Jak zrychlovat algoritmy

úzké hrdlo
opakované výpočty
pomocné tabulky, pole či soubory
šetřit aritmetické operace
předzpracovat si data
amortizace

Jana Finkeová

Které části programu má smysl zrychlovat

- nejprve je potřeba zjistit, v které části programu stráví výpočet nejvíce času (**úzké hrdlo**)
- vyzkoušejte všechno možné, nejčastěji je potřeba si do programu přidat pomocné sledování časové spotřeby
- je potřeba mít jasno, které části programu má smysl optimalizovat
- největší zrychlení je možné dosáhnout lepším algoritmem
- ale většinou za to platíme větší paměťovou náročností

Jana Finkeová

Odstranění opakovaných výpočtů

- s nějakou hodnotou je potřeba provést několik různých úkonů, ale tato hodnota není přímo součástí vstupních údajů a musí se vypočítat
- je zbytečné takovou hodnotu počítat pokaždé, když ji potřebujeme
- je dobré si ji spočítat pouze jednou a uložit si ji do pomocné proměnné pro opakované použití
- výpočet to urychlí, spotřebuje se nějaká paměť navíc

Předpočítání si výsledků do paměti

- nejrychlejší program je ten, který nic nepočítá a přímo vypíše správný výsledek
- výsledky je možné si spočítat dopředu uložit si je do tabulky
  - tabulka předpočítaných výsledků může být přímo ve zdrojovém kódu programu (např. jako statické inicializované pole)
  - nebo ji na začátku můžeme načíst ze souboru do paměti
- program pouze vyhledá správnou odpověď v tabulce
- nejde to tak vždy - tabulka nesmí být příliš velká

příklad

- pro zadané přirozené číslo máme spočítat funkční hodnotu, tady je vhodné předpočítané funkční hodnoty si uložit do tabulky
- některé algoritmy vyžadují znalost prvočísel, prvočísla není nutné pokaždé vyhledávat, je možné opět si je připravit do tabulky a příslušné prvočíslo vyhledat

Výpočet hodnoty

- Hornerovo schéma
  - pro polynomy
  - při výpočtu můžeme částečně povytýkat *x*
  - ušetříme několik aritmetických operací
  - při velkém počtu vyčíslení se to vyplatí
- výpočet na základě předchozí
  - typicky pro polynom
  - známe *P(x)* a chceme spočítat hodnotu *P(x+1)*
  - pro polynom druhého stupně to znamená připočítat k *P(x)* výraz *2ax+(b+1)*
  - *? pro váš projekt?*

poznámka: na většině počítačů je instrukce pro násobení časově náročnější než instrukce pro sčítání...

Jana Finkeová

Předzpracování dat

- někdy je vhodné si předem data upravit do vhodné podoby
- a teprve potom počítat výstup
- dvě fáze
  - předzpracování a uložení mezivýsledků (zvýšení nároků na paměť)
  - vlastní výpočet

Příklad: máme pole *n* celých čísel a máme najít to, které se v zadaném poli vyskytuje nejčastěji

- **řešení 1**: pro každý prvek pole spočítat, kolikrát se v poli vyskytuje, pro každý prvek je potřeba projít celé pole a najít největší počet výskytů, časová složitost tohoto algoritmu je $O(n^2)$, *n*-krát procházíme pole o *n* prvcích
- **řešení 2**: předzpracovat si pole tak, že si ho setřídíme, setřídění pole má časovou složitost příznivější ($O(n \log n)$), pak pro nalezení nejčastěji se vyskytující hodnoty stačí jeden průchod pole ($O(n)$), celková složitost je dána součtem složitostí obou fází

$$O(n^2) > O(n \log n) + O(n)$$

Jana Finkeová

Přímé generování požadovaných údajů

- potřebujeme nalézt všechny údaje dané vlastnosti
- můžeme postupně procházet všechny údaje a každý údaj otestovat, zda vyhovuje požadované vlastnosti
- tento přístup bývá pomalý
- zkusit najít způsob, jak přímo generovat údaje dané vlastnosti
- nebo alespoň zúžit výběr pouze na údaje, u kterých je naděje, že daná vlastnost bude splněna

  příklad: rozložte zadané kladné číslo *N* na součet třetích mocnin kladných čísel (časová složitost $O(N^2)$)
  - **řešení 1**: prozkoumat všechny dvojice čísel z daného rozmezí
  - **řešení 2**: snížit počet zkoumaných dvojic, stačí uvažovat čísla do třetí odmocniny z daného *N* (časová složitost $O(N^{2/3})$)
  - **řešení 3**: nezkoumat čísla, která jsou „hodně malá“, můžeme zkoumání začít s jedním číslem rovným 1 a s druhým rovným celé části z třetí odmocniny z N, pokud je součet třetích mocnin větší než N, pak větší číslo snížíme o 1, pokud je součet menší, zvětšíme menší číslo o jedničku, takto postupujeme až do okamžiku, kdy se obě čísla potkají (časová složitost $O(N^{1/3})$)

Jana Finkeová

Psaní efektivního kódu

- z hlediska hardwaru
  - práce s pamětí může být časově náročná, hlavně pokud se pracuje s velkými daty, někdy je lepší si velká data rozdělit a zpracovávat je postupně
  - zapisování a čtení ze souboru je časově náročné, lepší je zapisovat či číst méněkrát po větších kusech
  - podobně zapisování na obrazovku je časově náročné
  - vyhodnocování podmínek je časově náročné
  - některé aritmetické operace jsou časově náročné (násobení, dělení)

Jana Finkeová

Amortizace

- je důležité vědět, kterou úlohu budeme dělat často a kterou méně často
- typicky se musíme zamyslet nad úlohou vyhledávání
- musíme volit zda budeme vyhledávat v datech pouze jednou či jen několikrát
- nebo jestli budeme v datech vyhledávat mnohokrát
- v případě, kdy budeme vyhledávat mnohokrát, vyplatí se data setřídit

Proč?

- protože vyhledávací algoritmy v setříděných datech jsou rychlejší
- proto pojem AMORTIZACE – vyplatí se (amortizuje se) když si nejprve data setřídíme

Jana Finkeová

Amortizace

- data se mohou dynamicky měnit
- přidáváme, ubíráme položky v datech
- i to je potřeba vzít do úvahy
- protože je programově náročnější přidávat data do uspořádaného seznamu
- musíme je vřadit na správné místo

Jana Finkeová

Amortizace

vyhledávání

neuspořádaná data	uspořádaná data

algoritmus

lineární (sekvenční) vyhledávání	binární vyhledávání

časová složitost

n	log n

pokud vyhledávám pouze jednou,
nesetříděná data musím prohledávat lineárním prohledáváním
ve chvíli, kdy si je setřídím, mohu použít binární vyhledávání

n	n log n + log n

vyhledávám k-krát

k n	n log n + k log n

Jana Finkeová

Datové typy

- i ty hrají svou úlohu
- je potřeba si vyzkoušet, protože záleží i na procesoru

obecně lze říci:

- algoritmy běží rychleji na celočíselných datech než na datech s desetinnou čárkou
- instrukce, které zpracovávají celočíselná data, jsou rychlejší

Jana Finkeová

Literatura

- Přednášky na MIT: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00-introduction-to-computer-science-and-programming-fall-2008/