252-0027
Einführung in die Programmierung

5. Rekursion

Manuela Fischer, Malte Schwerhoff

Departement Informatik
ETH Zürich

5. Rekursion

5.1 Definition und Call Stack

5.2 Lösungsstrategien

Rekursion in …

- … **EBNF:** Name kommt auf LHS und RHS vor

```
<number>  ←  ( 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ) [ <number> ]
```

- … **Mathematik:** Definition bezieht sich auf sich selbst

$$n! = \begin{cases} 1, & \text{if } n \leq 1 \\ n \cdot (n-1)!, & \text{otherwise} \end{cases}$$

Abbruch

- … **Java:** Methode ruft sich selbst auf

```java
public static void f() {
  f();
}
```

Google recursion

All Images Videos News Web Maps Books More

See results about  Recursion
Computer science

Did you mean: ***recursion***

Recursion **occurs when the definition of a concept or process depends on a simpler or previous version of itself**. Recursion is used in a variety of disciplines ranging from linguistics to logic.

Endliche Rekursion in Java

- Mit Abbruch: einen Weg durch Methode ohne rekursiven Aufruf
  - **Basisfall** («**base case**»): Argument, das zu Abbruch führt
  - **Rekursiver Fall**: Argument, das zu rekursivem Aufruf führt

```
public static void f(int x) {
  if (x <= 0) {
    ...
  } else {
    f(x - 1);
  }
}
```

Basisfall

Rekursiver Fall

Beispiel: Fakultät

$$n! = \begin{cases} 1, & \text{if } n \leq 1 \\ n \cdot (n-1)!, & \text{otherwise} \end{cases}$$

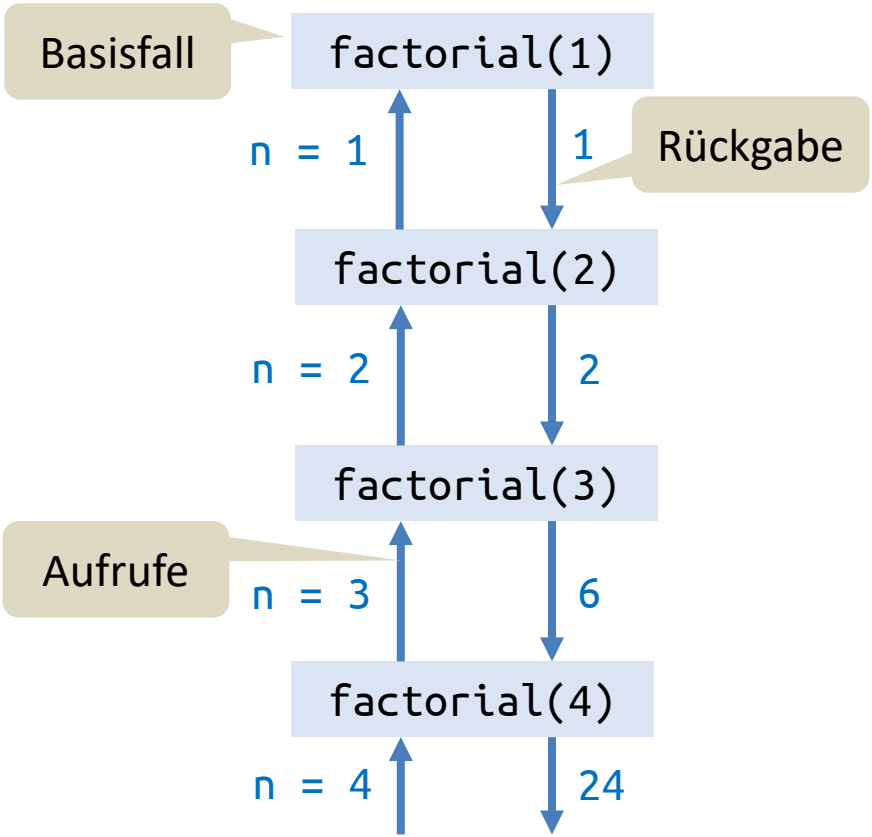
```
public static int factorial(int n) {
  if (n <= 1) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

Basisfall: $n \leq 1$

Rekursiver Fall: $n \geq 2$

Idee der Rekursion:
Lösung für n basiert auf Lösung für n-1.
Wenn ich Lösung für n-1 habe, habe ich auch Lösung für n.

Beispiel: Fakultät



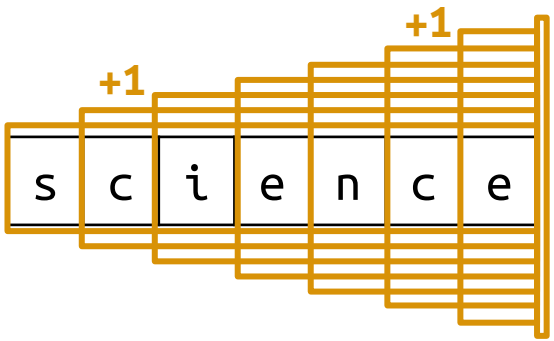

