Rekursion

Sven Eric Panitz

Methoden, die sich selbst aufrufen, werden als rekursiv bezeichnet.

Einfachstes Beispiel:

```
int bottom(){
   return bottom();
}
```

```
jshell> int bottom(){return bottom();}
|  created method bottom()

jshell> bottom()
|  Exception java.lang.StackOverflowError
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
|        at bottom (#1:1)
```

Problem

- Das Programm läuft nicht endlos
- Die Ausführung bricht nach einiger Zeit mit einem Fehler ab

Terminierung

```
int nMal(      ){
        return                  nMal(    );
}
```

Terminierung

```
int nMal(int n){
        return                    nMal(n-1);
}
```

Terminierung

```
int nMal(int n){
        return n<=0 ? 1 : nMal(n-1);
}
```

Abhängig von einem Parameter muss es einen terminierenden Fall geben, in dem die Funktion nicht rekursiv aufgerufen wird.

- Basisfall
- Rekursionsanker
- Terminierender Fall
- Rekursionsende

Implementierung rekursiver Funktionen

$$fac(n) = \begin{cases} 1 & \text{für } n \leq 0 \\ n * fac(n-1) & \text{für } n > 0 \end{cases}$$

```
long fac(long n){
  return n<=0 ? 1 : n*fac(n-1);
}
```

Evaluation Aufrufe rekursiver Methoden

fac(5)
→ 5 * fac(5-1)
→ 5 * (4 * fac(4-1))
→ 5 * (4 * (3 * fac(3-1)))
→ 5 * (4 * (3 * (2 * fac(2-1))))
→ 5 * (4 * (3 * (2 * (1 * fac(1-1)))))
→ 5 * (4 * (3 * (2 * (1 * 1))))
→ 5 * (4 * (3 * (2 * 1)))
→ 5 * (4 * (3 * 2))
→ 5 * (4 * 6)
→ 5 * 24
→ 120

Wachsender Stack

5 * (4 * (3 * (2 * f(1))))

- Nach dem Aufruf von f(1), sind noch die Aufrufe von f(2), f(3), f(4), f(5) zu beenden.
- Diese Information muss Java während der Laufzeit auch speichern.
- Hierzu wird der Stack verwendet.
- Bei zu tiefen Rekursionen kann der Speicherbereich des Stacks nicht reichen:
- Es kommt dann zum Programmabbruch.

Baumrekursion

Wenn zur Berechnung mehrere rekursive Aufrufe notwendig sind,
Spricht man von einer Baumrekursion.

```
long prod(int from, int to){
  var middle = (from+to)/2;
  return from==to ? from
         : prod(from,middle)
           * prod(middle+1,to);
}
```

Reduktion einer Baumrekursion

prod(1,10)

→ prod(1,5)*prod(6,10)

→ (prod(1,3)*prod(4,5)) * (prod(6,8) * prod(9,10))

→ ((prod(1,2)*prod(3,3))*(prod(4,4)*prod(5,5)))
  * ((prod(6,7)*prod(8,8)) * (prod(9,9)*prod(10,10)))

→ (((prod(1,1)*prod(2,2))*prod(3,3))*(prod(4,4)*prod(5,5)))
  * (((prod(6,6)*prod(7,7))*prod(8,8)) * (prod(9,9)*prod(10,10)))

→ (((1*2)*3)*(4*5)*((6*7))*8) *(9*10)

→ ((2*3)*20)*((42*8)*90)

