

Programowanie i projektowanie obiektowe

Techniki programowania

Paweł Daniluk

Wydział Fizyki

Jesień 2012



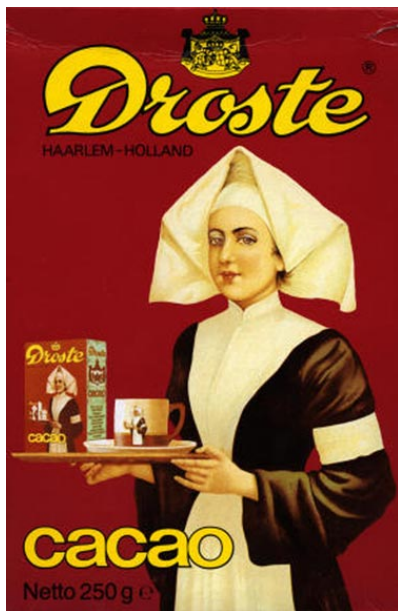
Recursion

See “Recursion”.

Recursion

If you still don't get it, see "Recursion".

Efekt Droste



Silnia

$$n! = 1 \cdot 2 \cdot \dots \cdot n$$

$$\text{silnia}(i) = \begin{cases} 1 & i = 1 \\ \text{silnia}(i - 1) \cdot i & \text{w p.p.} \end{cases}$$

Silnia

$$n! = 1 \cdot 2 \cdot \dots \cdot n$$

$$\text{silnia}(i) = \begin{cases} 1 & i = 1 \\ \text{silnia}(i - 1) \cdot i & \text{w p.p.} \end{cases}$$

Rozwiązanie rekurencyjne

```
static long silnia(int n) {  
    if(n==1) return 1;  
    else return silnia(n-1)*n;  
}
```

Silnia c.d.

Rozwiązanie rekurencyjne

```
static long silnia(int n) {  
    if(n==1) return 1;  
    else return silnia(n-1)*n;  
}
```

Rozwiązanie iteracyjne

```
static long silniaIter(int n) {  
    long res=1;  
    for(int i=1; i<=n; i++) {  
        res*=i;  
    }  
    return res;  
}
```

Liczby Fibonacciego

$$F(i) = \begin{cases} 1 & i = 1 \\ 1 & i = 2 \\ F(i-2) + F(i-1) & \text{w p.p.} \end{cases}$$

Rozwiązanie rekurencyjne

```
static long fibonacci(int n) {  
    if(n==1 || n==2) return 1;  
    else return fibonacci(n-2)+fibonacci(n-1);  
}
```


Zalety rekurencji

- prostota implementacji
- dobra do rozwiązywania problemów, które wymagają stosu
- często stosowana w metodzie “dziel i zwyciężaj”

Uwagi

Zalety rekurencji

- prostota implementacji
- dobra do rozwiązywania problemów, które wymagają stosu
- często stosowana w metodzie “dziel i zwyciężaj”

Wady rekurencji

- wywołanie funkcji jest drogie w językach preferujących konstrukcje iteracyjne
- może łatwo nastąpić przepełnienie

Uwagi

Zalety rekurencji

- prostota implementacji
- dobra do rozwiązywania problemów, które wymagają stosu
- często stosowana w metodzie “dziel i zwyciężaj”

Wady rekurencji

- wywołanie funkcji jest drogie w językach preferujących konstrukcje iteracyjne
- może łatwo nastąpić przepełnienie

Iteracja i rekurencja są wzajemnie równoważne.

Sensowne zastosowania

- Quicksort
- obchodzenie drzew (grafów)
- wieże Hanoi (ćw.)
- algorytmy “dziel i zwyciężaj”

Uwagi c.d.

Sensowne zastosowania

- Quicksort
- obchodzenie drzew (grafów)
- wieże Hanoi (ćw.)
- algorytmy “dziel i zwyciężaj”

Istnieją również rekurencyjne struktury danych (np. listy, drzewa).

