

Wstęp do Programowania potok funkcyjny

Marcin Kubica

2010/2011

Outline

- 1 Analiza kosztów
 - Złożoność czasowa i pamięciowa
 - Przykłady

Rzędy funkcji

Definition

$$f(x) = \Theta(g(x)) \Leftrightarrow \exists_{c_1, c_2 \in \text{real}, c_1, c_2 > 0, n_0 \in \text{nat}} \forall_{n \in \text{nat}, n \geq n_0} \\ c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

$$f(x) = O(g(x)) \Leftrightarrow \exists_{c \in \text{real}, c > 0, n_0 \in \text{nat}} \forall_{n \in \text{nat}, n \geq n_0} 0 \leq f(n) \leq c \cdot g(n)$$

$$f(x) = \Omega(g(x)) \Leftrightarrow \exists_{c \in \text{real}, c > 0, n_0 \in \text{nat}} \forall_{n \in \text{nat}, n \geq n_0} 0 \leq c \cdot g(n) \leq f(n)$$

Fact

Dla funkcji nieujemnych zachodzi:

$$f(x) = \Theta(g(x)) \Leftrightarrow f(x) = \Omega(g(x)) \wedge f(x) = O(g(x))$$

$$f(x) = \Omega(g(x)) \Leftrightarrow g(x) = O(f(x))$$

Rzędy funkcji

Definition

$$f(x) = \Theta(g(x)) \Leftrightarrow \exists_{c_1, c_2 \in \text{real}, c_1, c_2 > 0, n_0 \in \text{nat}} \forall_{n \in \text{nat}, n \geq n_0} c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

$$f(x) = O(g(x)) \Leftrightarrow \exists_{c \in \text{real}, c > 0, n_0 \in \text{nat}} \forall_{n \in \text{nat}, n \geq n_0} 0 \leq f(n) \leq c \cdot g(n)$$

$$f(x) = \Omega(g(x)) \Leftrightarrow \exists_{c \in \text{real}, c > 0, n_0 \in \text{nat}} \forall_{n \in \text{nat}, n \geq n_0} 0 \leq c \cdot g(n) \leq f(n)$$

Fact

Dla funkcji nieujemnych zachodzi:

$$f(x) = \Theta(g(x)) \Leftrightarrow f(x) = \Omega(g(x)) \wedge f(x) = O(g(x))$$

$$f(x) = \Omega(g(x)) \Leftrightarrow g(x) = O(f(x))$$

Rzędy funkcji

Example

Jak się mają do siebie funkcje:

- $n^2 + 100\,000$,
- $0.000001n^{\frac{1}{2}}$,
- 4 ,
- 2^{2n} ,
- $\log_2 n$,
- $2\sqrt{4n}$,
- n^n ,
- $\ln(10n)$,
- 4^n ,
- $n!$,
- 10^n ,
- 0 .

Przedstawienie kosztu jako funkcji

Definition

- Koszt, to ilość (określonego) zasobu potrzebnego do obliczenia wyniku.
- Ilość ta zależy od konkretnych danych.
- Interesuje nas ilość potrzebnych zasobów w zależności od określonego aspektu danych, np.:
 - rozmiaru danych,
 - rozmiaru macierzy,
 - jeśli dane to jedna liczba, to od samych danych,
 - wymaganej dokładności przybliżenia (liczba miejsc dziesiętnych).
- Oznaczenia
 - Zbiór danych — D .
 - Ilość zasobu — $\varphi : D \rightarrow \mathbb{N}$.
 - Aspekt danych — $\mu : D \rightarrow \mathbb{N}$.

Przedstawienie kosztu jako funkcji

Definition

- Związek między aspektem danych i ilością zasobu:

$$\vec{\varphi} \circ \mu^{-1} : N \rightarrow \mathcal{P}(N)$$

- Koszt pesymistyczny (możemy porównywać rząd wielkości):

$$\max \circ \vec{\varphi} \circ \mu^{-1} : N \rightarrow N$$

- Koszt oczekiwany (średni):
 - D , $\varphi(D)$ i $\mu(D)$ — zmienne losowe,
 - $E(\varphi(D) \mid \mu(D) = n)$.
- Programy niedeterministyczne — φ nie jest funkcją, ale relacją lub zmienną losową.
Zasadnicze wzory pozostają bez zmian.

