

Datové struktury I

4. přednáška: Paměťová hierarchie

Jirka Fink

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

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

Zimní semestr 2026/27

Licence: Creative Commons BY-NC-SA 4.0

Zjednodušený model paměti

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

Zjednodušený model paměti

- Uvažujeme pouze dvě úrovně paměti: pomalý disk a rychlou keš (cache)
- Paměť je rozdělena na bloky (stránky) velikosti B
- Velikost keše je M , takže keš má $P = \frac{M}{B}$ bloků
- Procesor může přistupovat pouze k datům uloženým v keši
- Paměť je plně asociativní
- Data se mezi diskem a keší přesouvají po celých blocích a cílem je určit počet bloků načtených do keše

Cache-aware algoritmus

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

Zjednodušený model paměti

- Uvažujeme pouze dvě úrovně paměti: pomalý disk a rychlou keš (cache)
- Paměť je rozdělena na bloky (stránky) velikosti B
- Velikost keše je M , takže keš má $P = \frac{M}{B}$ bloků
- Procesor může přistupovat pouze k datům uloženým v keši
- Paměť je plně asociativní
- Data se mezi diskem a keší přesouvají po celých blocích a cílem je určit počet bloků načtených do keše

Cache-aware algoritmus

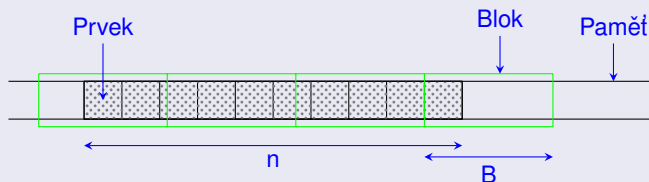
Algoritmus zná hodnoty M a B a podle nich nastavuje parametry (např. velikost vrcholu B-stromu při ukládání dat na disk).

Cache-oblivious algoritmus

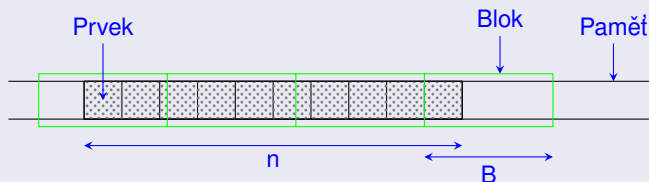
Algoritmus musí efektivně fungovat bez znalostí hodnot M a B . Důsledky:

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

Přečtení souvislého pole (výpočet maxima, součtu a podobně)

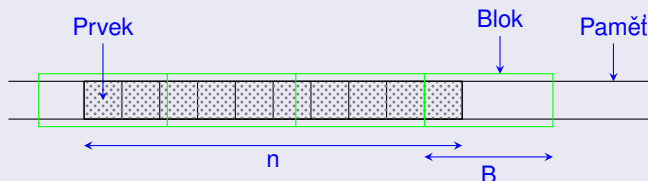


Přečtení souvislého pole (výpočet maxima, součtu a podobně)



- Minimální možný počet přenesených bloků je $\lceil n/B \rceil$
- Skutečný počet přenesených bloků je nejvýše $\lceil n/B \rceil + 1$
- Předpokládáme, že máme k dispozici $\mathcal{O}(1)$ registrů k uložení iterátoru a maxima

Přečtení souvislého pole (výpočet maxima, součtu a podobně)



- Minimální možný počet přenesených bloků je $\lceil n/B \rceil$
- Skutečný počet přenesených bloků je nejvýše $\lceil n/B \rceil + 1$
- Předpokládáme, že máme k dispozici $\mathcal{O}(1)$ registrů k uložení iterátoru a maxima

Obrácení pole

- Potřebujeme si v keši držet dva bloky, jinak je analýza stejná jako u přečtení souvislého pole
- Skutečný počet přenesených bloků je nejvýše $\lceil n/B \rceil + 1$ za předpokladu, že $P \geq 2$

Binární vyvážený strom

- Jeden vrchol je uložen v nejvýše 2 blocích
- Výška stromu je $\mathcal{O}(\log n)$
- Na cestě z kořene do listu načteme $\mathcal{O}(\log n)$ bloků

