

Analýza algoritmů. Složitost.

Matematické algoritmy (K611MA)

Jan Přikryl

7. přednáška K611MA
čtvrtek 13. listopadu 2008

verze: 2008-11-18 13:03

1 Úvod

1.1 Analýza algoritmů

Algoritmy a programy Co je co

Algoritmus:

- myšlenka řešení nějakého problému
- konečný počet kroků řešení
- vyjadřujeme nejčastěji slovně nebo pseudokódem

Program:

- implementace algoritmu ve zvoleném programovacím jazyce

Bez algoritmu nelze napsat program.

Analýza složitosti algoritmů Proč nás to vlastně zajímá

Ideální kombinace: *efektivní algoritmus + efektivní implementace*

Analýza algoritmu:

- poskytne **předpověď výkonnosti** algoritmu
- je **vhodnější než experimenty**
- umožní výběr **vhodné varianty** řešení

Asymptotická složitost **paměťová** $\times$ složitost **časová**

Analýza složitosti algoritmů (2) Předpověď chování

Analýza efektivity: Identifikace efektivních a neefektivních částí algoritmu umožní soustředit se na ty části, jejichž úprava přinese nejvyšší nárůst výkonu.

Předpověď výkonnosti programu

Velké projekty potřebují apriorní odhad výkonnosti pro daný hardware – je třeba učinit odhad bez znalosti detailů programového kódu.

Identifikace úzkých míst a jejich vhodné ošetření ještě před vlastním naprogramováním.

Analýza složitosti algoritmů (3) Ověření vlastností

Vhodnější než experimenty

Experimenty ověří chování pouze ve vybraných krizových případech – místo „dokázali jsme, že daný algoritmus funguje správně“ lze pouze tvrdit: „*nenalezli jsme způsob, jak prokázat, že algoritmus je špatně*“.

Záruky funkčnosti poskytuje jediné **formální analýza**.

Výběr vhodné varianty

Ne vždy je nejvhodnější varianta ta, jenž je v počtu instrukcí nejefektivnější a tedy nejrychlejší – *např. implementace pro jednočipový počítač $\times$ pracovní stanice*.

2 Vývojová stádia algoritmu

Ilustrační příklad Fibonacciho posloupnost

Poprvé popsána italským matematikem Leonardem z Pisy, známým také jako Fibonacci (1202).

Růstu populace králíků za poněkud idealizovaných podmínek.

Číslo $F(n)$ popisuje velikost populace po n měsících, předpokládáme-li, že

- první měsíc se narodí jediný pár,
- nově narozené páry jsou produktivní od druhého měsíce svého života,
- každý měsíc zplodí každý produktivní pár jeden další pár,
- králíci nikdy neumírají, nemají predátory.

Ilustrační příklad Stavů populace králíků

Fibonacciho číslo	Stav
$F(1) = 1$	začínáme s jedním párem
$F(2) = 1$	ještě jsou příliš mladí
$F(3) = 2$	tento měsíc již zplodí první potomky
$F(4) = 3$	druhý pár potomků
$F(5) = 5$	první potomci třetí generace

Obecně

$$F(n) = F(n - 1) + F(n - 2)$$

Jak efektivně zjistit $F(n)$ pro zvolené n ?

2.1 Algoritmus pomocí explicitního řešení

Explicitní řešení Přímé vyjádření $F(n)$

Explicitní nerekurzivní vztah pro n -tý člen Fibonacciho posloupnosti je

$$F(n) = \frac{\phi^n - (1 - \phi)^n}{\sqrt{5}},$$

kde ϕ je hodnota zlatého řezu,

$$\phi = \frac{1 + \sqrt{5}}{2} = 1,61803398874989 \dots \approx 1,618.$$

Algoritmus 1

Spočti

$$F(n) = \frac{\phi^n - (1 - \phi)^n}{\sqrt{5}}.$$

Analýza Algoritmu 1 Aneb co všechno může dopadnout špatně

Reprezentace čísel v plovoucí řádové čárce má svá omezení. V počítači budeme vztah reprezentovat jako

$$F(n) = \frac{1,61803^n - 0,61803^n}{2,23606}.$$

Jaké budou výsledky?

$F(2) = 1,00000$ je v pořádku $F(3) = 1,78884$ zaokrouhlíme na 2 $F(20) = 6764,69$ ještě zaokrouhlíme na 6765 $F(21) = 10945,4$ už zaokrouhlíme na 10945 místo 10946 $F(25) = 75020,6$ by mělo být 75025

Existuje přesnější varianta výpočtu?

