Musterlösungen Übungsblatt 5

Thomas Graf
EF / WF Informatik 2018-2019
Algorithmen

21. Oktober 2018

1 lineare Suche($\star$)

1.1 Beweis der Korrektheit

Wir zeigen die Korrektheit des Algorithmus in der Aufgabenstellung unter Verwendung einer loop-Invariante:

- **loop-Invariante**:
  Zu Beginn jeder Iteration des for loops (Zeilen 2-5) ist die loop-Invariante, dass es keinen Index $k < j$ gibt, so dass $A[k] = v$.

- **Initialisierung**:
  Vor Beginn der ersten loop-Iteration (Zeile 1) ist v noch nicht gefunden worden und der Index i korrekterweise auf NIL gesetzt.

- **Maintenance**:
  Falls der loop in Zeile 5 verlassen wird, dann haben wir gerade einen korrekten Index i (für den gilt: $A[i] = v$) gefunden.

- **Termination**:
  Falls der loop komplett durchlaufen wird, ohne, dass Zeile 5 jemals ausgeführt wurde, dann haben wir alle Einträge von A überprüft und können sicher sein, dass v nicht in A enthalten ist. Richtigerweise bleibt dadurch i auf den Wert NIL gesetzt.

1.2 Laufzeitanalyse

Seien c_i die Kosten der i-ten Zeile, $i = 1, ..., 6$ (also c_1 die Kosten der Zeile 1, c_2 die Kosten der Zeile 2 usw.).

1.2.1 best-case

Im besten Fall (best-case) hat das erste Element von A gerade den gesuchten Wert v. In diesem Fall führen wir die Zeilen 1, 2, 3, 4 und 5 je genau einmal aus. Die Gesamtkosten betragen also $c_1 + c_2 + c_3 + c_4 + c_5$ für das best-case-Szenario.

1.2.2 worst-case

Es gibt zwei verschiedene Fälle, welche zu Laufzeiten führen, die sich nur durch eine Konstante unterscheiden. (a):
Der Wert v ist nicht in A enthalten. Dann müssen wir Zeile 1 einmal, Zeile 2 $(n+1)$ mal, Zeile 3 (n) mal, Zeilen 4 und 5 gar nicht und Zeile 6 einmal ausführen. Die Kosten sind damit $c_1 + (n+1)c_2 + nc_3 + c_6 = (c_2 + c_3)n + c_1 + c_2 + c_6$.
(b):
Der Wert v wird zum ersten Mal an der letzten Stelle (Position $A[n]$) angenommen. Dann müssen wir Zeile 1 einmal, Zeile 2 und 3 (n) mal, Zeilen 4 und 5 einmal und Zeile 6 nie ausführen. Die Kosten sind damit $c_1 + nc_2 + nc_3 + c_4 + c_5 = (c_2 + c_3)n + c_1 + c_4 + c_5$.

Offensichtlich unterscheiden sich die beiden Laufzeiten nur um Konstanten und beide sind lineare Funktionen in n.

2 best-case running time ($\star\star$)

Wir haben ein Problem und einen Algorithmus A, welcher das Problem löst. Wie können wir den Algorithmus A nun modifizieren, damit er eine "schnell" best-case-Laufzeit hat?
Wir konstruieren einen Algorithmus, welcher zuerst zufällig einen Output generiert und dann überprüft, ob dieser Output gerade eine Lösung des Problems ist. Falls dies zutreffen sollte (was natürlich im Allgemeinen höchst unwahrscheinlich ist), dann haben wir bereits eine Lösung des Problems gefunden und sind fertig. Falls der zufällig generierte Output keine Lösung des Problems ist, dann lösen wir das Problem ganz normal mit Hilfe von A. Der so konstruierte Algorithmus hält immer und produziert die korrekte Lösung: falls die

zufällig generierte Ausgabe gerade eine Lösung ist, sind wir fertig und haben eine korrekte Lösung gefunden. Ansonsten übernimmt Algorithmus A die Arbeit. Dieser hält in endlicher Zeit und produziert eine korrekte Lösung (dies folgt aus der Definition von A als Algorithmus).
Im best-case-Szenario ist aber der zufällig generierte Output bereits die Lösung und der Algorithmus. Die best-case-Laufzeit ist also nur die Zeit die benötigt wird, um einen zufälligen Output zu generieren und diesen auf Korrektheit zu überprüfen.

Wir möchten die Idee am Beispiel des Sortierens einer Liste illustrieren.
Sei A ein Algorithmus, der eine Liste sortiert. Wir modifizieren A so, dass wir zuerst die Elemente von L zufällig permutieren und anschliessend überprüfen, ob die dadurch entstandene Liste bereits sortiert ist. Im best-case wäre dies der Fall und wir wären schon fertig.

3 Vergleich der Laufzeiten zweier Algorithmen ($\star\star$)

Die Gleichung

$$150n^2 = 2^n$$

kann nicht algebraisch gelöst werden. Durch *Pröbeln* finden wir, dass der kleinste Wert für n, welcher die geforderte Bedinung erfüllt $n = 16$ ist.