Liczby Fibonacciego

$$F(i) = \begin{cases} 1 & i = 1 \\ 1 & i = 2 \\ F(i-2) + F(i-1) & \text{w p.p.} \end{cases}$$

Rozwiązanie rekurencyjne

```
static long fibonacci(int n) {  
    if(n==1 || n==2) return 1;  
    else return fibonacci(n-2)+fibonacci(n-1);  
}
```

Rozwiązanie rekurencyjne jest niewydajne (złożoność wykładnicza).

Liczby Fibonacciego c.d.

Pomysł

Przechowujmy wyniki pośrednie zamiast wielokrotnie je liczyć.

Rozwiązanie iteracyjne

```
static long fibonacciDynamic(int n) {  
    if(n==1 || n==2) return 1;  
    long F[]=new long[n+1];  
  
    F[1]=1; F[2]=1;  
  
    for(int i=3; i<=n; i++) {  
        F[i]=F[i-2]+F[i-1];  
    }  
    return F[n];  
}
```

Wystarczy przechowywać dwie ostatnie wartości.

Uliniowanie sekwencji

Para sekwencji DNA

ACACACTA

AGCACACA

Uliniowanie sekwencji

Wstawienie odstępów, tak aby zmaksymalizować jakość uliniowania mierzoną podobieństwem aminokwasów i ilością odstępów.

A-CACACTA

AGCACAC-A

Uliniowanie sekwencji c.d.

Miara jakości uliniowania

$$w : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \mathbb{R}$$

Na przykład:

$$w(a, b) = \begin{cases} 2 & a = b \\ 0 & a = - \text{ lub } b = - \\ -1 & \text{w p.p.} \end{cases}$$

Algorytm Smitha-Watermana

$$S_1 = a_1 a_2 \dots a_n$$

$$S_2 = b_1 b_2 \dots b_m$$

$H(i, j)$ – miara najlepszego uliniowania sekwencji $a_1 a_2 \dots a_i, b_1 b_2 \dots b_j$

Definicja rekurencyjna

$$H(i, 0) = \sum_{k=1}^i w(a_k, -), \quad 0 \leq i \leq m$$

$$H(0, j) = \sum_{k=1}^j w(-, b_k), \quad 0 \leq j \leq n$$

$$H(i, j) = \max \left\{ \begin{array}{l} H(i-1, j-1) + w(a_i, b_j) \\ H(i-1, j) + w(a_i, -) \\ H(i, j-1) + w(-, b_j) \end{array} \right. \left. \begin{array}{l} \text{Dopasowanie} \\ \text{Delecia} \\ \text{Insercja} \end{array} \right\}, \quad 1 \leq i \leq$$

$$m, \quad 1 \leq j \leq n$$

Algorytm Smitha-Watermana c.d.

Przykład

$$H = \begin{pmatrix} - & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix}$$

Zadanie 1 – Fibonacci

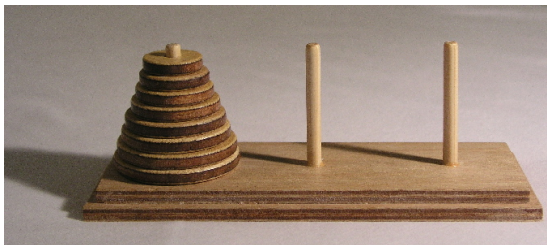
Zadanie

Zaimplementować funkcje obliczające n -tą liczbę Fibonacciego metodami rekurencyjną i programowania dynamicznego (z tablicą i bez).

Zadanie 2 – Wieże Hanoi

Zadanie

Zaimplementować algorytm rozwiązujący problem przenoszenia n krążków z pierwszego kołka na trzeci zgodnie z regułami.



Zadanie 2 – Wieże Hanoi c.d.

Schemat rozwiązania

```
//tworzy nową instancję problemu z n krążkami
//na pierwszym kołku
HanoiPegs pegs = new HanoiPegs(n);

// zdejmuję krążek z pierwszego kołka
int k=pegs.A.pop();

// wkłada k-ty krążek na trzeci kołek
pegs.C.push(k);
```

Krążek z numerem 1 jest najmniejszy.

Zadanie 3 – Uliniowanie sekwencji

Zadanie

Zaimplementować algorytm uliniawiania sekwencji DNA.