Prof. Dr. Leif Kobbelt
Stefan Dollase, Ira Fesefeldt, Alexandra Heuschling, Gregor Kobsik

Lösung - Übung 3

Tutoraufgabe 1 (Laufzeitanalyse):

Gegeben sei ein Algorithmus, der für ein Array von Integern überprüft, ob die Einträge aufsteigend sortiert sind:

```
1 def is_sorted(a):
2     i = 1
3     while i < len(a):
4         if a[i - 1] > a[i]:
5             return False
6         i += 1
7     return True
```

Bei Betrachtung der Laufzeit wird angenommen, dass die Vergleiche jeweils eine Zeiteinheit benötigen (siehe Zeilen 3 und 4). Die Laufzeit aller weiteren Operationen wird vernachlässigt.
Sei $n \in \mathbb{N}$ die Länge des Arrays a.

a) Bestimmen Sie in Abhängigkeit von n die Best-case Laufzeit $B(n)$. Begründen Sie Ihre Antwort. Geben Sie für jedes n ein Beispielarray an, bei dem diese Laufzeit erreicht wird.

b) Bestimmen Sie in Abhängigkeit von n die Worst-case Laufzeit $W(n)$. Begründen Sie Ihre Antwort. Geben Sie für jedes n ein Beispielarray an, bei dem diese Laufzeit erreicht wird.

c) Bestimmen Sie in Abhängigkeit von n die Average-case Laufzeit $A(n)$. Begründen Sie Ihre Antwort. Hierzu nehmen wir für $n \geq 2$ folgende Verteilung der Eingaben an:

- $\Pr(\texttt{a[0]} = 1) \;=\; 1$,
- $\Pr(\texttt{a[n} - 1\texttt{]} = 0) \;=\; 1$ und
- für alle $\texttt{i} \in \{1, \ldots, n-2\}$ gilt: $\Pr(\texttt{a[i]} = 0) \;=\; \Pr(\texttt{a[i]} = 1) \;=\; 0.5$.

Der erste Eintrag ist also immer 1 und der letzte Eintrag ist immer 0. Die übrigen Einträge sind mit gleicher Wahrscheinlichkeit entweder 0 oder 1. Insbesondere ist die Wahrscheinlichkeit, dass das Array aufsteigend sortiert ist immer 0. Für $n < 2$ ist die Verteilung der Eingaben nicht relevant.

Hinweise:

- Wir gehen davon aus, dass die Indizierung von Arrays mit 0 beginnt.
- Geben Sie die geforderten Laufzeiten in geschlossener Form (d.h. ohne Summenzeichen, $\Pr(\cdots)$ oder ähnliches) an.

Lösung

a) Der Vergleich in Zeile 3 wird auf jeden Fall einmal ausgeführt. Wir unterscheiden zwei Fälle:

- Falls $n < 2$, so gilt die Schleifenbedingung nicht und der Algorithmus terminiert sofort nach insgesamt einem Vergleich. Beispieleingaben: `a=[]` (für $n = 0$) und `a=[1]` (für $n = 1$).
- Falls $n \geq 2$ gilt die Schleifenbedingung und wir führen zusätzlich den Vergleich in Zeile 4 aus. Falls $\texttt{a[0]} > \texttt{a[1]}$ terminiert der Algorithmus nach insgesamt zwei Vergleichen. Beispieleingaben: `E=[2, 1, ...]`

Wir erhalten also:

$$B(n) = \begin{cases} 1 & \text{, falls } n < 2 \\ 2 & \text{, falls } n \geq 2 \end{cases}$$

b) Wir unterscheiden wieder zwei Fälle:

- Falls $n < 2$, gilt wie bei **a)**, dass der Algorithmus nach insgesamt einem Vergleich terminiert. Beispieleingabe: `a = []` (für $n = 0$) und `a = [1]` (für $n = 1$).
- Falls $n \geq 2$ wird die Worst-case Laufzeit genau dann erreicht, wenn das Array bereits sortiert ist. Nur dann wird die Schleifenbedingung (Zeile 3) genau n mal überprüft und die `if`-Bedingung (Zeile 4) genau $n-1$ mal. Beispieleingabe: `a = [1, 2, 3, ..., n]`

