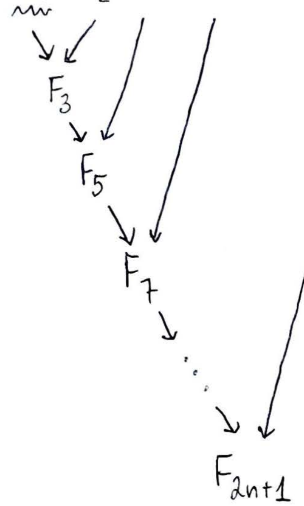
Liczby Fibbonacciego

$$F_{n+1} = F_n + F_{n-1}, \text{ gdzie } F_1 = F_2 = 1 \text{ (czasami } F_0 = 1)$$

(1) $F_2 + F_4 + F_6 + \ldots + F_{2n} = F_{2n+1} - 1 \quad /+1$

$$F_1 + F_2 + F_4 + F_6 + \ldots + F_{2n} = F_{2n+1}$$



(2) $F_{n+m} = F_{n-1} \cdot F_m + F_n \cdot F_{m+1}$

Dowód przez indukcję po m:

Podst: $m=0 \qquad F_{n+m} = F_n = F_{n-1} \cdot 0 + F_n \cdot 1 = F_n$

$(m=1 \qquad F_{n+1} = F_{n-1} + F_n)$

Krok: Załóżmy, że prawdziwe dla $m-1, m$. Pokażemy dla $m+1$

$$\begin{array}{rcl} F_{n+m-1} & = & F_{n-1} F_{m-1} + F_n F_m \\ + \; F_{n+m} & = & F_{n-1} F_m + F_n F_{m+1} \\ \hline F_{n+m+1} & = & F_{n-1} F_{m+1} + F_n F_{m+2} \end{array}$$

kolejne liczby Fibbonacciego

(3)

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right), \quad \left|\left(\frac{1-\sqrt{5}}{2}\right)^n\right| \leqslant 1$$

z tego wynika, że $F_n \sim \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$

Sortowanie przez scalanie

(1) Scalanie (posortowane ciągi a_n, b_m dają posortowany ciąg c_{n+m})

$$\underbrace{a_1 a_2 \ldots\ldots a_k}, \quad \underbrace{b_1 b_2 \ldots\ldots b_l} \rightarrow \underbrace{c_1 c_2 \ldots\ldots c_{k+l}}$$

Czas: $O(k+l)$

Max porównań: $k+l-1$

ostatni element na swoim miejscu

Mergesort (i,j) {sortuje elementy $a_i \ldots a_j$} $\quad a_1 \ldots a_n$

if $i=j$ then return (a_i)

$m \leftarrow \lfloor (i+j-1)/2 \rfloor$

return Merge (Mergesort (i,m), Mergesort $(m+1,j)$)

$T(n)$ - czas działania procedury Mergesort (uproszczony)

$T(n) = 2T(n/2) + c \cdot n$

$f(n)$ - max liczba porównań Mergesort

$f(n) = f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + n - 1$

Dowód $T(n)$:

zał. $n = 2^k$, $T(1) = d$ (jakaś stała)

$$\begin{aligned} T(n) &= cn + 2T(n/2) \\ &= cn + 2c(n/2) + 4T(n/4) \\ &= \underbrace{cn + cn + cn + \ldots + cn}_{k = \log n} + 2^k T(n/2^k) \\ &= cn \cdot \log n + n \cdot d \\ &= O(n \log n) \end{aligned}$$

Mnożenie liczb n-cyfrowych

$O(n^2) \to$ mnożenie pisemne

$O(n^{1.58}) \to$ algorytm Karacuby

$O(n \cdot \log n \cdot \log\log n) \to$ algorytm Schönhage - Strassena

Algorytm Karacuby (mnożenie dużych liczb)

$A \cdot B$ {A, B - liczby n-cyfrowe}

if $n < 32$ then return $A \cdot B$

Podziel A na połówki n/2-cyfrowe : $A = A_1 \cdot 2^{n/2} + A_0$

Podziel B na połówki n/2-cyfrowe : $B = B_1 \cdot 2^{n/2} + B_0$

$M_0 \leftarrow A_0 B_0$; $M_1 \leftarrow A_1 B_1$; $M_2 \leftarrow (A_0 + A_1)(B_0 + B_1)$ // $M_2 = A_0 B_0 + A_0 B_1 + A_1 B_0 + A_1 B_1$

$\Rightarrow A_0 B_1 + A_1 B_0 = M_2 - M_0 - M_1$

return $2^n M_1 + 2^{n/2}(M_2 - M_0 - M_1) + M_0$

Wynik mnożenia $A \cdot B = (A_1 \cdot 2^{n/2} + A_0)(B_1 \cdot 2^{n/2} + B_0)$

$= A_1 B_1 \cdot 2^n + (A_1 B_0 + A_0 B_1) \cdot 2^{n/2} + A_0 B_0$

T(n) - czas mnożenia liczb n-cyfrowych

3T(n/2) to czas potrzebny na wymnożenie liczb M_0, M_1, M_2 z podanego algorytmu

$$T(n) = 3T(n/2) + cn$$

$$= cn + 3c(n/2) + 9T(n/4)$$

$$= cn + \frac{3}{2}cn + \frac{9}{4}cn + \frac{3^{k-1}}{2^{k-1}}cn + T(n/2^k) \cdot 3^k \qquad (k = \log n)$$

$$\leq 3^k T(1) + \frac{3^k}{2^k} cn \left(\frac{2}{3} + \frac{2^2}{3^2} + \frac{2^3}{3^3} + \dots \right)$$

$$= 3^{\log n} \left(T(1) + \frac{\frac{2}{3}c}{1 - \frac{2}{3}} \right)$$

$$\boxed{3^{\log n} = 2^{\log 3 \log n} = n^{\log 3}}$$

$$T(n) \leq n^{\log_2 3} \left(T(1) + \frac{\frac{2}{3}c}{1 - \frac{2}{3}} \right) = C \cdot n^{\log_2 3} = O(n^{\log_2 3}) \approx O(n^{1.58})$$

