FGI-1 – Formale Grundlagen der Informatik I

Logik, Automaten und Formale Sprachen

Musterlösung 7: P, NP und NP-Vollständigkeit

Präsenzteil am 19.-22. Mai – Abgabe am 2.-5. Juni 2015

Präsenzaufgabe 7.1: Sei $A \in P$, $B \in NP$, C NP-schwierig und D NP-vollständig. Gelten die folgenden Aussagen unter der Annahme $P \neq NP$ (sofern nicht anders angegeben)? Begründen Sie kurz Ihre Antwort.

1. A ist in Polynomialzeit auf C reduzierbar.
2. B kann nicht in Polynomialzeit entschieden werden.
3. C ist in Polynomialzeit lösbar, falls $P = NP$ gilt.
4. Eine Lösung für D kann in Polynomialzeit verifiziert werden.

Lösung:

1. Da C NP-schwierig ist, ist jedes Problem in NP in Polynomialzeit auf C reduzierbar. Da ferner $P \subseteq NP$ gilt, gilt dies auch für A. Die Aussage gilt also.
2. Diese Aussage gilt nicht allgemein. Dass B in NP ist, schließt auch unter der Annahme $P \neq NP$ nicht aus, dass B in P ist (um dies auszuschließen hätte explizit $B \in NP \setminus P$ gefordert sein müssen). Ist B aber in P, so ist B in Polynomialzeit lösbar.
3. Auch diese Aussage ist falsch. Da C nur NP-schwierig ist, muss C nicht zwingend in NP liegen. Daher kann aus $P = NP$ dann nicht gefolgert werden, dass C nun in P ist und daher in Polynomialzeit gelöst werden kann.
4. Nach (der alternativen) Definition von NP kann eine Lösung für D in Polynomialzeit verifiziert werden. Ein nichtdeterministischer Algorithmus würde die möglichen Lösungen raten und dann verifizieren, dass eine davon stimmt. Die beiden Definitionen sind daher äquivalent (wobei die Richtung von nichtdeterministichem Algorithmus zu verifzierendem Algorithmus hier nicht diskutiert wurde).

   Sofern die Vermutung $P \neq NP$ stimmt, kann eine Lösung für D aber nicht in Polynomialzeit berechnet werden.

Präsenzaufgabe 7.2: Das Independent-Set-Problem ist wie folgt definiert: Gegeben ein ungerichteter Graph $G = (V, E)$ und eine Zahl k. Gibt es k Knoten, die alle paarweise nicht miteinander verbunden sind? D.h. gibt es eine Menge $V' \subseteq V$ mit $|V'| = k$ derart, dass für je zwei Knoten $v_1, v_2 \in V'$ (mit $v_1 \neq v_2$) keine Kante zwischen ihnen verläuft, also $\{v_1, v_2\} \notin E$ gilt?

Zeigen Sie, dass das Independent-Set-Problem NP-vollständig ist.

Hinweis: Wir haben in der Vorlesung nur für ein Graphenproblem nachgewiesen, dass es NP-vollständig ist. Probieren Sie es mit einer Reduktion von diesem. (Sie können auch eine Reduktion von SAT probieren. Das ist aber sehr viel kniffliger.)

Lösung: Wir bezeichnen das Independent-Set-Problem mit IS. Zu zeigen ist

1. dass IS in NP ist,
2. dass eine Reduktion von einem NP-vollständigen Problem auf IS existiert und
3. dass diese Reduktion in Polynomialzeit berechenbar ist.

Dass IS in NP liegt, kann man schnell einsehen: Eine NTM kann die Menge V' der Größe k raten und im Anschluss überprüfen, dass die k Knoten tatsächlich ein *independent set* sind. Um V' aufzubauen, sind nur k Schritte nötig. Anschließend sind k^2 viele Kanten zu überprüfen (bzw. $(k^2-k)/2$, wenn man die reflexiven Kanten ignoriert und beachtet, dass ein ungerichteter Graph vorliegt). Dies geht also in polynomieller Zeit.