Wir erhalten also:

$$W(n) = \begin{cases} 1 & \text{, falls } n < 2 \\ 2n-1 & \text{, falls } n \geq 2 \end{cases}$$

c) Wir unterscheiden wieder zwei Fälle:

- Falls $n < 2$, gilt wie bei **a)**, dass der Algorithmus nach insgesamt einem Vergleich terminiert.
- Falls $n \geq 2$, betrachten wir für jedes $k \in \{1, \ldots, n-1\}$ den Fall dass

$$\texttt{a[k]} = 0 \text{ und für alle } j < k\colon \texttt{a[j]} = 1.$$

Hinweis:

– Zur einfachen Berechnung ist es hier wichtig, dass die Fälle disjunkt sind, d.h. die Eingaben überschneiden sich nicht für verschiedene k. Nur so können wir die Wahrscheinlichkeit und die Laufzeit für jedes k einzeln bestimmen und am Ende aufsummieren.

In so einem Fall stellen wir fest, dass die `if`-Bedingung in Zeile 4 zum ersten mal gilt, wenn `i` $= k$ ist. Bis dahin durchlaufen wir die Schleife genau k mal und in jedem Durchlauf werden zwei Vergleiche vorgenommen. Die Laufzeit beträgt also $2 \cdot k$.

Als nächstes berechnen wir die Wahrscheinlichkeit eines solchen Falles.

– Falls $k = n-1$, gilt $\texttt{a} = [1, 1, \ldots, 1, 0]$. Da die Wahrscheinlichkeit eines einzelnen Eintrages unabhängig von den restlichen Einträgen ist, hat diese Eingabe die Wahrscheinlichkeit:

$$\underbrace{\Pr(\texttt{a[0]} = 1)}_{=1} \cdot \underbrace{\Pr(\texttt{a[1]} = 1) \cdot \ldots \cdot \Pr(\texttt{a[n-2]} = 1)}_{= 0.5 \cdot \ldots \cdot 0.5 = 0.5^{n-2}} \cdot \underbrace{\Pr(\texttt{a[n-1]} = 0)}_{=1} = 0.5^{n-2}$$

– Falls $k < n-1$, gilt $\texttt{a} = [1, 1, \ldots, 1, 0, \bullet, \ldots, \bullet]$, wobei $\bullet$ für einen beliebigen Eintrag steht. Wieder ist die Wahrscheinlichkeit eines einzelnen Eintrages unabhängig von den restlichen Einträgen. Somit ist die Wahrscheinlichkeit für eine solche Eingabe:

$$\underbrace{\Pr(\texttt{a[0]} = 1)}_{=1} \cdot \underbrace{\Pr(\texttt{a[1]} = 1) \cdot \ldots \cdot \Pr(\texttt{a[k-1]} = 1) \cdot \Pr(\texttt{a[k]} = 0)}_{= 0.5 \cdot \ldots \cdot 0.5 \cdot 0.5 = 0.5^{k}} = 0.5^{k}$$

Hinweis:

– Wenn wir richtig gerechnet haben, müssten sich die Wahrscheinlichkeiten aller Fälle ($k \in \{1, \ldots, n-1\}$) zu 1 aufsummieren. Wir testen dies (und nutzen die bekannte[a] Gleichung zur

geometrische Reihe):

$$\begin{aligned} 0.5^{n-2} + \sum_{k=1}^{n-2} 0.5^k &= 0.5^{n-2} + \left(\sum_{k=0}^{n-2} 0.5^k\right) - 0.5^0 \\ &= 0.5^{n-2} + \frac{1 - 0.5^{n-1}}{1 - 0.5} - 1 \\ &= 0.5^{n-2} + 2 \cdot \left(1 - 0.5^{n-1}\right) - 1 \\ &= 0.5^{n-2} + 2 - 0.5^{n-2} - 1 \; = \; \mathbf{1} \end{aligned}$$