Koszt czasowy

Definition

- Dokładny koszt czasowy = koszt operacji elementarnych.
- Operacje elementarne:
 - odczytanie środowiska, wartość stałej — 1,
 - wywołanie procedury:
 - koszt obliczenia wszystkich elementów (w tym procedury) +
 - koszt wyliczenia treści procedury +
 - liczba elementów kombinacji,obliczenie treści procedury wbudowanej — 1, chyba że powiedziane inaczej,
 - if-then-else:
 - koszt obliczenia warunku +
 - koszt obliczenia odpowiedniej części +
 - 1,
 - λ -abstrakcja (function) — 1,

Koszt czasowy

Definition

- Operacje elementarne (c.d.):
 - match-with:
 - obliczenie dopasowywanej wartości +
 - łączna długość wzorców +
 - obliczenie wyrażenia odpowiadającego dopasowanemu przypadkowi.
 - let-in:
 - koszt wyliczenia definiowanych wartości +
 - liczba definiowanych wartości +
 - obliczenie wyrażenia po in ,
 - rekurencja — równanie rekurencyjne na kosztach.
- Oznaczenie $T(n)$.
- Nie liczymy dokładnego kosztu, tylko z dokładnością do rzędu.
- Pomijamy odśmiecanie — nie wpływa na rząd kosztów.

Koszt czasowy

Example

```
let f =  
  let g x = 3 * x  
  in  
    function x -> g (2 * x);;  
f 7;;
```

Example

```
let rec silnia n =  
  if n < 2 then 1 else n * silnia (n - 1);;
```

Koszt czasowy

Example

```
let f =  
  let g x = 3 * x  
  in  
    function x -> g (2 * x);;  
f 7;;
```

Example

```
let rec silnia n =  
  if n < 2 then 1 else n * silnia (n - 1);;
```

Koszt pamięciowy

Definition

- Koszt pamięciowy = maksymalna pamięć zajmowana przez:
 - ramki zawierające potrzebne symbole,
 - tworzone wartości,
 - jeśli potrzebna jest wartość złożona (np. lista, lub para), to potrzebne są również jej składowe.
- Na potrzeby tego wykładu, nie wliczamy danych.
Wliczamy wartości pośrednie i wynik.
- Potrzebne symbole — jak w odśmiecaniu, występują w środowiskach:
 - globalnym,
 - wszystkich, w których są obliczane wyrażenia,
 - wskazywanych przez potrzebne wartości proceduralne.
- Rekurencja ogonowa — tylko środowisko związane z ostatnim wywołaniem jest potrzebne.

Koszt pamięciowy

Definition

- Rozmiar wartości:
 - liczby, znaki, wartości logiczne, unit — 1,
 - n -tka — suma wielkości składowych,
 - lista — długość (liczba rekordów) + wielkość elementów +1,
 - wariant — 1+ wielkość ew. argumentu,
 - wartość proceduralna — $2 \times$ liczba argumentów (nie liczymy kodu).
- Oznaczenie $M(n)$.

Koszt pamięciowy

- Brak ogólnej zasady liczenia kosztu pamięciowego.
- Analiza sposobu obliczania zgodnie z modelem środowiskowym.
- Odśmiecanie nie wpływa na rząd kosztu pamięciowego.
- Uwaga na wartości współdzielone.
- Bywa skomplikowane — zasada 80%–20%.
- Nie liczymy dokładnego kosztu, tylko od razu z dokładnością do rzędu.
- Uproszczenia:
 - koszt stały = $\Theta(1)$,
 - w równaniach rekurencyjnych, składniki bez odwołań rekurencyjnych możemy zastąpić równymi co do rzędu,
 - czasem prościej niezależnie oszacować rząd od góry i dołu $f(n) = \Omega(g(n)) \wedge f(n) = O(g(n)) \implies f(n) = \Theta(g(n))$.
- Koszt pamięciowy nie może być większy niż koszt czasowy.