```java
public static int factorial(int n) {
  if (n <= 1) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

- **Aufrufstapel** («**call stack**»):
  - Stapel der Aufrufe
  - Oberster Aufruf wird abgearbeitet
  - Bei return wird Aufruf von Stapel entfernt und bei neu oberstem Aufruf weitergefahren (mit Rückgabewert)

Ein altes Beispiel: Zeichen zählen

Wenn ich Lösung für Teilproblem hätte, dann könnte ich…

- **Rekursive** Methode `count(char c, String s)`, die zählt wie oft ein Zeichen c in einem String s vorkommt




```
count('c', "science")
= count('c', "cience")
= 1 + count('c', "ience")
= 1 + count('c', "ence")
= 1 + count('c', "nce")
= 1 + count('c', "ce")
= 1 + 1 + count('c', "e")
= 1 + 1 + count('c', "")
= 1 + 1 + 0
= 2
```

+1 falls erstes Zeichen ein c

Ein altes Beispiel: Zeichen zählen

- **Rekursive** Methode `count(char c, String s)`, die zählt wie oft ein Zeichen c in einem String s vorkommt

```
public static int count(char c, String s) {
  if (s.length() == 0) {
    return 0;
  } else if (s.charAt(0) == c) {
    return 1 + count(c, s.substring(1));
  } else {
    return count(c, s.substring(1));
  }
}
```

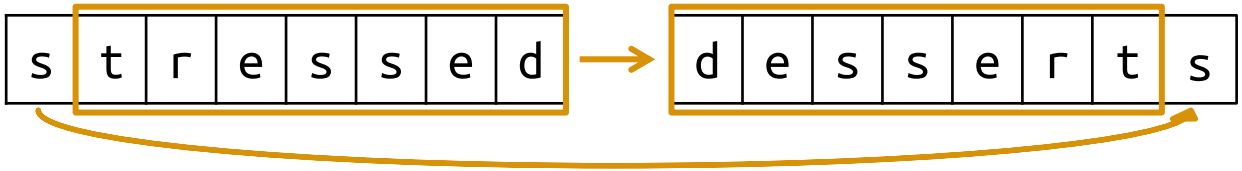
Basisfall

Gesuchtes Zeichen an erster Stelle

Beispiel: String umkehren

Wenn ich Lösung für Teilproblem hätte, dann könnte ich…

- **Rekursive** Methode `reverse(String s)`, die `s` umkehrt




```
reverse("stressed") = reverse("tressed") + 's'
= reverse("ressed") + 't' + 's'
= reverse("essed") + 'r' + 't' + 's'
= reverse("ssed") + 'e' + 'r' + 't' + 's'
= reverse("sed") + 's' + 'e' + 'r' + 't' + 's'
= reverse("ed") + 's' + 's' + 'e' + 'r' + 't' + 's'
= reverse("d") + 'e' + 's' + 's' + 'e' + 'r' + 't' + 's'
= "d" + 'e' + 's' + 's' + 'e' + 'r' + 't' + 's'
```

