Datové struktury I

5. přednáška: Kompetitivita LRU strategie

Jirka Fink

https://ktiml.mff.cuni.cz/~fink/

Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta
Univerzita Karlova v Praze

Zimní semestr 2024/25

Cache-oblivious (Frigo, Leiserson, Prokop, Ramachandran, 1999)

Zjednodušený model paměti

- Uvažujme pouze na dvě úrovně paměti: pomalý disk a rychlá cache
- Paměť je rozdělená na bloky (stránky) velikosti B ①
- Velikost cache je M, takže cache má $P = \frac{M}{B}$ bloků
- Procesor může přistupovat pouze k datům uložených v cache
- Paměť je plně asociativní ②
- Data se mezi diskem a cache přesouvají po celých blocích a cílem je určit počet bloků načtených do cache

Cache-aware algoritmus

Algoritmus zná hodnoty M a B a podle nich nastavuje parametry (např. velikost vrcholu B-stromu při ukládání dat na disk).

Cache-oblivious algoritmus

Algoritmus musí efektivně fungovat bez znalostí hodnot M a B. Důsledky:

- Není třeba nastavovat parametry programu, který je tak přenositelnější
- Algoritmus dobře funguje mezi libovolnými úrovněmi paměti (L1 – L2 – L3 – RAM)

1. Pro zjednodušení předpokládáme, že jeden prvek zabírá jednotkový prostor, takže do jednoho bloku se vejde B prvků.
2. Předpokládáme, že každý blok z disku může být uložený na libovolné pozici v cache. Tento předpoklad výrazně zjednodušuje analýzu, i když na reálných počítačích moc neplatí, viz `https://en.wikipedia.org/wiki/CPU_cache#Associativity`.

Cache-oblivious analýza: Transpozice matic: Triviální přístup

Strategie pro výměnu stránek v cache

- OPT: Optimální off-line algoritmus předpokládající znalost všech přístupů do paměti
- FIFO: Z cache smažeme stránku, která je ze všech stránek v cachi nejdelší dobu
- LRU: Z cache smažeme stránku, která je ze všech stránek v cachi nejdéle nepoužitá

Srovnání LRU a OPT strategií

- I/O složist jsme počítali vůči OPT strategii
- O kolik je LRU strategie horší?

Cvičení

Najděte libovolně dlouhou posloupnost přístupů do paměti, při které má LRU strategie $\Theta(P)$-krát více přenesených bloků paměti než OPT.

Věta (Sleator, Tarjan, 1985)

- Nechť $s_1, \dots, s_k$ je posloupnost přístupů do paměti ①
- Nechť P_{OPT} a P_{LRU} je počet bloků v cache pro strategie OPT a LRU ②
- Nechť F_{OPT} a F_{LRU} je počet přenesených bloků ③
- $P_{\mathrm{LRU}} > P_{\mathrm{OPT}}$

Pak $F_{\mathrm{LRU}} \leq \frac{P_{\mathrm{LRU}}}{P_{\mathrm{LRU}} - P_{\mathrm{OPT}}} F_{\mathrm{OPT}} + P_{\mathrm{OPT}}$

Důsledek

Pokud LRU může uložit dvojnásobný počet bloků v cache oproti OPT, pak LRU má nejvýše dvojnásobný počet přenesených bloků oproti OPT (plus P_{OPT}). ④

Zdvojnásobení velikosti cache většinou nemá vliv na asymptotický počet přenesených bloků