Potęgowanie

Example

Pierwsze podejście:

$$b^n = b \cdot b^{n-1}, \quad b^0 = 1$$

```
let rec potega b n =  
  if n = 0 then 1  
  else b * potega b (n-1);;
```

- Aspekt: n .
- n kroków, każdy ma stały koszt czasowy i pamięciowy.
- Koszty czasowy i pamięciowy są rzędu $T(n) = M(n) = \Theta(n)$.

Potęgowanie

Example

Drugie podejście:

```
let potega b n =  
  let rec iter n a =      (* = a · bn *)  
    if n = 0 then a else iter (n-1) (a*b)  
  in  
    iter n 1;;
```

- Rekurencja ogonowa — $M(n) = \Theta(1)$.
- Złożoność czasowa bez zmian, $T(n) = \Theta(n)$.

Potęgowanie

Example

Trzecie podejście:

$$b^0 = 1, \quad b^{2^n} = (b^2)^n, \quad b^{2^{n+1}} = b \cdot b^{2^n}$$

```
let potega b n =
```

```
  let rec pot b n a = (* = a · bn *)
```

```
    if n = 0 then a
```

```
    else if parzyste n then pot (square b) (n / 2) a
```

```
    else pot (square b) ((n - 1)/2) (a * b)
```

```
  in
```

```
    pot b n 1;;
```

- Rekurencja ogonowa — $M(n) = \Theta(1)$.
- Stały koszt czasowy pojedynczego kroku.
- Jeżeli $2^k \leq n < 2^{k+1}$, to algorytm wymaga $k + 2$ kroków.
Stąd, $T(n) = \Theta(\log n)$.

Algorytm mnożenia rosyjskich chłopów

Example

Rekurencyjny algorytm mnożenia, używany przez chłopów na pewnych obszarach Syberii do mnożenia w pamięci.

$$x \cdot y = \frac{x}{2} \cdot 2y \quad \text{dla } x \text{ parzystych}$$

$$x \cdot y = (x - 1) \cdot y + y \quad \text{dla } x \text{ nieparzystych}$$

```

let rec razy x y =
  let rec pom x1 y1 a = (* x1 ≥ 0 ∧ x1 · y1 + a = x · y *)
    if x1 = 0 then a
    else if parzyste x1 then
      pom (x1 / 2) (2 * y1) a
    else
      pom (x1 - 1) y1 (a + y1)
  in
  if abs x > abs y then razy y x else
  if x > 0 then pom x y 0 else pom (-x) (-y) 0;;

```

Algorytm mnożenia rosyjskich chłopów

Example

- Aspekt: $|x| + |y|$.
- Rekurencja ogonowa — $M(|x| + |y|) = \Theta(1)$.
(Inaczej nie dałoby się stosować do mnożenia w pamięci.)
- $T(n) = \Theta(\max(\log(|x| + |y|), 1))$:
 - Wystarczy rozważyć procedurę pom.
 $x_1 = \max(|x|, |y|)$,
 $x_1 \leq |x| + |y| \leq 2 \cdot x_1$,
 $x_1 = \Theta(|x| + |y|)$
 - Stały koszt czasowy pojedynczego kroku.
 - Jeśli x_1 jest nieparzyste, to w kolejnym kroku jest parzyste.
Przynajmniej w co drugim kroku x_1 jest parzyste.

Algorytm mnożenia rosyjskich chłopów

Example

- Jeśli x_1 parzyste, to w kolejnym kroku maleje o połowę (z wyjątkiem ostatniego kroku, gdy $x_1 = 0$).
Co najwyżej $2(\log x_1 + 1)$ kroków.
 $T(n) = O(\max(\log x_1, 1))$.
- Jeśli x_1 nieparzyste, w kolejnym kroku maleje o 1.
Co najmniej $\log x_1 + 1$, kroków.
 $T(n) = \Omega(\max(\log x_1, 1))$.
- Reasumując, $T(n) = \Theta(\max(\log(|x| + |y|), 1))$.

