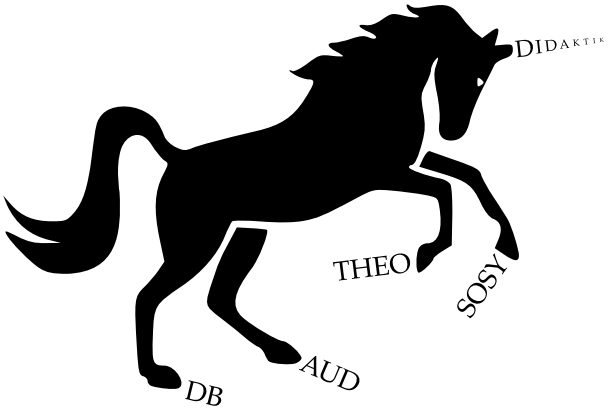
66115
Herbst
2016

Theoretische Informatik / Algorithmen (vertieft)

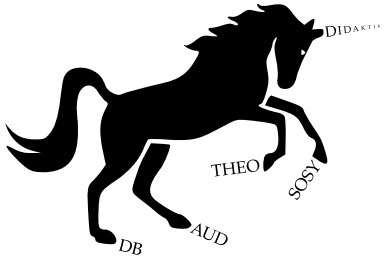
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 1

Aufgabe 2 [Chomsky-Hierarchie]

Ordnen Sie die folgenden Sprachen über $\Sigma = \{a, b\}$ bestmöglich in die Chomsky-Hierarchie ein und geben Sie jeweils eine kurze Begründung (1-2 Sätze).

(a) $L_1 = \{ a^n b^n \mid n \geq 1 \}$

Lösungsvorschlag

Typ-2-Sprache: Die Sprache L_1 ist kontextfrei, denn die Sprache braucht einen Speicher, da sie sich die Anzahl der a's merken muss, um die gleiche Anzahl an b's produzieren zu können. Dies ist mit einem Kellerautomaten möglich. Eine Grammtik der Sprache ist $G = (\{S\}, \Sigma, \{S \to aSb \mid \varepsilon\}, S)$

(b) $L_2 = \{ a^n b^n \mid$ die Turingmaschine mit Gödelnummer n hält auf leerer Eingabe $\}$

Lösungsvorschlag

Typ-0-Sprache: Die Sprache hat eine Typ-0-Grammatik, da sie offensichtlich semi-entscheidbar, aber nicht entscheidbar ist.

(c) $L_3 = {}^* \setminus L_1$

Lösungsvorschlag

Typ-2-Sprache: Die Sprache L_3 ist kontextfrei, da ein PDA existiert, der nicht aktzeptiert, wenn er L_1 aktzeptiert. (Ausgänge umgepolt)

(d) $L_4 = {}^* \setminus L_2$

Lösungsvorschlag

Nicht in der Hierachie: Das Komplement einer semi- aber unentscheidbaren Sprache kann nicht semi-entscheidbar sein, da L sonst entscheidbar wäre.

(e) $L_5 = \{ a^n b^m \mid n + m$ ist ein Vielfaches von drei $\}$

Lösungsvorschlag

Typ-3-Sprache: regulär, 2 Teilautomaten mit je 3 Zuständen (modulo 2 mal)

(f) $L_6 = \{ a^n b^n \mid n$ Quadratzahl $\}$

Lösungsvorschlag

nicht regulär, nicht kontextfrei (Pumping-Lemma)

Aufgabe 4 [Zahl der Inversionen von A]

Es sei $A[0 \ldots n-1]$ ein Array von paarweise verschiedenen ganzen Zahlen.

Wir interessieren uns für die Zahl der Inversionen von A; das sind Paare von Indices (i, j), sodass $i < j$ aber $A[i] > A[j]$. Die Inversionen im Array $[2, 3, 8, 6, 1]$ sind $(0, 4)$, da $A[0] > A[4]$ und weiter $(1, 4)$, $(2, 3)$, $(2, 4)$, $(3, 4)$. Es gibt also 5 Inversionen.

(a) Wie viel Inversionen hat das Array $[3, 7, 1, 4, 5, 9, 2]$?

