

## Rekursion

Dr. Philipp Wendler

Zentralübung zur Vorlesung

„Einführung in die Informatik: Programmierung und Softwareentwicklung“

<https://www.sosy-lab.org/Teaching/2018-WS-InfoEinf/>

WS18/19

### Rekursion Allgemein

Ein Algorithmus ist rekursiv, wenn in seiner Beschreibung derselbe Algorithmus wieder aufgerufen wird.

Das Prinzip der Rekursion wird folgendermaßen zur Lösung von Problemen eingesetzt (informell):

- In einfachen Fällen: "Ich weiß das Ergebnis sofort."
- In schwierigeren Fällen:  
"Wenn ich die Lösung eines kleineren Problems kenne (rekursiv), kann ich das Gesamtergebnis daraus berechnen."

z.B. aus der Vorlesung:

- einfacher Fall:  $0! = 1$
- schwieriger Fall: Wenn ich den Wert von  $(n-1)!$  kenne, gilt:  $n! = (n-1)! * n$

Dieses Vorgehen muss für alle  $n$  gleich sein!

### Divide et Impera im Römischen Reich

- Die Lenkung des gesamten Römischen Reichs bedarf...
  - ... großer Entscheidungen für das ganze Reich,
  - ... kleiner Entscheidungen für bestimmte Gebiete.
- Divide et Impera: Teile das Reich in kleinere Gebiete.
  - Koordiniere nur das Zusammenspiel aller Gebiete.
  - Delegiere die Lenkung jedes Gebiets an einen Verantwortlichen.
- **Rekursion:**  
Jeder Verantwortliche teilt seinen Verantwortungsbereich wieder in kleinere Gebiete auf, **solange bis** der Bereich überschaubar ist.



### Rekursion am Beispiel

#### Welche Aufgabe kann **nicht** rekursiv gelöst werden?

- a) Löffelweise Suppe essen:  
Falls der Teller leer ist, fertig;  
Ansonsten iss einen Löffel Suppe und beginne von vorne.
- b) Ein Auto am "Fließband" zusammenbauen  
Falls das Auto komplett ist, fertig;  
Ansonsten baue einen bestimmten Teil und beginne von vorne.
- c) Ein Viereck zeichnen  
Falls schon vier Kanten gezeichnet sind, fertig;  
Ansonsten zeichne eine Kante, drehe das Blatt um  $90^\circ$  und beginne von vorne.

## Aufgabe 1: Potenzfunktion (Beispiel 5<sup>4</sup>)

Schreiben Sie eine Methode, die für zwei nicht-negative Zahlen *a* und *n* vom Typ `int` die Potenz *a<sup>n</sup>* berechnet.

$$5^4 = 5 * 5 * 5 * 5^1 \quad (= 5 * 5 * 5 * 5 * 5^0)$$

$$5^4 = 5 * 5 * 5^2$$

$$5^4 = 5 * 5^3$$

Präzisierung:  $5^n = \overbrace{5 * \dots * 5}^{n\text{-mal}} = 5 * 5^{n-1}$

Per Definition:  $5^0 = 1$

Induktive Definition:

$$a^n = 1, \quad \text{falls } n = 0,$$

$$a^n = a * a^{n-1}, \quad \text{falls } n > 0$$

## Aufgabe 1: Potenzfunktion (rekursiv)

```
public static int potrek(int a, int n)
{
    if (n == 0)
        return 1;
    else
        return a * potrek(a, n - 1);
}
```

Induktive Definition:

$$a^n = 1, \quad \text{falls } n = 0,$$

$$a^n = a * a^{n-1}, \quad \text{falls } n > 0$$

## Aufgabe 1: Potenzfunktion (rekursiv): Beispiel 5<sup>3</sup>

```
public static int potrek(int a, int n) {
    if (n == 0)
        return 1;
    else
        return a * potrek(a, n - 1);
}
```

1. Aufruf: *a*=5, *n*=3

2. Aufruf: *a*=5, *n*=2

3. Aufruf: *a*=5, *n*=1

4. Aufruf: *a*=5, *n*=0

Rückgabewert: 1

potrek(5, 3)

5 \* potrek(5, 2)

5 \* (5 \* potrek(5, 1))

5 \* (5 \* (5 \* potrek(5, 0)))

5 \* (5 \* (5 \* 1));

Ausrechnen 125

## Aufgabe 1: Potenzfunktion (iterativ)

```
public static int potiter (int a, int n) {
    int akk = 1;
    while (n > 0) {
        akk = a * akk;
        n = n - 1;
    }
    return akk;
}
```

Induktive Definition:

$$a^n = 1, \quad \text{falls } n = 0,$$

$$a^n = a * a^{n-1}, \quad \text{falls } n > 0$$

## Aufgabe 1: Potenzfunktion (iterativ): Beispiel $5^3$

```
public static int potiter (int a, int n) {
    int akk = 1;
    while (n > 0) {
        akk = a * akk;
        n = n - 1;
    }
    return akk;
}
```

Aufruf:  $a=5, n=3$       `potiterativ(5, 3);`

1. Anweisung	<code>akk = 1,</code>	<code>n = 3</code>
1. Schleifendurchlauf	<code>akk = 5*1 = 5,</code>	<code>n = 2</code>
2. Schleifendurchlauf	<code>akk = 5*5 = 25,</code>	<code>n = 1</code>
3. Schleifendurchlauf	<code>akk = 5*25 = 125,</code>	<code>n = 0</code>

Schleifenabbruch!