Brakująca wartość na liście liczb [Bentley]

Example

- Problem:
 - Dane: liczba n i lista mniej niż n liczb całkowitych od 1 do n .
 - Znaleźć jedną z brakujących wartości z zakresu od 1 do n .
- Z zasady szufladkowej Dirichleta, taka wartość istnieje.
- Technika bisekcji:
 - Dzielimy przedział $[1, n]$ na dwa przedziały.
 - W jednym z nich musi brakować jakiejś wartości.
 - Dzielimy listę na dwie, stosownie do dwóch przedziałów.
 - Zawężamy poszukiwania do krótszego przedziału i listy.

Brakująca wartość na liście liczb [Bentley]

Example

```
let szukaj l n =  
  let rec binary l a b =  
    if l = [] then a  
    else  
      let c = (a + b) / 2  
      in  
        let l1 = filter (fun x -> x <= c) l  
        and l2 = filter (fun x -> x > c) l  
        in  
          if length l1 <= length l2 then  
            binary l1 a c  
          else  
            binary l2 (c+1) b  
  in  
    binary l 1 n;;
```

Brakująca wartość na liście liczb [Bentley]

Example

- Rekurencja ogonowa — $M(n) = \Theta(n)$.
- Złożoność czasowa:

$$T(1) = 1$$

$$T(n) \leq T\left(\lceil \frac{n}{2} \rceil\right)$$

$$T(n) \leq n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots = O(n)$$

Pierwsze wywołanie procedury binary — $\Omega(n)$.

$$T(n) = \Theta(n).$$

Algorytm Euklidesa przez odejmowanie

Example

Dla $x, y > 0$ chcemy obliczyć $(\text{nwd } x \ y) = \text{NWD}(x, y)$.

```
let rec nwd x y =  
  if x = y then x else  
    if x > y then  
      nwd (x - y) y  
    else  
      nwd x (y - x);;
```

- Aspekt: $n = x + y$.
- Liniowa liczba kroków o stałym koszcie czasowym — $T(n) = \Theta(n)$.
- Rekurencja ogonowa — $M(n) = \Theta(1)$.

Algorytm Euklidesa przez dzielenie

Example

```
let nwd a b =  
  let rec e a b =  
    if b = 0 then a else e b (a mod b)  
  in  
    if a > b then e a b else e b a;;
```

Algorytm Euklidesa przez dzielenie

Lemma (Lame)

Oznaczmy przez (a_i, b_i) pary wartości a i b , dla których powyższy algorytm wykonuje i kroków. Wówczas $b_i \geq \text{Fib}_{i-1}$.

Dowód.

Dowód indukcyjny.

- 1 jeden krok: $b_1 = 0$,
- 2 dwa kroki: $b \geq 1$,
- 3 więcej kroków: $(a_{k+1}, b_{k+1}) \rightarrow (a_k, b_k) \rightarrow (a_{k-1}, b_{k-1})$,
 $a_k = b_{k+1}$, $a_{k-1} = b_k$, $b_{k-1} = a_k \bmod b_k$,
 $a_k = qb_k + b_{k-1}$ dla $q \geq 1$,
 $b_{k+1} \geq b_k + b_{k-1}$.



Algorytm Euklidesa przez dzielenie

Example

- $Fib_n \approx \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n}{\sqrt{5}}$,
- Z lematu Lame: $T(a + b) = O(\log(a + b))$.
- Dla $a = Fib_{n+1}$ i $b = Fib_n$ algorytm wykonuje n kroków, bo:

$$Fib_{n+1} \bmod Fib_n = (Fib_n + Fib_{n-1}) \bmod Fib_n = Fib_{n-1}$$

- $T(a + b) = \Theta(\log(a + b))$.
- Rekurencja ogonowa — $M(n) = \Theta(1)$.

Algorytm Euklidesa przez parzystość

Example

```
let nwd x y =  
  let rec pom x y a =  
    if x = y then a * x  
    else if parzyste x && parzyste y then  
      pom (x / 2) (y / 2) (2 * a)  
    else if parzyste x then pom (x / 2) y a  
    else if parzyste y then pom x (y / 2) a  
    else if x > y then pom (x - y) y a  
    else pom x (y - x) a  
  in pom x y 1;;
```

- Jedynie odejmowanie i mnożenie/dzielenie/modulo 2.
- Dla bardzo dużych liczb — koszt operacji arytmetycznych: $O(\text{długość zapisu liczby})$.