Binární vyvážený strom

- Jeden vrchol je uložen v nejvýše 2 blocích
- Výška stromu je $\mathcal{O}(\log n)$
- Na cestě z kořene do listu načteme $\mathcal{O}(\log n)$ bloků

(a,b)-strom

- Předpokládejme, že můžeme zvolit $b = \Theta(B)$
- Jeden vrchol je uložen v nejvýše 2 blocích a má $\Theta(B)$ synů
- Výška stromu je $\Theta(\log_B n)$
- Na cestě z kořene do listu načteme $\Theta(1 + \log_B n)$ bloků
- Toto je cache-aware přístup

Binární vyvážený strom

- Jeden vrchol je uložen v nejvýše 2 blocích
- Výška stromu je $\mathcal{O}(\log n)$
- Na cestě z kořene do listu načteme $\mathcal{O}(\log n)$ bloků

(a,b)-strom

- Předpokládejme, že můžeme zvolit $b = \Theta(B)$
- Jeden vrchol je uložen v nejvýše 2 blocích a má $\Theta(B)$ synů
- Výška stromu je $\Theta(\log_B n)$
- Na cestě z kořene do listu načteme $\Theta(1 + \log_B n)$ bloků
- Toto je cache-aware přístup

Reprezentace binárního stromu pomocí van Emde Boas

- Cache-oblivious reprezentace
- Na cestě z kořene do listu načteme $\Theta(1 + \log_B n)$ bloků
- Podrobnosti na Datových strukturách II

Binární halda v poli: Průchod od kořene k listu



Binární halda v poli: Průchod od kořene k listu



- Cesta má $\Theta(\log n)$ vrcholů
- Posledních $\Theta(\log B)$ vrcholů leží v nejvýše dvou blocích
- Ostatní vrcholy jsou uloženy v po dvou různých blocích
- $\Theta(1 + \log n - \log B) = \Theta(1 + \log \frac{n}{B})$ přenesených bloků

Binární halda v poli: Průchod od kořene k listu



- Cesta má $\Theta(\log n)$ vrcholů
- Posledních $\Theta(\log B)$ vrcholů leží v nejvýše dvou blocích
- Ostatní vrcholy jsou uloženy v po dvou různých blocích
- $\Theta(1 + \log n - \log B) = \Theta(1 + \log \frac{n}{B})$ přenesených bloků

B-regulární halda v poli: Průchod od kořene k listu

- Opět předpokládáme, že do jednoho bloku se vejde B prvků
- Cesta má $\Theta(1 + \log_B n)$ vrcholů
- Na cestě z kořene do listu načteme $\Theta(1 + \log_B n)$ bloků

Binární vyhledávání

- Uvažujeme neúspěšné vyhledávání, protože úspěšné může skončit dříve
- Porovnáváme $\Theta(\log n)$ prvků s hledaným prvkem
- Posledních $\Theta(\log B)$ prvků je uloženo v nejvýše dvou blocích
- Ostatní prvky jsou uloženy v po dvou různých blocích
- $\Theta(1 + \log \frac{n}{B})$ přenesených bloků

Mergesort

- 1 Slití polí velikostí n_1 a n_2 potřebuje $\frac{2}{B}(n_1 + n_2) + \mathcal{O}(1)$ přenosů

Mergesort

- 1 Slití polí velikostí n_1 a n_2 potřebuje $\frac{2}{B}(n_1 + n_2) + \mathcal{O}(1)$ přenosů
- 2 Uvažujme (binární) strom rekurzivního volání
- 3 Na jedné hladině stromu potřebujeme $\mathcal{O}(n/B + 1)$ přenosů
- 4 Počet hladin stromu je $\mathcal{O}(\log n)$
- 5 Celkový počet přenosů je $\mathcal{O}((n/B + 1) \log n)$

