

### Beispiel: Fibonacci-Zahlen

- Unendliche Reihe: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$F_n = \begin{cases} 0 & , n = 0 \\ 1 & , n = 1 \\ F_{n-1} + F_{n-2} & , \text{sonst} \end{cases}$$

Fibonacci-Kaninchen:

Number of pairs: 1, 1, 2, 3, 5

L. P. Fibonacci (1170 - 1250)

G. Zachmann Informatik 1 - WS 05/06 Rekursion 23

### Fibonacci Zahlen in der Natur

Pinienzapfen

Blumenkohl

Nach 2 Monaten Wachstum produziert ein Zweig jeden Monat eine Abzweigung

G. Zachmann Informatik 1 - WS 05/06 Rekursion 24

### Mögliche Gefahren bei Rekursion

- Ist Rekursion schnell?
  - Ja. Bsp.: sehr effizientes Programm für ggT
  - Nein. Bsp.: Rekursive Berechnung von  $F_n$  ist extrem ineffizient!
- Das Programm:
 

```
def F(n):
    if n == 0 or n == 1:
        return n
    else:
        return F(n-1) + F(n-2)
```
- Beobachtung: es dauert sehr lange, um  $F(40)$  zu berechnen!

G. Zachmann Informatik 1 - WS 05/06 Rekursion 25

F(40)
   
F(39) F(38)
   
F(38) F(37) F(37) F(36)
   
F(37) F(36) F(36) F(35) F(36) F(35) F(35) F(34)

F(39) wird einmal berechnet.  
 F(38) wird 2-mal berechnet.  
 F(37) wird 3-mal berechnet.  
 F(36) wird 5-mal berechnet.  
 F(35) wird 8-mal berechnet.  
 ...  
 F(0) wird 165,580,141-mal berechnet.  
 331,160,281 rekursive Funktionsaufrufe

G. Zachmann Informatik 1 - WS 05/06 Rekursion 26

### Lösungen für ineffiziente Rekursionen

- Allgemeine Lösung, um nicht wiederholt Zwischenergebnisse zu berechnen:
  - Speichere Zwischenergebnisse in einer Tabelle
  - Schaue zuerst in dieser Tabelle nach, bevor Rekursion gemacht wird
- Spezielle Lösung:
  - Finde direkte Berechnungsvorschrift
  - Bsp. Fibonacci:
 
$$F_n = \frac{\phi^n - (1 - \phi)^n}{\sqrt{5}}$$

$$\phi = 1.61803398875\dots$$

$$\phi^2 = \phi + 1$$

G. Zachmann Informatik 1 - WS 05/06 Rekursion 27

### Beispiel: Modulare Exponentiation

- Gegeben positive ganze Zahlen  $n$ ,  $e$ , und  $m$ , berechne  $n^e \bmod m$ 
  - Bsp:  $11^{13} \bmod 53 = 52$
  - Ist der Verschlüsselungsschritt bei RSA
    - $n$  = Klartext, alle Zahlen im Bereich 1024 Bits
  - Teil des probabilistischen Miller-Rabin-Primzahltests (evtl. später)
- Einfache iterative Lösung:
 

```
def modexp(n, e, m):
    prod, i = 1, 0
    while i < e:
        prod *= n
        prod %= m
    return prod
```

  - Zwischenzahlen können 64 Bits sein
  - In jedem Durchlauf mod machen, um Überlauf zu vermeiden
- Aufwand:  $N$  Multiplikationen und mods.

G. Zachmann Informatik 1 - WS 05/06 Rekursion 28

### Rekursive Lösung:

$$a^N = \begin{cases} 1 & , N = 0 \\ a^{\frac{N}{2}} \times a^{\frac{N}{2}} & , N \text{ gerade} \\ a \times a^{\frac{N-1}{2}} \times a^{\frac{N-1}{2}} & , N \text{ ungerade} \end{cases}$$

Beispiel:  $17^{47} = 17 \times 17^{23} \times 17^{23}$   
Nur einmal berechnen

- Implementierung:
 

```
def modexp(n, e, m):
    if e == 0:
        return 1
    t = modexp(n, e/2, m)
    c = (t * t) % m
    if e % 2 == 1:
        c = (c * n) % m
    return c
```
- Aufwand:
  - max.  $2 \cdot \log n$
  - viele Mult./Mod.

