Informatik I

20. Iteration vs. Rekursion

Jan-Georg Smaus

Albert-Ludwigs-Universität Freiburg

1. Februar 2011

20.1 Iteration vs. Rekursion

20.1 Iteration vs. Rekursion

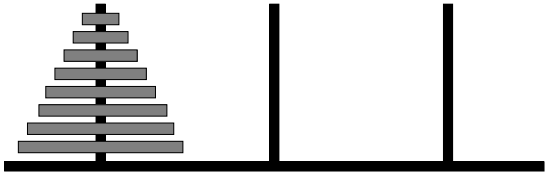
- Iterativer Algorithmus
- Beweis der Korrektheit
- Python-Programm
- Zusammenfassung

Iteration vs. Rekursion

- Wir haben am Beispiel von der Fakultätsfunktion und den Methoden für verlinkte Listen gesehen, dass man manche Probleme sowohl mit einer rekursiven als auch mit einer iterativen Funktion lösen kann.
- Die iterative ist meistens effizienter.
- Die rekursive Formulierung ist häufig eleganter und verständlicher.
- Wir werden heute ein Problem betrachten, das sich sehr einfach rekursiv lösen lässt, wohingegen die iterative Lösung überhaupt nicht offensichtlich ist: Türme von Hanoi.

Spielbeschreibung (Erinnerung)

Der Spielzustand besteht aus drei Säulen 1, 2, 3, auf denen insgesamt n Scheiben unterschiedlicher Größe aufgestapelt sind.



Die Aufgabe des Puzzles ist es, den Turm auf einen der anderen Pfähle zu bewegen, allerdings unter folgenden Einschränkungen:

- Es wird nur eine Scheibe auf einmal bewegt.
- Es werden immer nur kleinere auf größere Scheiben gelegt.

Rekursive Prozedur (Erinnerung)

Folgende rekursive Scheme-Prozedur löst das Problem:

```
(define hanoi
  (lambda (n)
    (if (zero? n)
        empty
        (let ((one-less (hanoi (- n 1))))
          (append
           (renumber-moves one-less 3 2)
           (cons (make-hanoi-move 1 3)
                 (renumber-moves one-less 1 2)))))))
```


Iterativer Algorithmus für Türme von Hanoi

- Buneman und Levy haben 1980 einen iterativen Algorithmus für die Türme von Hanoi vorgeschlagen [BL80].
- Es ist nicht offensichtlich, dass dieser Algorithmus korrekt ist.
- Mein Python-Programm zu diesem Algorithmus hat knapp über 100 Zeilen.
- Dagegen lässt sich der Algorithmus in Pseudocode sehr leicht kommunizieren.

Pseudocode

- Pseudocode ist ein Zwischending zwischen natürlicher Sprache und einer Programmiersprache. Er dient dazu, einem menschlichen Leser einen Algorithmus zu kommunizieren.
- Er sollte so präzise sein, dass ein Informatiker ihn unzweideutig versteht.
- Er sollte so abstrakt sein, dass ein Informatiker leicht den Überblick behält und sich nicht in Details verliert.
- Er sollte nur Konstrukte einer Programmiersprache verwenden, die so gängig sind, dass ein Informatiker sie in jedem Fall kennt. Häufig verwendet man Syntax, die sich an die Programmiersprache Pascal anlehnt. Wir lehnen uns aber hier an Python an.
- Natürlich gibt es keine allgemein anerkannte Definition von Pseudocode.

Iterativer Algorithmus in Pseudocode

```
if #n gerade:
    step = 1
else:
    step = 2
while #noch nicht fertig
    #bewege kleinste Scheibe von ...
    #... i nach (i+step) modulo 3
    if #gemaess Spielregeln moeglich:
        #bewege nichtkleinste Scheibe
```

Probieren wir's aus für $n = 1, 2, 3, 4 \ldots$

Beweis der Korrektheit

- Es ist überhaupt nicht offensichtlich, dass der Algorithmus korrekt ist.
- Selbst wenn er korrekt ist, stellt sich die Frage, ob er das Problem optimal löst.
- Beide Aussagen werden wir jetzt beweisen mittels vollständiger Induktion.

Das Beweisschema der vollständigen Induktion (Erinnerung)

Sei $P(n)$ eine Eigenschaft einer Zahl $n \in \mathbb{N}$ (Prädikat).
Zeige $(\forall n \in \mathbb{N})\ P(n)$.
Definiere $M := \{n \in \mathbb{N} \mid P(n) \text{ gilt}\} \subseteq \mathbb{N}$.
Induktionsaxiom: Falls $0 \in M$ und $(\forall n)\ n \in M \Rightarrow n' \in M$ dann $M = \mathbb{N}$.