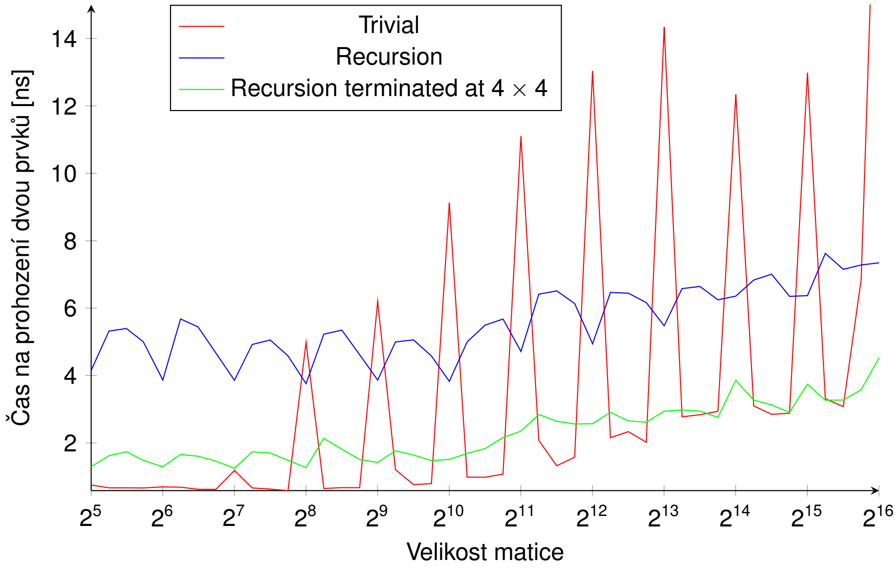
- Transpozice matic: $\mathcal{O}(n^2/B)$
- Mergesort: $\mathcal{O}\left(\frac{n}{B} \log \frac{n}{M}\right)$
- Funnelsort: $\mathcal{O}\left(\frac{n}{B} \log_P \frac{n}{B}\right)$
- The van Emde Boas layout: $\mathcal{O}(\log_B n)$

1. s_i značí blok paměti, se kterým program pracuje, a proto musí být načten do cache. Posloupnost $s_1, \dots, s_k$ je pořadí bloků paměti, ve kterém algoritmus pracuje s daty. Při opakovaném přístupu do stejného bloku se blok v posloupnosti opakuje.
2. Představme si, že OPT strategie pustíme na počítači s P_{OPT} bloky v cache a LRU strategie spustíme na počítači s P_{OPT} bloky v cache.
3. Srovnáváme počet přenesených bloků OPT strategie na počítači s P_{OPT} bloky a LRU strategie na počítači s P_{OPT} bloky.
4. Formálně: Jestliže $P_{\text{LRU}} = 2P_{\text{OPT}}$, pak $F_{\text{LRU}} \leq 2F_{\text{OPT}} + P_{\text{OPT}}$.

Důkaz ($F_{\text{LRU}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}} F_{\text{OPT}} + P_{\text{OPT}}$)

1. Rozdělíme posloupnost $s_1, \ldots, s_k$ na podposloupnosti tak, že LRU přenese P_{LRU} bloků v každé podposloupnosti kromě poslední
2. Jestliže v podposloupnosti s potřebuje LRU přenést P_{LRU} bloků, tak s obsahuje alespoň P_{LRU} různých bloků, a tedy OPT potřebuje alespoň $P_{\text{OPT}} - P_{\text{LRU}}$ přenosů
3. Jestliže F'_{OPT} and F'_{LRU} jsou počty přenesených bloků při zpracování podposloupnosti s, pak $F'_{\text{LRU}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}} F'_{\text{OPT}}$ (kromě poslední)
   - OPT přenese $F'_{\text{OPT}} \geq P_{\text{LRU}} - P_{\text{OPT}}$
   - LRU přenese $F'_{\text{LRU}} = P_{\text{LRU}}$
   - Dosazením dostáváme $\frac{F'_{\text{LRU}}}{F'_{\text{OPT}}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}}$
4. V poslední posloupnosti platí $F''_{\text{LRU}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}} F''_{\text{OPT}} + P_{\text{OPT}}$
   - OPT přenese $F''_{\text{OPT}} \geq F''_{\text{LRU}} - P_{\text{OPT}}$
   - Platí $1 \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}}$
   - Dosazením dostáváme $F''_{\text{LRU}} \leq F''_{\text{OPT}} + P_{\text{OPT}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}} F''_{\text{OPT}} + P_{\text{OPT}}$
5. Sečtením odhadů na F'_{LRU} a F''_{LRU} přes všechny podposloupnosti dostáváme $F_{\text{LRU}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}} F_{\text{OPT}} + P_{\text{OPT}}$