Wir geben nun eine Reduktion von *Clique* auf IS an. Die Idee ist, dass eine Clique in einem Graphen G gerade ein *independent set* in dem Graphen ist, der aus G hervorgeht, wenn man Kanten und Nicht-Kanten austauscht. Sei also (G, k) eine Instanz von *Clique*. Wir transformieren dies in eine Instanz von IS wie folgt: Aus G entsteht ein Graph $\overline{G}$, indem wir für je zwei Knoten v_1, v_2 mit $v_1 \neq v_2$, die durch eine Kante verbunden sind, diese Kante löschen, und für je zwei Knoten v_1, v_2 mit $v_1 \neq v_2$, die nicht durch eine Kante verbunden sind, eine Kante hinzufügen. Die Instanz von IS ist das Tupel $(\overline{G}, k)$. Wir müssen nun noch zeigen: $(G, k) \in \mathit{Clique} \Leftrightarrow (\overline{G}, k) \in IS$. Sei also $(G, k) \in \mathit{Clique}$, dann hat G eine Clique aus k Knoten. Seien v_1 und v_2 zwei beliebige, aber verschiedene Knoten der Clique. Dann gibt es in G eine Kante zwischen ihnen. In $\overline{G}$ gibt es aber nach Konstruktion keine Kante zwischen v_1 und v_2 und da dies für alle Knoten der Clique gilt, sind die k Knoten, die in G eine Clique bilden in $\overline{G}$ ein *independent set*. Folglich ist $(\overline{G}, k) \in IS$.

Ebenso argumentiert man, dass wenn $(\overline{G}, k) \in IS$ ist, also ein *independent set* der Größe k existiert, dieses gerade eine Clique in G ist und damit $(G, k) \in \mathit{Clique}$ gilt.

Zuletzt ist die Reduktion leicht in Polynomialzeit berechenbar, da lediglich für je zwei Knoten getestet werden muss, ob eine Kante existiert und diese gesetzt oder gelöscht werden muss. Dies ist leicht in $O(|V|^2)$ möglich.

Übungsaufgabe 7.3:

von
4

1. Zeigen Sie, dass aus $L_1, L_2 \in P$ auch $L_1 \cap L_2 \in P$ und $\overline{L_1} \in P$ folgt, dass also P gegenüber Durchschnitts- und Komplementbildung abgeschlossen ist.

2. Zeigen Sie, dass aus $L_1, L_2 \in NP$ auch $L_1 \cap L_2 \in NP$ folgt.

Lösung:

1. Seien $L_1, L_2 \in P$ und seien A_1 und A_2 Algorithmen, die L_1 bzw. L_2 in Polynomialzeit entscheiden (d.h. auf jeder Eingabe $x \in \Sigma^*$ anhalten und mit 1 (ja) oder 0 (nein) antworten je nachdem ob $x \in L_1$ gilt oder nicht (und entsprechend für L_2)). Ein Algorithmus A, der $L_1 \cap L_2$ entscheidet arbeitet wie folgt: Bei Eingabe $x \in \Sigma^*$ wird zunächst der Algorithmus A_1 gestartet mit Eingabe x gestartet. Liefert dieser als Ergebnis 0, so terminiert A mit Ergebnis 0 (da $x \notin L_1$ und damit auch $x \notin L_1 \cap L_2$ gilt). Ist hingegen $A_1(x) = 1$, also $x \in L_1$, so startet A als nächstes A_2 mit Eingabe x (hierfür ist es u.U. nötig x zu Anfang zu kopieren, da A_1 die Eingabe x u.U. manipuliert). Antwortet A_2 mit 0, so liefert A als Ergebnis wieder 0. Liefert A_2 als Ergebnis aber 1, so wissen wir wegen $A_1(x) = 1$, dass $x \in L_1$ gilt und wegen $A_2(x) = 1$, dass auch $x \in L_2$ gilt, insgesamt also $x \in L_1 \cap L_2$ gilt und A als Ergebnis 1 liefern kann.

   Die Korrektheit von A folgt aus obiger Beschreibung. Die Laufzeit von A ist polynomiell, da die Laufzeit von A_1 als auch A_2 polynomiell ist und diese lediglich hintereinander ausgeführt werden müssen. Die zusätzlichen Operationen von A (das anfängliche Kopieren von x, der Test ob A_1 bei Eingabe x eine 1 zurückliefert usw.) sind alle in Polynomialzeit oder sogar in konstanter Zeit möglich so dass sich insgesamt eine polynomielle Laufzeit für A ergibt.

