# Velké otázky

1. Definujte Splay strom. Popište, jak na něm probíhají operace Splay, Find, Insert a Delete. Popište výhody a nevýhody oproti jiným datovým strukturám, zejména vyváženým vyhledávacím stromům. Vyslovte a dokažte větu o amortizované složitosti operace Splay.
   - strom, ve kterém všechny operace vykonáváme pomocí Splay
   - neřešíme vyváženost, ale dokážeme, že Splay se vždy uamortizuje na log
   - **Splay**: vybraný vrchol přesunu do kořene pomocí rotací, přičemž preferuji dvojrotace
     - každý Splay použije nejvýše jednu jedno-rotaci
   - **Find**: Najdu vrchol jako v BVS, vysplayuji nalezený nebo poslední navštívený do vrcholu
   - **Insert**: Insert jako do normálního BVS, vysplayuji nalezený/vložený vrchol do kořene
     - přidání listu může být drahé
       - zvýším rank všem vrcholům na cestě do kořene
       - $$\Delta\Phi = r'(v_{t+1}) + \sum_{i=1}^{t} r'(v_i) - r(v_i)$$
       - jejich rank se zvýší o 1, což se dá odhadnout zhora původním rankem jejich rodiče – $r'(v_i) \leq r(v_{i-1})$
       - $r'(v_{t+1}) = 0$ (je to list)
       - $$\Delta\Phi \leq r'(v_1) - r(v_t)$$
   - **Delete**: Najdu vrchol k odstranění, vysplayuji do vrcholu, odstraním, vysplayuji nejpravější vrchol levého podstromu, spojím dohromady
   - výhody
     - sequential access
     - working set
     - $\Longrightarrow$ vyhledání $m$ prvků z podmnožiny velikosti $s$ je $\mathcal{O}(n \log n + m + m \log s)$
   - **složitost Splay**:
     - zavedeme $r(x)$ (rank vrcholu) jako $\log(s(x))$
     - potenciál je $\Phi = \sum_{x_i \in V} r(x_i)$
     - dvojrotace mají amortizovanou cenu $A = 3r'(x_i) - 3r(x_i)$, jednorotace $A = 3r'(x_i) - 3r(x_i) + 1$ ($r$ je před rotací, $r'$ je po)
     - celá Splay je pak posloupnost rotací jednoho vrcholu, $A = \left(\sum_i 3r_i(x) - 3r_{i-1}(x)\right) + 1$, všechny až na poslední rotaci jsou dvojrotace, teleskopická suma se posčítá na

       $$A = 3r_n(x) - 3r_0(x) + 1$$

       (kde +1 je z poslední rotace)
     - **zig-zag**:
       - $A = 2 + r'(x) + r'(w) + r'(y) - r(x) - r(w) - r(y)$
       - $s'(w) + s'(y) \leq s'(x)$
       - $r'(w) + r'(y) \leq 2\log(s'(w) + s'(y)) - 2 \leq 2r'(x) - 2$
       - $A \leq 3r'(x) - r(x) - r(w) - r(y)$
       - $r(x) \leq r(w), r(x) \leq r(y)$
       - $A \leq 3r'(x) - 3r(x)$
     - **zig-zig**:
       - $A = 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$
       - $s(x) + s'(z) \leq s'(x)$
       - $r(x) + r'(z) \leq 2\log(s(x) + s'(z)) - 2 \leq 2r'(x) - 2$
       - $A \leq 3r'(x) + r'(y) - 2r(x) - r(y) - r(z)$
       - $r'(x) \geq r'(y), r(x) \leq r(y), r(z) = r'(x)$

- $A \leq 3r'(x) + r'(x) - 2r(x) - r(x) - r'(x)$
- $A \leq 3r'(x) - 3r(x)$

▸ **zig**:
- $A = 1 + r'(x) + r'(y) - r(x) - r(y)$
- $r(x) \leq r(y), r'(x) \geq r'(y)$
- $A \leq 1 + 2r'(x) - 2r(x)$
- $r'(x) - r(x) \geq 0$
- $A \leq 1 + 3r'(x) - 3r(x)$