Doba transpozice matic na reálném počítači



Srovnání rychlosti čtení a zápisu z paměti

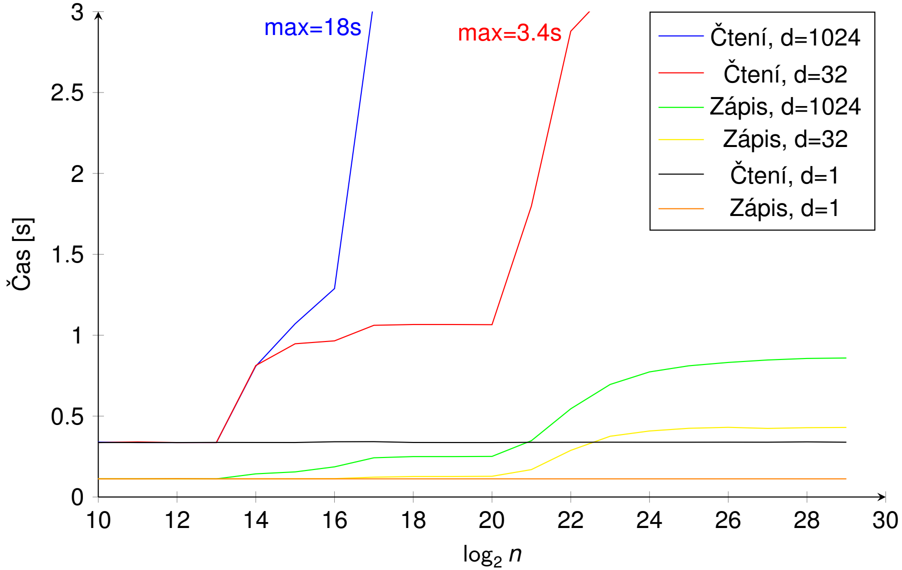
Čtení z paměti

```
# Inicializace pole 32-bitových čísel velikosti n
for (i=0; i+d<n; i+=d) do
    A[i] = i+d # Vezmeme každou d-tou pozici a vytvoříme cyklus
A[i=0]=0
# Měříme dobu průběhu cyklu v závislosti na parametrech n a d
for (j=0; j< 2^28; j++) do
    i = A[i] # Dokola procházíme cyklus d-tých pozic
```

Zápis do paměti

```
# Měříme dobu průběhu cyklu v závislosti na parametrech n a d
for (j=0; j< 2^28; j++) do
    A[(j*d) % n] = j # Dokola zapisujeme na d-té pozice
```

Srovnání rychlosti čtení a zápisu z paměti



Pár triků na závěr

Která varianta je rychlejší a o kolik?

```
# Použijeme modulo:
for (j=0; j< 2^28; j++) do
    A[(j*d) % n] = j
# Použijeme bitovou konjunkci:
mask = n − 1 # Předpokládáme, že n je mocnina dvojky
for (j=0; j< 2^28; j++) do
    A[(j*d) & mask] = j
```

Jak dlouho poběží výpočet vynecháme-li poslední řádek?

```
for (i=0; i+d<n; i+=d) do
    A[i] = i+d
A[i=0]=0
# Měříme dobu průběhu cyklu v závislosti na parametrech n a d
for (j=0; j< 2^28; j++) do
    i = A[i]
printf("%d\n", i);
```

Hešovaní

Základní pojmy

- Máme univerzum U všech prvků
- Chceme uložit podmnožinu $S \subseteq U$ velikosti n
- Uložíme S do pole velikosti m pomocí hešovací funkce $h : U \to M$, kde $M = \{0, 1, \ldots, m-1\}$
- Dva prvky $x, y \in S$ kolidují, jestliže $h(x) = h(y)$
- Hešovací funkce h je perfektní na S, jestliže h nemá žádnou kolizi S

Separované řetězce

- Vytvoříme tabulku velikost $m \approx n$ a hešovací funkci $h : U \to M$
- $M[y]$ obsahuje spojový seznam prvků x splňující $h(x) = y$
- INSERT(x): Přidáme prvek x do seznamu $M[h(x)]$
- FIND(x): Projdeme seznam $M[h(x)]$
- DELETE(x): Smažeme prvek ze seznamu $M[h(x)]$

Otázky

- Jak získat hešovací funkci?
- Jaké jsou další způsoby řešení kolizí?