## Vergleich von Rekursion und Iteration

### Welche Aussage ist richtig?

- Bei der Ausführung eines rekursiven Programms müssen möglicherweise viele Werte zwischengespeichert werden, da keine Zwischenergebnisse berechnet werden können.
- Mit Rekursion können mehr Probleme gelöst werden als mit Iteration (Schleifen).
- Ein rekursives Programm terminiert immer, ein iteratives Programm möglicherweise nicht.

## Aufgabe 2: Gewinnchance beim Lotto

Wie viele Möglichkeiten gibt es, 6 Zahlen aus gegebenen 49 Zahlen auszuwählen?

Entwickeln Sie einen rekursiven Algorithmus.



## Aufgabe 2: Lotto (klassische Berechnung)

Aus der Statistik ist bekannt, dass es  $\binom{n}{k}$  Teilmengen mit  $k$  Elementen aus einer Menge mit  $n$  Elementen gibt:

$\text{moeglichkeiten}(k, n) = \binom{n}{k}$  für  $1 \leq k \leq n$

Berechnung mit Binomialkoeffizient:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$\text{moeglichkeiten}(6, 49) = \binom{49}{6} = \frac{49!}{6!(49-6)!} = \frac{(49 * 48 * 47 * \dots * 1)}{(6 * 5 * \dots * 1) * (43 * 42 * \dots * 1)}$$

## Aufgabe 2: Lotto (rekursive Lösungsidee)

- Jede Auswahl „6 aus 48“ ist ebenfalls eine gültige Auswahl für „6 aus 49“  
(das sind alle Auswahlen, in denen 49 nicht vorkommt)
- Jede Auswahl „5 aus 48“ zusammen mit der festen sechsten Zahl 49 ist ebenfalls eine Auswahl für „6 aus 49“  
(das sind alle Auswahlen, in denen 49 vorkommt)
- a. + b. liefert alle Auswahlen für „6 aus 49“

Also:  $\text{moeglichkeiten}(6, 49) =$   
 $\text{moeglichkeiten}(6, 48) + \text{moeglichkeiten}(5, 48)$

## Aufgabe 2: Lotto (rekursive Lösungsidee)

Also:  $\text{moeglichkeiten}(6, 49) =$   
 $\text{moeglichkeiten}(6, 48) + \text{moeglichkeiten}(5, 48)$

Allgemein gilt für  $1 \leq k \leq n$ :

$\text{moeglichkeiten}(k, n) = n,$  falls  $k=1$

$\text{moeglichkeiten}(k, n) = 1,$  falls  $k=n$

$\text{moeglichkeiten}(k, n) =$

$\text{moeglichkeiten}(k, n-1) + \text{moeglichkeiten}(k-1, n-1),$  sonst

## Aufgabe 2: Lotto (rekursiv)

```
public static int moeglichkeiten(int k, int n) {
    if (k == 1) return n;
    else if (k == n) return 1;
    else return moeglichkeiten(k, n-1) + moeglichkeiten(k-1, n-1);
}
```

Allgemein gilt für  $1 \leq k \leq n$ :

$\text{moeglichkeiten}(k, n) = n,$  falls  $k=1$

$\text{moeglichkeiten}(k, n) = 1,$  falls  $k=n$

$\text{moeglichkeiten}(k, n) =$   
 $\text{moeglichkeiten}(k, n-1) + \text{moeglichkeiten}(k-1, n-1),$  sonst

## Aufgabe 2: Lotto (rekursiv)

Die Methode kann folgendermaßen aufgerufen werden:

```
public class Lotto {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        System.out.println(moeglichkeiten(6, 49));
        long ende = System.currentTimeMillis();
        System.out.println(ende - start);
    }

    public static int moeglichkeiten(int k, int n) {...}
}
```

Nach 11 Millisekunden berechnet das Programm das Ergebnis:

**13.983.816 Möglichkeiten**

## Aufgabe 2: Lotto (rekursiv)

### Welchen Nachteil hat diese rekursive Lösung?

```
public static int moeglichkeiten(int k, int n) { 1 ≤ k ≤ n
    if (k == 1) return n;
    else if (k == n) return 1;
    else return moeglichkeiten(k,n-1) + moeglichkeiten(k-1,n-1);
}
```

- a) Das Programm terminiert für manche erlaubte Werte nicht.
- b) Das Programm liefert für manche erlaubte Werte ein falsches Ergebnis.
- c) Es werden Werte doppelt berechnet.