Mergesort

- 1 Slití polí velikostí n_1 a n_2 potřebuje $\frac{2}{B}(n_1 + n_2) + \mathcal{O}(1)$ přenosů
- 2 Uvažujme (binární) strom rekurzivního volání
- 3 Na jedné hladině stromu potřebujeme $\mathcal{O}(n/B + 1)$ přenosů
- 4 Počet hladin stromu je $\mathcal{O}(\log n)$
- 5 Celkový počet přenosů je $\mathcal{O}((n/B + 1) \log n)$

k -cestný mergesort

- 1 Slití polí velikostí $n_1, \dots, n_k$ potřebuje $\frac{2}{B}(n_1 + \dots + n_k) + \mathcal{O}(1)$ přenosů
- 2 Ke slévání potřebujeme k vstupních bloků a jeden výstupní blok, proto volíme $k + 1 \leq B$

Mergesort

- 1 Slití polí velikostí n_1 a n_2 potřebuje $\frac{2}{B}(n_1 + n_2) + \mathcal{O}(1)$ přenosů
- 2 Uvažujme (binární) strom rekurzivního volání
- 3 Na jedné hladině stromu potřebujeme $\mathcal{O}(n/B + 1)$ přenosů
- 4 Počet hladin stromu je $\mathcal{O}(\log n)$
- 5 Celkový počet přenosů je $\mathcal{O}((n/B + 1) \log n)$

k -cestný mergesort

- 1 Slití polí velikostí $n_1, \dots, n_k$ potřebuje $\frac{2}{B}(n_1 + \dots + n_k) + \mathcal{O}(1)$ přenosů
- 2 Ke slévání potřebujeme k vstupních bloků a jeden výstupní blok, proto volíme $k + 1 \leq P$
- 3 Uvažujme k -ární strom rekurzivního volání
- 4 Na jedné hladině stromu potřebujeme $\mathcal{O}(n/B + 1)$ přenosů
- 5 Počet hladin stromu je $\mathcal{O}(\log_k n)$
- 6 Celkový počet přenosů je $\mathcal{O}((n/B + 1) \log_k n)$
- 7 Volbou $k = P - 1$ dostáváme $\mathcal{O}((n/B + 1) \log_P n)$ přenosů

Případ $n \leq M/2$

Celé pole se vejde do keše, takže přenášíme $2n/B + \mathcal{O}(1)$ bloků.

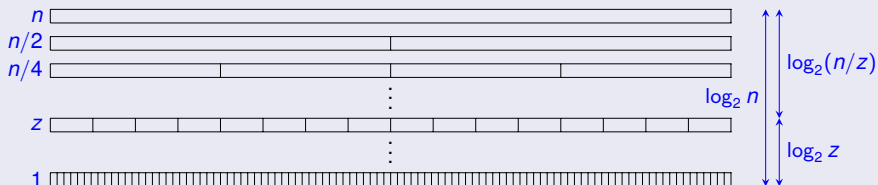
Případ $n \leq M/2$

Celé pole se vejde do keše, takže přenášíme $2n/B + \mathcal{O}(1)$ bloků.

Schéma

Délka spojovaných polí

Výška stromu rekurze



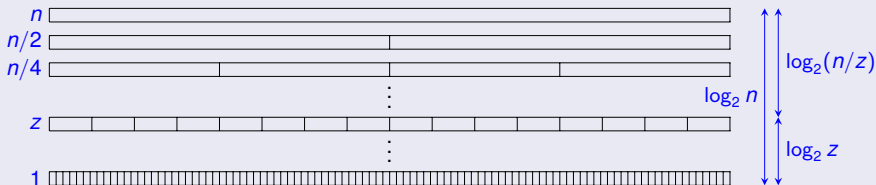
Případ $n \leq M/2$

Celé pole se vejde do keše, takže přenášíme $2n/B + \mathcal{O}(1)$ bloků.

Schéma

Délka spojovaných polí

Výška stromu rekurze



Případ $n > M/2$

- 1 Nechť z je maximální velikost pole, které lze v keši setřídit
- 2 Platí $z \leq \frac{M}{2} < 2z$

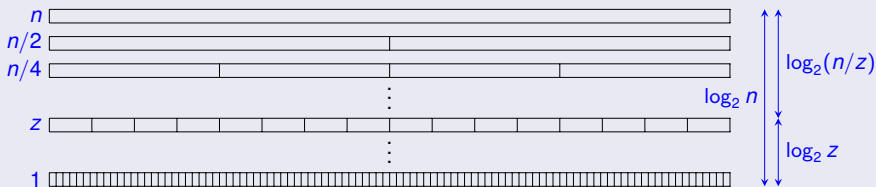
Případ $n \leq M/2$

Celé pole se vejde do keše, takže přenášíme $2n/B + \mathcal{O}(1)$ bloků.

