Technische Universität Darmstadt
Fachgebiet Theoretische Informatik

Prof. Johannes Buchmann
Nabil Alkeilani Alkadri
Nina Bindel
Patrick Struck

Algorithmen und Datenstrukturen

SS 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Lösungsblatt — 16.04.2018

P1 Korrektheit

Gegeben sind die folgenden Probleme für ein int-Array $A[0..n-1]$:

1. Minimum des Arrays berechnen.
2. Mittelwert der Einträge des Arrays berechnen.
3. Bestimmung des maximalen Index m, so dass für eine gegebene Schranke x gilt: $\sum_{i=0}^{m} A[i] \leq x$. Falls ein solcher Index m zu der gegebenen Schranke x nicht existiert, wird $m = -1$ gesetzt.

Befolgen Sie für jedes dieser Probleme diese zwei Schritte:

(a) Formulieren Sie einen Algorithmus, der das Problem löst.

(b) Bestimmen Sie eine geeignete Schleifeninvariante und benutzen Sie diese, um die Korrektheit Ihres Algorithmus zu beweisen.

Lösung. Im Folgenden bedeutet der Ausdruck "i-ter Schleifendurchlauf", dass der Wert der Schleifenvariable i ist.

1. (a) Der Algorithmus lautet

```
Minimum(A)
1  min=A[0]
2  for i=1 to n-1
3     if A[i] < min
4        min = A[i]
5  return min
```

   (b) Schleifeninvariante: Vor dem i-ten Schleifendurchlauf ist `min` das Minimum der Elemente $A[0...i-1]$.

   - Vor dem ersten Schleifendurchlauf ($i = 1$) gilt `min` $= A[0]$, also ist die Schleifeninvariante erfüllt.
   - Während des i-ten Schleifendurchlaufs ändert sich `min` nur, wenn das Element $A[i]$ kleiner als `min` und damit kleiner als alle Elemente in $A[0...i-1]$ ist. Nach dem i-ten Durchlauf ist `min` also das Minimum der Elemente $A[0...i]$ und es gilt: Falls die Invariante vor dem i-ten Durchlauf erfüllt ist, so ist sie es auch vor dem $i+1$-ten Durchlauf.
   - Die Schleife und der Algorithmus terminieren mit $i = n$. Durch Einsetzen von n in die Schleifeninvariante erhält man das gewünschte Ergebnis, nämlich dass `min` das Minimum des Arrays $A[0...n-1]$ ist.

2. (a) Der Algorithmus lautet

```
Mittelwert(A)
1  summe = A[0]
2  for i=1 to n-1
3     summe = summe + A[i]
```

```
4 mittelwert = summe / n
5 return mittelwert
```

(b) Schleifeninvariante: Vor dem i-ten Schleifendurchlauf ist `summe` die Summe der Elemente $A[0...i-1]$.

- Vor dem ersten Schleifendurchlauf ($i = 1$) gilt `summe` $= A[0]$. Also ist die Schleifeninvariante erfüllt.
- Während des i-ten Schleifendurchlaufs wird $A[i]$ zur Summe addiert. `summe` ist dann die Summe der Elemente $A[0...i]$ und die Invariante ist auch vor dem $i+1$-ten Durchlauf erfüllt.
- Die Schleife und der Algorithmus terminieren mit $i = n$. Damit ist `summe` die Summe der Elemente $A[0],\ldots,A[n-1]$. Dieser Wert wird noch durch n geteilt, um den Mittelwert aller Elemente des Arrays zu erhalten.

3. (a) Der Algorithmus lautet

```
MAX-INDEX (A)
1 m= -1
2 summe =0
3 for i=0 to n-1
4   summe = summe + A[i]
5   if summe <= x
6     m=i
7 return m
```

(b) Schleifeninvariante: Vor dem i-ten Schleifendurchlauf ist m der maximale Index in $\{0,\ldots,i-1\}$, so dass $\sum_{i=0}^{m} A[i] \leq x$ gilt oder $m = -1$, falls ein solcher Index nicht existiert.