Proč nestačí zvolit jednu triviální funkci?

Funkce $h(x) = x \mod m$

- Pokud hešujeme náhodná data, tak tato funkce stačí
- Pokud jsou prvky násobky m, pak padnou do stejné přihrádky
- V praxi nikdy nedostaneme dostatečně náhodná data

Pozorování (Nepřátelská podmnožina)

Pokud $|U| \geq mn$, pak pro každou hešovací funkci h existuje $S \subseteq U$ velikosti n taková, že h hešuje všechny prvky z S do jedné přihrádky. ①

Proč nestačí jedna hešovací funkce?

- Pokud nepřítel zná naši hešovací funkci (open source), tak si může předpočítat kolidující prvky
- Common Vulnerabilities and Exposures:
  - PHP: CVE-2011-4885
  - Ruby: CVE-2011-4815
  - Apache Geronimo: CVE-2011-5034
- Musíme zkonstruovat systém hešovacích funkcí, ze které budeme náhodně vybírat

1. Dirichletův princip (Balls-and-binds): Pokud hodíme mn míčů do m přihrádek (košů), tak v alespoň jedné přihrádce bude alespoň n míčů, které označíme S.

Universální hešování

Cíl

Sestrojit systém $\mathcal{H}$ hešovacích funkcí $f : U \to M$ takový, že náhodně zvolená funkce $f \in \mathcal{H}$ hešuje libovolnou množinu S „většinou dobře".

Úplně náhodná hešovací funkce

- Systém $\mathcal{H}$ obsahuje všechny funkce $f : U \to M$
- Platí $P[h(x) = z] = \frac{1}{m}$ pro všechna $x \in U$ a $z \in M$
- Náhodné přihrádky $h(x)$ a $h(y)$ jsou nezávislé pro různé $x, y \in U$
- Nepraktické: k zakódování funkce z $\mathcal{H}$ potřebujeme $\Theta(|U| \log m)$ bitů
- Někdy se používá k analýze hešování

c-universální hešovací systém

c-universální systém (ekvivalentní definice)

Systém hešovacích funkcí $\mathcal{H}$ je c-universální, jestliže ①

- počet hešovacích funkcí $h \in \mathcal{H}$ splňujících $h(x) = h(y)$ je nejvýše $\frac{c|\mathcal{H}|}{m}$ pro všechna různá $x, y \in U$
- náhodně zvolená $h \in \mathcal{H}$ splňuje $P[h(x) = h(y)] \leq \frac{c}{m}$ pro každé $x, y \in U$ a $x \neq y$. ② ③

Příklad c-universálního hešovacího systému

- Parametry: p a m, kde $p > u$ je prvočíslo
- Hešovací funkce

$$h_a(x) = (ax \mod p) \mod m$$

  je závislá na hodnotě a
- Hešovací systém $\mathcal{H} = \{h_a;\ 0 < a < p\}$ je c-universální
- Hešovací funkce ze systému $\mathcal{H}$ je určena hodnotou a
- Tedy náhodný výběr hešovací funkce z $\mathcal{H}$ je náhodné vygenerování $a \in \{1, \ldots, p-1\}$

1. Navíc obvykle vyžadujeme, aby hešovací funkci šlo spočítat v čase $\mathcal{O}(1)$ a aby funkci bylo možné popsat $\mathcal{O}(1)$ parametry.
2. Náhodný výběr hešovací funkce má vždy rovnoměrné rozdělení na celém systému.
3. Úplně náhodný hešovací systém je 1-universální, protože $h(x)$ padne do nějaké přihrádky a $h(y)$ má uniformní distribuci nezávislou na $h(x)$, a proto $P[h(x) = h(y)] = \frac{1}{m}$.