Algorytm Euklidesa przez parzystość

Example

- (a_i, b_i) — pary, dla których algorytm wykonuje i kroków.
- $a_{i+1} \geq a_i$, $b_{i+1} \geq b_i$, $b_1 = a_1 > 0$.
- W pierwszych trzech przypadkach $a_{i+1}b_{i+1} \geq 2a_i b_i$.
W czwartym przypadku $a_{i+1}b_{i+1} \geq 2a_{i-1}b_{i-1}$.
- Przez indukcję można pokazać, że: $a_i b_i \geq 2^{\lfloor \frac{i-1}{2} \rfloor}$.
Algorytm wykona $O(\log(ab)) = O(\log \max(a, b))$ kroków.

Algorytm Euklidesa przez parzystość

Example

- Z drugiej strony, $a_{i+1} + b_{i+1} \leq 2(a_i + b_i)$,
 $a_i + b_i \leq 2^{i-1}(a_1 + b_1)$.
- $\Omega(\log(a + b)) = \Omega(\log \max(a, b))$ kroków.
 $\Theta(\log \max(a, b))$ kroków.
- Stały koszt czasowy pojedynczego kroku.
 $T(\max(a, b)) = \Theta(\log \max(a, b))$.
- Rekurencją ogonowa — $M(\max(a, b)) = \Theta(1)$.

Liczby Fibonacciego

Example

Najprostszy algorytm opiera się na rekurencyjnym wzorze definiującym liczby Fibonacciego:

$$Fib_0 = 0 \quad Fib_1 = 1 \quad Fib_{n+1} = Fib_n + Fib_{n-1}$$

```
let rec fib n =
  if n < 2 then n
  else fib (n - 1) + fib (n - 2);;
```

- Złożoność czasowa:
 - Rozwijając rekurencję uzyskujemy sumę (Fib_n) jedynek i (nie więcej niż Fib_n) zer.
 - $T(n) = \Theta(Fib_n) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$.
- Złożoność pamięciowa:
 - Brak rekurencji ogonowej.
 - $M(n) = \text{stała} \cdot \text{głębokość rekurencji} = \Theta(n)$.

Liczby Fibonacciego

Example

Bardziej efektywny algorytm pamięta dwie kolejne liczby Fibonacciego i wykorzystuje rekurencję ogonową:

- z parami i rekurencją ogonową,

```
let fib n =  
  let rec fibpom a b n =  
    if n = 0 then  
      a  
    else  
      fibpom b (a + b) (n - 1)  
  in  
    fibpom 0 1 n;;
```

- Rekurencja ogonowa — $M(n) = \Theta(1)$.
- $n + 1$ kroków, każdy w stałym czasie — $T(n) = \Theta(n)$.

Liczby Fibonacciego

Example

Jeszcze szybsze rozwiązanie uzyskamy wykorzystując mnożenie macierzy i następujące tożsamości:

$$F = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad F \times \begin{pmatrix} Fib_i \\ Fib_{i+1} \end{pmatrix} = \begin{pmatrix} Fib_{i+1} \\ Fib_{i+2} \end{pmatrix}$$

$$F^n \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} Fib_n \\ Fib_{n+1} \end{pmatrix} \quad F^n = \begin{pmatrix} Fib_{n-1} & Fib_n \\ Fib_n & Fib_{n+1} \end{pmatrix}$$

Problem sprowadza się do obliczenia macierzy F^n .

Analogicznie do potęgowania liczb:

- $T(n) = \Theta(\log n)$,
- $M(n) = \Theta(1)$ — rekurencja ogonowa.

Liczby Fibonacciego

Example

```
let mnoz ((x11, x12), (x21, x22)) ((y11, y12), (y21, y22)) =
  ((x11 * y11 + x12 * y21, x11 * y12 + x12 * y22),
   (x21 * y11 + x22 * y21, x21 * y12 + x22 * y22));;
let square x = mnoz x x;;
let f = ((0, 1), (1, 1));;
let id = ((1, 0), (0, 1));;

let potega b n =
  let rec pot b n a =
    if n = 0 then a
    else if parzyste n then pot (square b) (n / 2) a
    else pot b (n - 1) (mnoz a b)
  in
    pot b n id;;

let fib n =
  let ((_, v), _) = potega f n
  in v;;
```

Logarytm całkowitoliczbowy

Example

- Dane: dodatnia liczba całkowita n .

