Rekursion I

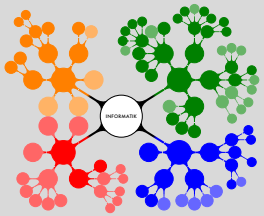
Kasparov versus Deep Blue

Till Tantau

Institut für Theoretische Informatik
Universität zu Lübeck

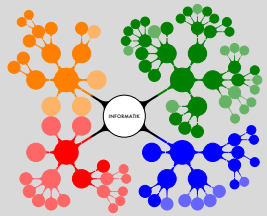
18. Vorlesung zu Informatik A für MLS

14. Dezember 2006

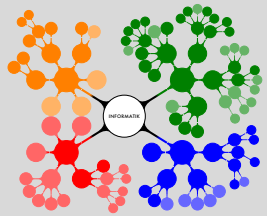


Die Lernziele der heutigen Vorlesung und der Übungen.

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



Gliederung

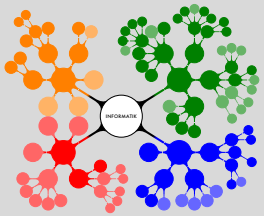


Kasparov versus Deep Blue.

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?



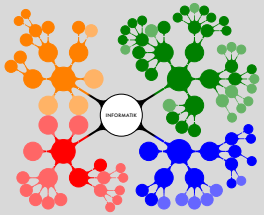
Steht der Weltuntergang unmittelbar bevor?

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

Sie bewegend 100 goldene Scheiben vom ersten von drei Stäben auf der 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.

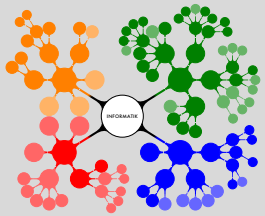
Author Marubatsu, public domain



5-Minuten-Aufgabe

Übungsaufgabe

Lösen Sie das Türme-von-Hanoi-Problem für drei Scheiben.

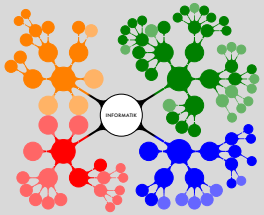


Ein Lösungsalgorithmus für das Türme-von-Hanoi-Problem

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

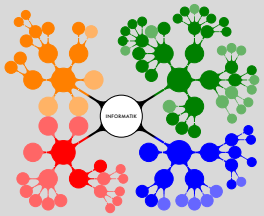
Algorithmus `calcMoves(int n, int from, int to)`

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`.



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.

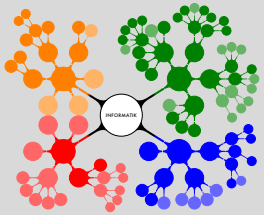


Die Rekursion im Lösungsalgorithmus für das Türme-von-Hanoi-Problem

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

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`.

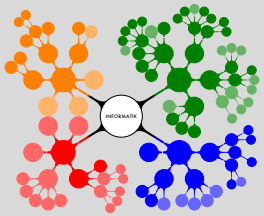


Die Rekursion im Lösungsalgorithmus für das Türme-von-Hanoi-Problem

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

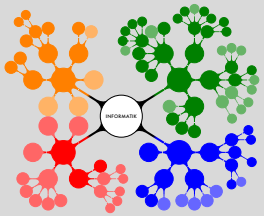
Algorithmus `calcMoves(int n, int from, int to)`

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`.



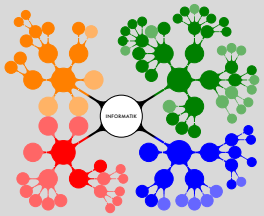
Rekursion in Java.