2. Sei $L_1 \in P$ und sei A_1 ein Algorithmus, der L_1 entscheidet. Ob $\overline{L_1} \in P$ zu zeigen ist es lediglich nötig die Ausgabe von A_1 zu invertieren. Ein Algorithmus A für $\overline{L_1}$ startet bei Eingabe x also zunächst A_1 (mit Eingabe x) und invertiert dann das Ergebnis, d.h. liefert A_1 als Rückgabe 1, so liefert A eine 0, liefert A_1 eine 0, so liefert A eine 1.

   Gilt also $x \in L_1$ (in diesem Fall ist $A_1(x) = 1$), so ist $A(x) = 0$, was $x \notin \overline{L_1}$ bedeutet (was wegen $x \in L_1$ korrekt ist). Entsprechend für $x \notin L_1$.

   Dass A in polynomieller Zeit läuft, wenn A_1 dies tut, ist klar, da nur ein weiterer Schritt nötig ist.

3. Seien $L_1, L_2 \in NP$ und seien A_1 und A_2 Algorithmen für L_1 bzw. L_2, die in polynomieller Zeit unter Nutzung eines Zertifikats *verifizieren*, ob ein $x \in \Sigma^*$ tatssächlich in L_1 (bzw. L_2) liegt. Ein Verifikationsalgorithmus A für $L_1 \cap L_2$ arbeitet wie folgt: Die Instanz sei mit x bezeichnet, das Zertifikat ist ein Tupel (y_1, y_2). A akzeptiert die Eingabe $(x, (y_1, y_2))$ genau dann, wenn $A_1(x, y_1) = 1$ und $A_2(x, y_2) = 1$ gilt. Da A_1 und A_2 in polynomieller Zeit arbeiten, tut dies auch A. Da y_1 und y_2 in $O(x^c)$ für ein festes c liegen, ist auch das Tupel (y_1, y_2) in $O(x^c)$ und A damit ein NP-Algorithmus für $L_1 \cap L_2$.

   Einfacher sieht man dies, wenn man mit *Nichtdeterminismus* arbeitet. Ein Algorithmus A' für $L_1 \cap L_2$ startet bei Eingabe x dann zunächst den NP-Algorithmus A_1' für L_1. Bei jenen Berechnungen, auf denen A_1' akzeptiert, wird nachfolgend der NP-Algorithmus A_2' für L_2 mit Eingabe x gestartet. (Wie bei P oben ist es auch hier u.U. nötig die Eingabe x zu Beginn einmal zu kopieren.) Akzeptiert auch A_2' die Eingabe x bei mindestens einer Rechnung, so akzeptiert A'. In A' gibt es so genau dann eine akzeptierende Rechnung, wenn sowohl A_1'

als auch A_2' akzeptieren, wenn also $x \in L_1 \cap L_2$ gilt. Die Laufzeit von A' ist polynomiell, da sowohl A_1' als auch A_2' eine polynomielle Laufzeit haben.

Übungsaufgabe 7.4: Zeigen Sie, dass das Färbungsproblem (siehe unten) in NP liegen, indem Sie einen Verifikationsalgorithmus mit polynomialer Laufzeit für dieses Problem angeben. Beachten Sie dabei, dass Sie das Zertifikat spezifizieren müssen.

von
2

- Das Färbungsproblem
  - Eingabe: Ein ungerichteter Graph $G = (V, E)$ und ein $k \in \mathbb{N}$.
  - Frage: Kann G mit k Farben gefärbt werden, d.h. gibt es eine Funktion $c : V \to \{1, \ldots, k\}$ derart, dass $c(u) \neq c(v)$ für jede Kante $\{u, v\} \in E$ gilt?

Lösung: Ein Verifikationsalgorithmus A für das Färbungsproblem erhält als Eingabe einen ungerichteten Graphen G und die Zahl $k \in \mathbb{N}$. Das Zertifikat ist eine Folge von $|V|$ Zahlen aus $\{1, \ldots, k\}$. Die i-te Zahl gibt dabei die Farbe an, die dem i-ten Knoten zugewiesen wird (das Zertifikat ist also gerade eine Färbung der Knoten). A überprüft dann, ob die Färbung korrekt ist, d.h. für jede Kante wird überprüft, ob den Endknoten unterschiedliche Farben zugewiesen wurden. Ist dies der Fall akzeptiert A, sonst nicht.

Der Zertifikat hat eine polynomielle Länge in der Länge von G und k. Ferner ist die Laufzeit von A polynomiell, da lediglich alle Kanten betrachtetet werden müssen. Das Problem ist damit in NP.

von
4

Übungsaufgabe 7.5: Betrachten Sie das folgende Problem:

Gegeben: Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$.