Beispiel: String umkehren

- **Rekursive** Methode `reverse(String s)`, die `s` umkehrt

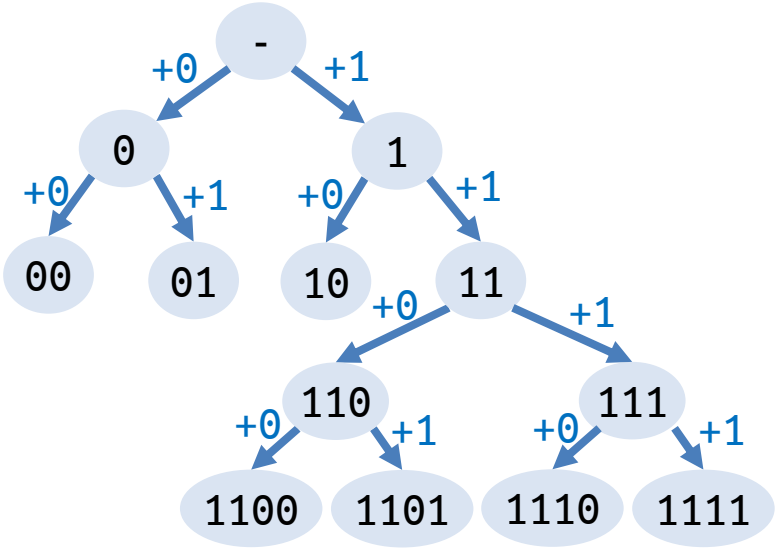
```
public static String reverse(String s) {
  if (s.length() <= 1) {
    return s;
  } else {
    return reverse(s.substring(1)) + s.charAt(0);
  }
}
```

Wenn ich weiss, wie ich Substring ohne 1. Zeichen umkehren kann, kann ich auch ganzen String umkehren!

Beispiel: Bitstrings enumerieren

- **Rekursive** Methode, die alle Bitstrings der Länge n ausgibt

```
0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111,
1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
```



Beispiel: Bitstrings enumerieren

- **Rekursive** Methode `bitstrings(String prefix, int n)`, die alle Bitstrings der Länge `n` mit Präfix `prefix` ausgibt

```
public static void bitstrings(String prefix, int n) {
  if (prefix.length() == n) {
    System.out.println(prefix);
  } else {
    bitstrings(prefix + '0', n);
    bitstrings(prefix + '1', n);
  }
}
```

Basisfall

2 rekursive Aufrufe

5. Rekursion

5.1 Definition und Call Stack

5.2 Lösungsstrategien

Lösungsstrategien

- **Iterativ (direkt lösen!)** `for i = 1 ... n: solve(i)`
  - Das Problem wird in ganz viele kleine gleich grosse Teilprobleme zerlegt
  - Teilprobleme werden nacheinander in einer Schleife gelöst
- **Rekursiv (delegieren!)**
  - Decrease and Conquer `recSolve(n-1)`
    - Problem wird jeweils in **ein** etwas **kleineres Teilproblem** umgeformt und delegiert
    - Teilprobleme werden kleiner und kleiner, bis zum Basisfall (direkte Lösung)
  - Divide and Conquer `recSolve(n/2) and recSolve(n/2)`
    - Problem wird in **zwei gleich grosse Teilprobleme** zerlegt und dann delegiert
    - Teilprobleme werden kleiner und kleiner, bis zum Basisfall (direkte Lösung)