$NWD(a,b) = \max\{d \in \mathbb{N} : d|a \wedge d|b\}$ (dla $a,b \in \mathbb{N} \cup \{0\}$)

$NWW(a,b) = \min\{c \in \mathbb{N} : a|c \wedge b|c\}$

Powyższe operacje są symetryczne, tj. $NWD(a,b) = NWD(b,a)$ oraz $NWW(a,b) = NWW(b,a)$.

$NWD(a,0) = 0$

$NWW(a,1) = a$

Dla $a \geq b$: $NWD(a,b) = NWD(a-b,b) = NWD(a-cb,b)$ $(c \in \mathbb{Z})$

Algorytm Euklidesa

```
NWD (a,b)
  if a = 0   then NWD ← b
  if a < b   then swap (a,b)
  else NWD ← NWD (a-b, b)
```

zamiast tego lepiej zamienić $(a-b)$ na $a - \lfloor \frac{a}{b} \rfloor \cdot b$ $\left(a - \lfloor \frac{a}{b} \rfloor b = a \text{ MOD } b,\ \lfloor \frac{a}{b} \rfloor = a \text{ DIV } b\right)$

```
NWD (a,b):
  while b>0:
    (a,b) ← (b, a MOD b)
  NWD ← a
```

dowód poprawności tego algorytmu można przeprowadzić używając pojęcia niezmiennika funkcji

Lemat: Niech (a_i, b_i) to wartości (a,b) w i-tej iteracji algorytmu Euklidesa.

Jeśli $i > 0$ oraz $a_i < F_{k+1}$ lub $b_i < F_k$, to:

$a_{i+1} < F_k$ lub $b_{i-1} < F_{k-1}$.

Z lematu wynika, że jeśli $a_0, b_0 < F_n$, to algorytm Euklidesa wykonuje co najwyżej n iteracji, czyli jeśli a_0, b_0 to liczby n-cyfrowe to liczba iteracji algorytmu Euklidesa jest $O(n)$.

Dowód lematu

Jeżeli $b_i < F_k$, to $a_{i+1} = b_i < F_k$.

Jeżeli $b_i \geq F_k$, to $a_i < F_{k+1}$, wtedy $b_{i+1} = a_i \text{ MOD } b_i = a_i - \lfloor \frac{a_i}{b_i} \rfloor b_i \leq$

$\leq a_i - b_i < F_{k+1} - F_k = F_{k-1}$ ∎

Rozszerzony algorytm Euklidesa

dla $a,b \in \mathbb{N}$ wylicza nie tylko $NWD(a,b)$, ale również:

$$x,y \in \mathbb{Z} : xa + yb = NWD(a,b), \quad (|x|<b, \ |y|<a)$$

Przykład: $a = 245$, $b = 168$

iteracja	a	b
	245	168
1	168	$245-168=77$
2	77	$168-2\cdot77=14$
3	14	$77-5\cdot14=7$
4	7	$7-7=0$

NWD

tutaj cofamy się od ostatniej do pierwszej iteracji algorytmu Euklidesa.

Znajdowanie x,y: $77-5\cdot14=7$

$77-5\cdot(168-2\cdot77)=7$

$11\cdot77-5\cdot168$

$11\cdot(245-168)-5\cdot168=7$

$11\cdot245-16\cdot168$

więc $x=11$, $y=-16$

Pierścień $\mathbb{Z}_n = \{0,1,2,\dots,n-1\}$, czyli zbiór reszt

Działania w $\mathbb{Z}_n$: $+_n, \cdot_n$, czyli $a +_n b = (a+b) \text{ MOD } n$

$a \cdot_n b = (a \cdot b) \text{ MOD } n$

Element odwrotny to a taki, a^{-1}, że $a^{-1} \cdot_n a = a \cdot_n a^{-1} = 1$.

Kiedy $a^{-1} \bmod n$ istnieje, jak to policzyć?

Lemat: Jeśli $NWD(a,n)>1$, to $a^{-1} \text{ MOD } n$ nie istnieje.

Dowód: $d = NWD(a,n)$, wtedy dla dowolnego a': $d \mid a' \cdot a$ i $d \mid n \Rightarrow$

$\Rightarrow d \mid a' \cdot_n a \neq 1 \Rightarrow a' \neq a^{-1} \bmod n$.

Lemat: Jeśli $a \perp n$*, czyli $NWD(a,n)=1$, to $a^{-1} \bmod n$ istnieje.

Dowód: Podstawiamy a i n w algorytm Euklidesa za a i b, algorytm wylicza $x,y \in \mathbb{Z}$: $xa+yb = NWD(a,n) = 1$, zatem $x \cdot_n a = 1$, zatem $a^{-1} = x \bmod n$.

* $a \perp b$ oznacza względnie pierwsze liczby a i b