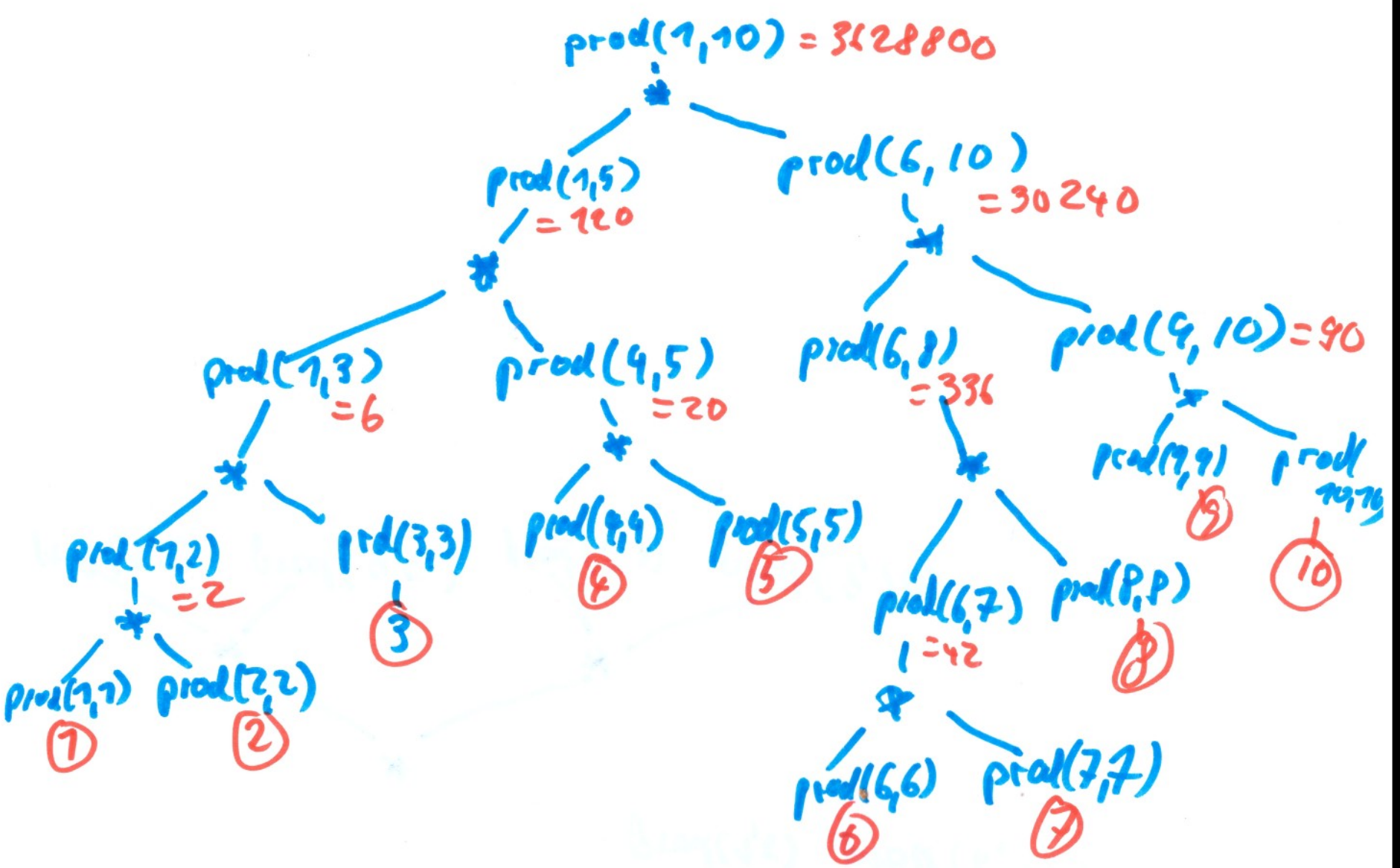
→ (6*20)*(336*90)

→ 120*30240

→ 3628800

Baumrekursion

- Bei einer Baumrekursion erhält man eine baumartige Berechnungsstruktur
- Die mehreren rekursiven Aufrufe sind können als Kinder des ursprünglichen Funktionsaufrufs notiert werden


prod(1,10) = 3628800
prod(1,5) = 120
prod(6,10) = 30240
prod(1,3) = 6
prod(4,5) = 20
prod(6,8) = 336
prod(9,10) = 90
prod(1,2) = 2
prd(3,3)
prod(4,4)
prod(5,5)
prod(7,9)
prod
prod(1,1)
prod(2,2)
prod(6,7) = 42
prod(8,8)
prod(6,6)
prod(7,7)
1
2
3
4
5
6
7
8
9
10

Teile und Herrsche

- Mit Baumrekursionen lassen sich Teile-und-Herrsche-Algorithmen realisieren
- Halbiere die zu lösende Aufgabe in zwei gleich schwierige Unteraufgaben, die rekursiv gelöst werden
- Baumrekursion lässt sich direkt parallelisieren

Endrekursion (iterative Rekursion)

- Wenn im rekursiven Fall nur direkt der rekursive Aufruf folgt, wird das als Endrekursion oder auch iterative Rekursion bezeichnet.

- ```
  long log2(long result, long n){
    return n<=1 ? result : log2(result+1, n/2);
  }
  ```
  ```

Reduktion einer Endrekursion

- log2(0, 1036)
  → log2(1, 518)
  → log2(2, 259)
  → log2(3, 129)
  → log2(4, 64)
  → log2(5, 32)
  → log2(6, 16)
  → log2(7, 8)
  → log2(8, 4)
  → log2(9, 2)
  → log2(10, 1)
  → 10

Von Endrekursion zur Schleife

```
long log2(long result,long n){
  while (!n<=1){
    result = result+1;
    n = n/2;
  }
  return result;
}
```

//solange nicht terminierender Fall

//setze die Argumente für
//den rekursiven Aufruf

//return mit terminierenden Fall

Von Endrekursion zur Schleife

```
ResultType f(Parameter){
   return  Terminierungsbedingung
         ? Basisfall
         : f(neueParameter);
}
```

Von Endrekursion zur Schleife

```
ResultType f(Parameter){
   while (! Terminierungsbedingung){
      Parameter für
      Rekursion setzen
   }
   return Basisfall;
}
```

Von Endrekursion zur Schleife

- Die meisten Programmiersprachen optimieren eine Endrekursion zu einer Schleife
- Dieses wird als Endrekursionsoptimierung bezeichnet (eng. Tail call optimization)
- Damit kommt es nicht mehr zum StackOverflowError
- Java führt diese Optimierung nicht durch!

Zusammenfassung

- Funktionen, die sich selbst aufrufen heißen rekursiv.
- Damit ermöglicht Rekursion mehrfach wiederholte Ausführung von Programmcode.
- Java kann nicht beliebig tiefe Rekursionen aufrufen.
- Es kann passieren, dass der Speicher des Stack nicht ausreicht.
- Baumrekursionen lassen sich direkt parallelisieren
- Endrekursionen können direkt zu Schleifen umgeschrieben werden.