Wynik: $\lfloor \log_2 n \rfloor$.

- Najprostsze rekurencyjne rozwiązanie:

```
let rec int_log n =  
  if n = 1 then 0  
  else 1 + int_log (n / 2);;
```

- Z każdym krokiem n jest dzielone przez 2.
Nie więcej niż $\log_2 n + 1$ kroków, a dla $n = 2^k$ jest ich dokładnie tyle. $T(n) = \Theta(\log n)$.
- $M(n) = \Theta(\log n)$ — rekurencja nie jest ogonowa.

Logarytm całkowitoliczbowy

Example

- Stosując rekurencję ogonową poprawiamy złożoność pamięciową do $M(n) = \Theta(1)$.
- Złożoność czasowa pozostaje bez zmian, $T(n) = \Theta(\log n)$.

```
let int_log n =  
  let rec pom n a =  
    if n = 1 then a  
    else pom (n / 2) (a + 1)  
  in pom n 0;;
```

Logarytm całkowitoliczbowy

Example

- Pomyślmy o zapisie binarnym liczby $\lfloor \log_2 n \rfloor$.
Najstarszy bit wyniku jest na pozycji i .
- Liczbę i wyznaczamy porównując n z liczbami:
 $2^1, 2^2, 2^4, \dots, 2^{2^i}, 2^{2^{i+1}}$.
- Znając $i, 2^i, 2^{2^i}$ stosujemy: $\lfloor \log_2 n \rfloor = 2^i + \lfloor \log_2 \frac{n}{2^{2^i}} \rfloor$

```

let int_log n =
  let rec pom i j k m acc =
    if m = 1 then acc
    else if square k > m then
      pom 0 1 2 (m / k) (acc + j)
    else
      pom (i+1) (2*j) (square k) m acc
  in pom 0 1 2 n 0;;

```

$= acc + \lfloor \log_2 m \rfloor$
 $j = 2^i, k = 2^j \leq m$

Logarytm całkowitoliczbowy

Example

- Złożoność czasowa:
 - Wyznaczenie pozycji i najstarszego bitu wymaga $i + 1$ kroków.
 - Wyznaczając kolejny bit nie korzystamy z wcześniej obliczonych wartości.
Jest to najstarszy bit $\frac{n}{2^{2^i}}$, itd.
 - $T(n) = O\left(\sum_{i=1}^{\log \log n} i\right) = O((\log \log n)^2)$
 - Dla $n = 2^{2^i - 1}$ osiągamy ten czas, czyli $T(n) = \Theta((\log \log n)^2)$.
- Złożoność pamięciowa: $M(n) = \Theta(1)$ — rekurencja ogonowa.

Logarytm całkowitoliczbowy

Example

- Wyznaczając pozycję najstarszego bitu obliczamy liczby:
 $2^1, 2^2, 2^4, \dots, 2^{2^i}, 2^{2^{i+1}}$.
Jeśli je zapamiętamy, to możemy je wykorzystać do wyznaczenia kolejnych bitów.
- Podzielmy algorytm na dwie fazy:
 - 1 Procedura gen tworzy listę par postaci:
 $[(2^i, 2^{2^i}); \dots (4, 2^4); (2, 2^2); (1, 2^1)]$,
 - 2 Procedura scan wyznacza wynik (od najstarszego do najmłodszego bitu).