2. Definujte (a,b)-strom. Popište, jak na něm probíhají operace Find, Insert a Delete. Vyslovte a dokažte větu o amortizovaném počtu změn uzlů pro operace Insert a Delete a (a,2a)-stromech. Jak se to liší pro (a,2a-1)-stromy? Popište výhody a nevýhody oproti jiným datovým strukturám, zejména vyváženým binárním vyhledávacím stromům.
- stromy, kde počet potomků každého vrcholu je mezi $a$ a $b$, kde $b \geq 2a - 1$
- **hloubka**: $\Theta(\log_b n)$, $\mathcal{O}(\log_a n)$
- **Find**: musíme projít $\mathcal{O}(h) = \mathcal{O}(\log_a n) = \mathcal{O}(\log n / \log a)$ vrcholů, v každém vrcholu vrchol najdeme binárním vyhledáváním v $\mathcal{O}(\log b)$, celkově $\mathcal{O}(\log n \cdot \log b / \log a)$
- **Insert**: přidáme klíč do nejnižšího vrcholu, splitujeme, je-li třeba, split trvá $\mathcal{O}(b)$, celkově $\mathcal{O}(b \cdot \log n / \log a)$
- **Delete**: odstraníme klíč a nahradíme maximem levého podstromu, buď se mergenem se sousedem, nebo z něj půjčíme klíč (soused $\longrightarrow$ rodič $\longrightarrow$ my)
- $m$ po sobě jdoucích insertů do prázdného stromu je v $\boldsymbol{\mathcal{O}(m)}$
  - každý split zvýší množství vrcholů, merge neděláme, vrcholů je $\mathcal{O}(m)$
- **věta**: pro (a,2a)-stromy taky $m$ po sobě jdoucích insertů a deletů do prázdného stromu je v $\boldsymbol{\mathcal{O}(m)}$
  - určíme si potenciál podle počtu klíčů ve vrcholu $(a, 2a) \longrightarrow (a - 1, 2a - 1)$
  - | $k$ | $a-2$ | $a-1$ | $a$ | ... | $2a-2$ | $2a-1$ | $2a$ |
    |---|---|---|---|---|---|---|---|
    | $f(k)$ | 2 | 1 | 0 | 0 | 0 | 2 | 4 |
  - přidání/odebrání vrcholu – $|f(i) - f(i+1)| \leq c$
  - split:
    - $f(2a) \longrightarrow f(a) + f(a-1) + \underbrace{c}_{\text{přidání vrcholu do rodiče}} + \underbrace{1}_{\text{skutečná cena}}$
    - $4 \geq 0 + 1 + 2 + 1$
  - merge:
    - $f(a-2) + f(a-1) \longrightarrow f(2a-2) + \underbrace{c}_{\text{ubrání vrcholu z rodiče}} + \underbrace{1}_{\text{skutečná cena}}$
    - $2 + 1 \geq 0 + 2 + 1$
  - **důsledek**: Libovolná posloupnost $m$ Insertů a Deletů je $\mathcal{O}(n + m)$ (můžu postavit posloupnost $n$ Insertů do prázdného stromu)
- výhody:
  - cache-aware
  - amort. konstantní složitost operací
- nevýhody??

3. Definujte I/O model pro správu cache a srovnejte cache-aware a cache-oblivious algoritmy. Vyslovte a dokažte Sleatorovu-Tarjanovu větu o kompetitivnosti LRU. Popište přínos této věty pro analýzu cache-oblivious algoritmů.
- externí a interní paměť
  - interní má omezenou velikost $M$

  - externí je neomezená, ale lze číst a zapisovat pouze načtením bloku velikosti $B$ do interní paměti
- I/O složitost se obvykle určuje jen ve čteních z externí paměti, protože zápisy jsou většinou čteními zhora omezené
- algoritmus neovládá, které bloky se vyhodí z keše, je použit nějaký online algoritmus
- algoritmus OPT vyhazuje vždy ten blok, který bude potřeba nejdále v budoucnosti – optimum
- algoritmus LRU vyhazuje ten blok, který byl použit nejdále v minulosti
  - umíme dokázat, že to není tak špatné:

$$C_{\text{LRU}} \leq \frac{M_{\text{LRU}}}{M_{\text{LRU}} - M_{\text{OPT}}} \cdot C_{\text{OPT}} + M_{\text{OPT}}$$