Induktionsschema

Falls $P(0)$ (Induktionsbasis, -anfang)
und
$(\forall n)\ P(n) \Rightarrow P(n')$ (Induktionsschritt)
dann
$(\forall n \in \mathbb{N})\ P(n)$. ($P(n)$ Induktionshypothese, -behauptung)

Satz

Satz

1. Die vom Algorithmus berechnete Zugfolge bewegt den Hanoi-Turm der Größe n vom ersten auf den dritten Pfahl.
2. Es gibt keine kürzere Zugfolge, die den Hanoi-Turm der Größe n vom ersten auf den dritten Pfahl bewegt.

Induktionsbasis

- Für $n = 1$ ist `step` $= 1$, da 1 ungerade ist.
- Die kleinste Scheibe wird von Pfahl 1 nach Pfahl 3 bewegt.
- Gemäß den Spielregeln ist es nicht möglich, eine nichtkleinste Scheibe zu bewegen, da es gar keine solche gibt.
- Das Problem ist in einem Zug gelöst, was offensichtlich optimal ist.

Induktionsschritt

Nimm an, für n gelte die Induktionshypothese. Sei $f(n)$ die Anzahl der berechneten Züge (optimal).
Betrachten wir nun zuerst den Fall, dass n gerade ist.
Wir wenden nun den Algorithmus auf $n+1$ an. Wir halten zunächst folgende Beobachtung fest:

GS fix

Die größte Scheibe kann nicht bewegt werden, bevor nicht der Turm bis auf die größte Scheibe vollständig von Pfahl 1 wegbewegt worden ist, sei es auf Pfahl 2 oder auf Pfahl 3.

Hilfssatz

Die ersten $f(n)$ Züge des Algorithmus für $n+1$ Scheiben sind die Züge des Algorithmus für n Scheiben, wobei Pfahl 2 und Pfahl 3 vertauscht sind.

Beweis des Hilfssatzes

Den Hilfssatz beweisen wir ebenfalls mittels vollständiger Induktion über die Anzahl der Züge. Wir bezeichen den Algorithmus für n als Alg(n) und für $n+1$ als Alg($n+1$).

Induktionsbasis (erster Zug):
Alg($n+1$) bewegt die kleinste Scheibe von Pfahl 1 nach Pfahl 3.
Alg(n) Scheiben bewegt die kleinste Scheibe von Pfahl 1 nach Pfahl 2.
Also: gleiche "Zugfolge" aber mit Pfahl 2 und Pfahl 3 vertauscht.

Induktionsschritt für den Hilfssatz Ia

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 1: der $(j+1)$te Zug ist

```
#bewege kleinste Scheibe von ...
#... i nach (i+step) modulo 3
```

- in Alg(n) liegt die kleinste Scheibe auf $i = 1$: dann bewegt Alg(n) die kleinste Scheibe nach Pfahl 2. Nach Induktionshypothese liegt die Scheibe in Alg($n+1$) ebenfalls auf $i = 1$ und wird nach Pfahl 3 bewegt.

Induktionsschritt für den Hilfssatz Ib

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 1: der $(j+1)$te Zug ist

```
#bewege kleinste Scheibe von ...
#... i nach (i+step) modulo 3
```

- in Alg(n) liegt die kleinste Scheibe auf $i = 2$: dann bewegt Alg(n) die kleinste Scheibe nach Pfahl 3. Nach Induktionshypothese liegt die Scheibe in Alg($n+1$) auf $i = 3$ und wird nach Pfahl 2 bewegt.

Induktionsschritt für den Hilfssatz Ic

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 1: der $(j+1)$te Zug ist

```
#bewege kleinste Scheibe von ...
#... i nach (i+step) modulo 3
```

- in Alg(n) liegt die kleinste Scheibe auf $i = 3$: dann bewegt Alg(n) die kleinste Scheibe nach Pfahl 1. Nach Induktionshypothese liegt die Scheibe in Alg($n+1$) auf $i = 2$ und wird nach Pfahl 1 bewegt.

Insgesamt gilt: der $(j+1)$te Zug von Alg(n) und Alg($n+1$) ist der gleiche, bis auf dass Pfahl 2 und Pfahl 3 vertauscht sind.