- Vor dem ersten Schleifendurchlauf ($i = 0$) gilt $m = -1$, also ist die Schleifeninvariante erfüllt.
- Während des i-ten Schleifendurchlaufs wird $m = i$ genau dann gesetzt, wenn $\sum_{i=0}^{m} A[i] \leq x$ gilt. Weil die Schleifeninvariante für i gilt, ist m der maximale Index in $\{0,\ldots,i\}$, so dass $\sum_{i=0}^{m} A[i] \leq x$ gilt oder $m = -1$, falls ein solcher Index nicht existiert. Die Invariante ist also vor dem nächsten Schleifendurchlauf ($i+1$) ebenfalls erfüllt.
- Die Schleife und der Algorithmus terminieren mit $i = n$. Durch Einsetzen in die Schleifeninvariante sieht man: m ist der maximale Index in $\{0,\ldots,n-1\}$, so dass $\sum_{i=0}^{m} A[i] \leq x$ gilt oder $m = -1$, falls ein solcher Index nicht existiert.

P2 Rekursion

Es sei der folgende Algorithmus ALGO mit Eingabewerten x und $y \in \mathbb{N}$ gegeben.

```
ALGO(x,y)
1 if y>0
2   if y=0 mod 2
3     return ALGO(x^2,y/2)
4   else
5     return x*ALGO(x,y-1)
6 else
7   return 1
```

(a) Berechnen Sie ALGO(2,3) und ALGO(5,7).

(b) Was berechnet der Algorithmus ALGO?

(c) Geben Sie eine nicht-rekursive Variante des Algortihmus ALGO an.

Lösung.

(a) ALGO(2,3) = 2 * ALGO(2,2)=2 * ALGO(4,1)= 2 * 4 * ALGO(4,0)=2 * 4 =8
ALGO(5,7) = 5 * ALGO(5,6) = 5 * ALGO(25,3) = 5* 25* ALGO (25,2) = 5*25*ALGO(625,1)=5*25*625=78.125

(b) ALGO(x,y) berechnet den Wert x^y (schnelle Exponentiation).

(c) SCHNELLE-EXPONENTIATION(x,y)
```
1 res=1
2 while y>0
3   if y=1 mod 2
4     res=res*x
5   y=y div 2
6   x=x^2
7 return res
```

P3 Asymptotische Notation

(a) Kreuzen Sie in der Tabelle an, ob $f = O(g)$, $f = \Omega(g)$ und/oder $f = \Theta(g)$ gilt. Dabei ist $k > 0$ eine Konstante.

f(n)	**g(n)**	O	Ω	Θ	**f(n)**	**g(n)**	O	Ω	Θ
$\log_2(3^n)$	$\log_2(5^n)$				e^n	e^{2n}			
2^n	$2^{n/2}$				$\log_2 n$	$\log_{10} n$			
$n \log n$	n^2				2^{n+k}	2^n			

Begründen Sie Ihre Wahl.

(b) Es seien die beiden Funktionen $f(n) = 4n^2 + 3n + 7$ und $g(n) = n^3$ gegeben. Bestimmen Sie eine Funktion $n_0(c)$ so, dass $f(n) \leq c \cdot g(n)\ \forall n \geq n_0$. Mit der Existenz einer solchen Funktion ist bewiesen, dass $f(n) = o(g(n))$ ist.

Lösung.

a)

f(n)	**g(n)**	O	Ω	Θ	**f(n)**	**g(n)**	O	Ω	Θ
$\log_2(3^n)$	$\log_2(5^n)$	X	X	X	e^n	e^{2n}	X		
2^n	$2^{n/2}$		X		$\log_2 n$	$\log_{10} n$	X	X	X
$n \log n$	n^2	X			2^{n+k}	2^n	X	X	X

b) Wir setzen $n_0(c) = \max\left(5, \frac{5}{c}\right)$. Wir müssen zeigen, dass $\frac{f(n)}{g(n)} \leq c$ für alle $n \geq n_0$ gilt.

$$\frac{f(n)}{g(n)} = \frac{4n^2 + 3n + 7}{n^3} \overset{n \geq 5}{\leq} \frac{5n^2}{n^3} = \frac{5}{n} \overset{n \geq \frac{5}{c}}{\leq} \frac{5}{\frac{5}{c}} = c.$$

Um eine exakte Lösung des Problems zu finden, müsste man eine Gleichung dritten Grades lösen.

