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

Lösung - Übung 3

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 $j \in \{2, \dots, n\}$, 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:

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

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):

$2a^2 <= a$ folgt aus der Induktionshypothese

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:

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

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 $T(n) \in O(n^{0.5})$ durch Regel von de L'Hospital zeigbar

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

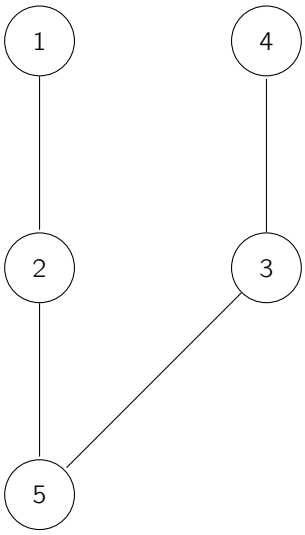
Während einer Klausur sollen die Prüflinge verschiedene Aufgaben bekommen, um das Risiko von Täuschungen zu vermindern. Dabei dürfen jedoch nur benachbarte Prüflinge nicht die gleiche Klausur bekommen. Ein ungerichteter Graph $G = (V, E)$ gibt an, welche Prüflinge benachbarte Plätze haben. Dabei sind 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.