Zuletzt summieren wir die Produkte aus Laufzeit und Wahrscheinlichkeit für jedes k auf (und nutzen die etwas unbekanntere Variante[b] der geometrischen Reihe):

$$\begin{aligned} 2 \cdot (n-1) \cdot 0.5^{n-2} + \sum_{k=1}^{n-2} 2 \cdot k \cdot 0.5^k &= 2 \cdot \left((n-1) \cdot 0.5^{n-2} + \sum_{k=0}^{n-2} k \cdot 0.5^k\right) \\ &= 2 \cdot \left((n-1) \cdot 0.5^{n-2} + \frac{(n-2) \cdot 0.5^n - (n-1) \cdot 0.5^{n-1} + 0.5}{(0.5-1)^2}\right) \\ &= 2 \cdot \left((n-1) \cdot 0.5^{n-2} + (n-2) \cdot 0.5^{n-2} - (n-1) \cdot 0.5^{n-3} + 0.5^{-1}\right) \\ &= 2 \cdot \left((n-1) \cdot 0.5^{n-2} + (n-2) \cdot 0.5^{n-2} - (n-1) \cdot 0.5^{n-2} \cdot 2 + 2\right) \\ &= 2 \cdot \left(((n-1) + (n-2) - 2(n-1)) \cdot 0.5^{n-2} + 2\right) \\ &= 2 \cdot \left((n - 1 + n - 2 - 2n + 2) \cdot 0.5^{n-2} + 2\right) \\ &= 2 \cdot \left(-1 \cdot 0.5^{n-2} + 2\right) \\ &= 4 - 0.5^{n-3} \end{aligned}$$

Wir erhalten also:

$$A(n) = \begin{cases} 1 & \text{, falls } n < 2 \\ 4 - 0.5^{n-3} & \text{, falls } n \geq 2 \end{cases}$$

[a] https://de.wikipedia.org/wiki/Geometrische_Reihe#Berechnung_der_(endlichen)_Partialsummen_einer_geometrischen_Reihe
[b] https://de.wikipedia.org/wiki/Geometrische_Reihe#Verwandte_Summenformel_1

Tutoraufgabe 2 (Rekursionsgleichung):

Finden Sie für die Rekursionsgleichung F, die wir als

$$F(1) = 1 \text{ und } F(n) = F(n-1) + n^2 + n \text{ für } n > 1$$

definieren, ein Polynom $g : \mathbf{R}_+ \to \mathbf{R}_+$ als obere Abschätzung an, welche nicht rekursiv definiert ist. Beweisen Sie anschließend per vollständiger Induktion, dass ihr Vorschlag tatsächlich eine obere Abschätzung ist, also dass $F(n) \leq g(n)$ ist. Geben Sie schließlich auch eine Abschätzung der Funktion g in O-Notation an.

Lösung

Wir haben die Vermutung, dass die Formel sich kubisch verhalten könnte. Daher schätzen wir die Rekursionsgleichung durch die kubische Formel $g(n) = an^3 + bn^2 + cn + d$ ab. Hierfür benötigen wir jedoch noch Werte für alle Koeffizienten. Diese versuchen wir während des Beweis für die vollständige Induktion zu sammeln.
Für den Induktionsanfang haben wir: $F(1) = 1 \leq a + b + c + d$.
Nun nehmen wir an, die Aussage gelte für ein beliebiges aber festes n als unsere Induktionshypothese.

Für den Induktionsschritt haben wir:

$$\begin{aligned} F(n+1) &= F(n) + (n+1)^2 + (n+1) \\ &\leq an^3 + bn^2 + cn + d + (n+1)^2 + (n+1) && \text{(Induktions Hypothese)} \\ &\leq an^3 + bn^2 + cn + d + n^2 + 2n + 1 + n + 1 && \text{(Vereinfachen)} \\ &= an^3 + (b+1)n^2 + (c+3)n + d + 2 && \text{(Vereinfachen)} \end{aligned}$$

Auf der anderen Seite haben wir:

$$\begin{aligned} g(n+1) &= a(n+1)^3 + b(n+1)^2 + c(n+1) + d \\ &= an^3 + 3an^2 + 3an + a + bn^2 + 2bn + b + cn + c + d && \text{(Auflösen)} \\ &= an^3 + (3a+b)n^2 + (3a+2b+c)n + a + b + c + d && \text{(Polynom Normalform herbeiführen)} \end{aligned}$$

Damit nun $F(n+1) \leq g(n+1)$ ist, müssen folgende Bedingungen gelten:

$$\begin{aligned} 1 &\leq a + b + c + d \\ b + 1 &\leq 3a + b \\ c + 3 &\leq 3a + 2b + c \\ d + 2 &\leq a + b + c + d \end{aligned}$$