Lösungsstrategien: Beispiel Crowdfunding

Crowdfunding 1’000’000 CHF mit 1 CHF pro Person

- **Iterativ:**

Iterativ: Der Reihe nach…

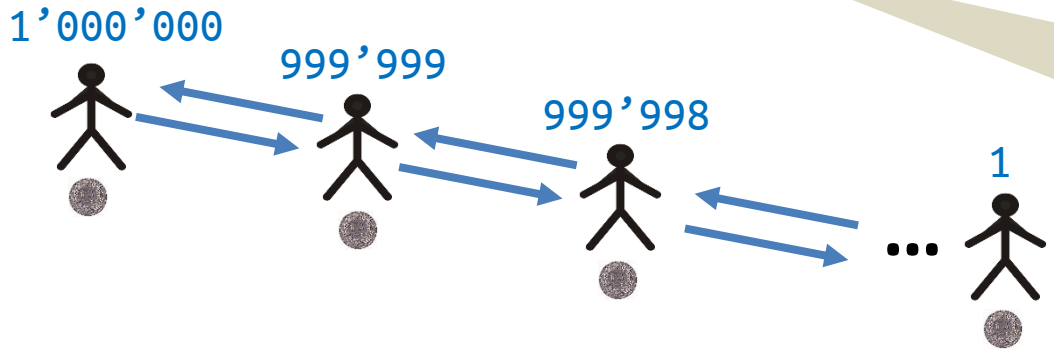
```
For i = 1 ... 1’000’000: Frage Person i nach 1 CHF
```

Lösungsstrategien: Beispiel Crowdfunding

Crowdfunding 1’000’000 CHF mit 1 CHF pro Person

- **Rekursiv mit Decrease and Conquer:**
  Crowdfunding von x CHF:

```
Falls Betrag x nur 1 CHF ist: zahle selbst!
Sonst zahle 1 CHF und frage Person nach x-1 CHF (delegieren!).
```



Decrease and Conquer: Delegiere den um 1 kleineren Betrag an andere Person.

Lösungsstrategien: Beispiel Crowdfunding

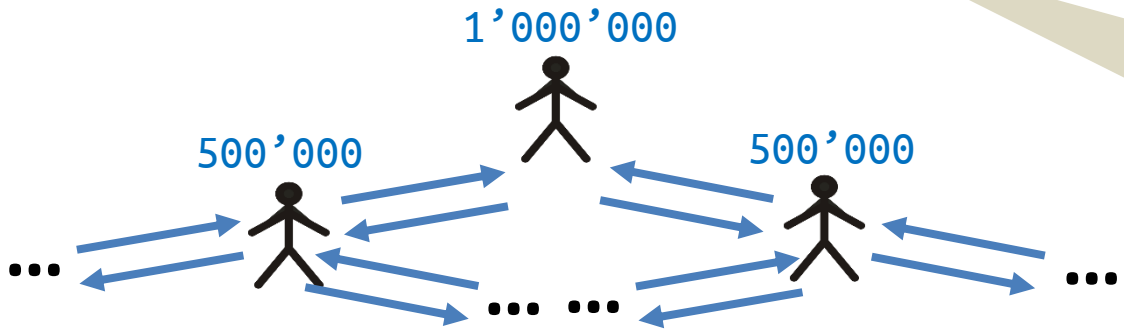
Crowdfunding 1’000’000 CHF mit 1 CHF pro Person

- **Rekursiv mit Divide and Conquer:**
  Crowdfunding von x CHF:

```
Falls Betrag x nur 1 CHF ist: zahle selbst!
Sonst frage 2 Personen nach je x/2 CHF (delegieren!)
```

$\lfloor \frac{x}{2} \rfloor$ und $\lceil \frac{x}{2} \rceil$ falls x ungerade.

Divide and Conquer: Delegiere die Hälfte des Betrags je an zwei Personen.



Potenzberechnung auf 3 Arten!

Methode `power(int b, int p)`, die die p-te Potenz von b zurückgibt.

- **Iterativ (direkt lösen!)**

```
public static int power(int b, int p) {
  int result = 1;
  for (int i = 1; i <= p; i++) {
    result *= b;
  }
  return result;
}
```

Potenzberechnung auf 3 Arten!

Methode `power(int b, int p)`, die die p-te Potenz von b zurückgibt.

