Einführung in die Informatik: Übung 1

Alexander Waldenmaier

11. November 2020

Präsenzaufgabe

Als Input für den Algorithmus könnte eine Liste mit n Zahlen dienen, wobei das i-te Listenelement als a_i bezeichnet wird (mit $0 \leq i < n$). Der Algorithmus könnte die folgenden Schritte beinhalten:

(a) Setze $i = 0$ (1E)

(b) Teste $i < (n - 1)$ (1E)

(c) Gebe aus "nein, a_i ist die letzte Zahl!"(1E)

(d) Setze $j = i + 1$, Teste $a_i < a_j$ (2E)

(e) Gebe aus "ja, a_j ist größer!", Setze $i = i + 1$ (2E)

(f) Setze $j = j + 1$, Teste $j < n$ (2E)

(g) Gebe aus "nein, es gibt keine größere Zahl!", Setze $i = i + 1$ (2E)

Zu Beginn des Programmdurchlaufs wird in Schritt a die Laufvariable $i = 0$ gesetzt. In Schritt b wird geprüft, ob bereits das letzte Element der Liste erreicht ist. Wenn ja, dann wird in Schritt c eine entsprechende Meldung herausgegeben und das Programm terminiert. Wenn nicht, wird in Schritt d die Laufvariable $j = i + 1$ gesetzt und geprüft, ob das j-te Element größer als da i-te Element ist. Wenn ja, wird in Schritt e eine entsprechende Meldung herausgegeben, i inkrementiert und wieder zu Schritt b zurückgekehrt. Wenn nicht, wird in Schritt f j inkrementiert und geprüft, ob damit das Ende der Liste erreicht ist. Wenn ja, wird in Schritt g eine entsprechende Meldung herausgegeben, i inkrementiert und wieder zu Schritt b zurückgekehrt. Wenn nicht, wird direkt zu Schritt d zurückgekehrt.

Im besten Fall handelt es sich um eine Liste mit n Zahlen, bei denen jede Zahl größer als ihr Vorgänger ist (oder die erste ist). Dann liegt die minmale Ausführungszeit vor, da für jedes Listenelement X nur das nächste Element Y überprüft werden muss und dabei stets sofort $Y > X$ gilt.

$$\begin{aligned} T_{min} &= 1 * a + n * b + 1 * c + (n-1) * d + (n-1) * e + (n-1) * f \\ &= (1 + n + 1 + 2(n-1) + 2(n-1) + 2(n-1))E \\ &= (7n - 4)E \end{aligned}$$

Im schlechtesten Fall ist die Liste aber in fallender Reihenfolge sortiert. Dann muss für jedes Listenelement X jedes darauffolgende Element bis an das Ende der Liste geprüft werden. Dann gilt:

$$\begin{aligned} T_{max} &= 1 * a + n * b + 1 * c + n\frac{n-1}{2} * d + (n-1)\frac{n-2}{2} * f + (n-1) * g \\ &= (1 + n + 1 + 2\frac{n^2 - n}{2} + 2\frac{n^2 - 3n + 2}{2} + 2(n-1))E \\ &= (2n^2 - n + 2)E \end{aligned}$$

Im Fall von $n = 5$ gilt also $T_{min} = 31E$ und $T_{max} = 47E$. Aufgrund des quadratischen Wachstums von T_{max} ist dieser Algorithmus bei hohen n nicht praktikabel.

Aufgabe 1: Horner-Schema

Umrechnungen:

- $1000010_{(2)} = (((((1 * 2 + 0) * 2 + 0) * 2 + 0) * 2 + 0) * 2 + 1) * 2 + 0 = 66_{(10)}$
- $10222_{(8)} = (((1 * 8 + 0) * 8 + 2) * 8 + 2) * 8 + 2 = 4.242_{(10)}$
- $ABBA_{(16)} = ((10 * 16 + 11) * 16 + 11) * 16 + 10 = 43.962_{(10)}$

Schema rückwärts:

$$11_{(10)} = 1 * 2^3 + 0 * 2^2 + 1^* 2^1 + 1 * 2^0 = 1011_{(2)}$$

Aufgabe 2: Algorithmenkonstruktion

> Finde die minimale, maximale und durchschnittliche Abarbeitungszeit!

a) Problemspezifikation

Was ist die Abarbeitungszeit? Die Abarbeitungszeit ist auf Programmniveau ein Zahlenwert aus der gegebenen Menge. Folglich sind „minimal“, „maximal“ und „durschnittlich“ nun als mathematische Begriffe zu verstehen, die auf die in der Menge befindlichen Zahlenwerte anzuwenden sind.

b) Problemabstraktion

- Gegeben: Eine Menge X von positiven Float-Zahlen. Die Menge enthält n Elemente, jedes Element sei x_i genannt (wobei $0 \leq i < n$)
- Gesucht: Das Minimum $min(X)$, das Maximum $max(X)$ sowie der Durschnitt $sum(X)/n$ der Arbarbeitungszeiten.

c) Algorithmenentwurf

Algorithmus „Abarbeitungszeiten“:

(a) Lese die Menge X als Liste von Float-Zahlen x_i mit Länge n ein. (2E)

(b) Setze die Variablen $x_{min} = x_0$, $x_{max} = x_0$, $x_{sum} = 0$ und $i = 0$ (4E)

(c) Solange $i < n$, wiederhole: (1E)

  (a) Wenn $x_i < x_{min}$ (1E)
    i. Setze $x_{min} = x_i$ (1E)
  (b) Wenn $x_i > x_{max}$ (1E)
    i. Setze $x_{max} = x_i$ (1E)
  (c) Setze $x_{sum} = x_{sum} + x_i$ (1E)
  (d) Setze $x_i = x_i + 1$ (1E)

(d) Gebe aus:

  (a) "Minimale Ausführungsdauer = x_{min}"(1E)
  (b) "Maximale Ausführungsdauer = x_{max}"(1E)
  (c) "Durschnittliche Ausführungsdauer = x_{sum}/n"(1E)

d) Korrektheitsnachweis, Verifikation

Behauptung: Der Algorithmus „Abarbeitungszeiten“ terminiert und liefert die geforderten Ergebnisse.