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



Der Hanoi-Algorithmus in Java

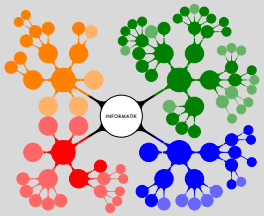
```
static void calculateMoves(int n, int from, int to)
{
  if (n == 1)
    System.out.println ("Move␣top␣disk␣from␣" +
                        from + "␣to␣" + to + ".");
  else
  {
    // Calculate other stack using evil trickery:
    int other_stack = 6 - from - to;

    calculateMoves(n-1, from, other_stack);

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

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

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.

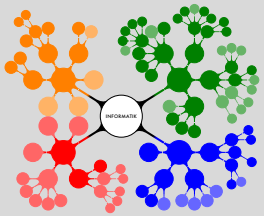
Rekursive Lösung:

```
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;
}
```

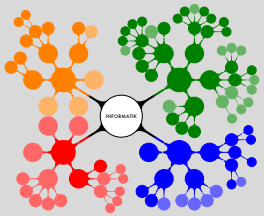
- Es gab (gibt?) einen Glaubenskrieg, ob Rekursion oder Iteration besser ist.
- Hier ist Iteration besser, da schneller und eventuell klarer.
- In komplexeren Situationen ist aber Rekursion einfacher und natürlicher.



10-Minuten-Aufgabe

Übungsaufgabe

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



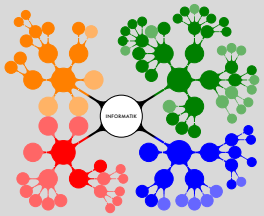
10-Minuten-Aufgabe

Entschlüsseln Sie den Da-Vinci-Code...

Die Fibbonacci-Folge beginnt mit zweimal 1. Die nächste Zahl in der Folge ist immer die Summe der beiden vorherigen:

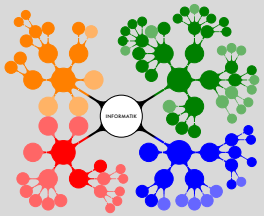
1, 1, 2, 3, 5, 8, 13, 21, ...

Geben Sie den Code einer rekursiven Methode an, die die *n*-te Fibbonacci-Zahl berechnet.



Binäre Suche mit Rekursion.

```
static int search (String[] strings,
                   String   value,
                   int      lower_bound,
                   int      upper_bound)
{
  if (lower_bound == upper_bound)
    // Recursion break
    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);
  }
}
```

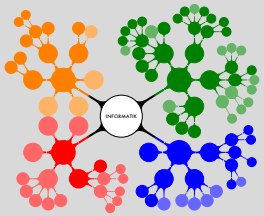
Das Tic-Tac-Toe-Spiel.

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

1. Kann ich mit einem Zug gewinnen, so ist dies ein guter Zug.
2. Sonst schaue ich für alle möglichen Züge, bei welchem dem andere Spieler nur schlechte Züge bleiben.
3. Diesen Zug nehme ich.



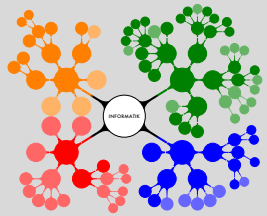
Tic-Tac-Toe-Algorithmus

Algorithmus `findOptimalComputerMove`

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

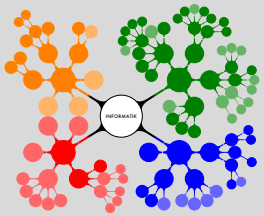
Algorithmus `findOptimalHumanMove`

1. Falls ein Zug zum sofortigen Gewinn führt, gib diesen zurück.
2. Sonst tue folgendes:
   1. Berechne für jeden möglichen Menschenzug den optimalen Zug des Computers.
   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 ist eine Programmiermethode.
- Dabei wird ein Problem gelöst, indem es auf eine einfachere Instanz zurückgeführt wird.
- Eine Rekursion muss immer eine Abbruchbedingung haben.
- Wechselseitige Rekursion wird oft bei Spielen benutzt.

H. P. Gumm, M. Sommer.
Einführung in die Informatik, Oldenbourg Verlag, 2004.
Kapitel 2.8