

Rekursive Definition der Fibonacci-Zahlen

$$f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2} \quad (n \geq 2)$$

Erste Werte

n	0	1	2	3	4	5	6	7	8	9	10	...	25	...
f_n	0	1	1	2	3	5	8	13	21	34	55	...	75025	...

Exakte Formel (de Moivre, 1718)

$$f_n = \frac{\varphi^n - \hat{\varphi}^n}{\sqrt{5}} \quad \text{mit} \quad \begin{cases} \varphi = \frac{1+\sqrt{5}}{2} \approx 1.61803 \dots \text{ "goldener Schnitt"} \\ \hat{\varphi} = \frac{1-\sqrt{5}}{2} \approx -0.61803 \dots \end{cases}$$

Bemerkung: der "goldene Schnitt" φ ist die positive Lösung der Gleichung:

$$x^2 = 1 + x$$

$\hat{\varphi} = 1 - \varphi$ ist die negative Lösung dieser Gleichung.

Die Folge $(f_{n+1}/f_n)_{n \geq 1}$ von Quotienten aufeinanderfolgender Fibonacci-Zahlen konvergiert und es gilt

$$\lim_{n \rightarrow \infty} \frac{f_{n+1}}{f_n} = \varphi$$

Abschätzung mittels goldenem Schnitt

$$\varphi^{n-2} \leq f_n \leq \varphi^{n-1}$$

Alternative Abschätzung (GA:Lemma 1.3)

$$2^{\lfloor \frac{n-1}{2} \rfloor} \leq f_n \leq 2^{n-2} \quad (n > 2)$$

andere interessante Eigenschaften der Fibonacci-Zahlen

- $\text{ggT}(f_m, f_n) = f_{\text{ggT}(m,n)}$
- $f_{n+1}f_{n-1} - f_n^2 = (-1)^n$
- $f_{n+m} = f_m f_{n+1} + f_{m-1} f_n$
- $\varphi^n = f_{n+1} - \hat{\varphi} \cdot f_n$, $\varphi^{n+1} = \varphi \cdot f_{n+1} + f_n$
- $\varphi^n + \hat{\varphi}^n = 2f_{n+1} - f_n$

Die Fibonacci-Zahlen treten in der Mathematik und Algorithmik (in der Kunst und in der Natur...) an vielen verschiedenen Stellen auf. Wichtig ist der Zusammenhang mit dem Algorithmus von Euklid.

Satz von Lamé (1844):

Sind $m, n \in \mathbb{N}$ mit $m, n \leq f_k$, so benötigt der Algorithmus von Euklid zur Berechnung von $\text{ggT}(m, n)$ maximal $k + 1$ Divisionsschritte.

Dieses Resultat kann man als die (vermutlich) historisch früheste quantitative Analyse des Laufzeitverhaltens eines Algorithmus ansehen.

Rekursive Berechnung der Fibonacci-Zahlen (gofer)

```
fib1 :: Int -> Int
fib1 1 = 1
fib1 2 = 1
fib1 n = fib1 (n-1) + fib1 (n-2)
```

$C_{rek}(n)$ = Anzahl der arithmetischen Operationen (Additionen, Subtraktionen, Multiplikationen, Divisionen) zur Berechnung von f_n mit dem Programm fib1

$$C_{rek}(1) = C_{rek}(2) = 0$$

$$C_{rek}(n) = 3 + C_{rek}(n-1) + C_{rek}(n-2) \quad (n \geq 3)$$

Tatsache (GA Lemma 1.1): Die Rekursion

$$D(1) = d$$

$$D(2) = d$$

$$D(n) = D(n - 1) + D(n - 2) \quad (n \geq 3)$$

hat die Lösung $D(n) = d \cdot f_n$ ($n \geq 3$).

Resultat (GA, Korollar 1.2):

$$C_{rek}(n) = 3(f_n - 1)$$

Zahlenbeispiel (GA): $n = 100 \Rightarrow C_{rek}(n) \approx 5 \cdot 10^{14}$

Iterative Berechnung der Fibonacci-Zahlen (gofer)

Idee: Iteration der Beziehung $(f_n, f_{n-1}) = (f_{n-1} + f_{n-2}, f_{n-1})$

```
fib2 n :: Int -> Int
fib2 n = x where (x,y) = g n

g :: Int -> (Int,Int)
g 1 = (1,undefined)
g 2 = (1,1)
g n = (x+y,x) where (x,y) = g (n-1)
```

$C_{iter}(n)$ = Anzahl der arithmetischen Operationen zur Berechnung von f_n mit dem Programm fib2

$$C_{iter}(1) = C_{iter}(2) = 0$$

$$C_{iter}(n) = 2 + C_{iter}(n - 1) \quad (n \geq 3)$$

Lösung der Rekursion (GA, Lemma 1.4): $C_{iter}(n) = 2(n - 2) \quad (n \geq 2)$

Schnelle Exponentiation - die Idee:

um für eine Zahl $a = 3$ und einen Exponenten $n = 155$ die Potenz 3^{155} zu berechnen, kann man

- 154 Multiplikationen mit dem Faktor 3 ausführen, oder
- die Binärdarstellung $(n)_2 = 10011011$ geschickt ausnutzen, indem man

– sukzessive

$3^2, 3^4 = (3^2)^2, 3^8 = (3^4)^2, \dots, 3^{128} = (3^{64})^2$ berechnet, sowie

$3^3 = 3 \times 3^2, 3^{11} = 3^3 \times 3^8, 3^{27} = 3^{11} \times 3^{16}, 3^{155} = 3^{27} \times 3^{128}$

(macht 7 Quadrierungen und 4 Multiplikationen)