    tedy, pokud $M_{\text{LRU}} = 2 \cdot M_{\text{OPT}}$:

$$C_{\text{LRU}} \leq 2C_{\text{OPT}} + M_{\text{OPT}}$$

  - **důkaz**
    - rozdělme posloupnost přístupů k blokům do *epoch* $e_0, ..., e_n$, tak, aby pro každou epochu kromě $e_0$ platilo $C_{\text{LRU}}(e_i) = M_{\text{LRU}}$, pro $e_0$: $C_{\text{LRU}}(e_0) \leq M_{\text{LRU}}$
    - pro každou epochu kromě $e_0$ platí jedna z možností:
      - všechny missnuté prvky jsou různé, takže $C_{\text{OPT}}(e_i) \geq M_{\text{LRU}} - M_{\text{OPT}}$
      - některý prvek byl missnut vícekrát, takže mezi jeho načteními muselo být přistoupeno ke všem ostatním prvkům keše, takže opět $C_{\text{OPT}}(e_i) \geq M_{\text{LRU}} - M_{\text{OPT}}$
    - tedy $C_{\text{LRU}}(e_i) = M_{\text{LRU}}$ a $C_{\text{OPT}}(e_i) \geq M_{\text{LRU}} - M_{\text{OPT}}$:

$$\frac{C_{\text{LRU}}(e_i)}{C_{\text{OPT}}(e_i)} \leq \frac{M_{\text{LRU}}}{M_{\text{LRU}} - M_{\text{OPT}}}$$

    - pro $e_0$ pak platí:
      - pokud keše začínají prázdné, všechny missy LRU jsou různé a OPT je všechny musí načíst taky, proto $C_{\text{OPT}}(e_0) \geq C_{\text{LRU}}(e_0)$
      - pokud LRU obsahuje bloky, nějaký miss se může proměnit v hit, ale výsledný LRU seznam se nezmění
      - pokud OPT obsahuje bloky, budou to přesně ty, které bude v $e_0$ potřebovat, tedy $C_{\text{OPT}}(e_0) \geq C_{\text{LRU}}(e_0) - M_{\text{OPT}}$
    - po zprůměrování přes epochy $e_1 ... e_n$ a $M_{\text{OPT}}$ z $e_0$ získáme chtěný vztah
- **důsledek**: ve většině zkoumaných algoritmů je závislost na $M$ taková, že vynásobení konstantou $cM$ (v tomto případě $c = 2$) se asymptotika nezmění, takže lze předpokládat OPT, i když se použije LRU

4. Definujte c-univerzální a k-nezávislé rodiny hešovacích funkcí. Uveďte příklady, kde nestačí c--universální rodina hešovacích funkcí, ale musíme použít k-nezávislou rodinu. Formulujte a dokažte větu o střední délce řetězce v hešování s řetězci. Ukažte příklady c-univerzálních a k--nezávislých rodin pro hešování přirozených čísel. Pro jeden příklad dokažte c-universalitu nebo k-nezávislost, pro k ≥ 2.
   - pro danou konstantu $c$, je rodina hashovacích funkcí $\mathcal{H} : \mathcal{U} \to [m]$ $c$-univerzální, pokud pro všechny různé dvojice $x, y \in \mathcal{U}$:

$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \frac{c}{m}$$

   - pro dané $k$ a konstantu $c$, je rodina hashovacích funkcí $\mathcal{H} : \mathcal{U} \to [m]$ $(k, c)$-nezávislá, pokud pro každou $k$-tici různých $x_1 ... x_k \in \mathcal{U}$ a $y_1 ... y_k \in [m]$:

$$\Pr_{h\in\mathcal{H}}[h(x_1) = y_1 \wedge ... \wedge h(x_k) = y_k] \leq \frac{c}{m^k}$$

