

## Obsah

- 1 Třídění v paměti
- 2 Proveditelné algoritmy
- 3 Třídění na disku
- 4 Sublineární třídění

Třídění ►

2 / 27

## Třídění

### Implementace databázových systémů

Pavel Rychlý

pary@fi.muni.cz

26. září 2008

Třídění ►

1 / 27

## Obsah

- 1 Třídění v paměti
- 2 Proveditelné algoritmy
- 3 Třídění na disku
- 4 Sublineární třídění

Třídění ► Třídění v paměti

4 / 27

## Rychlost operační paměti

Zpracování matic

```
int i, j;
for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
        matrix[size * j + i] = cnt++;
```

Třídění ►

3 / 27

## CountSort, RadixSort

Potřebujeme seřadit 100 známek (A-F).

- spočítáme počty jednotlivých známek
- zapíšeme výsledek

RadixSort

- rekurzivní třídění po jednotlivých bajtech
- první průchod zjistí počty
- druhý průchod umístí data na správné místo

Třídění ► Třídění v paměti

6 / 27

## Třídění v paměti

Pro třídění v paměti známe celou řadu algoritmů:

- QuickSort

(Pěkná analýza je na <http://video.google.com/videoplay?docid=-1031789501179533828>)

- BubleSort, InsertSort
- HeapSort, MergeSort
- RadixSort, CountSort

Některé mají asymptotickou časovou složitost  $O(n \log n)$ , jiné  $O(n^2)$  nebo  $O(n)$ .

Třídění ► Třídění v paměti

5 / 27

## Třídění v praxi

Které algoritmy používáte ve svých programech?

- 1 QuickSort
- 2 BubleSort, InsertSort
- 3 HeapSort, MergeSort
- 4 jiný
- 5 ani nevím, co v té knihovně mají

“Pozorování” o složitosti třídění:

- Teorie říká (a máme takové algoritmy), že je  $O(n \log n)$ .
- Ve většině praktických případů je  $O(n)$ .
- Mnoho programátorů používá algoritmy s  $O(n^2)$ .

Třídění ► Třídění v paměti

8 / 27

## Vlastnosti třídících algoritmů

- předpokládají data v operační paměti
- většinou třídí na místě (in-place)
- potřebují málo ( $\log n$ ) další paměti

Třídění ► Třídění v paměti

7 / 27

# Obsah

- 1 Třídění v paměti
- 2 Proveditelné algoritmy
- 3 Třídění na disku
- 4 Sublineární třídění

# Jaká časová složitost je únosná?

Jaká (v jakých řádech) musí být časová složitost, abychom mohli algoritmus použít v praxi? Co nám říká teorie?

- třídy  $P$  a  $NP$
- polynomiální
- $n^{10}$  – určitě ne
- $n^3$  – někdy
- $n \log n - \log n$  je pro většinu případů *malá* konstanta
- $n$  – ideální

# Jak to je v praxi?

- Dokonce některé exponenciální algoritmy mohou být použitelné, protože je vždy potřebujeme pouze pro malá  $n$ .
- Existují lineární algoritmy (dále ukážeme), které nejsou použitelné.
- Nezajímá nás asymptotická složitost ale čas (v sekundách).
- Budeme tedy počítat, kolik "přesně" (drahých) operací algoritmus potřebuje.

# Které operace počítat

Nemůžeme (a ani nemusíme) počítat všechny operace. Člověk má malou rozlišovací schopnost.

- 1:2 rozlišíme dvojnásobek od čtyřnásobku
- 1:5 někdy rozlišíme pětinasobek od desetinásobku
- 1:50 špatně rozlišíme padesátinasobek od stonásobku
- 1:1000 nerozlišíme tisícinásobek od dvoutisícinásobku

Stačí jen ty důležité/drahé/pomalé operace. Musíme vědět, jaký model použít, které operace jsou drahé.

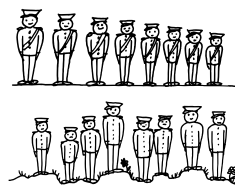
# Obsah

- 1 Třídění v paměti
- 2 Proveditelné algoritmy
- 3 Třídění na disku
- 4 Sublineární třídění

# Model třídění na disku

Mooreův zákon nám říká: důležité jsou náhodné přístupy na disk.

Pro názornost použijeme model vojenské jednotky.



- Vojáci mají nastoupit seřazení podle velikosti (od nejmenšího).
- Velitel jedním pohledem zjistí chybu.
- Připravit se musí mimo rovnou plochu. Mají jen prkno pro 10 vojáků.