Logarytm całkowniczy

Example

```

let int_log n =
  let rec gen i j k acc =
    if k > n then
      acc
    else
      gen (i+1) (2*j) (square k) ((j, k) :: acc)
  and scan l m =
    match l with
    []          -> 0 |
    (j, k)::t  ->
      if k <= m then
        j + scan t (m / k)
      else
        scan t m
  in scan (gen 0 1 2 []) n;;

```

$$j = 2^i, \quad k = 2^j, \quad \sqrt{k} \leq n,$$

$$acc = [(2^{i-1}, 2^{2^{i-1}}); \dots (2, 2^2); (1, 2^1)]$$

$$l = [(2^i, 2^{2^i}); \dots (2, 2^2); (1, 2^1)],$$

$$m < 2^{2^{i+1}}$$

Logarytm całkowitoliczbowy

Example

- Złożoność czasowa (gen i scan): $T(n) = \Theta(\log \log |n|)$.
- Złożoność pamięciowa:
 - Rozmiar konstruowanej listy.
 - Brak rekurencji ogonowej w procedurze scan.

$$M(n) = \Theta(\log \log |n|).$$

- Polepszenie złożoności czasowej kosztem pogorszenia złożoności pamięciowej (ang. *time-memory trade-off*). Zwykle, bardziej zależy nam na złożoności czasowej. To rozwiązanie jest lepsze od poprzedniego.

Logarytm całkowitoliczbowy

Example

Jak zbić złożoność pamięciową do $M(n) = \Theta(1)$?

- Problem: łatwo możemy przejść od $(i, 2^i, 2^{2^i})$ do $(i + 1, 2^{i+1}, 2^{2^{i+1}})$, ale nie w drugą stronę. Musimy pamiętać ciąg liczb.
- Rozważmy piątki liczb postaci: $(i, Fib_i, Fib_{i+1}, 2^{Fib_i}, 2^{Fib_{i+1}})$. Możemy łatwo przejść do piątki dla $i + 1$ lub $i - 1$. Wystarczy pamiętać tylko jedną taką piątkę liczb.
- Intuicja: kolejne bity wyniku w systemie Zeckendorfa.
- Podobnie jak poprzednio, mamy dwie fazy:
 - ① Procedura gen wyznacza takie liczby $i, Fib_i, Fib_{i+1}, 2^{Fib_i}, 2^{Fib_{i+1}}$, że $2^{Fib_i} \leq n < 2^{Fib_{i+1}}$.
 - ② Procedura scan wyznacza wynik (w kolejności od najbardziej, do najmniej znaczącego bitu w systemie Zeckendorfa).

Logarytm całkowitoliczbowy

Example

```

let int_log n =
  let rec gen i j k l m =
    if m > n then
      scan i j k l m n 0
    else
      gen (i+1) k (j+k) m (l*m)
  and scan i j k l m nn acc =
    if nn = 1 then acc
    else if l <= nn then
      scan (i-1) (k-j) j (m/l) l (nn/l) (acc+j)
    else
      scan (i-1) (k-j) j (m/l) l nn acc
  in
  gen 1 1 1 2 2;;

```

$$j = \text{Fib}_i, \quad k = \text{Fib}_{i+1},$$

$$l = 2^{\text{Fib}_i}, \quad m = 2^{\text{Fib}_{i+1}}, \quad l \leq n$$

$$j = \text{Fib}_i, \quad k = \text{Fib}_{i+1},$$

$$l = 2^{\text{Fib}_i}, \quad m = 2^{\text{Fib}_{i+1}}, \quad m > nn,$$

$$[\log_2 n] = \text{acc} + [\log_2 nn]$$

Logarytm całkowitoliczbowy

Example

- Złożoność czasowa: $T(n) = \Theta(\log \log n)$.
(Ciąg liczb Fibonacciego rośnie wykładniczo szybko.)
- Złożoność pamięciowa: $M(n) = \Theta(1)$.
(Rekurencja ogonowa.)

Test na liczby pierwsze

Example

- Jeżeli liczba n nie jest pierwsza, to ma dzielnik $\leq \sqrt{n}$.

```
let min_dzielnik n =  
  let rec dziel k =  
    if square k > n then n  
    else if (n mod k) = 0 then k  
    else dziel (k + 1)  
  in dziel 2;;
```

- Rekurencja ogonowa — $M(n) = \Theta(1)$.
- Liczba kroków nie przekracza \sqrt{n} , a dla liczb pierwszych osiąga $\lfloor \sqrt{n} \rfloor$ — $T(n) = \Theta(\sqrt{n})$.

