Übungen zur Vorlesung

Algorithmen und Datenstrukturen

Übungsblatt 1 – Lösungen der Präsenzübungen

Arbeitsgruppe Kryptographie und Komplexitätstheorie
Prof. Dr. Marc Fischlin
Dr. Christian Janson
Patrick Harasser
Felix Rohrbach

Sommersemester 2019
Veröffentlicht am: 27.04.2019

Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

P1 (Gruppendiskussion)

Nehmen Sie sich etwas Zeit, um die folgenden Fachbegriffe in einer Kleingruppe zu besprechen, sodass sie anschließend in der Lage sind, die Begriffe dem Rest der Übungsgruppe zu erklären:

(a) Algorithmus
(b) Schleifeninvariante
(c) Totale Ordnung

P2 (Insertion Sort)

(a) In der Vorlesung haben Sie den Algorithmus Insertion Sort kennengelernt, der die Elemente einer Liste in aufsteigender Reihenfolge sortiert. Schreiben Sie diesen so um, dass er die Elemente in absteigender Reihenfolge sortiert. Beachten Sie, dass Ihr neuer Algorithmus weiterhin stabil bleiben soll.

(b) Sortieren Sie mithilfe dieses Algorithmus das folgende Array von Strings nach ihrer Länge: [“auf”,“Baum”,“Daten”, “Haus”,“sortieren”]. Geben Sie dabei die Zwischenschritte mit an.

Lösung. (a) Die Grundidee hier ist die Abbruchsbedingung der inneren While-Schleife so unzuschreiben, dass der Schlüsselwert so lange verschoben wird, bis zum ersten Mal ein größerer oder gleich großer Wert auftaucht:

AbsteigenderInsertionSort(A)

for $j = 1$ to A.length $- 1$ **do**
  $key = A[j]$
  $i = j - 1$
  while $i \geq 0$ and $A[i] < key$ **do**
    $A[i+1] = A[i]$
    $i = i - 1$
  $A[i+1] = key$

(b) Hier das Ausgangsarray und die Zwischenschritte der Algorithmus, für $1 \leq j \leq 4$:

[	“auf”	“Baum”	“Daten”	“Haus”	“sortieren”]
[	**“Baum”**	“auf”	“Daten”	“Haus”	“sortieren”]
[	**“Daten”**	“Baum”	“auf”	“Haus”	“sortieren”]
[	“Daten”	“Baum”	**“Haus”**	“auf”	“sortieren”]
[	**“sortieren”**	“Daten”	“Baum”	“Haus”	“auf”]

□

P3 (Eigenschaften von Algorithmen)

Beschreiben Sie jeweils für die folgenden Algorithmen kurz, was diese berechnen, und prüfen Sie, welche Eigenschaften (Determiniertheit, Determinismus, Terminierung, Korrektheit, Effizienz) der Algorithmus erfüllt.

Algorithmus1(int[] array)

```
int n = array.length;
while true {
  bool sorted = true ;
  for (int i = 0; i < n − 1; i + +){
    if array[i] < array[i + 1]{
      sorted = false ;
    }
  }
  if sorted{
    return array;
  }
  int n1 = random.nextInt(n); // 0 ≤ n1 < n
  int n2 = random.nextInt(n);
  int tmp = array[n1];
  array[n1] = array[n2];
  array[n2] = tmp;
}
```

Algorithmus2(int n)

```
if n%2 == 0{
  return false
}
for (int i = 3; i < n; i = i + 2){
  if (n%i == 0){
    return false ;
  }
}
return true ;
```

Lösung.

- Algorithmus 1 ist ein Sortieralgorithmus (auch bekannt als *Bogosort*). Algorithmus 1 ist determiniert und korrekt, da immer ein sortiertes Array zurückgegeben wird, jedoch nicht deterministisch, da die Operationen vom Zufall abhängig ist. Es ist möglich, dass der Algorithmus unendlich lange weiterläuft, dementsprechend ist er nicht terminiert. Effizient ist der Algorithmus definitiv nicht.