Dies ist zum Beispiel für die Belegung $a = \frac{1}{3}$, $b = 1$, $c = \frac{2}{3}$ und $d = -1$ der Fall. Damit bekommen wir dann den Induktionsbeweis:
Induktionsanfang: $F(1) = 1 = \frac{1}{3} + 1 + \frac{2}{3} - 1$
Induktionshypothese gelte für ein beliebiges, aber festes n.
Induktionsschritt:

$$\begin{aligned} g(n+1) &= \frac{1}{3}(n+1)^3 + (n+1)^2 + \frac{2}{3}(n+1) - 1 \\ &= \frac{1}{3}n^3 + n^2 + n + \frac{1}{3} + n^2 + 2n + 1 + \frac{2}{3}n + \frac{2}{3} - 1 && \text{(Auflösen)} \\ &= \frac{1}{3}n^3 + n^2 + \frac{2}{3}n^2 - 1 + n^2 + 3n + 2 && \text{(Neu Anordnen)} \\ &= F(n) + n^2 + 3n + 2 && \text{(Induktionshypothese)} \\ &= F(n) + (n+1)^2 + (n+1) && \text{(Neu Anordnen)} \\ &= F(n+1) && \text{(Definition)} \end{aligned}$$

Damit haben wir nun dass $F(n) \in O(n^3)$ ist da auch $g(n) \in O(n^3)$ ist.

Tutoraufgabe 3 (Master Theorem):

Benutzen Sie das Master Theorem um für folgende Rekursionsgleichungen $T(n)$ eine möglichst gute O-Klasse anzugeben, in der $T(n)$ liegt.

a)

$$T(1) = 1$$
$$T(n) = 16 \cdot T(\frac{n}{2}) + n^5 + \log_4(n) \qquad \text{for } n > 1$$

b)

$$T(1) = 2$$
$$T(n) = 27 \cdot T(\frac{n}{3}) + n^3 + n^2 \qquad \text{for } n > 1$$

c)

$T(1) = 3$

$T(n) = 26 \cdot T(\frac{n}{5}) + n \cdot \log_2 n \qquad \text{for } n > 1$

Lösung

a) Wir wählen $p = 5$, dann ist $n^5 + \log_4(n) \in O(n^5)$ und $16 < 32 = 2^5$. Somit ist $T(n) \in O(n^5)$ nach dem Master Theorem.

b) Wir wählen $p = 3$, dann ist $n^3 + n^2 \in O(n^3)$ und $27 = 3^3$. Somit ist $T(n) \in O(n^3 \cdot \log(n))$ nach dem Master Theorem.

c) Wir wählen $p = 2$, dann ist $n \cdot \log_2(n) \in O(n^2)$ und $26 > 25 = 5^2$. Somit ist $T(n) \in O(n^{\log_5(26)})$ nach dem Master Theorem.

Aufgabe 4 (Laufzeitanalyse): 10 Punkte

Betrachten Sie den folgenden Algorithmus, welcher für ein Array `a` der Länge n mit Zahlen zwischen 0 und $k-1$ zurückgibt, ob in `a` eine Zahl mehrfach enthalten ist.

```
1 seen = [False] * k
2 for i in a
3   if seen[i]:
4     return True
5   else:
6     seen[i] = True
7 return False
```

Hinweis:

- `[False]*k` erzeugt ein Array der Länge k in dem alle Einträge `False` sind.

Bestimmen Sie die Best-Case, Worst-Case, und Average-Case Laufzeit unter der Annahme, dass

- `a` genau die Einträge 0 bis $k-1$ sowie eine zusätzliche 0 enthält (`a` hat also die Länge $n = k+1$) und
- jede Reihenfolge gleich wahrscheinlich ist.

Zur Einfachheit nehmen wir an, dass das Prüfen der `if`-Bedingung in Zeile 3 genau c Zeiteinheiten benötigt (für eine Konstante $c > 0$) und alle weiteren Operationen keine Zeit benötigen. Begründen Sie Ihre Antworten kurz.

Hinweise:

- Das Tauschen der beiden 0en ändert die Reihenfolge der Liste nicht.
- Ihre Lösung beim Average-case darf Summenzeichen enthalten.