Lösungsvorschlag

- $(0, 1)$: $3 > 1$
- $(0, 6)$: $3 > 2$
- $(1, 2)$: $7 > 1$
- $(1, 3)$: $7 > 4$
- $(1, 4)$: $7 > 5$
- $(1, 6)$: $7 > 2$
- $(3, 6)$: $4 > 2$
- $(4, 6)$: $5 > 2$

(b) Welches Array mit den Einträgen $\{1, \ldots, n\}$ hat die meisten Inversionen, welches hat die wenigsten?

Lösungsvorschlag

Folgt nach der 1 eine absteigend sortiere Folge, so hat sie am meisten Inversionen, z. B. $\{1, 7, 6, 5, 4, 3, 2\}$. Eine aufsteigend sortierte Zahlenfolge hat keine Inversionen, z. B. $\{1, 2, 3, 4, 5, 6, 7\}$.

(c) Entwerfen Sie eine Prozedur `int merge(int[] a, int i, int h, int j);`

welche das Teilarray a[i.,j] sortiert und die Zahl der in ihm enthaltenen Inversionen zurückliefert, wobei die folgenden Vorbedingungen angenommen werden:

- $0 \leq i \leq h \leq j < n$, wobei n die Länge von a ist (`n = a.length`).
- $a[i \ldots h]$ und $a[h+1 \ldots j]$ sind aufsteigend sortiert.
- Die Einträge von $a[i \ldots j]$ sind paarweise verschieden.

Ihre Prozedur soll in linearer Zeit, also $\mathcal{O}(j - i)$ laufen. Orientieren Sie sich bei Ihrer Lösung an der Mischoperation des bekannten Mergesort-Verfahrens.

(d) Entwerfen Sie nun ein Divide-and-Conquer-Verfahren zur Bestimmung der Zahl der Inversionen, indem Sie angelehnt an das Mergesort-Verfahren einen Algorithmus `ZI` beschreiben, der ein gegebenes Array in sortierter Form liefert und gleichzeitig dessen Inversionsanzahl berechnet. Im Beispiel wäre also

$$ZI([2, 3, 8, 6, 1]) = ([1, 2, 3, 6, 8], 5)$$

Die Laufzeit Ihres Algorithmus auf einem Array der Größe n soll $\mathcal{O}(n \log(n))$ sein.

Sie dürfen die Hilfsprozedur merge aus dem vorherigen Aufgabenteil verwenden, auch, wenn Sie diese nicht gelöst haben.

(e) Begründen Sie, dass Ihr Algorithmus die Laufzeit $\mathcal{O}(n \log(n))$ hat.

(f) Geben Sie die Lösungen folgender asymptotischer Rekurrenzen (in O-Notation) an:

(i) $T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(\log n)$

(ii) $T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n^2)$

(iii) $T(n) = 3 \cdot T(\frac{n}{2}) + \mathcal{O}(n)$

Thema Nr. 2

Aufgabe 3 [Registermaschinen (RAMs)]

Sei $M_0, M_1, \ldots$ eine Registermaschinen (RAMs). Beantworten Sie folgende Fragen zur Aufzählbarkeit und Entscheidbarkeit. Beweisen Sie Ihre Antwort.

Exkurs: Registermaschinen (RAMs)
Die Random Access Machine (kurz RAM) ist eine spezielle Art von Registermaschine. Sie hat die Fähigkeit der indirekten Adressierung der Register.
Die Random Access Machine besteht aus:

- einem Programm bestehend aus endlich vielen durchnummerierten Befehlen (beginnend mit Nummer 1)
- einem Befehlszähler b
- einem Akkumulator $c(0)$
- und einem unendlich großen Speicher aus durchnummerierten Speicherzellen (Registern) $c(1), c(2), c(3), \ldots$

Jedes Register (einschließlich b und $c(0)$) speichert eine beliebig große natürliche Zahl.

(a) Ist folgende Menge entscheidbar?

$A = \{x \in \mathbb{N} \mid x = 100 \text{ oder } M_x \text{ hält bei Eingabe } x\}$

Lösungsvorschlag

Ja, $x \geq 100$ ist entscheidbar und aufgrund des „oder“ ist die 2. Bedingung nur für $x < 100$ relevant. Da $x < 100$ eine endliche Menge darstellt, kann eine endliche Liste geführt werden und ein Experte kann für jeden Fall entscheiden, ob M_x hält oder nicht, somit ist A entscheidbar.