Frage: Gibt es Teilmengen $V \subseteq V_1$ und $E \subseteq E_1$ derart, dass $|V| = |V_2|$ und $|E| = |E_2|$ gilt und eine bijektive Abbildung $f : V_2 \to V$ existiert mit $\{u, v\} \in E_2$ genau dann, wenn $\{f(u), f(v)\} \in E$?

1. Zeigen Sie, dass das Problem in NP liegt, indem Sie einen NP-Algorithmus angeben, der das Problem löst.
2. Beweisen Sie, dass das Problem NP-hart (und damit insgesamt NP-vollständig) ist, indem Sie eine Reduktion von einem Ihnen bekannten NP-vollständigen Problem angeben.

Lösung:

1. In vielen Fällen ist es recht einfach, einen NP-Algorithmus für ein Problem anzugeben, sofern einer existiert. Meist wird ein Objekt mit bestimmten Eigenschaften gesucht. Dieses Objekt kann man nichtdeterministisch raten und dann in Polynomialzeit (oft sogar deterministisch) die Eigenschaften überprüfen. Sobald eines der geratenen Objekte die Eigenschaften erfüllt, wird 'ja' ausgegeben. Da man alle möglichen Objekte rät, muss eines, das die Eigenschaften erfüllt, dabei sein, sofern denn eines existiert.

   Hier wird eine Teilmenge V der Knotenmenge V_1, eine Teilmenge E der Kantenmenge E_1 und eine Funktion $f : V_2 \to V$ mit speziellen Eigenschaften gesucht. Man könnte nun alle diese Teilmengen und alle Funktionen raten und dann überprüfen, ob sie die Eigenschaften erfüllen. In diesem Fall kann man aber sofort einige der Eigenschaften in den Rateprozess übernehmen. So braucht man bspw. nicht alle Teilmengen $V \subseteq V_1$ zu raten, sondern nur solche, die aus genau $|V_2|$ Elementen bestehen.

   Im einzelnen arbeitet ein NP-Algorithmus für obiges Problem wie folgt:

   (a) Wähle (bzw. rate) nichtdeterministisch Teilmengen $V \subseteq V_1$ und $E \subseteq E_1$ mit $|V| = |V_2|$ und $|E| = |E_2|$ (die Werte von $|V_2|$ und $|E_2|$ lassen sich dabei leicht aus der Eingabe ermitteln).

   (b) Wähle (bzw. rate) nichtdeterministisch eine Bijektion $f : V_2 \to V$. (Damit f eine Bijektion ist, wird jedem Element aus V_2 ein Element aus V zugewiesen, aber jedes Element aus V tritt dabei nur einmal auf. Da $|V_2| = |V|$ gilt, ist dies dann eine Bijektion.)

   (c) Für je zwei Knoten $u, v \in V_2$ mit $u \neq v$ prüfe deterministisch, dass für $\{u, v\} \notin E_2$ auch $\{f(u), f(v)\} \notin E$ und für $\{u, v\} \in E_2$ auch $\{f(u), f(v)\} \in E$ gilt. (Man beachte, dass es hier sogar genügt nur die Kanten aus G_2 zu betrachten, also nur eine Richtung zu testen, um die 'genau dann, wenn' Beziehung zu testen. Dies liegt daran, dass man E rät, E genau so groß wie E_2 ist und f bijektiv ist.)

   (d) Wird der dritte Schritt erfolgreich absolviert, akzeptiere, ansonsten lehne ab.

   Der Algorithmus löst obiges Problem, denn sollte es Teilmengen und eine Funktion mit den gewünschten Eigenschaften geben, dann werden sie im ersten und zweiten Schritt auch geraten. Ebenso akzeptiert der Algorithmus nur dann, wenn solche Teilmengen und die Funktion existiert. Der Algorithmus läuft ferner in Polynomialzeit, da das nichtdeterministische Raten in $O(V_2 + E_2 + V_2) = O(V_2 + E_2)$ möglich ist (zumindest dann, wenn die benutzten Datenstrukturen einem schnell erlauben zu überprüfen, ob ein gewählter Knoten schon in der Menge ist; ansonsten kann es hier zu quadratischem Mehraufwand kommen, was aber

immer noch ein Polynom in der Eingabegröße ist) und der anschließende Test im dritten Schritt in $O(V_2^2)$ ist.