H1 Komplexität

Schätzen Sie die Laufzeit der folgenden Algorithmen in Abhängigkeit von der Eingabe ab. Benutzen Sie die O-Notation.

(a) A ist eine $m \times n$ Matrix, v ein n vector.

```
MULTIPLY (A,v)
1  for i=0 to m-1
2    res[i]=0
3    for j=0 to n-1
4      res[i]=res[i]+A[i][j]*v[j]
5  return res
```

(b) n ist eine natürliche Zahl

```
SUM-OF-SQUARES(n)
1  if n =0
2    return 0
3  else
4    a=n^2
5    return a+ SUM-OF-SQUARES (n-1)
```

Lösung.

(a)

MULTIPLY(A,v)	cost	times
1 **for** i=0 to m-1	c_1	m+1
2 res[i]=0	c_2	m
3 **for** j=0 to n-1	c_3	m*(n+1)
4 res[i]=res[i]+A[i][j]*v[j]	c_4	m*n
5 **return** res	c_5	1

$$\begin{aligned} T(m,n) &= (c_3+c_4)mn + (c_1+c_2+c_3)m + c_1 + c_5 \\ &= O(mn) \end{aligned}$$

If $m = O(n)$ we get $T(n) = O(n^2)$.

(b)

SUM-OF-SQUARES(n)	cost	times
1 **if** n =0	c_1	1
2 **return** 0	c_2	O(1)
3 **else**		
4 a=n^2	c_3	O(1)
5 **return** a+ SUM-OF-SQUARES(n-1)	c_4 + T(n-1)	O(1)

$$T(n) = \begin{cases} c_1 + c_2 & \text{, falls } n = 0 \\ T(n-1) + c_1 + c_3 + c_4 & \text{, sonst} \end{cases}$$

Für $n \geq 1$ wird die Rekursion n mal aufgerufen. Also ist $T(n) = n \cdot (c_1 + c_3 + c_4) + c_1 + c_2 = O(n)$.

H2 Asymptotische Notation

Beweise oder widerlege:

(a) $n^n = O(e^n)$

(b) $\log_2 n! = \Theta(n \cdot \log_2 n)$

Lösung.

(a) $n^n \neq O(e^n)$. Beweis:
$n^n/e^n = \left(\frac{n}{e}\right)^n \overset{n\to\infty}{\longrightarrow} \infty$. Der Quotient divergiert also, wenn n gegen unendlich geht. Hieraus folgt, dass es kein c, n_0 geben kann, so dass für alle $n > n_0$ gilt $n^n \leq c \cdot e^n$.

(b) $\log_2 n! = \Theta(n \cdot \log_2 n)$. Beweis:
a) $\log_2 n! \leq \log_2 n^n = n \cdot \log_2 n$. Damit gilt für $c = 1$ und alle $n \geq 1$: $\log_2 n! \leq c \cdot n \log_2 n$. Also gilt $\log_2 n! = O(n \cdot \log_2 n)$.
b) $\log_2 n! = \sum_{i=1}^{n} \log_2 i = 0 + \sum_{i=2}^{\lfloor n/2 \rfloor} \log_2 i + \sum_{i=\lfloor n/2 \rfloor + 1}^{n} \log_2 i \geq \lfloor n/2 \rfloor - 1 + \lceil n/2 \rceil \cdot \log_2 \frac{n}{2} \geq \lfloor n/2 \rfloor - 1 + (\lfloor n/2 \rfloor - 1) \cdot (\log_2 n - 1) = (\lfloor n/2 \rfloor - 1) \log_2 n \overset{n \geq 8}{\geq} \frac{1}{3} n \cdot \log_2 n$.
Damit gilt $\log_2 n! = \Omega(n \cdot \log_2 n)$, zusammen mit Teil (a) also $\log_2 n! = \Theta(n \cdot \log_2 n)$.