Schéma

Délka spojovaných polí

Výška stromu rekurze



Případ $n > M/2$

- 1 Nechť z je maximální velikost pole, které lze v keši setřídít
- 2 Platí $z \leq \frac{M}{2} < 2z$
- 3 Slití jedné úrovně vyžaduje $2\frac{n}{B} + 2\frac{n}{z} + \mathcal{O}(1) = \mathcal{O}(\frac{n}{B})$ přenosů.
- 4 Počet přenesených bloků je $\mathcal{O}(\frac{n}{B}) (1 + \log_2 \frac{n}{z}) = \mathcal{O}(\frac{n}{B} \log \frac{n}{M})$.

Strategie nahrazování stránek v keši

- OPT:** Optimální off-line algoritmus předpokládající znalost všech přístupů do paměti
- FIFO:** Z keše smažeme stránku, která v keši zůstává ze všech stránek nejdéle
- LRU:** Z keše smažeme stránku, která nebyla ze všech stránek v keši nejdéle použita

Strategie nahrazování stránek v keši

- OPT:** Optimální off-line algoritmus předpokládající znalost všech přístupů do paměti
- FIFO:** Z keše smažeme stránku, která v keši zůstává ze všech stránek nejdéle
- LRU:** Z keše smažeme stránku, která nebyla ze všech stránek v keši nejdéle použita

Triviální algoritmus pro transpozici matice A velikosti $k \times k$

```
1 for  $i \leftarrow 1$  to  $k$  do  
2   for  $j \leftarrow i + 1$  to  $k$  do  
3      $\text{Swap}(A_{ij}, A_{ji})$ 
```

Strategie nahrazování stránek v keši

- OPT:** Optimální off-line algoritmus předpokládající znalost všech přístupů do paměti
- FIFO:** Z keše smažeme stránku, která v keši zůstává ze všech stránek nejdéle
- LRU:** Z keše smažeme stránku, která nebyla ze všech stránek v keši nejdéle použita

Triviální algoritmus pro transpozici matice A velikosti $k \times k$

```
1 for  $i \leftarrow 1$  to  $k$  do  
2   for  $j \leftarrow i + 1$  to  $k$  do  
3      $\text{Swap}(A_{ij}, A_{ji})$ 
```

Předpoklady

Uvažujeme pouze případ, kdy

- $B < k$: Do jednoho bloku keše se nevejde celý řádek matice
- $P < k$: Do keše se nevejde celý sloupec matice

Příklad: Reprezentace matice 5×5 v paměti

11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45	51	52	53	54	55

Příklad: Reprezentace matice 5×5 v paměti

11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45	51	52	53	54	55
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

LRU a FIFO strategie

Při čtení matice po sloupcích si keš pamatuje posledních P řádků, takže při čtení prvku $A_{3,2}$ prvek $A_{3,1}$ již v keši není. Počet přenesených bloků je $\Omega(k^2)$.

Příklad: Reprezentace matice 5×5 v paměti

11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45	51	52	53	54	55
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

LRU a FIFO strategie

Při čtení matice po sloupcích si keš pamatuje posledních P řádků, takže při čtení prvku $A_{3,2}$ prvek $A_{3,1}$ již v keši není. Počet přenesených bloků je $\Omega(k^2)$.

OPT strategie

- 1 Transpozice prvního řádku/sloupce vyžaduje alespoň $k - 1$ přenosů.
- 2 Nejvýše P prvků z druhého sloupce zůstane v keši.
- 3 Proto transpozice druhého řádku/sloupce vyžaduje alespoň $k - P - 2$ přenosů.
- 4 Transpozice i -tého řádku/sloupce vyžaduje alespoň $\max\{0, k - P - i\}$ přenosů.
- 5 Celkový počet přenosů je alespoň $\sum_{i=1}^{k-P} (k - P - i) = \Omega((k - P)^2)$.

