Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

Übungsblatt 02

Algorithmen I - Sommersemester 2022

Abgabe im ILIAS bis 04.05.2022, 14:00 Uhr

Bitte beschrifte Deine Abgabe gut sichtbar mit Deinem Namen und Deiner Matrikelnummer. Achte insbesondere bei handschriftlichen Abgaben auf Lesbarkeit und genügend Platz für Korrektur-Anmerkungen. Die Abgabe erfolgt über das Übungsmodul in der Gruppe Deines Tutoriums im ILIAS. Gib Deine Ausarbeitungen im PDF-Format ab.

Aufgabe 1 - Rekurrenzen aufstellen (7 Punkte)

Gegeben seien die beiden Algorithmen algOne und algTwo.
Achtung: Wir fordern für diese Aufgabe, dass für den initialen Aufruf von algOne der zweite Eingabeparameter stets größer ist als der erste.

```
1: algOne(x: ℕ, y: ℕ): ℕ
2:     if x = 0 ∨ x = y then
3:         return 1
4:     end
5:     return algOne(x - 1, y - 1) + algOne(x, y - 1)
```

```
1: algTwo(x: ℕ, y: ℕ): ℕ
2:     if x ≥ y then
3:         return x
4:     end
5:     total : ℕ = algTwo(x, ⌊(y+3x)/4⌋)
6:     total += algTwo(⌈(y+3x)/4⌉, ⌊(y+x)/2⌋)
7:     total += algTwo(⌈(y+x)/2⌉, ⌊(3y+x)/4⌋)
8:     return total
```

1. Wir haben Laufzeiten und Rekurrenzen in Abhängigkeit von Problemgrößen kennengelernt. Beschreibe für beide Algorithmen, was ihre Problemgröße ist. (2 Punkte)

2. Stelle nun für beide Algorithmen die Rekurrenzgleichung auf. (2 Punkte)

3. Ist das Mastertheorem für die Rekurrenzgleichung von algOne anwendbar? Begründe deine Antwort kurz. (1 Punkt)

4. Bestimme eine eine möglichst enge asymptotische obere Schranke für die Laufzeit von algOne. (1 Punkt)

5. Bestimme eine eine möglichst enge asymptotische obere Schranke für die Laufzeit von algTwo. (1 Punkt)

Lösung 1

Diese Aufgabe wurde aus der Wertung gestrichen. Die Lösung wird in Übung zwei erläutert.

Aufgabe 2 - Rekurrenzen Lösen (6 Punkte)

In Vorlesung 2 haben wir das Mastertheorem zum Lösen von Rekurrenzen kennen gelernt. Besonders interessant ist hier die Herleitung, bei der die Kosten des Rekursionsbaums in jeder Ebene betrachtet wurden.

Verwende daher zum Lösen der folgenden Rekurrenzen die gleiche Technik und überlege analog zu Folie 13

- Wie viele Knoten sind auf Lage i?
- Wie groß ist das n auf Lage i?
- Wie viel Zeit kostet ein Knoten in Lage i?

um somit die Gesamtkosten für Lage i und hieraus die Gesamtkosten abzuleiten.

1. $$T_1(n) = \begin{cases} 1 & \mid n = 1 \\ 4 \cdot T_1(\frac{n}{2}) + \sqrt{3n} + 1 & \mid \text{sonst} \end{cases}$$

2. $$T_2(n) = \begin{cases} 1 & \mid n = 1 \\ 14 \cdot T_2(\frac{n}{4}) + T_1(n) & \mid \text{sonst} \end{cases}$$

3. $$T_3(n) = \begin{cases} 1 & \mid n = 1 \\ 4 \cdot T_3(\frac{n}{2}) + n^2 & \mid \text{sonst} \end{cases}$$

4. $$T_4(n) = \begin{cases} 1 & \mid n = 1 \\ 4 \cdot T_4(\frac{n}{3}) + n^2 \log n & \mid \text{sonst} \end{cases}$$

Hinweis: Für $0 \leq c < 1$ gilt sowohl $0 < \sum_{i=0}^{\infty} c^i < \infty$ als auch $0 < \sum_{i=0}^{\infty} i \cdot c^i < \infty$.