Induktionsschritt für den Hilfssatz IIa

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 2: der $(j+1)$te Zug ist

```
if #gemaess Spielregeln moeglich:
    #bewege nichtkleinste Scheibe
```

- in Alg(n) liegt die zweitkleinste oben liegende Scheibe auf Pfahl 1, und die drittkleinste auf Pfahl 2: dann macht Alg(n) einen Zug von Pfahl 1 nach Pfahl 2. Nach Induktionshypothese liegt in Alg($n+1$) die zweitkleinste oben liegende Scheibe auf Pfahl 1, und die drittkleinste auf Pfahl 3, und Alg(n) macht einen Zug von Pfahl 1 nach Pfahl 3.

Induktionsschritt für den Hilfssatz IIb

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 2: der $(j+1)$te Zug ist

```
if #gemaess Spielregeln moeglich:
    #bewege nichtkleinste Scheibe
```

- in Alg(n) liegt die zweitkleinste oben liegende Scheibe auf Pfahl 1, und die drittkleinste auf Pfahl 3: dann macht Alg(n) einen Zug von Pfahl 1 nach Pfahl 3. Nach Induktionshypothese liegt in Alg($n+1$) die zweitkleinste oben liegende Scheibe auf Pfahl 1, und die drittkleinste auf Pfahl 2, und Alg(n) macht einen Zug von Pfahl 1 nach Pfahl 2.

Induktionsschritt für den Hilfssatz IIc

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 2: der $(j+1)$te Zug ist

```
if #gemaess Spielregeln moeglich:
    #bewege nichtkleinste Scheibe
```

- in Alg(n) liegt die zweitkleinste oben liegende Scheibe auf Pfahl 2, und die drittkleinste auf Pfahl 1: dann macht Alg(n) einen Zug von Pfahl 2 nach Pfahl 1. Nach Induktionshypothese liegt in Alg($n+1$) die zweitkleinste oben liegende Scheibe auf Pfahl 3, und die drittkleinste auf Pfahl 1, und Alg(n) macht einen Zug von Pfahl 3 nach Pfahl 1.

Induktionsschritt für den Hilfssatz IId

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 2: der $(j+1)$te Zug ist

```
if #gemaess Spielregeln moeglich:
    #bewege nichtkleinste Scheibe
```

- in Alg(n) liegt die zweitkleinste oben liegende Scheibe auf Pfahl 2, und die drittkleinste auf Pfahl 3: dann macht Alg(n) einen Zug von Pfahl 2 nach Pfahl 3. Nach Induktionshypothese liegt in Alg($n+1$) die zweitkleinste oben liegende Scheibe auf Pfahl 3, und die drittkleinste auf Pfahl 2, und Alg(n) macht einen Zug von Pfahl 3 nach Pfahl 2.

Induktionsschritt für den Hilfssatz IIe

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 2: der $(j+1)$te Zug ist

```
if #gemaess Spielregeln moeglich:
    #bewege nichtkleinste Scheibe
```

- in Alg(n) liegt die zweitkleinste oben liegende Scheibe auf Pfahl 3, und die drittkleinste auf Pfahl 1: dann macht Alg(n) einen Zug von Pfahl 3 nach Pfahl 1. Nach Induktionshypothese liegt in Alg($n+1$) die zweitkleinste oben liegende Scheibe auf Pfahl 2, und die drittkleinste auf Pfahl 1, und Alg(n) macht einen Zug von Pfahl 2 nach Pfahl 1.

Induktionsschritt für den Hilfssatz IIf

Der Hilfssatz gelte für eine Zugfolge der Länge $j < f(n)$.

Fall 2: der $(j+1)$te Zug ist

```
if #gemaess Spielregeln moeglich:
    #bewege nichtkleinste Scheibe
```

- in Alg(n) liegt die zweitkleinste oben liegende Scheibe auf Pfahl 3, und die drittkleinste auf Pfahl 2: dann macht Alg(n) einen Zug von Pfahl 3 nach Pfahl 2. Nach Induktionshypothese liegt in Alg($n+1$) die zweitkleinste oben liegende Scheibe auf Pfahl 2, und die drittkleinste auf Pfahl 3, und Alg(n) macht einen Zug von Pfahl 2 nach Pfahl 3.

Insgesamt gilt: der $(j+1)$te Zug von Alg(n) und Alg($n+1$) ist der gleiche, bis auf dass Pfahl 2 und Pfahl 3 vertauscht sind.