- Algorithmus 2 überprüft, ob die Eingabe n eine Primzahl ist. Der Algorithmus ist deterministisch und damit auf determiniert. Weiterhin terminiert der Algorithmus für alle Eingaben. Jedoch ist der Algorithmus nicht korrekt: So erkennt er 2 nicht als Primzahl, jede ungerade negative Zahl und 1 wird jedoch als Primzahl erkannt. Der Algorithmus ist effizient gegenüber der trivialen Implementierung, da er alle geraden Zahlen beim Testen überspringt, jedoch könnte er noch deutlich effizienter sein (bspw. würde es reichen, Zahlen bis $\sqrt{n}$ zu überprüfen).

□

P4 (Laufzeiten)

In der Tabelle sind verschiede Laufzeitfunktionen $f(n)$ abhängig von der Größe der Eingabe n in Millisekunden angegeben, sowie verschiedene Zeitabschnitte. Tragen Sie ein, wie groß n jeweils maximal sein darf, damit die Berechnung innerhalb des Zeitabschnitts durchläuft. Nehmen Sie einfachheitshalber an, dass ein Tag aus 24 Stunden und ein Jahr aus 365 Tagen besteht.

	1 Sekunde	1 Minute	1 Stunde	1 Tag	1 Monat	1 Jahr	1 Jahrhundert
$\text{sqrt}(n)$							
n							
$n \log_2(n)$							
n^2							
n^3							
2^n							
$n!$							

Lösung. Sei $f(n)$ eine der links angegebenen Funktionen, und c einer der oben genannten Zeitintervalle (in Sekunden). Dann ist die entsprechende Lösung gegeben durch

$$\max\left\{n \in \mathbb{N} : \frac{f(n)}{1000} \leq c\right\}.$$

Hier die ausgefüllte Tabelle:

	1 Sekunde	1 Minute	1 Stunde	1 Tag	1 Monat	1 Jahr	1 Jahrhundert
$\text{sqrt}(n)$	1.000.000	$3,6 \cdot 10^9$	$13 \cdot 10^{12}$	$7,5 \cdot 10^{15}$	$6,7 \cdot 10^{18}$	$1 \cdot 10^{21}$	$10 \cdot 10^{24}$
n	1000	60.000	$3,6 \cdot 10^6$	$86 \cdot 10^6$	$2,592 \cdot 10^9$	$32 \cdot 10^9$	$3,2 \cdot 10^{12}$
$n \log_2(n)$	140	4.895	204.095	$3.9 \cdot 10^6$	$97 \cdot 10^6$	$1 \cdot 10^9$	$87 \cdot 10^9$
n^2	31	244	1.897	9.295	50.911	177.583	1.775.837
n^3	10	39	153	442	1.373	3.159	14.664
2^n	9	15	21	26	31	34	41
$n!$	6	8	9	11	12	13	15

□

P5 (Türme von Hanoi)

Die *Türme von Hanoi* sind ein bekanntes Geduldsspiel, dessen Regeln hier nochmal kurz erklärt werden.

Das Spiel besteht aus drei Stäben. Auf die Stäbe wird eine feste Anzahl gelochter Scheiben gelegt, alle unterschiedlicher Größe. Zu Beginn liegen alle Scheiben auf einem Stab, der Größe nach geordnet, mit der größten Scheibe ganz unten. Ziel des Spiels ist es, alle Scheiben vom Ausgangsstab auf einen anderen Stab zu versetzen. Dabei muss folgende Regel beachtet werden: Bei jedem Zug darf immer nur die oberste Scheibe eines beliebigen Stapels bewegt, und auf einen anderen Stab gelegt werden, unter der Voraussetzung, dass sich dort nicht schon eine kleinere Scheibe befindet.