Lösung 2

1. Jeder Knoten auf Lage $i-1$ wird in 4 Knoten aufgeteilt, welche je die Hälfte der Problemgröße erhalten. Also:
   - Anzahl Knoten auf Lage i: 4^i
   - Problemgröße auf Lage i: $\frac{n}{2^i}$
   - Kosten pro Knoten mit Problemgröße k auf Lage i: $\sqrt{3k}+1$

   Damit ergibt sich ein Gesamtaufwand von

$$\begin{aligned}\sum_{i=0}^{\log_2 n} 4^i \cdot (\sqrt{\frac{3n}{2^i}}+1) &= \sum_{i=0}^{\log_2 n} 4^i \cdot \sqrt{\frac{3n}{2^i}} + \sum_{i=0}^{\log_2 n} 4^i \\ &= \sum_{i=0}^{\log_2 n} \sqrt{\frac{16^i \cdot 3n}{2^i}} + \sum_{i=0}^{\log_2 n} 4^i \\ &= \sqrt{3n} \cdot \sum_{i=0}^{\log_2 n} \sqrt{8^i} + \sum_{i=0}^{\log_2 n} 4^i\end{aligned}$$

   Es gilt

$$\sum_{i=0}^{\log_2 n} \sqrt{8^i} \in \Theta(\sqrt{8^{\log_2 n}}) = \Theta(n^{1/2 \cdot \log_2 8}) = \Theta(n^{3/2})$$

   und damit

$$\sqrt{3n} \cdot \sum_{i=0}^{\log_2(n)} \sqrt{8^i} \in \Theta(n^{1/2} \cdot n^{3/2}) = \Theta(n^2)$$

   sowie

$$\sum_{i=0}^{\log_2 n} 4^i \in \Theta(4^{\log_2 n}) = \Theta(2^{\log_2(n^2)}) = \Theta(n^2)$$

   Also ist $T_1(n) \in \Theta(n^2)$.

2. Jeder Knoten auf Lage $i-1$ wird in 14 Knoten aufgeteilt, welche je ein Viertel der Problemgröße erhalten. Also:
   - Anzahl Knoten auf Lage i: 14^i
   - Problemgröße auf Lage i: $\frac{n}{4^i}$
   - Kosten pro Knoten mit Problemgröße k auf Lage i $\in \Theta(k^2)$

   Damit wächst der Gesamtaufwand asymptotisch genauso schnell wie

$$\sum_{i=0}^{\log_4 n} 14^i \cdot \left(\frac{n}{4^i}\right)^2 = n^2 \cdot \sum_{i=0}^{\log_4 n} \left(\frac{14}{16}\right)^i$$

Da $\frac{14}{16} < 1$ wird diese exponentielle Summe durch den ersten Summanden dominiert und liegt damit in $\Theta(1)$.
Also ist $T_2(n) \in \Theta(n^2)$.

3. Jeder Knoten auf Lage $i-1$ wird in 4 Knoten aufgeteilt, welche je die Hälfte der Problemgröße erhalten. Also:
   - Anzahl Knoten auf Lage i: 4^i
   - Problemgröße auf Lage i: $\frac{n}{2^i}$
   - Kosten pro Knoten mit Problemgröße k auf Lage i: k^2

   Damit ergibt sich ein Gesamtaufwand von

$$\begin{aligned}\sum_{i=0}^{\log_2 n} 4^i \cdot \left(\frac{n}{2^i}\right)^2 &= n^2 \cdot \sum_{i=0}^{\log_2 n} \left(\frac{4}{4}\right)^i \\ &= n^2 \cdot \sum_{i=0}^{\log_2 n} 1 \\ &= n^2 \cdot (\log_2 n + 1) \\ &\in \Theta(n^2 \cdot \log(n))\end{aligned}$$

4. Jeder Knoten auf Lage $i-1$ wird in 4 Knoten aufgeteilt, welche je ein Drittel der Problemgröße erhalten. Also:
   - Anzahl Knoten auf Lage i: 4^i
   - Problemgröße auf Lage i: $\frac{n}{3^i}$
   - Kosten pro Knoten mit Problemgröße k auf Lage i: $k^2 \cdot \log k$