2. Während man den Algorithmus in der ersten Teilaufgabe sogar formulieren kann, ohne dass man das Problem genau verstanden haben muss, ist es i.A. natürlich hilfreich, das Problem, das man betrachtet, zu verstehen. Insbesondere, wenn man eine Reduktion für einen Vollständigkeitsbeweis angeben will, ist dies nützlich, da man dann schneller eine Idee haben kann, welches Problem Ähnlichkeiten aufweist. Bei dem hier betrachteten Problem sind zwei Graphen gegeben und die Frage ist im Grunde genommen, ob der Graph G_2 in dem Graphen G_1 als Teilgraph auftritt. Die Bijektion dient lediglich dazu, dass die Knoten verschieden benannt oder in verschiedener Reihenfolge notiert sein können. Ist bspw. G_2 ein Dreieck und G_1 ein Kreis aus vier Knoten, so tritt G_2 in G_1 nicht auf. Enthält G_1 hingegen noch zwei weitere Kanten, so dass G_1 also der vollständige Graph auf vier Knoten ist (vier Knoten und alle sind paarweise miteinander verbunden), so tritt G_2 als Teilgraph in G_1 auf (nach Umbenennung der Knoten), auch wenn G_1 noch etliche Kanten mehr enthält.

   Dieses Problem des 'Findens eines Teilgraphen' ist sehr ähnlich zum Clique-Problem. Im Clique-Problem wird gefragt, ob ein gegebener Graph G eine Clique einer bestimmten Größe m enthält, d.h. ob G m Knoten enthält, die alle paarweise miteinander verbunden sind (vgl. Theorem 14.23). Tatsächlich kann man leicht eine Reduktion von Clique auf das obige Problem angeben. Sei dazu (G, m) eine Instanz des Clique-Problems. Wir setzen $G_1 = G$ und $G_2 = K^m$, wobei K^m der vollständige Graph auf m Knoten ist. Die Reduktion ist deterministisch in Polynomialzeit möglich, da lediglich G kopiert werden muss und da, um G_2 zu erzeugen, lediglich $O(m^2)$ viele Schritte nötig sind. Hier tritt nun allerdings ein Problem auf: Ist m eine Zahl der Eingabe, so ist die Eingabe*länge* hierfür lediglich in $O(\log m)$, so dass eine Transformation, die $O(m^2)$ Zeit benötigt, gar nicht polynomiell in der Eingabelänge ist, sondern exponentiell! Nun kann man aber getrost von $m \leq |V(G)|$ ausgehen und da $V(G)$ auch Teil der Eingabe ist (insb. die Eingabelänge mindestens $|V(G)|$ ist), ist $O(m^2)$ wieder polynomiell in der Eingabe. Im Fall $m > |V(G)|$ ist (G, m) sofort eine 'nein'-Instanz von Clique (ein Graph mit $|V(G)|$ Knoten kann keine Clique aus m Knoten enthalten, wenn $m > |V(G)|$ gilt) und kann auf eine (vorgefertigte) 'nein'-Instanz des obigen Problems abgebildet werden, bspw. auf das oben angesprochene Beispiel, in dem G_2 ein Dreieck und G_1 ein Kreis aus vier Knoten ist.

   Es ist also nur noch zu zeigen, dass (G, m) eine 'ja'-Instanz von Clique ist (d.h. G enthält m paarweise miteinander verbundene Knoten) genau dann, wenn (G_1, G_2) eine 'ja'-Instanz des neuen Problems ist. Dies sieht man aber schnell ein: Wenn G eine Clique aus m Knoten enthält, dann enthält $G_1 = G$ einen Teilgraphen, der zu $G_2 = K^m$ isomorph ist (d.h. die Abbildung f kann angegeben werden), da dies im Grunde genommen die gleiche Fragestellung ist. Ebenso enthält G_1 einen Teilgraphen, der zu G_2 isomorph ist, genau dann, wenn $G = G_1$ einen $K^m = G_2$ als Teilgraphen enthält, also eine Clique aus m Knoten. Damit ist die angegebene Reduktion korrekt und in der nötigen Zeitschranke berechenbar, woraus folgt, dass das Problem aus der Aufgabenstellung NP-hart ist. Mit der ersten Teilaufgabe folgt dann, dass es sogar NP-vollständig ist.

