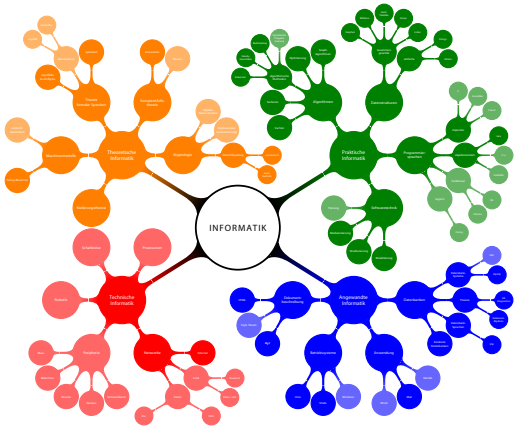
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 19

Einführung zur Rekursion

Kasparov versus Deep Blue

Vorlesung Einführung in die Informatik 1 vom 9. Januar 2014 von Till Tantau



Lernziele von Kapitel 19

1. Das Konzept der Rekursion verstehen
2. Rekursive Methoden implementieren können für einfache Probleme

Gliederung von Kapitel 19

Kasparov versus Deep Blue.

Copyright by IBM, free for non-commercial use

Unknown author, public domain

- Im Jahr 1997 hat Deep Blue gegen Kasparov im Schach gewonnen.
- Wie ging das?

Was ist Rekursion?

- *Rekursion* in der Programmierung stammt ursprünglich aus der Mathematik.
- Die Idee ist, ein Problem *»auf sich selbst«* zurückzuführen.
- Wichtig ist aber, dass man bei der Rückführung
  1. *einfacher* wird und
  2. es eine *Abbruchbedingung* gibt.

Ein praktisches Beispiel: Fahrtplanung.

Problemstellung

Gesucht: Zugverbindung von Simbach nach Lübeck.

Da es einen Regionalexpress von Hamburg nach Lübeck gibt, *reduziert* sich das Problem auf ein *einfacheres*:

Einfachere Problemstellung

Gesucht: Zugverbindung von Simbach nach Hamburg.

Da ein ICE von München nach Hamburg fährt, *reduziert* sich das Problem noch weiter:

Noch einfachere Problemstellung

Gesucht: Zugverbindung von Simbach nach München.

Hier fährt direkt ein Zug.

Ein mathematisches Beispiel: Rekursive Berechnung einer Summe.

Problemstellung

Gesucht: Summe der ersten 42 Zahlen.

Diese Summe erhalten wir, wenn wir auf die Summe der ersten 41 Zahlen 42 addieren. Also *reduziert* sich das Problem auf:

Einfachere Problemstellung

Gesucht: Summe der ersten 41 Zahlen.

Diese Summe erhalten wir, wenn wir auf die Summe der ersten 40 Zahlen 41 addieren. Also *reduziert* sich das Problem auf:

Noch einfachere Problemstellung

Gesucht: Summe der ersten 40 Zahlen.

Und so weiter, bis die Summe der ersten wenigen Zahlen (zum Beispiel der ersten drei Zahlen) bekannt ist.

Rekursion in Java.

- Eine Methode darf auch sich selbst aufrufen.
- Es gibt keine spezielle Syntax hierfür.
- Jeder rekursive Aufruf erzeugt einen neuen Scope.

Rekursive Berechnung der Summe in Java.


```
static int sum(int n)
{
  if (n == 1) {           // Abbruchbedingung
    return 1;
  }
  else {
    return sum(n-1) + n;
      // Rückführung auf ein einfacheres Problem
  }
}
```

Jede Rekursion hat zwei Elemente:

1. *Die Abbruchbedingung.*
2. Die Rückführung auf ein einfacheres Problem.

Rekursive Berechnung der Summe in Java.


```
static int sum(int n)
{
  if (n == 1) {            // Abbruchbedingung
    return 1;
  }
  else {
    return sum(n-1) + n;
      // Rückführung auf ein einfacheres Problem
  }
}
```

Jede Rekursion hat zwei Elemente:

1. Die Abbruchbedingung.
2. *Die Rückführung auf ein einfacheres Problem.*

Rekursive Berechnung der Fakultät.

Die *Fakultät* einer Zahl *n* ist das Produkt der ersten *n* Zahlen.

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

Vergleich von Rekursion und Iteration.

Rekursive Lösung:

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

Iterative Lösung:

```
static int fac(int n)
{
  int product = 1;
  for (int i = 1; i <= n;
       i++) {
    product = product * i;
  }
  return product;
}
```

- Es gab (gibt?) einen *Glaubenskrieg*, ob Rekursion oder Iteration besser ist.
- *Hier* ist Iteration besser, da schneller und eventuell klarer.
- In *komplexeren* Situationen ist aber Rekursion einfacher und natürlicher.

5-Minuten-Aufgabe

Zur Übung

Geben Sie eine rekursive und eine iterative Methode an, die die Summe der ersten *n* Quadratzahlen berechnet.