# TPMMS

Two-Phase, Multiway Merge-Sort  
Dvoufázové třídění vícecestným sléváním

- 1. fáze  
třídění "bloků" v op. paměti
- 2. fáze  
slévání všech proudů

# Dvoufázové třídění - 1. fáze

Opakovaně provádíme:

- naplnění "celé" paměti daty (sekvenční čtení)
- třídění v paměti
- zápis celého proudu na disk sekvenční zápis
- 1 čtení + 1 zápis pro každý blok dat

Vojáci po 10 naběhnou na prkno a seřadí se.

## Dvoufázové třídění - 2. fáze

- 1 blok paměti pro každý proud
- blok(y) paměti pro výstup
- načtení vstupních bufferů
- *na prkne stojí vždy první z každé skupiny*
- opakované
  - nalezení nejmenší hodnoty ze všech proudů
  - přesun do výstupního bloku
  - *odchází vždy nejmenší voják*
  - aktualizace čela použitého proudu
  - plný výst. buffer → zápis
  - prázdný vstupní buffer → čtení
  - *odešlého nahradí další ze stejné skupiny*
- 1 čtení + 1 zápis pro každý blok dat

Třídění ► Třídění na disku

17 / 27

## Dvoufázové třídění - analýza

- $B$  = počet bloků s tříděnými daty
- 1. fáze:  $B * (1 \text{ čtení} + 1 \text{ zápis})$
- 2. fáze:  $B * (1 \text{ čtení} + 1 \text{ zápis})$
- celkem:  $\max 4 * B$  náhodných přístupů na disk
- Složitost TMPPS je  $O(n)$ ,  $n$  je počet bloků na disku.
- Přesněji: maximálně  $4B$ , může být méně minimálně  $B$ .

Třídění ► Třídění na disku

18 / 27

## Dvoufázové třídění-omezení

- parametry (v bajtech)
  - $K$  = velikost bloku
  - $M$  = velikost dostupné paměti
  - $R$  = velikost záznamu
- max počet bufferů:  $M/K$
- max počet proudů:  $M/K - 1$
- počet záznamů v každém kroku 1. fáze:  $M/R$
- max počet tříděných záznamů  
 $(M/R)(M/K - 1) = M^2/RK$
- max velikost tříděných dat:  $M^2/K$

Třídění ► Třídění na disku

19 / 27

## Dvoufázové třídění-omezení

- max velikost tříděných dat:  $MM/K$
- $M$  (paměť) = 128MB =  $2^{27}$
- $K$  (blok) = 128kB =  $2^{17}$
- $MM/K = 2^{27+2-17} = 2^{37} = 128\text{GB}$
- pokud to nestačí  
→ třífázové třídění

Třídění ► Třídění na disku

20 / 27

## Třífázové třídění

1. fáze stejně jako u dvoufázové třídění
  2. fáze třídíme pouze tolik proudů, kolik se vejde do paměti  
každý běh vytvoří nový větší proud
  3. fáze třídíme všechny "velké" proudy z 2. fáze
- DŮ: Jaké jsou limity třífázového třídění?  
Obecně (v praxi asi ne) pro větší  $n$  nemusí stačit 3 fáze, musíme tedy přidat další. Celkově  $\log_{M/B} B$  fází.

Třídění ► Třídění na disku

21 / 27

## Vylepšení TPMMS

- Jak setřídíte vojáky?
  - každého změříme, setřídíme na papíře, vyvoláme seřazené
  - nefunguje
  - náhodný přístup pro každého vojáka (záznam)
- komprese dat
  - funguje, až pro větší data (100 MB)
  - komprese na disku
  - komprese v paměti

Třídění ► Třídění na disku

22 / 27

## Nepoužitelný lineární algoritmus

Vytváření invertovaného souboru (reverzního indexu) rozsáhlého textu.

- pro každé slovo zaznamenáme všechny pozice výskytu
- víme, kam máme zapisovat, můžeme zapisovat přímo
- nepoužitelné

Třídění ► Třídění na disku

23 / 27

## Obsah

- 1 Třídění v paměti
- 2 Proveditelné algoritmy
- 3 Třídění na disku
- 4 Sublineární třídění

Třídění ► Sublineární třídění

24 / 27

## Můžeme třídit v méně než lineárním čase?

- na běžných počítačích **NE**
- změna modelu: masivní paralelizace
- máme dostatek počítačů pro libovolně velká data

## Google MapReduce

Obecný systém pro zpracování velkých dat.

Příklad třídění

- $10^{10}$  100bytových záznamů
- $10^{12}$  B = 1TB
- 840 sekund = 14 minut
- $10^9$  B/s = 1GB/s
- třídíme rychleji, než dokážeme číst z disku

## Závěry

- Pro praktickou proveditelnost algoritmů není důležitá asymptotická složitost, ale počet drahých operací (náhodných přístupů na disk).
- V "databázích" umíme pomocí TPMMS třídit v lineárním čase.
- Na paralelních systémech lze třídit i v sublineárním čase.