   Man beachte, dass, so wie das Problem gestellt ist, eine Reduktion von Independent Set nicht so einfach möglich ist. Nimmt man als Graph G_2 dann einen Graphen mit m isolierten Knoten und fragt, dann ob (G_1, G_2) eine 'ja'-Instanz ist, so ist die Antwort bereits positiv, sobald G_1 m Knoten enthält. Dies liegt daran, dass man als Menge E die leere Menge wählen kann, die Bedingung an f und die Kanten ist dann erfüllt. Was einen aber dann wahrscheinlich eher interessiert ist, ob in G_1 m Knoten auftreten, die paarweise nicht miteinander verbunden sind (evtl. aber noch mit anderen Knoten). Um das Problem so umzuformulieren, dass diese Eigenschaft überprüft wird, entfernt man E aus der Frage und überprüft dann nicht

$\{u, v\} \in E_2$ gdw. $\{f(u), f(v)\} \in E$, sondern $\{u, v\} \in E_2$ gdw. $\{f(u), f(v)\} \in E_1$. Dann klappt sowohl eine Reduktion von Clique wie oben als auch eine von Independent Set mit der gleichen Idee. Gesucht werden dann nicht mehr nur Teilgraphen, sondern es wird geprüft, ob Graphen mit der exakt gleichen Struktur wie von G_2 vorgegeben in G_1 auftreten.

von
2

Übungsaufgabe 7.6: Sei A ein Algorithmus, der eine konstante Anzahl von Aufrufen von Unterroutinen enthält. Zählt man jeden dieser Aufrufe als einen Schritt, so sei A ein Polynomialzeit-Algorithmus. Zeigen Sie, dass A insgesamt in polynomialer Zeit läuft, wenn die Unterroutinen in polynomialer Zeit laufen. Zeigen Sie ferner, dass ein Algorithmus mit exponentieller Laufzeit entstehen kann, wenn ein Algorithmus eine polynomiale Anzahl von Aufrufen von Unterroutinen mit polynomialer Laufzeit enthält.

Lösung: Es macht zunächst den Anschein, als könne eine polynomielle Anzahl von Aufrufen ($p(n)$ Aufrufe, wobei die Eingabe die Länge n hat) einer Unterroutine mit polynomieller Laufzeit (sei diese durch das Polynom q gegeben) nicht exponentiell werden, da $p(n) \cdot q(n)$ wieder ein Polynom ist. Allerdings kann die Größe der Eingabe für die Unterroutine stark wachsen.

Um dies zu illustrieren nehmen wir an die Unterroutine foo hat eine Laufzeit von $q(x) = x^2$. Nehmen wir ferner an, sie erwartete als Eingabe eine Zeichenkette der Länge x und produziert als Ausgabe eine Zeichenkette der Länge x^2 (eine längere Zeichenkette kann nicht als Ausgabe produziert werden, da die Laufzeit ja durch x^2 beschränkt ist!). Wenn wir nun anfänglich eine Zeichenkette *str* der Länge n als Eingabe (für das gesamte Programm) erhalten und mit diesem als Argument foo aufrufen, so erhalten wir nun eine Zeichenkette der Länge n^2 zurück. Benutzen wir diese nun *wieder als Argument* für foo, rufen also insgesamt $foo(foo(str))$ auf, so haben wir bereits eine Zeichenkette der Länge $(n^2)^2 = n^4$. Wiederholt man dies, so erhält man Zeichenketten der Länge n^8, n^{16}, n^{32} usw. Allgemein erhält man so nach m Aufrufen eine Zeichenkette der Länge $(n^2)^m = n^{2 \cdot m}$. Ist die Anzahl der Aufrufe nun durch ein Polynom p wie ganz oben angesprochen gegeben, so erhält man eine Länge von $n^{2 \cdot p(n)}$, was exponentiell ist (da $p(n)$ im Exponenten steht). Da die Laufzeit der Unterroutine foo durch $q(x) = x^2$ gegeben war ergibt sich hier dann auch eine insgesamt exponentielle Laufzeit.

Hat man hingegen nur eine konstante Anzahl m an Aufrufen so erreicht man auch so maximal eine Laufzeit von $q(n)^m$, was weiterhin eine polynomielle Laufzeit ist, da m eine Konstante ist.

Informationen und Unterlagen zur Veranstaltung unter:

`http://www.informatik.uni-hamburg.de/TGI/lehre/vl/SS15/FGI1`