(a) Geben Sie einen rekursiven Algorithmus an, der die Lösung des Spiels beschreibt, und berechnen Sie dessen Laufzeit.
(b) Einer Legende zufolge gibt es irgendwo in der Stadt Hanoi einen Brahma-Tempel, in dem Mönche dieses Spiel mit 64 Scheiben aus purem Gold spielen. Die Legende besagt, dass die Welt in Schutt und Asche fallen wird, sobald sie mit dem Spiel fertig sind. Angenommen, die Mönche verwenden Ihren Algorithmus aus (a) und verschieben eine Scheibe pro Sekunde, wie lange wird es dauern, bis sie das Spiel zu Ende gespielt haben? Geben Sie eine grobe Abschätzung.

(c*) *Zusatzaufgabe:* Zeigen Sie, dass Ihr Algorithmus aus (a) "optimal" ist: Es gibt keinen korrekten Algorithmus der es ermöglicht, das Spiel in weniger Schritten zu lösen. (Falls dies nicht der Fall sein sollte, revidieren Sie Ihre Angabe.) Was bedeutet das für die Mönche aus (b)? Müssen wir uns über den "Weltuntergang durch die Türme von Hanoi" Gedanken machen?[1]

Lösung. (a) Es sei $n \in \mathbb{N}$ die Anzahl der Scheiben, nummeriert von der kleinsten zur größten (die erste Scheibe ist also die kleinste, und die n-te die größte). Wir nummerieren auch die drei Stäbe via 0, 1, und 2.
Der Algorithmus TürmeHanoi bekommt als Eingabe die Anzahl n der Scheiben, sowie den Ausgangs- und Zielstab $i, j \in \{0, 1, 2\}$. Falls $n = 0$ oder $i = j$ muss der Algorithmus nichts machen. Anderenfalls verschiebt er zunächst die ersten $n-1$ Scheiben auf den dritten (Hilfs)Stab $k = 3 - i - j$ (via TürmeHanoi$(n-1, i, k)$), bewegt dann die n-te Scheibe von i nach j, und gewinnt schließlich das Spiel, indem er die $n-1$ Scheiben von k nach j versetzt. In Pseudocode also:

TürmeHanoi(n, i, j)

if $n > 0$ and $i \neq j$ **then**
  $k \leftarrow 3 - i - j$
  TürmeHanoi$(n-1, i, k)$
  Bewege Scheibe n von i nach j
  TürmeHanoi$(n-1, k, j)$

Zur Laufzeit: Es sei $M_{n,i,j}$ die Anzahl der vom Algorithmus benötigten Spielzüge, um n Scheiben von Stab i auf Stab j zu verschieben. Falls $n = 0$ oder $i = j$ gilt offensichtlich $M_{n,i,j} = 0$. Anderenfalls folgt aus der rekursiven Definition des Algorithmus, dass $M_{n,i,j} = M_{n-1,i,k} + 1 + M_{n-1,k,j}$, also $M_{n,i,j} = 2M_{n-1,i,j} + 1$, da die Nummerierung der Stäbe wegen der Symmetrie des Problems für die Laufzeit irrelevant ist. Wir erhalten also, für $i \neq j$, folgende lineare Rekursionsgleichung mit entsprechendem Anfangswert:

$$\begin{cases} M_{n,i,j} = 2M_{n-1,i,j} + 1 & \text{für } n > 0, \\ M_{0,i,j} = 0. \end{cases}$$

Führt man nun die Variable $P_{n,i,j} = M_{n,i,j} + 1$ ein, so erhält man folgende äquivalente lineare Rekursionsgleichung:

$$\begin{cases} P_{n,i,j} = 2P_{n-1,i,j} & \text{für } n > 0, \\ P_{0,i,j} = 1. \end{cases}$$

Die Lösung dieses Anfangsproblems ist bekannterweise $P_{n,i,j} = 2^n$, also $M_{n,i,j} = 2^n - 1$. Somit erhalten wir:

$$M_{n,i,j} = \begin{cases} 0 & \text{für } n = 0 \text{ oder } i = j, \\ 2^n - 1 & \text{sonst.} \end{cases}$$

