Analýza algoritmů

Matematické algoritmy (K611MA)

Jan Přikryl, Miroslav Vlček

Ústav aplikované matematiky
Fakulta dopravní ČVUT

9. přednáška K611MA
čtvrtek 6. prosince 2007

Obsah přednášky

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 mne to vlastně zajímá

Ideální kombinace: efektivní algoritmus + efektivní implementace

Analýza efektivity algoritmu:

- **vhodnější než experimenty**, neboť experiment neodhalí všechna možná úskalí
- výběr **vhodné varianty** řešení, protože možností je vždycky více
- poskytuje **předpověď výkonnosti** algoritmu, neboť experiment neodhalí všechna možná úskalí
- **analýza slabých míst** algoritmu

Analýza složitosti algoritmů (2)

Předpověď chování

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 slabých míst

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

Analýza složitosti algoritmů (2)

Předpověď chování

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 slabých míst

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

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 jedině **formální analýza**.

Výběr vhodné varianty

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

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 jedině **formální analýza**.

Výběr vhodné varianty

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

Obsah přednášky

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

Stavy 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?

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,

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

Algoritmus 1

Spočti

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

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,

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

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?

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?

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?

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?

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?

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: $\quad y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $\quad y \Leftarrow 1$
5: **else**
6: $\quad y \Leftarrow F(n-1) + F(n-2)$
7: **end if**

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus
Výpočet podle definice

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

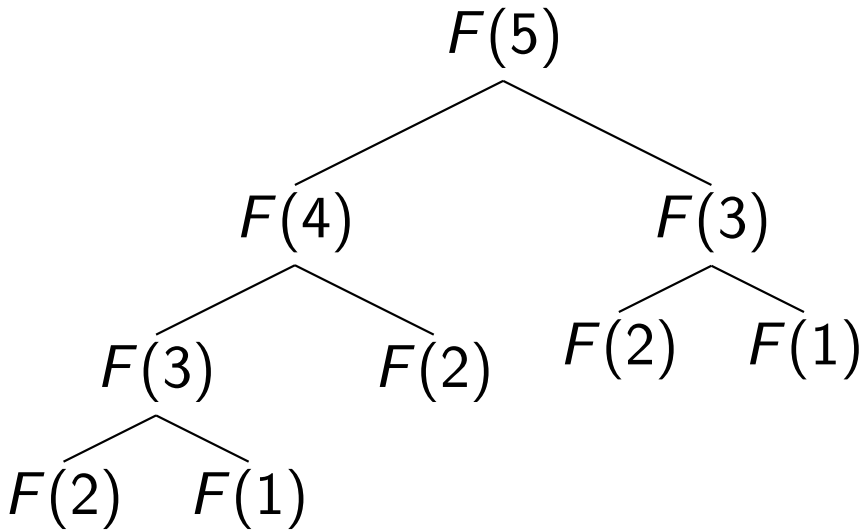
Algoritmus 2

Require: $n \geq 0$
Ensure: $y = F(n)$
1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else if** $n \leq 2$ **then**
4: $\quad y \Leftarrow 1$
5: **else**
6: $\quad y \Leftarrow F(n-1) + F(n-2)$
7: **end if**

Jak efektivní je daný algoritmus?

Rekurzivní algoritmus
Efektivita

Posloupnost výpočtu $F(5)$:



Výpočet $F(3)$ potřebuje dvě hodnoty v listech a dvě rekurze.
Výpočet $F(4)$ potřebuje tři hodnoty v listech a čtyři rekurze.
Výpočet $F(5)$ potřebuje pět hodnot v listech a osm rekurzí.
Výpočet $F(6)$ potřebuje osm hodnot v listech a čtrnáct rekurzí.

Počet kroků pro $F(n)$ je $\tau(n) \approx 3F(n) - 1$, přičemž $F(50) = 12586269025$.

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: $n \geq 0$, mapa $f = \{0 \to 0, 1 \to 1\}$
Ensure: $y = F(n)$
1: **if** $f[n]$ neexistuje **then**
2: $f[n] \Leftarrow f[n-1] + f[n-2]$
3: **end if**
4: $y \Leftarrow f[n]$

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: $n \geq 0$, mapa $f = \{0 \to 0, 1 \to 1\}$

Ensure: $y = F(n)$

1: **if** $f[n]$ neexistuje **then**

2: $f[n] \Leftarrow f[n-1] + f[n-2]$

3: **end if**

4: $y \Leftarrow f[n]$

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

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

Asymptotická složitost
Velká O notace

Jakým způsobem se bude chování algoritmu měnit v závislosti na velikosti (počtu) 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.

Asymptotická složitost
Velká O notace

Vliv asymptotické časové složitosti

Pro $O(N^2)$ má zdvojnásobení velikosti vstupu za následek čtyřnásobnou dobu vykonávání algoritmu.
Pro $O(\log N)$ může mít čtyřnásobná velikosti vstupu za následek dvojnásobnou dobu vykonávání algoritmu.
Pro $O(1)$ je dobu 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 $O(2^N)$ má čtyřnásobná velikost vstupu zosmínásobí nároky na paměť.

Asymptotická složitost

Velká O notace

Doba vykonání alogoritmu pro různá O(N)

Potřebujeme zpracovat data o $N = 10^6$ bitech na procesoru, jenž jednu bitovou instrukci vykoná za $1\mu s$.

Složitost	Doba vykonání
$O(1)$	$1\mu s$
$O(N)$	$10^6 \times 1\mu s = 1s$
$O(N^2)$	$10^{12} \times 1\mu s = \frac{10^{12}}{10^6 \cdot 3600 \cdot 24} \approx 11{,}6$ dne
$O(N^3)$	$10^{18} \times 1\mu s = \frac{10^{18}}{10^6 \cdot 3600 \cdot 24 \cdot 365} \approx 31709$ let

Maticová varianta

Jiná explicitní forma

Pro členy Fibonacciho posloupnosti platí také

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^n = \begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix}$$

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

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

Maticová varianta

Pomocí opakovaného mocnění

Algoritmus 5

Require: $n \geq 0$

Ensure: $y = F(n)$

1: **if** $n = 0$ **then**
2: $\quad y \Leftarrow 0$
3: **else**
4: $\quad \mathbf{M} \Leftarrow \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
5: $\quad \mathbf{M} \Leftarrow \text{matpow}(\mathbf{M}, n-1)$
6: $\quad y \Leftarrow M[0,0]$
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: $\quad \mathbf{B} \Leftarrow \text{matpow}(\mathbf{A}, n/2)$
3: $\quad \mathbf{B} \Leftarrow \mathbf{BA}$
4: **end if**
5: **if** n je liché **then**
6: $\quad \mathbf{B} \Leftarrow \mathbf{A} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
7: **end if**

Obsah přednášky

Porovnání variant

Místo závěru

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