   Damit ergibt sich ein Gesamtaufwand von

$$\begin{aligned}\sum_{i=0}^{\log_3 n} 4^i \cdot \left(\frac{n}{3^i}\right)^2 \cdot \log\left(\frac{n}{3^i}\right) &= \sum_{i=0}^{\log_3 n} \left(\frac{4}{9}\right)^i \cdot n^2 \cdot \log\left(\frac{n}{3^i}\right) \\ &= n^2 \cdot \sum_{i=0}^{\log_3 n} \left(\frac{4}{9}\right)^i \cdot (\log(n) - \log(3^i)) \\ &= n^2 \cdot \sum_{i=0}^{\log_3 n} \left(\frac{4}{9}\right)^i \cdot \log(n) - n^2 \cdot \sum_{i=0}^{\log_3 n} \left(\frac{4}{9}\right)^i \cdot i \cdot \log(3) \\ &= n^2 \log(n) \cdot \sum_{i=0}^{\log_3 n} \left(\frac{4}{9}\right)^i - \log(3) \cdot n^2 \cdot \sum_{i=0}^{\log_3 n} i \cdot \left(\frac{4}{9}\right)^i\end{aligned}$$

   Da $\sum_{i=0}^{\log_3 n} \left(\frac{4}{9}\right)^i \in \Theta(1)$ und $\sum_{i=0}^{\log_3 n} i \cdot \left(\frac{4}{9}\right)^i \in \Theta(1)$ (siehe obigen Hinweis), ist $T_4(n) \in \Theta(n^2 \cdot \log(n))$.

Aufgabe 3 - Dr. Meta sieht alles (6 Punkte)

Das Böse schläft nie. Auch der gewiefte und wissbegierige Superbösewicht Dr. Meta hat seine Augen überall. Das gilt insbesondere für sein eigenes geheimes Labor, welches er streng überwacht. Dazu hat er Sicherheitskameras über das gesamte Labor verteilt. Jede Kamera hat eine eindeutige Koordinate $(x, y) \in \mathbb{N}^2$. Jede Koordinate hat dabei genau vier Nachbarn. Zwei Koordinaten (x, y) und (x', y') sind benachbart, wenn $|x - x'|+|y - y'| = 1$ gilt.
Dr. Metas Vorsicht ist begründet: In den letzten Wochen wurden immer wieder verdächtige Spuren gefunden, die nur von einem Eindringling stammen können. Um zu verstehen, wie dieser entkommen konnte, will Dr. Meta herausfinden, wo sein Weg endete. Dazu will er die Spuren entlang benachbarter Koordinaten verfolgen.
Natürlich kann Dr. Meta als vielbeschäftigter Schurke diese Aufgabe nicht übernehmen und gibt sie an dich weiter. Es liegt also an dir, den Eindringling dingfest zu machen!
Dazu steht die Hilfsmethode

traceObserved(x: $\mathbb{N}$, y : $\mathbb{N}$): Bool

bereit. Diese gibt zurück, ob von der Kamera an der übergebenen Koordinate Spuren aufgenommen wurden. Falls an der übergebenen Koordinate keine Sicherheitskamera angebracht wurde, gibt sie den Wert `false` zurück.
Der Eindringling war sehr geschickt und scheint darauf geachtet zu haben, das Labor so schnell wie möglich zu durchqueren. Du darfst deswegen davon ausgehen, dass der Eindringling auf seinem Weg keine Koordinate mehr als ein Mal besucht hat und sich in der Nachbarschaft jeder Koordinate höchstens zwei Spuren befinden.
Hinweis: In der Nachbarschaft der Koordinaten, die den Beginn und das Ende eines Weges bezeichnen, befindet sich jeweils nur eine Koordinate mit Spuren.

1. Angenommen, der Weg führt von Koordinate (x, y) zur benachbarten Koordinate (x', y'). Wie findet man von (x', y') aus die korrekte nächste Koordinate? (2 Punkte)

2. Beschreibe, wie die Koordinate bestimmt werden kann, an der der Weg endet, der an einer gegebenen Koordinate (x, y) beginnt. (1 Punkt)

3. Formuliere nun einen rekursiven Algorithmus in Pseudocode, der das gleiche Problem löst. Beschreibe kurz, welchen Nutzen die Eingabeparameter deines Algorithmus haben. (1 Punkt)

4. Sei wieder eine Koordinate (x, y) gegeben, an der ein Weg beginnt. Wie muss dein Algorithmus aus 3. initial aufgerufen werden, um diesen Weg zu verfolgen?