Cache-aware algoritmus pro transpozici matice A velikosti $k \times k$

```
# Rozdělíme danou matici na submatice velikosti  $z \times z$ 
1 for ( $i = 0; i < k; i += z$ ) do
2   for ( $j = i; j < k; j += z$ ) do
3     # Transponujeme submatici začínající na pozici  $(i, j)$ 
4     for ( $ii = i; ii < \min(k, i + z); ii ++$ ) do
5       for ( $jj = \max(j, ii + 1); jj < \min(k, j + z); jj ++$ ) do
        Swap( $A_{ii,jj}, A_{jj,ii}$ )
```

Cache-aware algoritmus pro transpozici matice A velikosti $k \times k$

```
# Rozdělíme danou matici na submatice velikosti  $z \times z$ 
1 for ( $i = 0; i < k; i += z$ ) do
2   for ( $j = i; j < k; j += z$ ) do
3     # Transponujeme submatici začínající na pozici  $(i, j)$ 
4     for ( $ii = i; ii < \min(k, i + z); ii ++$ ) do
5       for ( $jj = \max(j, ii + 1); jj < \min(k, j + z); jj ++$ ) do
        Swap( $A_{ii,jj}, A_{jj,ii}$ )
```

Analýza

- Zvolíme $z = B$ a předpokládáme, že $4B \leq P$
- K transpozici jedné submatice potřebujeme $\mathcal{O}(z)$ přenosů
- Počet submatic je $\mathcal{O}((k/z)^2)$
- K transpozici potřebujeme $\mathcal{O}(k^2/B)$ přenosů, což je optimální
- Při správně zvolené hodnotě z bývá tento postup nejrychlejší

Idea

Rekurzivně rozdělíme matici na submatice

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad A^T = \begin{pmatrix} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{pmatrix}$$

- Matice A_{11} a A_{22} se transponují podle stejného schématu
- Matice A_{12} a A_{21} se prohazují
- Transpozice a prohození matic A_{12} a A_{21} se provádí najednou

```
1 Procedure transpose_on_diagonal ( $A$ )
2   if Matice  $A$  je malá then
3     Transponujeme matici  $A$  triviálním postupem
4   else
5      $A_{11}, A_{12}, A_{21}, A_{22} \leftarrow$  souřadnice submatic
6     transpose_on_diagonal ( $A_{11}$ )
7     transpose_on_diagonal ( $A_{22}$ )
8     transpose_and_swap ( $A_{12}, A_{21}$ )
9 Procedure transpose_and_swap ( $A, B$ )
10  if Matice  $A$  a  $B$  jsou malé then
11    Prohodíme a transponujeme matice  $A$  a  $B$  triviálním postupem
12  else
13     $A_{11}, A_{12}, A_{21}, A_{22}, B_{11}, B_{12}, B_{21}, B_{22} \leftarrow$  souřadnice submatic
14    transpose_and_swap ( $A_{11}, B_{11}$ )
15    transpose_and_swap ( $A_{12}, B_{21}$ )
16    transpose_and_swap ( $A_{21}, B_{12}$ )
17    transpose_and_swap ( $A_{22}, B_{22}$ )
```

Analýza počtu přenesených bloků

- 1 Předpoklad „Tall cache“: $M \geq 4B^2$, tj. počet bloků je alespoň $4B$
- 2 Nechť z je největší velikost submatice, jejíž jeden řádek se vejde do jednoho bloku
- 3 Platí: $z \leq B \leq 2z$

Analýza počtu přenesených bloků

- 1 Předpoklad „Tall cache“: $M \geq 4B^2$, tj. počet bloků je alespoň $4B$
- 2 Nechť z je největší velikost submatice, jejíž jeden řádek se vejde do jednoho bloku
- 3 Platí: $z \leq B \leq 2z$
- 4 Jedna submatice $z \times z$ je uložena v nejvýše $2z \leq 2B$ blocích
- 5 Dvě submatice $z \times z$ se vejdou do keše
- 6 Transpozice matice typu $z \times z$ vyžaduje nejvýše $4z$ přenosů