Kolik máme očekávat kolizí?

Pozorování (Narozeninový paradox)

Pokud n míčů hodíme do $m \geq n$ košů, pak pravděpodobnost, že v každém koši je nejvýše jeden míč, je

$$\prod_{i=1}^{n-1} \frac{m-i}{m} \sim e^{-\frac{n^2}{2m}}$$

a očekávaný počet kolizí je

$$\binom{n}{2}\frac{1}{m} \sim \frac{n^2}{2m}.$$

Důkaz

- $\prod_{i=0}^{n-1} \frac{m-i}{m} = \prod_{i=1}^{n-1} \left(1 + \frac{-i}{m}\right) \sim \prod_{i=1}^{n-1} e^{-\frac{i}{m}} = e^{-\frac{\sum_{i=1}^{n-1} i}{m}} = e^{-\frac{\binom{n}{2}}{m}} \sim e^{-\frac{n^2}{2m}}$ ①
- $E[\#\text{ kolizí}] = \sum_{\{x,y\}} P[h(x) = h(y)] = \binom{n}{2}\frac{1}{m} < \frac{n^2}{2m}$ ②

1. Předpokládáme, že se každým míčem trefíme do právě jednoho koše, do každého koše se trefíme se stejnou pravděpodobností a jednotlivé hody jsou nezávislé. i-tý míč padne do prázdného koše s pravděpodobností $\frac{m-i+1}{m}$, takže pravděpodobnost, že v každém koši bude nejvýše jeden míč, je $\prod_{i=1}^{n-1} \frac{m-i}{m}$. Použitím aproximaci prvního řádu funkce $e^x \sim 1 + x$ dostáváme

$$\prod_{i=1}^{n-1} \left(1 + \frac{-i}{m}\right) \sim \prod_{i=1}^{n-1} e^{-\frac{i}{m}} = e^{-\frac{\sum_{i=1}^{n-1} i}{m}} = e^{-\frac{\binom{n}{2}}{m}} \sim e^{-\frac{n^2}{2m}}.$$

2. Pravděpodobnost kolize dvou prvků je $1/m$ a počet dvojic různých prvků je $\binom{n}{2}$. Důkaz plyne z linearity střední hodnoty.

Statické perfektní hešování

Lemma (cvičení)

Čekáme-li na událost, která nastane v jednom kroku s pravděpodobností p (nezávisle na ostatních krocích), pak $E[\#\text{ kroků}] = \frac{1}{p}$.

Markovova nerovnost

Jestliže X je nezáporná náhodná veličina a $d > 1$, pak $P[X < dE[X]] > 1 - \frac{1}{d}$.

Pozorování: Statické perfektní hešování

Pro danou podmnožinu $S \subseteq U$ velikosti n lze najít perfektní hešovací funkci do tabulky velikosti $m = \Omega(n^2)$ tak, že vyzkoušíme v průměru $\mathcal{O}(1)$ funkcí z c-universálního systému.

Důkaz

- Předpokládejme $m \geq an^2$ a nechť X značí počet kolizí
- $E[X] = \sum_{\{x,y\}} P[h(x) = h(y)] \leq \binom{n}{2} \frac{c}{m} < \frac{n^2}{2} \frac{c}{an^2} = \frac{c}{2a}$
- Markov: $P[X < 1] > P[X < \frac{2a}{c} E[X]] > 1 - \frac{c}{2a}$
- Očekávaný počet pokusů na nalezení perfektní hešovací funkce je nejvýše $\frac{1}{1-c/2a}$

Plán na další 3 přednášky

- Volba hešovací funkce
  - Jak hešovat čísla, řetězce, k-tice, pole, ...
- Řešení kolizí
  - Separované řetězce
  - Lineární přidávání
  - Kukaččí hešování
  - Bloom filtry