(b) Ist folgende Menge entscheidbar?

$B = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid M_x \text{ hält bei Eingabe } x \text{ genau dann, wenn } M_y \text{ bei Eingabe } y \text{ hält}\}$

Lösungsvorschlag

Nein. Dieses Problem entspricht der parallelen Ausführung des Halteproblems auf zwei Bändern. Das Halteproblem ist unentscheidbar, damit ist auch die parallele Ausfürhung des Halteproblems und damit B unentscheidbar.

(c) Ist folgende Menge aufzählbar?

$C = \{x \in \mathbb{N} \mid M_x \text{ hält bei Eingabe 0 mit dem Ergebnis 1}\}$

Lösungsvorschlag

Ja, die Menge ist aufzählbar, da die Menge aller Turningmaschinen aufzählbar und über natürliche Zahlen definiert ist (die wiederum aufzählbar sind).

Aufgabe 7 [Sortieren mit Quicksort]

(a) Gegeben ist die Ausgabe der Methode **Partition** (s. Pseudocode), rekonstruieren Sie die Eingabe.

Konkret sollen Sie das Array A = (_, _, 1, _, _) so vervollständigen, dass der Aufruf Partition(A, 1, 5) die Zahl 3 zurückgibt und nach dem Aufruf gilt, dass A = (1, 2, 3, 4, 5) ist.

Geben Sie A nach jedem Durchgang der for-Schleife in **Partition** an.

Lösungsvorschlag

```
 2  4  1  5  3  Eingabe
 2  4  1  5  3  zerlege
 2  4  1  5  3* markiere (i 4)
>2< 4  1  5  3  vertausche (i 0<>0)
 2 >4  1< 5  3  vertausche (i 1<>2)
 2  1 >4  5  3< vertausche (i 2<>4)
 2  1           zerlege
 2  1*          markiere (i 1)
>2  1<          vertausche (i 0<>1)
          5  4  zerlege
          5  4* markiere (i 4)
         >5  4< vertausche (i 3<>4)
 1  2  3  4  5  Ausgabe
```

(b) Beweisen Sie die Korrektheit von **Partition** (z. B. mittels einer Schleifeninvarianten)!

(c) Geben Sie für jede natürliche Zahl n eine Instanz I_n, der Länge n an, so dass QuickSort(I_n) $\Omega(n^2)$ Zeit benötigt. Begründen Sie Ihre Behauptung.

Lösungsvorschlag

$I_n = 1, 2, 3, \ldots, n$

Die Methode **Partition** wird n mal aufgerufen, weil bei jedem Aufruf der Methode nur eine Zahl, nämlich die größte Zahl, abgespalten wird.

- Partition(A, 1, n)
- Partition(A, 1, $n-1$)
- Partition(A, 1, $n-2$)
- Partition(A, 1, $\ldots$)
- Partition(A, 1, 1)

In der For-Schleife der Methode Partition wird bei jeder Wiederholung ein Vertauschvorgang durchgeführt (Die Zahlen werden mit sich selbst getauscht.)

```
 1  2  3  4  5  6  7  zerlege
 1  2  3  4  5  6  7* markiere (i 6)
```

```
>1< 2  3  4  5  6  7  vertausche (i 0<>0)
 1 >2< 3  4  5  6  7  vertausche (i 1<>1)
 1  2 >3< 4  5  6  7  vertausche (i 2<>2)
 1  2  3 >4< 5  6  7  vertausche (i 3<>3)
 1  2  3  4 >5< 6  7  vertausche (i 4<>4)
 1  2  3  4  5 >6< 7  vertausche (i 5<>5)
 1  2  3  4  5  6 >7< vertausche (i 6<>6)
 1  2  3  4  5  6     zerlege
 1  2  3  4  5  6*    markiere (i 5)
>1< 2  3  4  5  6     vertausche (i 0<>0)
 1 >2< 3  4  5  6     vertausche (i 1<>1)
 1  2 >3< 4  5  6     vertausche (i 2<>2)
 1  2  3 >4< 5  6     vertausche (i 3<>3)
 1  2  3  4 >5< 6     vertausche (i 4<>4)
 1  2  3  4  5 >6<    vertausche (i 5<>5)
 1  2  3  4  5        zerlege
 1  2  3  4  5*       markiere (i 4)
>1< 2  3  4  5        vertausche (i 0<>0)
 1 >2< 3  4  5        vertausche (i 1<>1)
 1  2 >3< 4  5        vertausche (i 2<>2)
 1  2  3 >4< 5        vertausche (i 3<>3)
 1  2  3  4 >5<       vertausche (i 4<>4)
 1  2  3  4           zerlege
 1  2  3  4*          markiere (i 3)
>1< 2  3  4           vertausche (i 0<>0)
 1 >2< 3  4           vertausche (i 1<>1)
 1  2 >3< 4           vertausche (i 2<>2)
 1  2  3 >4<          vertausche (i 3<>3)
 1  2  3              zerlege
 1  2  3*             markiere (i 2)
>1< 2  3              vertausche (i 0<>0)
 1 >2< 3              vertausche (i 1<>1)
 1  2 >3<             vertausche (i 2<>2)
 1  2                 zerlege
 1  2*                markiere (i 1)
>1< 2                 vertausche (i 0<>0)
 1 >2<                vertausche (i 1<>1)
```