Analýza počtu přenesených bloků

- 1 Předpoklad „Tall cache“: $M \geq 4B^2$, tj. počet bloků je alespoň $4B$
- 2 Nechť z je největší velikost submatice, jejíž jeden řádek se vejde do jednoho bloku
- 3 Platí: $z \leq B \leq 2z$
- 4 Jedna submatice $z \times z$ je uložena v nejvýše $2z \leq 2B$ blocích
- 5 Dvě submatice $z \times z$ se vejdou do keše
- 6 Transpozice matice typu $z \times z$ vyžaduje nejvýše $4z$ přenosů
- 7 Máme $(k/z)^2$ submatic velikosti $z \times z$
- 8 Celkový počet přenesených bloků je nejvýše $\frac{k^2}{z^2} \cdot 4z \leq \frac{8k^2}{B} = \mathcal{O}\left(\frac{k^2}{B}\right)$
- 9 Tento postup je optimální až na multiplikativní faktor

Věta (Sleator, Tarjan, 1985)

- Nechť $s_1, \dots, s_k$ je posloupnost přístupů do paměti
- Nechť P_{OPT} a P_{LRU} jsou počty bloků v keši pro strategie OPT a LRU
- Nechť F_{OPT} a F_{LRU} jsou počty přenesených bloků
- $P_{\text{LRU}} > P_{\text{OPT}}$

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

Věta (Sleator, Tarjan, 1985)

- Nechť $s_1, \dots, s_k$ je posloupnost přístupů do paměti
- Nechť P_{OPT} a P_{LRU} jsou počty bloků v keši pro strategie OPT a LRU
- Nechť F_{OPT} a F_{LRU} jsou počty přenesených bloků
- $P_{\text{LRU}} > P_{\text{OPT}}$

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

Důsledek

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

Věta (Sleator, Tarjan, 1985)

- Nechť $s_1, \dots, s_k$ je posloupnost přístupů do paměti
- Nechť P_{OPT} a P_{LRU} jsou počty bloků v keši pro strategie OPT a LRU
- Nechť F_{OPT} a F_{LRU} jsou počty přenesených bloků
- $P_{\text{LRU}} > P_{\text{OPT}}$

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

Důsledek

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

Zdvojnásobení velikosti keše většinou nemá vliv na asymptotický počet přenesených bloků

- Scanning: $\mathcal{O}(n/B + 1)$
- Mergesort: $\mathcal{O}(\frac{n}{B} \log \frac{n}{M} + 1)$
- Funnelsort: $\mathcal{O}(\frac{n}{B} \log_P \frac{n}{B} + 1)$
- Transpozice matice: $\mathcal{O}(n^2/B + 1)$

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

- 1 Pokud LRU má $f \leq P_{\text{LRU}}$ přenesených bloků v podposloupnosti s , pak OPT přeneše alespoň $f - P_{\text{OPT}}$ bloků v podposloupnosti s
 - Pokud LRU načte v podposloupnosti f různých bloků, pak podposloupnost obsahuje alespoň f různých bloků
 - Pokud LRU načte v podposloupnosti jeden blok dvakrát, pak podposloupnost obsahuje alespoň $P_{\text{LRU}} \geq f$ různých bloků
 - OPT má před zpracováním podposloupnosti nejvýše P_{OPT} bloků z podposloupnosti v keši a zbylých alespoň $f - P_{\text{OPT}}$ musí načíst

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

- 1 Pokud LRU má $f \leq P_{\text{LRU}}$ přenesených bloků v podposloupnosti s , pak OPT přenesse alespoň $f - P_{\text{OPT}}$ bloků v podposloupnosti s
 - Pokud LRU načte v podposloupnosti f různých bloků, pak podposloupnost obsahuje alespoň f různých bloků
 - Pokud LRU načte v podposloupnosti jeden blok dvakrát, pak podposloupnost obsahuje alespoň $P_{\text{LRU}} \geq f$ různých bloků
 - OPT má před zpracováním podposloupnosti nejvýše P_{OPT} bloků z podposloupnosti v keši a zbylých alespoň $f - P_{\text{OPT}}$ musí načíst
- 2 Rozdělíme posloupnost $s_1, \dots, s_k$ na podposloupnosti tak, že LRU přenesse P_{LRU} bloků v každé podposloupnosti (kromě poslední)