- **Rekursiv mit Decrease and Conquer (delegieren!)**

```
public static int power(int b, int p) {
  if (p == 0) {
    return 1;
  } else {
    return b * power(b, p-1);
  }
}
```

Potenzberechnung auf 3 Arten!

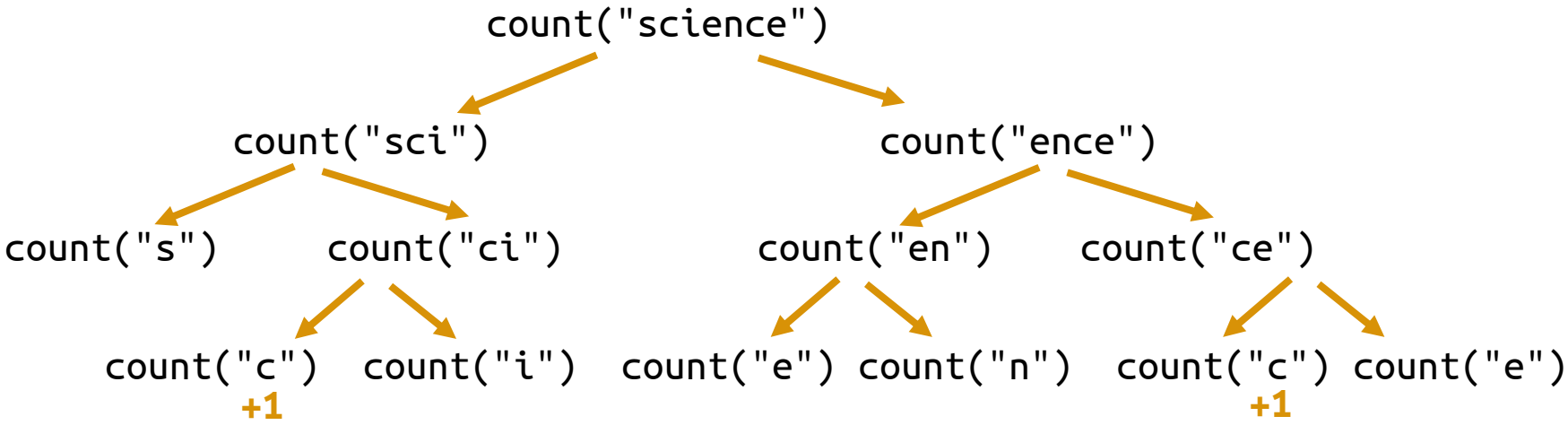
Methode `power(int b, int p)`, die die p-te Potenz von b zurückgibt.

- **Rekursiv mit Divide and Conquer (delegieren!)**

```
public static int power(int b, int p) {
  if (p == 0) {
    return 1;
  } else if (p == 1) {
    return b;
  }
  else {
    return power(b, p/2) * power(b, p - p/2);
  }
}
```

Es geht effizienter, wenn man Teilprobleme wiederverwendet. Mehr dazu in A&D!

Das alte Beispiel nochmals: Zeichen zählen



Das alte Beispiel nochmals: Zeichen zählen

```
public static int count(char c, String s) {
  if (s.length() == 0) {
    return 0;
  } else if (s.length() == 1) {
    return s.charAt(0) == c ? 1 : 0;
  } else {
    return count(c, s.substring(0, s.length()/2))
           + count(c, s.substring(s.length()/2));
  }
}
```

Basisfälle

2 rekursive Aufrufe

Iteration versus Rekursion

Bei schwierigeren Problemen
(insbesondere mit mehreren rekursiven Aufrufen):

- Rekursive Methoden sind oft einfacher!
  - kürzer und intuitiver
  - haben einfacheren Kontrollfluss
- Aufrufstapel bei Rekursion braucht viel Platz!
  - Kann zu Stack Overflow führen!
- Rekursion löst unter Umständen Teilprobleme wiederholt!
  - Ineffizient!

Best from both worlds:
Dynamische Programmierung
(siehe A&D)