– sukzessive

$3^2, 3^4 = (3^2)^2, 3^9 = 3 \times (3^4)^2, 3^{19} = 3 \times (3^9)^2, 3^{38} = (3^{19})^2,$

$3^{77} = 3 \times (3^{38})^2, 3^{155} = 3 \times (3^{77})^2$ berechnet (macht ebenfalls 7

Quadrierungen und 4 Multiplikationen)

Exponentiation mittels iteriertem Quadrieren:

Idee: in jeder Halbgruppe H kann man zur Berechnung von $(a, n) \mapsto a^n$ für $a \in H, n \in \mathbb{N}$ die Binärdarstellung des Exponenten n verwenden

$$a^n = \left\{ \begin{array}{ll} \left(a^{\frac{n}{2}}\right)^2 & \text{falls } n \text{ gerade} \\ a \cdot \left(a^{\frac{n-1}{2}}\right)^2 & \text{falls } n \text{ ungerade} \end{array} \right\} = a^{n \pmod{2}} \cdot \left(a^{n \div 2}\right)^2$$

Bezeichnungen für $n \in \mathbb{N}$:

- $(n)_2$: Binärdarstellung von n (ohne führende Nullen)
- $\ell(n)$: Länge der Binärdarstellung von $n = \lfloor \log(n) \rfloor + 1$
- $\#_1(n)_2$: Anzahl der Einsen in $(n)_2$

rekursives Programm zur Berechnung von $pot : (a, n) \mapsto a^n$ (gofer)

```
pot :: Int -> Int -> Int
pot a n | n == 1 = a
        | even n = p*p   where p = pot a (n/2)
        | odd  n = a*p*p where p = pot a (n/2)
```

Resultat (GA, Theorem 1.7): Die Berechnung von $pot(a, n)$ benötigt $\ell(n) + \#_1(n)_2 - 2$ Multiplikationen und $\ell(n) - 1$ Divisionen durch 2

Berechnung der Fibonacci-Zahlen mittels iteriertem Quadrieren

Idee: mit $F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ gilt

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} f_{n-1} + f_{n-2} \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = F \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix}$$

also ist (Induktion, GA, Lemma 1.5)

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = F^{n-2} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix} = F^{n-2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Folgerung (GA, Theorem 1.8):

Die Berechnung von f_n mittels iteriertem Quadrieren der Matrix F benötigt $13\lfloor \log(n-2) \rfloor + 12\#_1(n-2)_2 - 10$ arithmetische Operationen (Additionen, Subtraktionen, Multiplikationen, Divisionen durch 2)

Dies kann noch etwas verbessert werden, wenn man ausnützt, dass die Potenzen von F symmetrische Matrizen sind

```

h :: Int -> (Int,Int,Int,Int)
h n | n==1    = (1, 1,
                1, 0)
    | even n  = let (a,b,c,d)=h(n/2)
                in ((a*a)+(b*c), (a*b)+(b*d),
                    (c*a)+(d*c), (c*b)+(d*d))
    | odd  n   = let (a,b,c,d)=h(n/2)
                in ((a*a)+(b*c)+(a*b)+(b*d), (a*a)+(b*c),
                    (c*a)+(d*c)+(c*b)+(d*d), (c*a)+(d*c))

fib3 :: Int -> Int
fib3 n | n==1 || n==2 = 1
      | n>2           = a+c where (a,b,c,d) = h (n-2)

```

Zur Dauer der Berechnung von Fibonacci-Zahlen

Annahme: Dauer einer arithmetischen Operation = 1 Mikrosekunde

Variante/Zeitbedarf	1ms	1s	1m	1h
rekursiv	≈ 14	≈ 28	≈ 37	≈ 45
iterativ	≈ 500	$\approx 5 \cdot 10^5$	$\approx 3 \cdot 10^7$	$\approx 2 \cdot 10^9$
iteriertes Quadrieren	$\approx 10^{12}$	$\approx 10^{12.000}$	$\approx 10^{700.000}$	$\approx 10^{10^6}$

Da die beteiligten Operanden sehr schnell wachsen, ist obige Annahmen nicht realistisch!

Laufzeitvergleich für drei Methoden zur Berechnung der Fibonacci-Zahlen

(Quelle: Brassard, Bratley, *Algorithmics, Theory and Practice*, Prentice-Hall 1988, Sec. 1.7)

A: Berechnung modulo 7, gemessene Werte, Zeiten > 2 min geschätzt

n	10	20	30	50	10^2	10^4	10^6	10^8
fib1	8ms	1s	2m	21d	10^9 y	—	—	—
fib2	1/6ms	1/3ms	1/2ms	3/4ms	3/2ms	150ms	15s	25m
fib3	1/3ms	2/5ms	1/2ms	1/2ms	1/2ms	1ms	3/2ms	2ms

B: Laufzeitverhalten

$$\text{fib1 : } t_1(n) = \varphi^{n-20} \text{ sec}$$

$$\text{fib2 : } t_2(n) = 15 n \times 10^{-6} \text{ sec}$$

$$\text{fib3 : } t_3(n) = \frac{1}{4} \log n \times 10^{-3} \text{ sec}$$

C: gemessene Werte (in Sekunden) für exakte Berechnung (long integer)

n	5	10	15	20	25	100	500	10^3	$5 \cdot 10^3$	10^4
fib1	0.007	0.087	0.941	10.766	118.457	—	—	—	—	—
fib2	0.005	0.009	0.011	0.017	0.021	0.109	1.177	3.581	76.107	298.892
fib3	0.013	0.017	0.019	0.020	0.021	0.041	0.132	0.348	7.664	29.553