2.2 Rekurzivní algoritmus

Rekurzivní algoritmus Výpočet podle definice

Hodnoty $F(0)$ až $F(2)$ předpočítáme, zbytek lze s jejich pomocí vyjádřit.

Algoritmus 2

Require: $n \geq 0$

Ensure: $y = F(n)$

- 1: **if** $n = 0$ **then**
- 2: $y \leftarrow 0$
- 3: **else if** $n \leq 2$ **then**
- 4: $y \leftarrow 1$

```

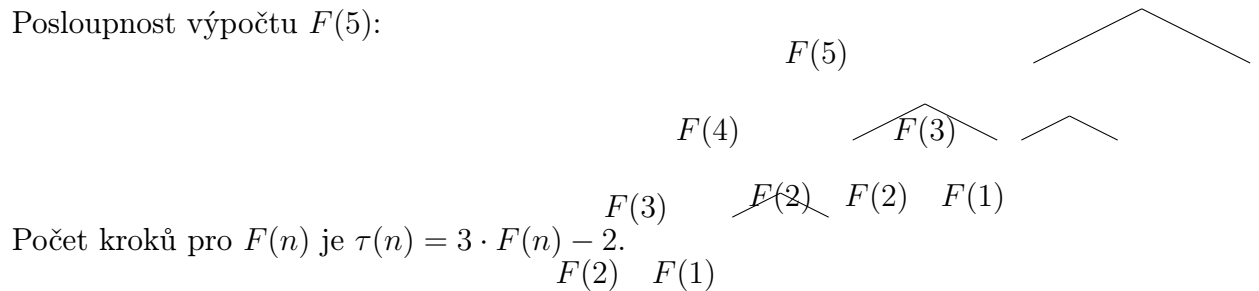
5: else
6:    $y \leftarrow F(n-1) + F(n-2)$ 
7: end if

```

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus Efektivita

Posloupnost výpočtu $F(5)$:



2.3 Dynamické programování

Dynamické programování Varianta „shora dolů“

Technika matematické optimalizace.

Dekompozice problému na identické podproblémy.

Dva základní přístupy:

- **shora dolů** – řešíme podproblémy postupně a pamatujeme si řešení
- **zdola nahoru** – vyřešíme všechny potřebné podproblémy a skládáme je

Algoritmus 3

Require: $n \geq 0$

Ensure: $y = F(n)$

```

1: Alokuj  $f[1 \dots n]$ 
2:  $f[0] \leftarrow 0$ 
3:  $f[2] \leftarrow f[1] \leftarrow 1$ 
4: for  $i = 3$  to  $n$  do
5:    $f[i] \leftarrow f[i-1] + f[i-2]$ 
6: end for
7:  $y \leftarrow f[n]$ 

```

Dynamické programování Varianta „zdola nahoru“

Algoritmus 3 potřebuje pole n prvků pro uchování minulých členů posloupnosti.

Jde to ale i bez něj.

Algoritmus 4

Require: $n \geq 0$

Ensure: $y = F(n)$

```
1: if  $n = 0$  then
2:    $y \leftarrow 0$ 
3: else
4:    $a \leftarrow 1, b \leftarrow 1$ 
5:   for  $i = 3$  to  $n$  do
6:      $c \leftarrow a + b$ 
7:      $a \leftarrow b, b \leftarrow c$ 
8:   end for
9:    $y \leftarrow b$ 
10: end if
```

Asymptotická složitost Velká O notace

Jakým způsobem se bude chování algoritmu měnit v závislosti na velikosti (počtu, objemu) vstupních dat?

Dva základní typy:

- **časová složitost** – vliv na dobu výpočtu
- **paměťová složitost** – nároky na operační paměť

Značíme:

- $O(N)$ – lineární složitost,
- $O(N^2)$ – kvadratická složitost,
- $O(\log N)$ – logaritmická složitost.

Složitost $O(N)$ znamená lineárně rostoucí nároky, $O(N) \sim k \cdot N + q$ pro nějaká $k, n \in \mathbb{N}$, pro $O(1)$ jsou nároky konstantní, $O(1) \sim q$.