5. Wie muss dein Algorithmus aus 3. abgeändert werden, um ihn zu einem iterativen Algorithmus umzuwandeln? (1 Punkt)

6. Gib eine Abschätzung für die Laufzeit deines Algorithmus im O-Kalkül an, mit der Dr. Meta im schlimmsten Fall rechnen muss. Die Laufzeit soll dabei in Abhängigkeit von der Anzahl an Koordinaten, an denen sich Spuren befinden, betrachtet werden. Unterscheiden sich die iterative und die rekursive Umsetzung bezüglich ihrer asymptotischen Laufzeit? (1 Punkt)

Lösung 3

1. Da wir annehmen, dass der Eindringling keine Koordinate mehr als ein Mal besucht, kann sich sein Weg nie gabeln. Somit müssen wir nur dafür sorgen, dass wir, wenn wir von (x, y) zu (x', y') kommen, nicht wieder zurück zu (x, y) gehen. Damit genügt es, sich im vorherigen Schritt zu merken, aus welcher Richtung man auf die aktuelle Koordinate gestoßen ist. Wechseln wir von (x, y) zu (x', y'), so speichern wir also $(x' - x, y' - y)$ als die Richtung ab, in die wir im nächsten Schritt nicht bewegen dürfen. Wir überprüfen dann mit traceObserved, ob eine der übrigen benachbarten Koordinaten eine Spur enthält.

2. Wir starten auf der eingegebenen Koordinate. Nun bestimmen wir die benachbarte Koordinate (x', y'), an der Spuren gesichtet wurden, d.h. für welche traceObserved den Wert `true` zurückliefert. Dann bestimmen wir, wie in 1. beschrieben, die Richtung, in die wir uns im nächsten Schritt nicht bewegen dürfen und setzen die aktuelle Koordinate auf (x', y'). Dieses Vorgehen wiederholen wir solange, bis wir die Koordinate finden, an der die Spur endet, d.h. die keine Nachbarkoordinate außer der in der gerade ausgeschlossenen Richtung hat, für die traceObserved `true` zurückliefert. Diese Koordinate ist die von uns gesuchte.

3. Eine mögliche Variante ist die folgende. Der Eingabeparameter cur_pos ist hier die aktuelle Koordinate, last_direction ist die Richtung, in der die zuvor besucht Koordinate liegt.

```
1: traceEnd(cur_pos: ℕ², last_direction: ℕ²): ℕ²
2:     for direction ∈ {(1,0), (−1,0), (0,1), (0,−1)} do
3:         next_pos : ℕ² = cur_pos + direction
4:         is_valid : Bool = direction ≠ last_direction ∧ next_pos ≥ (0,0)
5:         if is_valid ∧ traceObserved(next_pos) then
6:             return traceEnd(next_pos, (−1)· direction)
7:         end
8:     end
9:     return cur_pos
```

4. traceEnd$((x, y), (0, 0))$
für Startkoordinate (x, y). Da im ersten Schritt keine Richtung ausgeschlossen wird, übergeben wir $(0, 0)$ als "nicht erlaubte" Richtung.

5. Anstatt für jede Koordinate einen neuen Methodenaufruf zu tätigen, verwenden wir eine Schleife. Wie im rekursiven Algorithmus merken wir uns die aktuelle Koordinate und die aktuell unerlaubte Richtung. Pro Schleifendurchlauf wird dann die Koordinate auf einen ihrer Nachbarn gesetzt und die Richtung neu berechnet. Die Durchführung bricht ab, sobald in einem Schleifendurchlauf keine Nachbarkoordinate mit Spuren gefunden wurde.

6. Die asymptotische Laufzeit für beide Varianten liegt in $O(n)$, wobei n hier die Anzahl an Koordinaten ist, an denen Spuren gesichtet werden. Sowohl bei der iterativen als auch bei der rekursiven Variante wird pro Koordinate nur konstanter Zeitaufwand benötigt: traceObserved wird maximal vier Mal bei der Bearbeitung einer Koordinate aufgerufen. Das gleiche gilt für den Vergleich zweier Koordinaten. Pro Koordinate verringert sich in beiden Varianten die Länge der Spur, die noch verfolgt werden muss. Vor dem Eintritt in die Schleife in der iterativen Variante müssen nur Variablen initialisiert werden, was ebenfalls konstanten Zeitbedarf benötigt.

Sommereulenfest

Es ist soweit, am 14. Juli findet endlich wieder das Eulenfest statt. Dafür braucht die Fachschaft eure Hilfe. Das Eulenfest ist eine große Party auf dem Campus, die traditionell von Ersties für Ersties organisiert wird. Wenn du also Lust hast selbst mal ein Fest zu organisieren, schau einfach vorbei am Dienstag den **03. Mai um 19:30 Uhr** beim ersten **Eulenfestorgatreffen**.