5-Minuten-Aufgabe

Zur Übung

Die Fibbonacci-Folge beginnt mit zweimal 1. Die nächste Zahl in der Folge ist immer die Summe der beiden vorherigen:
1, 1, 2, 3, 5, 8, 13, 21, . . .
Geben Sie den Code einer rekursiven Methode an, die die *n*-te Fibbonacci-Zahl berechnet.

Steht der Weltuntergang unmittelbar bevor?

Author Marubatsu, public domain

Die Legende über die Türme von Hanoi besagt, dass sich die Mönche eines Klosters in Hanoi folgender Aufgabe widmen:

Sie bewegen 100 goldene Scheiben vom ersten von drei Stäben auf den dritten. Die Scheiben haben alle unterschiedliche Größen und es liegen immer kleinere auf größeren.

Wenn die Mönche die hundert Scheiben auf den dritten Stab geschichtet haben, so wird die Welt enden.

Ein Lösungsalgorithmus für das Türme-von-Hanoi-Problem

- Eingabe: Scheibenanzahl, Start-Turm, Ziel-Turm.
- Ausgabe: Folge der Verschiebungen, um die gewünschte Anzahl Scheiben vom Start-Turm zum Ziel-Turm zu bewegen.

Algorithmus `calcMoves(int n, int from, int to)`

1. Wenn $n = 1$, so schiebe einfach die eine Scheibe von `from` nach `to`.
2. Ansonsten sei `extra` der verbleibende Turm (also weder `from` noch `to`).
3. Schiebe $n - 1$ Scheiben von `from` nach `extra`.
4. Schiebe eine Scheibe von `from` nach `to`.
5. Schiebe $n - 1$ Scheiben von `extra` nach `to`.

Der Hanoi-Algorithmus in Java


```
static void calculateMoves(int n, int from, int to)
{
  if (n == 1) {
    System.out.println ("Move␣top␣disk␣from␣" +
                        from + "␣to␣" + to + ".");
  }
  else {
    // Calculate other stack using evil trickery:
    int other_stack = 6 - from - to;

    calculateMoves(n-1, from, other_stack);

    System.out.println ("Move␣top␣disk␣from␣" +
                        from + "␣to␣" + to + ".");

    calculateMoves(n-1, other_stack, to);
  }
}
```

Binäre Suche mit Rekursion.


```
static int search (String[] strings,
                   String   value,
                   int      lower_bound,
                   int      upper_bound)
{
  if (lower_bound == upper_bound) {
    return lower_bound;
  }
  else {
    int mid_point = (lower_bound + upper_bound)/2;

    if (strings[mid_point].compare(value) < 0) {
      return
          search(strings,value,mid_point+1,upper_bound);
    }
    else {
      return search(strings,value,lower_bound,mid_point);
    }
  }
}
```

Zur Übung

Identifizieren Sie alle rekursiven Aufrufe und den Rekursionsabbruch.

Das Tic-Tac-Toe-Spiel.

Das Spiel

- Gespielt wird auf einem 3-mal-3-Feld.
- Gezogen wird abwechselnd, ein Spieler setzt Kreuze, der andere Kreise in die Felder.
- Falls einer der Spieler drei gleiche Symbole in einer Reihe erzeugt, gewinnt er.

Strategie für einen guten Zug

- Kann ich mit einem Zug gewinnen,
  so ist dies ein *guter Zug*.
- Sonst schaue ich für alle möglichen Züge, bei welchem *dem andere Spieler* nur schlechte Züge bleiben,
- und nehme diesen Zug.

Tic-Tac-Toe-Algorithmus

Algorithmus `findOptimalComputerMove`

1. Falls ein Zug zum sofortigen Gewinn führt, gib diesen zurück.
2. Sonst tue folgendes:
   - 2.1 Berechne für jeden möglichen Computerzug den optimalen Zug des Menschen.
   - 2.2 Gib den Zug zurück, bei dem der optimale Zug des Menschen am schlechtesten ist.

Algorithmus `findOptimalHumanMove`

1. Falls ein Zug zum sofortigen Gewinn führt, gib diesen zurück.
2. Sonst tue folgendes:
   - 2.1 Berechne für jeden möglichen Menschenzug den optimalen Zug des Computers.
   - 2.2 Gib den Zug zurück, bei dem der optimale Zug des Computers am schlechtesten ist.

Wie funktionieren Schachprogramme?

- Man kann Schachprogramme im Prinzip genauso wie Tic-Tac-Toe programmieren.
- Dann würde die Berechnung aber viel zu lange dauern (viele, viele Mal länger, als das Universum alt ist; auf einem Computer, der so groß wie das Universum ist).
- Deshalb wird die Rekursion nach einer gewissen Anzahl Schritte abgebrochen.
- Und dann braucht man noch viele, viele Tricks.

Zusammenfassung

Rekursion

Rekursion ist eine Programmiermethode, bei der Probleme gelöst werden, indem jede Eingabe auf die Lösung zu einer *einfacheren Eingabe* zurückgeführt wird.

Typischer Aufbau einer Rekursion in Java

```
static ... rekursiveMethode (int n, ...)
{
  if (n <= 1) {
    return ...;
  } else {
    ...;
    ... rekursiveMethode(n-1, ...) ...;
    ...;
    return ...;
  }
}
```