Asymptotická složitost Velká O notace

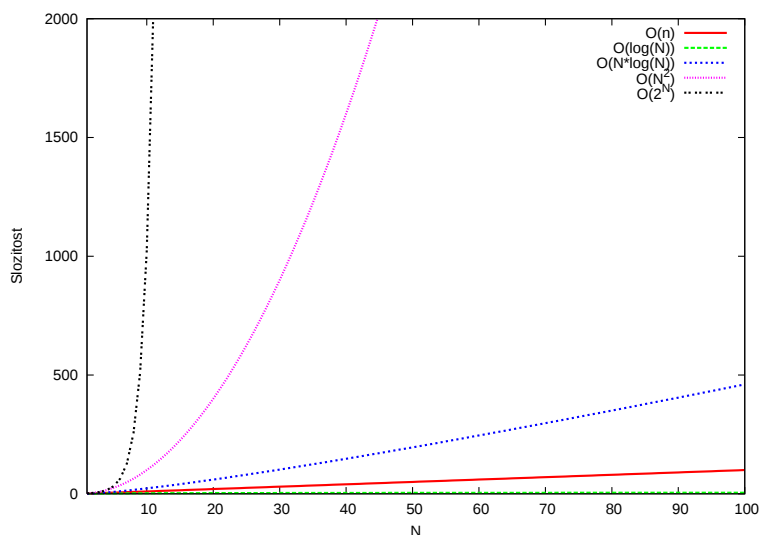
Vliv asymptotické časové složitosti

Pro $O(N^2)$ má zdvojnásobení objemu vstupních dat za následek čtyřnásobnou dobu vykonávání algoritmu. Pro $O(\log N)$ může mít čtyřnásobný počet dat na vstupu za následek dvojnásobnou dobu vykonávání algoritmu. Pro $O(1)$ je doba vykonávání algoritmu nezávislá na velikosti vstupu.

Vliv asymptotické paměťové složitosti

Pro $O(N)$ má zdvojnásobení velikosti vstupu za následek dvojnásob vysoké nároky na operační paměť. Pro $O(2^N)$ čtyřnásobná velikost vstupu zosminásobí paměťové nároky.

Asymptotická složitost Obrázek



Maticová varianta Jiná explicitní forma

Pro členy Fibonacciho posloupnosti platí také

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

Důkaz: Indukcí pro $n \rightarrow n + 1$.

Použitím opakovaného mocnění snížíme počet kroků na $O(\log n)$.

Důkaz začneme pro $n = 1$, kde platí

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix}$$

a vztah tedy pro $n = 1$ dává korektní výsledek. Za předpokladu, že vztah platí pro n , dostaneme pro $n + 1$

$$\begin{aligned} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n+1} &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} F_{n+1} + F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix} \end{aligned}$$

Odvození celého maticového předpisu vychází z teorie dvouprvkových rekurencí a na žádost některých z vás zde uvádím stručný výťah. K dispozici máte volně stažitelnou plnou verzi článku [1], z níž jsem čerpal.

Satz 1. *Množina $\mathcal{R}(a, b)$ Mějme dvojici reálných čísel a a $b \neq 0$ a posloupnost A_n definovanou rekurentním vztahem*

$$A_{n+2} = a \cdot A_{n+1} + b \cdot A_n.$$

Potom pro pevné hodnoty parametrů a a b označíme $\mathcal{R}(a, b)$ množinu všech takto parametrizovaných posloupností.

Jedním z význačných prvků $\mathcal{R}(1, 1)$ je Fibonacciho posloupnost F s počátečními členy $F_0 = 0$ a $F_1 = 1$.

Definice 2. Posloupnost A Všechny prvky posloupnosti $A_0, A_1, A_2, \dots$ tvořící vektor v $\mathbb{R}^\infty$ označíme jako *posloupnost A* .

Satz 3. *Operátor posunu doleva, označovaný $\triangleleft$, odstraní z posloupnosti A její nejlevější prvek.*

Z původní posloupnosti $A = A_0, A_1, A_2, A_3, \dots$ vznikne posunem doleva o jednu pozici posloupnost $\triangleleft A = A_1, A_2, A_3, A_4, \dots$.

Studiem vlastností dvouprvkových rekurencí snadno dospějeme k závěru, že vlastnosti rekurencí $A \in \mathcal{R}(a, b)$ jsou zcela určeny volbou A_0 a A_1 . Prostor $\mathcal{R}(a, b)$ je proto dvou-rozměrný a každé $A \in \mathcal{R}(a, b)$ musíme být schopni vyjádřit jako lineární kombinaci dvou bázových posloupností X a Y s prvky $X_0 = 1, X_1 = 0$ a $Y_0 = 0, Y_1 = 1$,

$$\begin{aligned} X &= 1, 0, b, ab, a^2b + b^2, \dots, \\ Y &= 0, 1, a, a^2 + b, a^3 + 2ab, \dots \end{aligned}$$