Beweis: Vollständige Induktion über n:

- Induktionsbehauptung:
  x_{min} ist das Minimum von X, x_{max} ist das Maximum von X und x_{sum}/n ist der Durchschnitt von X. Das Programm termininiert nach $n-1$ Durchläufen.
- Induktionsanfang ($n = 1$):
  Beweis durch Simulation (Schritte wie in Teil c)):
  (a) Es wird gesetzt: $X = x_0$, $n = 1$
  (b) Es wird gesetzt: $x_{min} = x_0$, $x_{max} = x_0$, $x_{sum} = 0$ und $i = 0$

(c) Schleife wird einmalig ausgeführt.

(d) Es wird ausgegeben:

(a) "Minimale Ausführungsdauer = $x_{min} = x_0$"

(b) "Maximale Ausführungsdauer = $x_{max} = x_0$"

(c) "Durschnittliche Ausführungsdauer = $x_{sum}/n = x_0$"

$\Rightarrow$ Die Behauptung gilt für $n = 1$.

- Induktionsschritt ($n \to n + 1$):
  Induktionssannahme: Induktionsbehauptung gelte für n, betrachte nun $n' = n + 1$ (Schritte wie in Teil c)):

  (a) Es wird gesetzt: X, $n = n'$

  (b) Es wird gesetzt: $x_{min} = x_0$, $x_{max} = x_0$, $x_{sum} = 0$ und $i = 0$

  (c) Schleife wird $n - 1$ mal ausgeführt. Dann gilt nach Induktionsannahme die Induktionsbehauptung.
  Bei der letzten Ausführung der Schleife gibt es folgende Abwägungen:

  (a) Wenn x_{n+1} kleiner als x_{min} ist, wird $x_{min} = x_{n+1}$ gesetzt.

  (b) Wenn x_{n+1} größer als x_{max}, wird $x_{max} = x_{n+1}$ gesetzt.

  (c) Es wird $x_{sum} = x_{sum} + x_{n+1}$ gesetzt.

  In jedem Fall gilt die Induktionsbehauptung für n'.

  (d) Es wird ausgegeben:

  (a) "Minimale Ausführungsdauer = x_{min}"

  (b) "Maximale Ausführungsdauer = x_{max}"

  (c) "Durschnittliche Ausführungsdauer = x_{sum}/n"

q.e.d.

e) Aufwandsanalyse

Die Schritte a und b werden immer exakt einmal ausgeführt. Anschließend wird Schritt c $n - 1$ mal ausgeführt. Schritt c-a wird ebenfalls $n - 1$ mal ausgeführt, c-a-i hingegen nur so häufig, wie x_i kleiner als x_{min} ist. Schritt c-b wird analog $n - 1$ mal ausgeführt, c-b-i hingegen nur so häufig, wie x_i größer als x_{max} ist. Anschließend werden immer, insgesamt $n - 1$ mal, die Schritte c-c und c-d ausgeführt. Schritte d-a, d-b und d-c werden jeweils exakt einmal ausgeführt.

Die minimale Ausführungsdauer ergibt sich für den Fall, dass alle Elemente x_i aus X exakt gleich sind. In diesem Fall schlägt weder Test c-a noch Test c-b an und folglich werden c-a-i und c-b-i nie ausgeführt (alle Elemente sind sowohl das größte als auch das kleinste Element). Im schlimmsten Fall sind alle Elemente unterschiedlich und ihrer Größe nach sortiert. Dann schlägt bei absteigender

Reihenfolge jedes Mal Test c-a an und c-a-i wird ausgeführt. Bei umgekehrter Sortierung wird stets Test c-b anschlagen und c-b-i ausgeführt.

$$\begin{aligned} T_{min} =& 1 * a + 1 * b + (n-1) * c + \\ & (n-1) * ([c-a] + [c-b] + [c-c] + [c-d]) + 1 * d \\ =& (1 * 2 + 1 * 4 + (n-1) * 1 + (n-1) * (1+1+1+1) + 1 * 3)E \\ =& (5n+4)E \end{aligned}$$

$$\begin{aligned} T_{max} =& 1 * a + 1 * b + (n-1) * c + \\ & (n-1) * ([c-a] + [c-b] + [c-a/b-i] + [c-c] + [c-d]) + 1 * d \\ =& (1 * 2 + 1 * 4 + (n-1) * 1 + (n-1) * (1+1+1+1+1) + 1 * 3)E \\ =& (6n+3)E \end{aligned}$$

Abschließende Bemerkungen

Wichtig für die korrekte Funktion des Algorithmus ist, dass sich in der Menge X der Abarbeitungszeiten sämtliche Szenarien wiederfinden, also insbesondere die „Extremfälle“ bei denen Abarbeitung besonders lange gedauert hat, oder sehr schnell vonstatten ging. Wurden zufällig nur "guteÄbarbeitungszeiten erwischt, wird der Algorithmus auch nur diese Situation wiedergeben.