G. Zachmann Informatik 1 - WS 05/06 Rekursion 29

### Weitere Gefahr bei Rekursion

- Resultat der Berechnung / des Algorithmus nicht offensichtlich
- Besonders "tückisches" Beispiel: McCarthys „91-Funktion“:
 
$$f(n) = \begin{cases} n-10, & \text{falls } n > 100 \\ f(f(n+11)), & \text{sonst} \end{cases}$$

$$f(100) = f(f(111)) = f(101) = 91$$

$$f(99) = f(f(110)) = f(100) = 91$$

$$f(98) = f(f(109)) = f(99) = 91$$

$$f(97) = f(f(108)) = f(98) = 91$$

$$\dots$$

$$f(91) = f(f(102)) = f(92) = 91$$

$$\dots$$

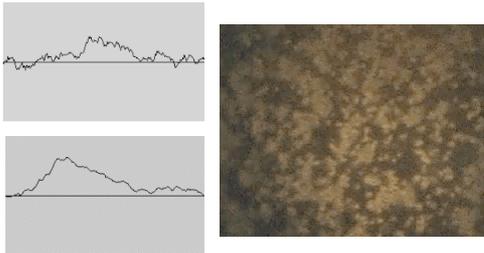
$$f(88) = f(f(99)) = f(91) = 91$$

$$\dots$$
 Also:  $f(n) = 91$  für alle  $n$

G. Zachmann Informatik 1 - WS 05/06 Rekursion 30

## Brown'sche Bewegung

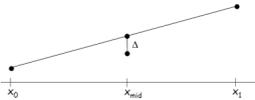
- Physikalischer Prozeß, der viele natürliche Phänome modelliert
  - Verteilung von Tinte in Wasser
  - Preis von Aktien
  - Formen der Berge und der Wolken (Fraktale)



G. Zachmann Informatik 1 - WS 05/06 Rekursion 31

## Simulation der Brown'schen Bewegung

- Random Midpoint Displacement-Methode
  - Führe ein Intervall mit Endpunkten  $(x_0, y_0)$  und  $(x_1, y_1)$
  - Starte mit zufälligen initialen  $(x_0, y_0)$  und  $(x_1, y_1)$
  - Wähle  $\Delta$  zufällig aus der Gaußschen Verteilung und proportional zur Länge des Segmentes  $(x_0, y_0), (x_1, y_1)$
  - Setze  $x_{mid} = \frac{1}{2}(x_0 + x_1)$  und  $y_{mid} = \frac{1}{2}(y_0 + y_1) + \Delta$
  - Wiederhole auf den linken und rechten Intervallen, bis Intervall "klein genug"

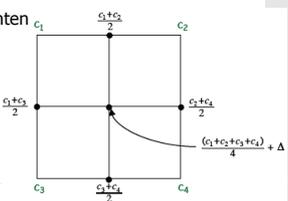


- A. Fournier, D. Fussell, and L. Carpenter: *Computer Rendering of Stochastic Models*. Communications of the ACM, 25:371-384, 1982.

G. Zachmann Informatik 1 - WS 05/06 Rekursion 32

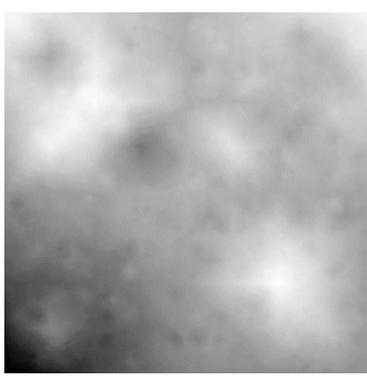
## Anwendung: Plasma-Wolke

- Starte mit Quadrat und zufälligen Intensitätswerten  $c_i$  an den Ecken
- Jede Ecke beschriftet mit Grayscale Wert
- Teile Quadrat in vier Quadranten
- Grayscale-Wert jeder neuen Ecke ist der Durchschnitt von anderen:
  - Mitte: Durchschnitt der vier Ecken + gelegentliche Versetzung  $\Delta$
  - andere: Durchschnitt von zwei originalen Ecken
- Wiederholen auf den vier Quadranten