Pro všechna $A \in \mathcal{R}(a, b)$ je tedy

$$A = A_0X + A_1Y.$$

Všimněte si, že pro $a = 1$ a $b = 1$ je $Y = F$ (půjde o Fibonacciho posloupnost) a také, že

$$\triangleleft X = b \cdot Y,$$

takže můžeme jakoukoliv posloupnost $A \in \mathcal{R}(a, b)$ psát jako posloupnost členů

$$A_n = A_0bY_{n-1} + A_1Y_n = A_0bF_{n-1} + A_1F_n.$$

Operátor posunu doleva lze v $\mathcal{R}(a, b)$ s bázovými prvky X a Y vyjádřit maticí $\mathbf{M}$ jako

$$\triangleleft \begin{bmatrix} X \\ Y \end{bmatrix} = \mathbf{M}^\top \begin{bmatrix} X \\ Y \end{bmatrix}.$$

Prvky matice $\mathbf{M}$ jsou dány vztahy

$$\begin{aligned} \triangleleft X &= b \cdot Y, \\ \triangleleft Y &= X + a \cdot Y, \end{aligned}$$

a proto

$$\mathbf{M} = \begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}.$$

Posloupnost $A \in \mathcal{R}(a, b)$ je v bázi $[X, Y]$ možno zapsat jako

$$A = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}$$

přičemž pro n -krát posunutou posloupnost bude platit

$$\triangleleft^n A = \begin{bmatrix} A_n \\ A_{n+1} \end{bmatrix}.$$

Vzhledem k definici transformační matice $\mathbf{M}$ je ale také $\triangleleft^n A = \mathbf{M}^n A$, a proto

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} = \begin{bmatrix} A_n \\ A_{n+1} \end{bmatrix}.$$

Pro n -násobné posuny bazové posloupnosti X a Y můžeme psát

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{21} \end{bmatrix} = \begin{bmatrix} X_n \\ X_{n+1} \end{bmatrix},$$

respektive

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{12} \\ m_{22} \end{bmatrix} = \begin{bmatrix} Y_n \\ Y_{n+1} \end{bmatrix},$$

z čehož vyplývá

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} X_n & Y_n \\ X_{n+1} & Y_{n+1} \end{bmatrix}.$$

Již dříve jsme si všimli, že $Y = F$ (a tedy $Y_n = F_n$) a že $\Lambda X = b \cdot F$ (a tedy $X_n = b \cdot F_{n-1}$).
V $\mathcal{R}(a, b)$ bude proto platit

$$\begin{bmatrix} 0 & 1 \\ b & a \end{bmatrix}^n = \begin{bmatrix} b \cdot F_{n-1} & F_n \\ b \cdot F_n & F_{n+1} \end{bmatrix}$$

a v $\mathcal{R}(1, 1)$ ekvivalentně

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n = \begin{bmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{bmatrix}.$$

Maticová varianta Pomocí opakovaného mocnění

Algoritmus 5

Require: $n \geq 0$

Ensure: $y = F(n)$

1: **if** $n = 0$ **then**

```

2:   $y \leftarrow 0$ 
3:  else
4:     $\mathbf{M} \leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ 
5:     $\mathbf{M} \leftarrow \text{matpow}(\mathbf{M}, n - 1)$ 
6:     $y \leftarrow M[0, 0]$ 
7:  end if

```

Algoritmus 5a

Require: $n \geq 0, \mathbf{A}[2 \times 2]$

Ensure: $\mathbf{B} = \text{matpow}(\mathbf{A}, n)$

```

1: if  $n > 1$  then
2:    $\mathbf{B} \leftarrow \text{matpow}(\mathbf{A}, n/2)$ 
3:    $\mathbf{B} \leftarrow \mathbf{B}\mathbf{A}$ 
4: end if
5: if  $n$  je liché then
6:    $\mathbf{B} \leftarrow \mathbf{A} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ 
7: end if

```

3 Porovnání variant

Porovnání variant Místo závěru

Algoritmus	Paměťové nároky	Časová složitost
1	$O(1)$	$O(\log N)$
2	$O(N)$	$O(F(N))$
3	$O(N)$	$O(N)$
4	$O(1)$	$O(N)$
5	$O(\log N)$	$O(\log N)$

Reference

- [1] Kalman, D. – Mena, R.: The Fibonacci Numbers—Exposed. Mathematics Magazine, vol. 76, no. 3, 2003, pp. 167–181. Preprint dostupný na WWW: <http://www.american.edu/cas/mathstat/People/kalman/pdf/files/fibpaper.pdf> (staženo 18.11.2008).