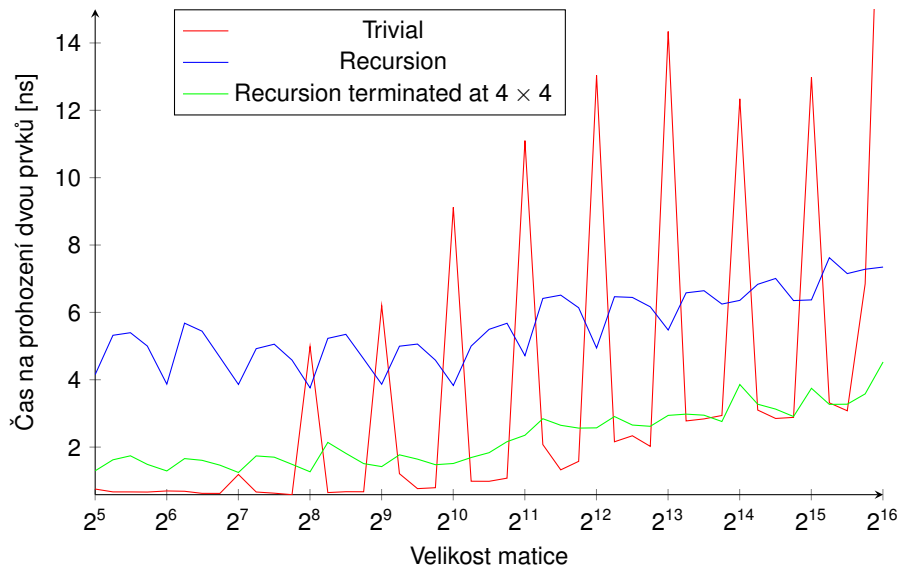
Důkaz ($F_{\text{LRU}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}} F_{\text{OPT}} + P_{\text{OPT}}$)

- ❶ Pokud LRU má $f \leq P_{\text{LRU}}$ přenesených bloků v podposloupnosti s , pak OPT přeneše alespoň $f - P_{\text{OPT}}$ bloků v podposloupnosti s
 - Pokud LRU načte v podposloupnosti f různých bloků, pak podposloupnost obsahuje alespoň f různých bloků
 - Pokud LRU načte v podposloupnosti jeden blok dvakrát, pak podposloupnost obsahuje alespoň $P_{\text{LRU}} \geq f$ různých bloků
 - OPT má před zpracováním podposloupnosti nejvýše P_{OPT} bloků z podposloupnosti v keši a zbylých alespoň $f - P_{\text{OPT}}$ musí načíst
- ❷ Rozdělíme posloupnost $s_1, \dots, s_k$ na podposloupnosti tak, že LRU přeneše P_{LRU} bloků v každé podposloupnosti (kromě poslední)
- ❸ Jestliže F'_{OPT} a F'_{LRU} jsou počty přenesených bloků při zpracování libovolné podposloupnosti kromě poslední, pak $F'_{\text{LRU}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}} F'_{\text{OPT}}$
 - OPT přeneše alespoň $P_{\text{LRU}} - P_{\text{OPT}}$ bloků v každé podposloupnosti
 - Tedy $\frac{F'_{\text{LRU}}}{F'_{\text{OPT}}} \leq \frac{P_{\text{LRU}}}{P_{\text{LRU}} - P_{\text{OPT}}}$

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

- ❶ Pokud LRU má $f \leq P_{LRU}$ přenesených bloků v podposloupnosti s , pak OPT přeneše alespoň $f - P_{OPT}$ bloků v podposloupnosti s
 - Pokud LRU načte v podposloupnosti f různých bloků, pak podposloupnost obsahuje alespoň f různých bloků
 - Pokud LRU načte v podposloupnosti jeden blok dvakrát, pak podposloupnost obsahuje alespoň $P_{LRU} \geq f$ různých bloků
 - OPT má před zpracováním podposloupnosti nejvýše P_{OPT} bloků z podposloupnosti v keši a zbylých alespoň $f - P_{OPT}$ musí načíst
- ❷ Rozdělíme posloupnost $s_1, \dots, s_k$ na podposloupnosti tak, že LRU přeneše P_{LRU} bloků v každé podposloupnosti (kromě poslední)
- ❸ Jestliže F'_{OPT} a F'_{LRU} jsou počty přenesených bloků při zpracování libovolné podposloupnosti kromě poslední, pak $F'_{LRU} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F'_{OPT}$
 - OPT přeneše alespoň $P_{LRU} - P_{OPT}$ bloků v každé podposloupnosti
 - Tedy $\frac{F'_{LRU}}{F'_{OPT}} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}}$
- ❹ V poslední podposloupnosti platí $F''_{LRU} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F''_{OPT} + P_{OPT}$
 - Platí $F''_{OPT} \geq F''_{LRU} - P_{OPT}$ a $1 \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}}$
 - Tedy $F''_{LRU} \leq F''_{OPT} + P_{OPT} \leq \frac{P_{LRU}}{P_{LRU} - P_{OPT}} F''_{OPT} + P_{OPT}$