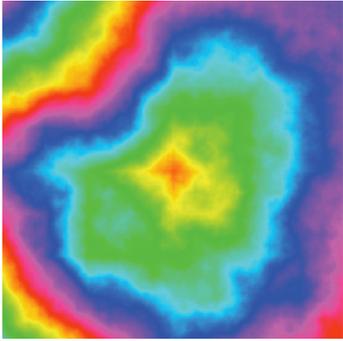
G. Zachmann Informatik 1 - WS 05/06

## Plasma-Wolke (Grayscale)



G. Zachmann Informatik 1 - WS 05/06 Rekursion 34

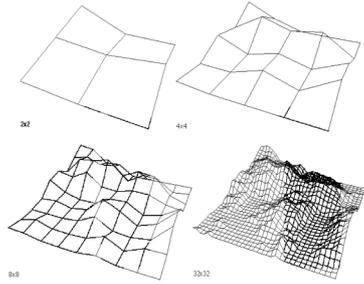
Plasma-Wolke (Farbe)



G. Zachmann Informatik 1 - WS 05/06 Rekursion 35

Fraktales Terrain

- Selbe Methode funktioniert auch, um Terrain zu generieren
- Interpretiere "Intensität" als "Höhe"



G. Zachmann Informatik 1 - WS 05/06 Rekursion 36

Mächtigkeit von Rekursion vs. Iteration

- Jede rekursive Funktion kann mit Iteration geschrieben werden
  - Compiler implementiert Funktionen mit einem Stack
  - Rekursion kann man durch Iteration + Stack ersetzen
- Jede iterative Funktion kann mit Rekursion geschrieben werden
  - Klar: verwende *Tail-Recursion*
- Effizienz: weder Iteration noch Rekursion sind *per se* schneller / langsamer
  - Auch der Overhead durch Funktionsaufruf spielt keine Rolle i.A.
- Soll man Iteration oder Rekursion verwenden?
  - Leichtigkeit und Klarheit der Implementierung
  - Time/space Effizienz

G. Zachmann Informatik 1 - WS 05/06 Rekursion 37

Rekursionsarten

- Lineare Rekursion**
  - Funktion ruft sich selbst **höchstens 1x** auf
  - Anzahl Aufrufe  $\leq$  Eingabegröße
  - Bsp.: rekursive Berechnung der Fakultät
  - Spezialfall *Tail Recursion*: Rekursion findet erst **am Ende** der Fkt statt
- Verzweigte Rekursion ("fat recursion")**
  - Funktion löst **zwei oder mehr** rekursive Aufrufe aus
  - Bsp.: rekursive Berechnung der Fibonacci-Zahlen
- Verschachtelte Rekursion ("compound recursion")**
  - Argument** für rekursiven Aufruf wird selbst durch rekursiven Aufruf bestimmt
  - Bsp.: 91-Funktion & Ackermann-Funktion
- Offene Rekursion (nicht-monotone Rekursion)**
  - Kontrollargument wird **nicht immer** in Richtung des Abbruchkriteriums verändert!
  - Folge: Der Rekursionsabbruch kann nicht sichergestellt werden
  - Bsp.: Ulam-Folge
- Wechselseitige Rekursion (indirekte Rekursion)**
  - Funktion *f* ruft Funktion *g* auf, diese ruft wieder *f* auf

G. Zachmann Informatik 1 - WS 05/06 Rekursion 38

## Die Ackermann-Funktion

- Definition:
 
$$A(0,n) = n+1$$

$$A(m+1, 0) = A(m,1)$$

$$A(m+1, n+1) = A(m, A(m+1,n))$$
- Z.B.
  - $A(0,0) = 1$
  - $A(0,1) = 2$
  - $A(1,1) = A(0, A(1,0)) = A(0, A(0,1)) = A(0,2) = 3$

G. Zachmann Informatik 1 - WS 05/06 Rekursion 39

## Wachstum der Ackermannfunktion :

m \ n	0	1	2	3	4	n
0	1	2	3	4	5	n + 1
1	2	3	4	5	6	n + 2
2	3	5	7	9	11	2n + 3
3	5	13	29	61	125	2 <sup>n+1</sup> · 3
4	13	65533	2 <sup>65533</sup> - 3	A <sub>3</sub> (2 <sup>65533</sup> - 3)	A <sub>3</sub> (A <sub>3</sub> , 3)	2 <sup>2<sup>n</sup> - 3</sup> 7! + 3 TWOS
5	65533	A <sub>4</sub> (65533)	A <sub>4</sub> (A <sub>4</sub> , 11)	A <sub>4</sub> (A <sub>4</sub> , 21)	A <sub>4</sub> (A <sub>4</sub> , 3)	
6	A <sub>5</sub> (1)	A <sub>5</sub> (A <sub>5</sub> , 1)	A <sub>5</sub> (A <sub>5</sub> , 11)	A <sub>5</sub> (A <sub>5</sub> , 21)	A <sub>5</sub> (A <sub>5</sub> , 3)	