Test Fermata

Theorem (Małe twierdzenie Fermata)

Jeśli n jest liczbą pierwszą, a jest dowolną dodatnią liczbą mniejszą niż n , to $a^{n-1} \equiv 1 \pmod{n}$.

Wniosek

Jeśli istnieje takie $1 < a < n - 1$, że $a^{n-1} \not\equiv 1 \pmod{n}$, to n nie jest liczbą pierwszą.

- Niestety istnieją takie liczby n , które nie są pierwsze, ale dla każdego $1 < a < n - 1$ zachodzi $a^{n-1} \equiv 1 \pmod{n}$.
- Liczby takie nazywamy liczbami *Carmichaela*.
Oto kilka pierwszych liczb Carmichaela: 561, 1105, 1729.

Nietrywialne pierwiastki z 1

Fact

Jeśli istnieje takie $1 < a < n - 1$, że $a^2 \equiv 1 \pmod{n}$, to n nie jest liczbą pierwszą.

Takie a nazywamy *nietrywialnym pierwiastkiem z 1*.

Dowód.

Niech a będzie jak wyżej.

$(a - 1) \cdot (a + 1) = a^2 - 1 \equiv 0 \pmod{n}$. Równocześnie $a - 1 \not\equiv 0 \pmod{n}$ i $a + 1 \not\equiv 0 \pmod{n}$.

\mathbb{Z}_n nie jest ciałem, czyli n nie jest liczbą pierwszą. □

Nietrywialne pierwiastki z 1

Example

Dla $n = 15$ mamy następujące pierwiastki jedynki:

$$1^2 = 1 \equiv 1 \pmod{15}$$

$$4^2 = 16 \equiv 1 \pmod{15}$$

$$11^2 = 121 \equiv 1 \pmod{15}$$

$$14^2 = 196 \equiv 1 \pmod{15}$$

1 i 14 są trywialnymi pierwiastkami z 1.

Algorytm Millera-Rabina

Example

- Algorytm polega na wylosowaniu liczby a i teście Fermata.
- Obliczając $a^{n-1} \pmod n$ szukamy nietrywialnych pierwiastków z 1.
- Jeżeli oba kryteria są spełnione, przyjmujemy, że n jest pierwsze.
- Możemy się pomylić, fałszywie przyjmując, że n jest pierwsze.

Fact

Jeżeli n jest liczbą nieparzystą i nie jest liczbą pierwszą, to przynajmniej dla połowy $1 < a < n - 1$ obliczenie $a^{n-1} \pmod n$ odkryje nietrywialny pierwiastek z 1.

Algorytm Millera-Rabina

Example

```
exception Pierwiastek;;

let rec expmod b k n =
  let test x =
    if not (x = 1) && not (x = n - 1)
      && (square x) mod n = 1 then
      raise Pierwiastek
    else x
  in
    if k = 0 then 1
    else if parzyste k then
      (square (test (expmod b (k / 2) n))) mod n
    else
      ((test (expmod b (k-1) n)) * b) mod n;;
```

Algorytm Millera-Rabina

Example

```
let randtest n =  
  if parzyste n then n = 2  
  else if n = 3 then true  
  else  
    try expmod (Random.int (n-3) + 2) (n-1) n = 1  
    with Pierwiastek -> false;;
```

Algorytm Millera-Rabina

Example

- Prawdopodobieństwo pomyłki $< \frac{1}{2}$.
- Jeśli prawdopodobieństwo pomyłki ma być $< \varepsilon$, powtarzamy test $-\lceil \log_2 \varepsilon \rceil$ razy (stała).
- Złożoność pojedynczego testu:
 - Czasowa: $T(n) = \Theta(\log n)$.
 - Pamięciowa: $M(n) = \Theta(\log n)$ — brak rekurencji ogonowej.

Algorytmy randomizowane

Example

Dwie klasy:

- Monte Carlo — złożoność zawsze dobra, ale z małym prawdopodobieństwem może dawać złe (lub zaburzone) wyniki.
- Las Vegas — zawsze daje dobre wyniki, ale z małym prawdopodobieństwem działa dłużej; oczekiwana złożoność musi być OK.