Lösung

Beachte: Aufgrund der gegebenen Annahmen ist es für die Einträge nur wichtig, ob der Wert gleich 0 oder ungleich 0 ist. Außerdem terminiert der Algorithmus immer sobald die zweite 0 gefunden wird. Im Best-Case ist der Wert der ersten beiden Einträge 0. Dann überprüfen wir zwei Einträge, d.h.

$$B(n) = 2 \cdot c.$$

Im Worst-Case wird die zweite 0 erst im letzten Schleifendurchlauf gefunden, d.h.

$$W(n) = n \cdot c.$$

Für den Average-Case betrachten wir für jedes $j \in \{2, \ldots, n\}$, den Fall dass die erste 0 an einer beliebigen Position $i < j$ steht und die zweite 0 an Position j. In dem Fall beträgt die Laufzeit $j \cdot c$. Wir berechnen nun die Wahrscheinlichkeit für jeden dieser Fälle in Abhängigkeit von j.

- Zunächst bestimmen wir die Anzahl der möglichen Anordnungen, die für diesen Fall auftreten können. Es gibt $j-1$ mögliche Positionen für die erste 0. Für die restlichen Einträge von 1 bis $k-1$ gibt es $(n-2)!$ mögliche Anordnungen. Insgesamt gibt es also

  $$(j-1) \cdot (n-2)!$$

  mögliche Anordnungen für den Fall, dass die zweite 0 an Position j steht (und die erste 0 irgendwo davor).

- Nun bestimmen wir die gesamte Anzahl der möglichen Anordnungen. Es gibt $n = k+1$ Einträge. Wir müssen aber beachten, dass das Vertauschen der beiden 0en die gleiche Anordnung ergibt. Daher gibt es $n!/2$ mögliche Anordnungen.

Da jede Anordnung gleich wahrscheinlich ist, ergibt sich die Wahrscheinlichkeit für den Fall "Die zweite 0 an Position j" durch Division der beiden Anzahlen:

$$(j-1) \cdot (n-2)! \cdot \frac{2}{n!} = \frac{2j-2}{n^2-n}.$$

Da die Fälle disjunkt sind (d.h. die betrachteten Eingaben überlappen sich nicht), können wir nun die Average-Case Laufzeit bestimmen, indem wir die Laufzeiten der einzelnen Fälle gewichtet mit der Wahrscheinlichkeit aufaddieren:

$$A(n) = \sum_{j=2}^{n} j \cdot c \cdot \frac{2j-2}{n^2-n}.$$

Aufgabe 5 (Rekursionsgleichung): 10 Punkte

Finden Sie für die Rekursionsgleichung F, die wir als

$$F(1) = 1 \text{ und } F(n) = 2 \cdot F\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \cdot F\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \text{ für } n > 1$$

definieren, eine Funktion $g : \mathbf{R}_+ \to \mathbf{R}_+$ als obere Abschätzung an, welche nicht rekursiv definiert ist. Beweisen Sie anschließend, dass ihr Vorschlag tatsächlich eine obere Abschätzung ist, also dass $F(n) \leq g(n)$ ist. Geben Sie schließlich auch eine Abschätzung der Funktion g in O-Notation an.

Hinweise:

- $\lfloor \ \rfloor$ ist die untere Gaußklammer, d.h. falls der Wert nicht natürlich ist, runden wir ihn nach unten ab.

Lösung

Wir vermuten, dass die Funktion exponential ist und wählen die Funktion $g(n) = a \cdot 2^n$ generisch, um anschließend a korrekt zu wählen.
Nun zeigen wir, dass es ein a gibt, sodass $F(n) \leq g(n)$ gilt per vollständiger Induktion. Wir benutzen hier im eine erweiterte Version der vollständigen Induktion, die es uns erlaubt unsere Induktionshypothese auf alle Werte kleiner als ein festes aber beliebiges n anzuwenden.
Induktionsanfang: Wir haben $F(1) = 1 \leq a \cdot 2^1 = g(n)$.