- Verwendung
  - Testen der Effizienz von Prozeduraufrufmechanismen
  - Beschreibung der Komplexität mancher Algos (Inverse der Ackermann-Fkt.,  $\alpha$ , spielt dabei oft eine Rolle)
  - Sonst keinen weiteren praktischen Nutzen
  - Ähnlich: Takeuchi-Funktion

G. Zachmann Informatik 1 - WS 05/06 Rekursion 40

## Exkurs: Primitiv-rekursive Funktion

- können auf bestimmte Art aus einfachen Grundoperationen zusammengesetzt werden
- der Begriff **primitiv-rekursive Funktion** wurde von der ungarischen Mathematikerin **Rózsa Péter** geprägt
- zeichnen sich durch eine gewisse Gutartigkeit aus
- Insbesondere: vor der Berechnung eines Funktionswertes kann man angeben, wie komplex diese Operation ist, d.h. auch, wie lange diese Berechnung dauern wird

G. Zachmann Informatik 1 - WS 05/06 Rekursion 41

## Definition

- Die Klasse  $Pr_k$  der primitiv-rekursiven Funktionen von  $\mathbb{N}^k \rightarrow \mathbb{N}$  umfasst zunächst die folgenden prim.-rek. **Grundfunktionen**:
  - konstante Funktion**:  $f_c^k(n_1, \dots, n_k) := c \quad c \in \mathbb{N}$
  - Projektion auf ein Argument**:  $\pi_i^k(n_1, \dots, n_k) := n_i \quad 1 \leq i \leq k$
  - Nachfolgefunktion** (auch Sukzessorfunktion genannt):  $\nu(n) := n + 1$
- Aus diesen werden mit folgenden **Operationen** alle weiteren primitiv-rekursiven Funktionen gewonnen:
  - Komposition**:  $f(n_1, \dots, n_k) := g(h_1(n_1, \dots, n_k), \dots, h_m(n_1, \dots, n_k))$   
wobei  $g, h_1, \dots, h_m \in Pr$
  - Primitive Rekursion**:
 
$$f(0, n_2, \dots, n_k) := g(n_2, \dots, n_k) \quad (\text{Basisfall})$$

$$f(n_1 + 1, n_2, \dots, n_k) := h(f(n_1, \dots, n_k), n_1, \dots, n_k)$$

$$h, g \in Pr$$

G. Zachmann Informatik 1 - WS 05/06 Rekursion 42

## Taxonomie

- Alle primitiv-rekursiven Funktionen sind Turing-berechenbar
- Frage: Wir wissen, jede primitiv-rekursive Funktion ist total. Wäre prima, wenn die primitiv-rekursiven Funktionen gerade die Turing-berechenbaren Funktionen wären!
- Traurige Tatsache: Es gibt totale Turing-berechenbare Funktionen, die nicht primitiv-rekursiv sind.
  - Z.B.  $A(m, m)$
- Theorem (o. Bew.):
  - Für jede primitiv-rekursive Funktion  $f$  gibt es ein  $m$ , so daß  $f(m) < A(m, m)$ .
  - So kann  $A$  nicht primitiv-rekursiv sein.

G. Zachmann Informatik 1 - WS 05/06 Rekursion 43

## Wachstumsraten

Das Diagramm zeigt die relative Wachstumsrate von Funktionen in konzentrischen Ecken, die von innen nach außen folgende Funktionen umschließt:

- Polynomfunktion:**  $2n+5$  und  $n^3+3n^2+2n+1$
- Exponentialfunktion:**  $3^n$  und  $n^n$
- iterierte Exponenten:**  $n^{n^n}$
- Ackermanfunktion und Freunde:**  $A(m, n)$

G. Zachmann Informatik 1 - WS 05/06 Rekursion 44

## Zusammenfassung

- Wie schreibt man einfache rekursive Programme?
  - Basis-Fall, Reduktions-Schritt
  - Rekursionsbaum aufzeichnen
  - Sonstige Zeichnungen
- Warum lernen wir Rekursion?
  - Neue algorithmische Denkweise
  - Sehr wichtige Algorithmentechnik
- Viele Probleme haben elegante *divide-and conquer* Lösungen
  - Adaptive Quadratur
  - Sortieren (Quicksort, später)
  - Matrix-Multiplikation
  - Quad-tree für effiziente N-body Simulation

G. Zachmann Informatik 1 - WS 05/06 Rekursion 45