(b) Wenn die Mönche den Algorithmus aus (a) verwenden, brauchen sie $2^{64} - 1$ Spielzüge, um alle Scheiben von einem Stab auf einen anderen zu verschieben. Bei einem Spielzug pro Sekunde dauert das Spiel also $(2^{64}-1)/(60 \cdot 60 \cdot 24 \cdot 365)$ Jahre, und

$$\frac{2^{64} - 1}{60 \cdot 60 \cdot 24 \cdot 365} \geq \frac{2^{63}}{2^6 \cdot 2^6 \cdot 2^5 \cdot 2^9} = 2^{37} = (2^{10})^3 \cdot 2^7 \geq (10^3)^3 \cdot 2^7 = 128 \cdot 10^9.$$

Nach einer ersten (sehr) groben Abschätzung dauert es also mindestens 128 Milliarden Jahre, bis die Mönche das Spiel zu Ende gespielt haben.

(c*) Wir müssen zeigen, dass jeder korrekte Algorithmus A mindestens $M_{n,i,j}$ Spielzüge benötigt, um n Scheiben von Stab i auf Stab j zu verschieben. Wenn $i = j$ ist dies offensichtlich der Fall, da hier $M_{n,i,j} = 0$. Ab jetzt sei also $i \neq j$.
Wir beweisen unsere Behauptung mittels Induktion. Sei A ein beliebiger Algorithmus, der das Spiel spielt, und sei $S \subseteq \mathbb{N}$ die Menge aller natürlichen Zahlen, sodass A mindestens $M_{n,i,j}$ Spielzüge braucht. Wir zeigen $S = \mathbb{N}$:

- Offensichtlich ist $0 \in S$, da in diesem Fall $M_{0,i,j} = 0$.

[1] Unser Universum ist ca. 13,8 Milliarden Jahre alt.

- Angenommen, $n \in S$. Zu zeigen ist, dass dann auch $n+1 \in S$. Falls A auf der Eingabe mit $n+1$ Scheiben nicht terminiert, dann ist die Aussage trivialerweise richtig. Wir können uns also auf den Fall beschränken, dass A terminiert.
  Die entscheidende Bemerkung ist nun: Nachdem $i \neq j$ und A korrekt ist und terminiert, muss die $n+1$-te Scheibe mindestens einmal vom Ausgangsstab i auf einen anderen Stab $r \in \{j, 3-i-j\}$ bewegt werden. Damit dies aber möglich ist, müssen zuerst die oberen n Scheiben auf den dritten Stab $3-i-r$ versetzt werden, denn nur so kann die $n+1$-te Scheibe überhaupt bewegt werden.
  Ein ähnliches Argument gilt auch für den letzten Spielzug, der die $n+1$-te Scheibe betrifft: Bewegt A Scheibe $n+1$ von einem Stab $s \in \{i, 3-i-j\}$ auf Stab j, so müssen sich davor alle anderen Scheiben auf Stab $3-j-s$ befinden, und können anschließend auf Stab j verschoben werden. Algorithmus A muss also mindestens:
  - Die ersten n Scheiben von Stab i auf Stab $3-i-r$ verschieben, wofür A nach Induktionsvoraussetzung mindestens $M_{n,i,3-i-r} = M_{n,i,j}$ Spielzüge braucht;
  - Einmal Scheibe $n+1$ verschieben (1 Spielzug);
  - Die ersten n Scheiben von Stab $3-j-s$ auf Stab j verschieben, wofür A nach Induktionsvoraussetzung wieder mindestens $M_{n,3-j-s,j} = M_{n,i,j}$ Spielzüge benötigt.

  In Summe beträgt die Laufzeit von A also mindestens $2M_{n,i,j} + 1 = M_{n+1,i,j}$, und die Behauptung ist gezeigt.

Für die Mönche aus (b) bedeutet dies, dass sie bereits einen optimalen Algorithmus verwenden: Es gibt keinen "besseren" Algorithmus, der es Ihnen ermöglicht, mit dem Spiel schneller fertig zu werden. Da es nach (b) also mindestens $128 \cdot 10^9$ Jahre bis zum "Weltuntergang durch die Türme von Hanoi" dauert, und unser Universum dagegen nur schlappe $13,8 \cdot 10^9$ Jahre alt ist, besteht diesbezüglich kein Grund zur Sorge. □