Hilfssatz und Korollar

Hilfssatz
Die ersten $f(n)$ Züge des Algorithmus für $n+1$ Scheiben sind die Züge des Algorithmus für n Scheiben, wobei Pfahl 2 und Pfahl 3 vertauscht sind.

Korollar
Die ersten $f(n)$ Züge des Algorithmus für $n+1$ Scheiben bewegen den Turm bis auf die größte Scheibe von Pfahl 1 nach Pfahl 2.

Turm n von Pfahl 1 nach Pfahl 2

Da die kleinste Scheibe nach $f(n)$ Schritten auf Pfahl 2 zuoberst aufliegt, muss der $f(n)$te Schritt ein Schritt gemäß

```
#bewege kleinste Scheibe von ...
#... i nach (i+step) modulo 3
```

gewesen sein.

Demnach ist aber im $f(n)+1$ten Schritt

```
if #gemaess Spielregeln moeglich:
    #bewege nichtkleinste Scheibe
```

möglich und kann nur bedeuten: bewege die größte Scheibe von Pfahl 1 nach Pfahl 3.

Turm n von Pfahl 2 nach Pfahl 3

Ganz analog zum Hilfssatz

Hilfssatz
Die ersten $f(n)$ Züge des Algorithmus für $n+1$ Scheiben sind die Züge des Algorithmus für n Scheiben, wobei Pfahl 2 und Pfahl 3 vertauscht sind.

kann man nun zeigen:

Hilfssatz 2
Der Algorithmus für $n+1$ Scheiben, angewandt auf den Spielzustand, in dem der Turm bis auf die größte Scheibe auf Pfahl 2 liegt, bewegt diesen Turm in $f(n)$ Zügen von Pfahl 2 auf Pfahl 3.

Gesamte Zugfolge

- Der Algorithmus für $n+1$ Scheiben bewegt also in $2 \cdot f(n)+1$ Zügen den Turm von Pfahl 1 nach Pfahl 3.
- Da $f(1)=1$, folgt $f(n)=2^n-1$ (einfacher Induktionsbeweis).
- Aufgrund von GS fix kann es nicht besser gehen.

Wir haben angenommen, dass n gerade ist. Der Fall, dass n ungerade ist, ist aber ganz analog.
Somit haben wir gezeigt:

Satz
1. Die vom Algorithmus berechnete Zugfolge bewegt den Hanoi-Turm der Größe n vom ersten auf den dritten Pfahl.
2. Es gibt keine kürzere Zugfolge, die den Hanoi-Turm der Größe n vom ersten auf den dritten Pfahl bewegt.

Das Hauptprogramm ist selbst in echtem Python-Code nicht sehr kompliziert:

python.py

```
if n % 2 == 0:
    step = 1
else:
    step = 2
state = State(n)
continu = True
while continu:
    state.move_smallest(step)
    continu = state.move_other()
```


Wieso $>$ 100 Zeilen?

python.py

```
class State:
    def __init__(self, n):
        self.one = empty
        self.two = empty
        self.three = empty
        for i in range(n, 0, -1):
            self.one = self.one.cons(i)
```

- Der Spielzustand ist mittels einer Klasse implementiert, die drei Attribute für die drei Pfähle hat. Es gibt 6 mögliche Züge ($1 \to 2$, $1 \to 3$, $2 \to 1$, $2 \to 3$, $3 \to 1$, $3 \to 2$), was zu allerhand Fallunterscheidungen mit symmetrischen Fällen führt.
- Besser wäre, man könnte mittels Indizes auf die Pfähle zugreifen, wie Sie es schon aus den Übungen kennen.

Zusammenfassung

- Wir haben ein Beispiel für ein Problem gesehen, dass eine einfache rekursive Lösung hat, und außerdem eine iterative Lösung, deren Korrektheit aber überhaupt nicht offensichtlich ist.
- Wir haben die Korrektheit und Optimalität der iterativen Lösung bewiesen, und somit die Äquivalenz zur rekursiven Lösung.
- Unser Beweis hat durchaus an einigen Stellen an die Intuition appelliert, erfüllt aber die Anforderungen an Strenge, die gemeinhin in der Informatik üblich sind.
- Der Zweck dieses Kapitels war einerseits, Iteration und Rekursion zu vergleichen, aber vor allem, noch mehr Übung im Beweisen zu erlangen.

Literatur I


Peter Buneman and Leon S. Levy.
The towers of hanoi problem.
Inf. Process. Lett., 10(4/5):243–244, 1980.