Induktionshypothese: Die Aussage gelte für alle natürliche Zahlen kleiner einem beliebigen, aber festen n.
Induktionsschritt: Für $F(n)$ haben wir:

$$\begin{aligned} F(n) &= 2 \cdot F\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \cdot F\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \\ &\leq 2 \cdot a \cdot 2^{\left\lfloor \frac{n}{2} \right\rfloor} \cdot a \cdot 2^{\left\lfloor \frac{n}{2} \right\rfloor} && \text{(Induktionshypothes)} \\ &\leq 2 \cdot a^2 \cdot 2^{\frac{n}{2}} \cdot 2^{\frac{n}{2}} && \text{(Obere Abschätzung der Gaußklammern)} \\ &= 2 \cdot a^2 \cdot 2^n && \text{(Vereinfachen)} \end{aligned}$$

Damit nun $F(n) \leq g(n)$ ist, muss wegen dem Induktionsanfang $1 \leq a \cdot 2$ und $2 \cdot a^2 \leq a$ gelten. Vereinfacht ergibt das die zwei Bedingungen $0.5 \leq a$ und $a \leq 0.5$ - somit muss $a = 0.5$ sein. Und tatsächlich erhalten wir (die sogar stärkere Aussage):
Induktionsanfang: Wir haben $F(1) = 1 = 0.5 \cdot 2^1 = g(n)$.
Induktionshypothese: Die Aussage gelte für alle natürliche Zahlen kleiner einem beliebigen, aber festen n.
Induktionsschritt: Für $F(n)$ haben wir:

$$\begin{aligned} F(n) &= 2 \cdot F\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \cdot F\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \\ &\leq 2 \cdot 0.5 \cdot 2^{\left\lfloor \frac{n}{2} \right\rfloor} \cdot 0.5 \cdot 2^{\left\lfloor \frac{n}{2} \right\rfloor} && \text{(Induktionshypothes)} \\ &\leq 0.5 \cdot 2^{\frac{n}{2}} \cdot 2^{\frac{n}{2}} && \text{(Obere Abschätzung der Gaußklammern)} \\ &= 0.5 \cdot 2^n && \text{(Vereinfachen)} \\ &= g(n) \end{aligned}$$

Schließlich haben wir $g(n) \in O(2^n)$ und somit auch $T(n) \in O(2^n)$.

Aufgabe 6 (Master Theorem): 2 + 2 + 2 + 2 + 2 = 10 Punkte

Benutzen Sie das Master Theorem um für folgende Rekursionsgleichungen $T(n)$ die beste O-Klasse anzugeben, in der $T(n)$ liegt. Anbei haben wir Ihnen erneut das Master Theorem angegeben:
Für $T(1) = c$ und $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ falls $n > 1$ gilt

Wenn	Dann
$f \in O(n^p)$ und $a < b^p$	$T(n) \in O(n^p)$
$f \in O(n^p)$ und $a = b^p$	$T(n) \in O(n^p \cdot \log(n))$
$f \in O(n^p)$ und $a > b^p$	$T(n) \in O(n^{\log_b(a)})$

a)

$$T(1) = 2$$
$$T(n) = 2 \cdot T(\frac{n}{4}) + \log_2(n) \qquad \text{for } n > 1$$

b)

$$T(1) = 5$$
$$T(n) = 8 \cdot T(\frac{n}{2}) + 3 \cdot n^2 + 5 \cdot n \qquad \text{for } n > 1$$

c)

$$T(1) = 10$$
$$T(n) = 9 \cdot T(\frac{n}{3}) + 2 \cdot n^5 + \log_3(n) \qquad \text{for } n > 1$$

d)

$T(1) = 7$

$T(n) = 9 \cdot T(\frac{n}{3}) + n \qquad \text{for } n > 1$

e)

$T(1) = 1$

$T(n) = 16 \cdot T(\frac{n}{4}) + 5 \cdot n^2 + \log_3(n) \qquad \text{for } n > 1$

Lösung

a) Fall 3: Wir wählen $p = 0.25$. Dann haben wir $\log_2(n) \in O(n^{0.25})$ und $a = 2 > \sqrt{2} = b^p$ und $\log_4(2) = 0.5$, somit gilt nach dem Master Theorem $T(n) \in O(n^{0.5})$

b) Fall 3: Wir wählen $p = 2$. Dann haben wir $3 \cdot n^2 + 5 \cdot n \in O(n^2)$ und $a = 8 > 4 = 2^2 = b^p$ und $\log_2(8) = 3$, somit gilt nach dem Master Theorem $T(n) \in O(n^3)$.