Doba transpozice matic na reálném počítači



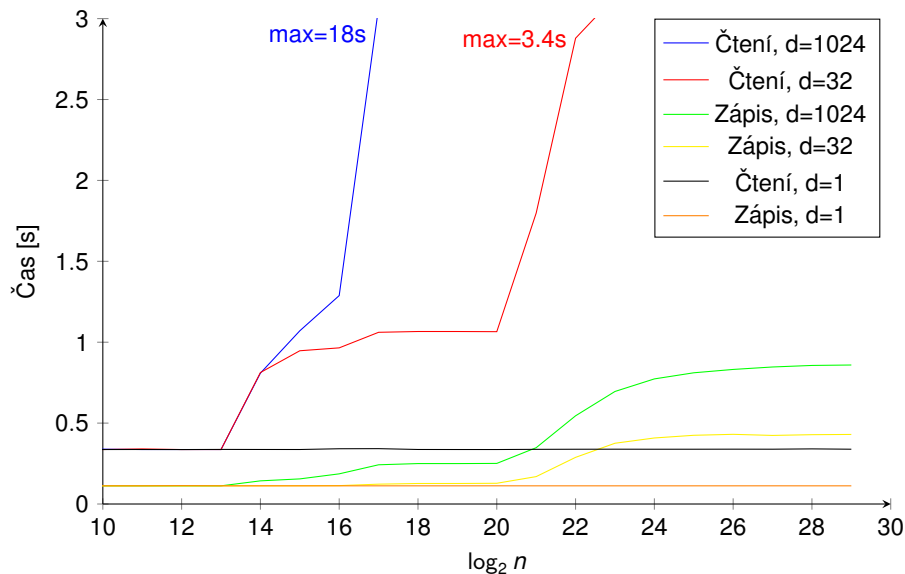
Čtení z paměti

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

Zápis do paměti

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

Srovnání rychlosti čtení z paměti a zápisu do paměti



Která varianta je rychlejší a o kolik?

```
# Použijeme modulo:
1 for ( $j=0; j < 2^{28}; j++$ ) do
2   |  $A[j*d \% n] = j$ 

# Použijeme bitovou konjunkci:
3  $mask = n - 1$  # Předpokládáme, že  $n$  je mocnina dvojky
4 for ( $j=0; j < 2^{28}; j++$ ) do
5   |  $A[j*d \& mask] = j$ 
```

Která varianta je rychlejší a o kolik?

```
# Použijeme modulo:
1 for ( $j=0; j < 2^{28}; j++$ ) do
2   |  $A[j*d \% n] = j$ 

# Použijeme bitovou konjunkci:
3  $mask = n - 1$  # Předpokládáme, že  $n$  je mocnina dvojky
4 for ( $j=0; j < 2^{28}; j++$ ) do
5   |  $A[j*d \& mask] = j$ 
```

Jak dlouho poběží výpočet, vynecháme-li poslední řádek?

```
1 for ( $i=0; i+d < n; i+=d$ ) do
2   |  $A[i] = i+d$ 
3  $A[i=0]=0$ 
# Měříme dobu průběhu cyklu v závislosti na parametrech  $n$  a  $d$ 
4 for ( $j=0; j < 2^{28}; j++$ ) do
5   |  $i = A[j]$ 
6 printf("%d\n", i);
```

Hešování: Jak funguje hešování různých objektů a jak řešit kolize.