(d) Was müsste Partition (in Linearzeit) leisten, damit QuickSort Instanzen der Länge n in $\mathcal{O}(n \cdot \log n)$ Zeit sortiert? Zeigen Sie, dass Partition mit der von Ihnen geforderten Eigenschaft zur gewünschten Laufzeit von QuickSort führt.

Exkurs: Master-Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

$a =$ Anzahl der rekursiven Aufrufe, Anzahl der Unterprobleme in der Rekursion ($a \geq 1$).

$\frac{1}{b} =$ Teil des Originalproblems, welches wiederum durch alle Unterprobleme repräsentiert wird, Anteil an der Verkleinerung des Problems ($b > 1$).

$f(n) =$ Kosten (Aufwand, Nebenkosten), die durch die Division des Problems und die Kombination der Teillösungen entstehen. Eine von $T(n)$ unabhängige und nicht negative Funktion.

Dann gilt:

1. Fall: $T(n) \in \Theta\left(n^{\log_b a}\right)$

falls $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$ für $\varepsilon > 0$

2. Fall: $T(n) \in \Theta\left(n^{\log_b a} \cdot \log n\right)$

falls $f(n) \in \Theta\left(n^{\log_b a}\right)$

3. Fall: $T(n) \in \Theta(f(n))$

falls $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ für $\varepsilon > 0$ und ebenfalls für ein c mit $0 < c < 1$ und alle hinreichend großen n gilt: $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$

Lösungsvorschlag

Die Methode **Partition** müsste die Instanzen der Länge n in zwei gleich große Teile spalten ($\frac{n-1}{2}$).

Allgemeine Rekursionsgleichung:

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$

Anzahl der rekursiven Aufrufe (a):

2

Anteil Verkleinerung des Problems (b):

um $\frac{1}{2}$ also $b = 2$

Laufzeit der rekursiven Funktion ($f(n)$):

n

Ergibt folgende Rekursionsgleichung:

$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$

1. Fall: $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$:

für $\varepsilon = 4$:
$f(n) = n \notin \mathcal{O}\left(n^{\log_2 2 - \varepsilon}\right)$

2. Fall: $f(n) \in \Theta\left(n^{\log_b a}\right)$:

$f(n) = n \in \Theta\left(n^{\log_2 2}\right) = \Theta(n)$

3. Fall: $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$:

$f(n) = n \notin \Omega\left(n^{\log_2 2 + \varepsilon}\right)$

$\Rightarrow T(n) \in \Theta\left(n^{\log_2 2} \cdot \log n\right) = \Theta(n \cdot \log n)$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

Funktion Quicksort(A, $l = 1$, $r = A$.length)

```
if l < r then
    m = Partition(A, l, r);
    Quicksort(A, l, m − 1);
    Quicksort(A, m + 1, r);
end
```

Funktion Partition(A, int l, int r)

```
pivot = A[r];
i = l;
for j = l to r − 1 do
    if A[j] ≤ pivot then
        Swap(A, i, j);
        i = i + l;
    end
end
```

Funktion Swap(A, int l, int r)

```
temp = A[i];
A[i] = A[y];
A[j] = temp;
```