c) Fall 1: Wir wählen $p = 5$. Dann haben wir $2 \cdot n^5 + \log_3(n) \in O(n^5)$ und $a = 9 < 243 = 3^5 = b^p$, somit gilt nach dem Master Theorem $T(n) \in O(n^5)$.

d) Fall 3: Wir wählen $p = 1$. Dann haben wir $n \in O(n)$ und $a = 9 > 3 = b^p$ und $\log_3(9) = 2$, somit gilt nach dem Master Theorem $T(n) \in O(n^2)$.

e) Fall 2: Wir wählen $p = 2$. Dann haben wir $5 \cdot n^2 + \log_3(n) \in O(n^2)$ und $a = 16 = 4^2 = b^p$, somit gilt nach dem Master Theorem $T(n) \in O(n^2 \cdot \log(n))$.

Aufgabe 7 (Greedy): 10 Punkte

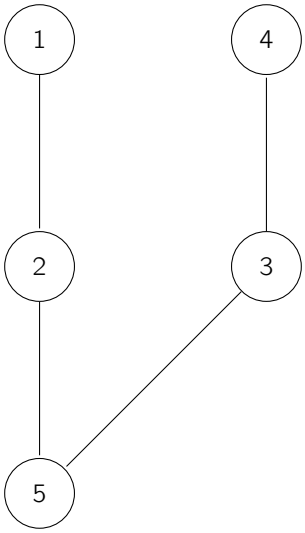
Während einer Klausur sollen die Prüflinge verschiedene Aufgaben bekommen, um das Risiko von Täuschungen zu vermindern. Dabei dürfen jedoch nur benachbarte Prüflinge nicht die gleiche Klausur bekommen. Ein ungerichteter Graph $G = (V, E)$ gibt an, welche Prüflinge benachbarte Plätze haben. Dabei sind V die Prüflinge und die Kanten $(i, j), (j, i) \in E$ geben an, dass die Prüflinge i und j benachbart sind.
Geben Sie einen Greedy Algorithmus an, der eine Abbildung von Prüflingen auf Klausuren berechnet, sodass zwei benachbarte Prüflinge nicht die gleiche Klausur erhalten. Sie müssen keinen Code angeben. Es reicht ihren Algorithmus ausreichend detailliert zu beschreiben.
Minimiert Ihr Algorithmus die Anzahl an nötigen unterschiedlichen Klausuren?

Lösung

Wir nummerieren die verschiedenen Klausuren durch und bilden stattdessen die Knoten V auf solche Klausurnummern ab. Weiterhin nehmen wir an, dass der Graph als Adjazenzliste gegeben ist, das heißt wir haben eine Abbildung von Knoten zu einer Liste an Kanten. Weiterhin nehmen wir zur Vereinfachung an, dass die Studenten ebenfalls durch natürliche Zahlen gegeben sind. Nun können wir die Klausuren wie folgt berechnen:

1. Falls allen Prüflingen eine Klausur zugewiesen wurde, gebe *exams* aus.
2. Wähle einen Prüfling v aus, der noch keine Klausur zugewiesen hat.
3. Lese aus, welche Klausuren die Nachbarn von v haben und speichere sie in $neighborExams$.
4. Wähle die kleinste Zahl aus, die nicht in $neighborExams$ gespeichert ist und speichere sie in $exams[v]$.
5. Wiederhole von 1.

Unser Algorithmus berechnet nicht notwendigerweise eine Zuteilung mit minimaler Anzahl an verschiedenen Klausuren. Es gibt jedoch stets eine Reihenfolge an Knoten, die uns eine optimale Lösung berechnet. Ein Beispiel, in dem eine Reihenfolge uns ein nicht optimales Ergebnis liefert wäre der Graph:



für den der Algorithmus die Zuteilung $1 \mapsto 0, 2 \mapsto 1, 3 \mapsto 0, 4 \mapsto 1, 5 \mapsto 2$ berechnet. Diese ist jedoch nicht optimal, da die Zuteilung $1 \mapsto 0, 2 \mapsto 1, 3 \mapsto 1, 4 \mapsto 0, 5 \mapsto 0$ weniger Klausuren benötigt.