht abgedeckt, wobei initial alle Knoten nicht abgedeckt sind.

Wir geben einen Algorithmus mit Branching an, der entscheidet, ob es ein Dominating Set der Größe k gibt. Sei v ein nicht ausgewählter Knoten. In einem Dominating Set muss entweder v oder einer seiner bis zu d Nachbarn ausgewählt werden. Der Algorithmus kann auf dieser Entscheidung mit $d + 1$ Möglichkeiten branchen. Da in jedem rekursiven Aufruf ein weiterer Knoten ausgewählt wird, ergibt sich eine maximale Rekursionstiefe von k.

Insgesamt ergibt sich eine Laufzeit von $(d + 1)^k \cdot n^{O(1)}$.

Aufgabe 4: Punkte und Geraden **7 Punkte**

Gegeben seien eine Menge von n Punkten $P = \{p_1, \ldots, p_n\}$ in der Ebene, sowie ein Parameter k. Das Problem Geradenüberdeckung besteht darin zu entscheiden, ob es k Geraden gibt, sodass jeder Punkt $p_i \in P$ auf mindestens einer der Geraden liegt. Gib einen FPT-Algorithmus für dieses parametrisierte Problem an.

Lösung 4:

Das Problem zu entscheiden, ob es eine k-Geradenüberdeckung gibt, lässt sich ähnlich wie Vertex Cover mit Kernbildung lösen.

Zuerst stellen wir fest, dass nur Geraden betrachtet werden müssen, die durch mindestens zwei Punkte gehen, da Geraden, die nur einen Punkt enthalten, ersetzt werden können ohne die Lösung zu vergrößern. Außerdem gilt, dass falls eine Gerade mit p Punkten nicht in der Überdeckung ist,

dann müssen stattdessen p andere Geraden (eine durch jeden Punkt) in der Überdeckung enthalten sein.

Somit können wir folgende Reduktionsregeln formulieren:

Reduktionsregel 1: Falls es eine Gerade mit mehr als k Punkten gibt, dann muss diese in der Lösung enthalten sein. Lösche die Gerade und die Punkte und verringere k um 1.

Reduktionsregel 2: Gibt es mehr als k^2 Punkte, aber nur noch Geraden, die höchstens k Punkte enthalten, dann kann es keine Geradenüberdeckung mit k Geraden geben. Gib in diesem Fall eine konstant große Nein-Instanz zurück.

Falls beide Reduktionsregeln nicht anwendbar sind, gibt es also höchstens k^2 Punkte und k^4 Geraden und wir erhalten einen Problemkern, dessen Größe nur von k abhängt. Ein FPT-Algorithmus für Geradenüberdeckung kann diese Kernbildung anwenden und anschließend das Problem per brute-force lösen.