- právě lineární adresování potřebuje 5-nezávislost pro konstantní čas, případně kukačky, also **[T]** se odkazuje na QuickSort
- **délka řetězce**:
  - věta: pro $c$-univerzální rodinu $\mathcal{H} : \mathcal{U} \to [m]$, $X = \{x_1, ..., x_n\}$ prvky v ní uložené, a $y$ prvek neuložený:

$$\mathbb{E}_{h\in\mathcal{H}}[\#i : h(x_i) = h(y)] \leq \frac{cn}{m}$$

  - dk: nechť $A_i$ je indikátorová veličina, že $h(x_i) = h(y)$, $\mathbb{E}A_i \leq \frac{c}{m}$ z $c$-univerzality, $\mathbb{E}\sum_i A_i = \sum_i \mathbb{E}A_i \leq \frac{cn}{m}$
  - důsledek:
    - neúspěšné vyhledávání projede celý řetízek s $h(y)$, takže $\leq \frac{cn}{m}$
    - úspěšný insert nejdřív provede neúspěšné vyhledávání, $\leq \frac{cn}{m}$
    - úspěšné vyhledávání projde nejvýše tolik prvků, kolik prošlo přidání daného prvku $\leq \frac{cn}{m}$
    - neúspěšný insert odpovídá úspěšnému vyhledávání, $\leq \frac{cn}{m}$
    - delete odpovídá úspěšnému/neúspěšnému vyhledávání $\leq \frac{cn}{m}$
  - tedy, pokud je $m \in \Omega(n)$ jsou operace na řetízkové hashovací tabulce konstantní
- příklady:
  - **Lineární kongruence**: pro prvočíslo $p$ a $m < p$,

$$\mathcal{L} : [p] \to [m] : \left\{h_{a,b} \mid a, b \in \mathbb{Z}_p\right\}$$

$$h_a, b(x) = ((ax + b) \bmod p) \bmod m$$

    - v $\mathbb{Z}_p$ existuje pro dané různé $x, y$ a soustavu:

$$r = (ax + b) \bmod p$$
$$s = (ay + b) \bmod p$$

      bijekce mezi páry $(r, s)$ a $(a, b)$
    - $\Pr\left[h_{a,b}(x) = r \wedge h_{a,b}(y) = s\right] = \frac{1}{p^2}$ pro rovnoměrně náhodně vybrané $a, b$
    - bez druhého mod $m$ je tedy rodina 2-nezávislá, lemma o modulení říká, že s ním to bude jen $2c$ horší
  - **Polynomiální hashování**:

$$\mathcal{P}_k : [p] \to [p] : \left\{h_{\boldsymbol{t}}(x) \mid \boldsymbol{t} \in \mathbb{Z}_p^k\right\}$$

$$h_{\boldsymbol{t}}(x) = \sum_{i=0}^{k-1} t_i x^i \bmod p$$

    - je $k$-nezávislé, neboť pro $x_1...x_k$ a $a_1...a_k$ existuje právě jeden polynom stupně nejvýše $k$ takový, že platí: $\forall i : p(x_i) = a_i$
    - takových polynomů je $p^k$, máme $(k, 1)$-nezávislost
    - pro $[p] \to [m]$, kde $p > 2km$ dává modulící lemma $(k, 2)$-nezávislost
  - Multiply-shift
  - Tabulkové hashování

5. Popište a analyzujte hešování s lineárním přidáváním s plně náhodnou hešovací funkcí a např. třetinovým zaplněním. Popište výhody a nevýhody oproti jiným datovým strukturám, zejména založeným na hešování.

- do každého kyblíku hashujeme pouze jeden prvek, v případě kolize hledáme $h(x) + i \bmod m$
- pro zaplněnost $\leq \frac{1}{3}$ (tedy $m \geq 3n$) a plně náhodnou hashovací funkci dokážeme konstantní délku „běhů“
  - nechť $n = \frac{m}{3}$ (horší už to být nemůže)
  - kyblíky $[m]$ rozdělíme postupně na po sobě jdoucí bloky velikosti $2^t$ (jako úplný binární strom)
  - blok je *kritický*, pokud se do něj **zahashuje** $\frac{2}{3} \cdot 2^t$ prvků
  - **Černovova nerovnost**: Pro nezávislé náhodné veličiny s hodnotami $\{0, 1\}$, jejich součet $X = X_1 + ... + X_k$, $\mu = \mathbb{E}[X]$ a $c > 1$:

$$\Pr[X \geq c\mu] = \left( \frac{e^{c-1}}{c^c} \right)^\mu$$

  - Pravděpodobnost, že blok $B$ velikosti $2^t$ je kritický je $\left(\frac{e}{4}\right)^{\left(\frac{1}{3}\right)^{2^t}} = q^{2^t}$, $q \doteq 0.879$
    - $$\mathbb{E}[X] = 2^t \cdot \frac{1}{3}$$
    - Černov: $c = 2$,

$$\Pr\left[X \geq 2^t \cdot \frac{1}{3} \cdot 2\right] = \left(\frac{e}{2^2}\right)^{2^t \cdot \frac{1}{3}}$$

  - Každý běh $R$, $|R| \geq 2^{l+2}$ obsahuje alespoň jeden kritický blok délky $2^l$
    - Běh $R$ zasahuje do 4 nebo 5 bloků velikosti $2^l$.
    - První blok $B_0$ obsahuje alespoň 1 prvek z $R$, $B_1 ... B_3$ obsahují $2^l$ prvků
    - Před $B_0$ je prázdno, takže všechny prvky v $L = B_0 ... B_3 \cap R$ se sem i zahashovaly (nepřetekly sem)
    - Do $L$ musí být zahashovaných alespoň $1 + 3 \cdot 2^t$, ale čtyři nekritické bloky obsahují nejvýše $4 \cdot \frac{2}{3} \cdot 2^t$.
  - Nechť $R$ je běh obsahující $h(x)$ a $|R| \in \left[2^{l+2}, 2^{l+3}\right)$, pak jeden z 12 okolních bloků je kritický, 8 před $h(x)$, 3 po
    - $R$ je 4 až 8 bloků dlouhý, ale nemusí být aligned, takže zasáhne 4 až 9 bloků
    - pokud končí blokem s $h(x)$, zabírá až 8 bloků před $h(x)$
    - pokud začíná blokem s $h(x)$, zabírá až 8 bloků za $h(x)$, ale kritický je jeden z prvních 4
  - Aby běh $R$ obsahující $h(x)$ byl dlouhý $\left[2^{l+2}, 2^{l+3}\right)$, musí z okolních 12 bloků být alespoň jeden kritický
    - Union bound – pravděpodobnost, že libovolný z 12 jevů nastane je nejvýše $12 \cdot p$
  - Tedy pro $R$ běh obsahující $h(x)$,

$$\Pr[\![|R| \in \left[2^{l+2}, 2^{l+3}\right)]\!] \leq 12 \cdot \Pr[\text{blok velikosti } 2^l \text{ je kritický}] = 12 \cdot q^{2^l}$$

  - Pro běh $R$ obsahující $h(x)$:

$$\mathbb{E}[\![R]\!] \leq 3 \cdot \Pr[\![|R| \leq 3]\!] + \sum_{l \geq 0} 2^{l+3} \cdot 12 \cdot q^{2^l}$$

$$\mathbb{E}[\![R]\!] \leq 3 \cdot 1 + 12 \cdot 8 \cdot \sum_{l \geq 0} 2^l \cdot q^{2^l}$$

$$\mathbb{E}[\![R]\!] \leq 3 \cdot 1 + 96 \cdot \sum_{i \geq 1} i \cdot q^i$$

    - poslední úprava – suma před úpravou dosazuje za $i$ jen mocniny dvojky, suma po úpravě dosazuje všechna přirozená čísla, což je určitě horní odhad
    - Gabrielovo schodiště konverguje pro libovolné $q$

6. Definujte vícerozměrné intervalové stromy (range trees). Rozeberte prostorovou složitost datové struktury a časovou složitost konstrukce a obdélníkových dotazů (bonus: včetně zrychlení zlomkovým kaskádováním).
   - Intervalové stromy jsou identické s BVS, jen při hledání pamatujeme interval vrcholů, které daný podstrom může obsahovat
     - **Vyhledávání intervalu $I$**: Rekurzí,
       - pokud je $v \in I$, reportuju $v$
       - pokud je $\text{inv}(v) \subseteq I$, reportuju celý podstrom a končím
       - rekurzím do potomků, které zasahují do $I$, pokud zasahují oba, v každém mi zůstane jen jeden konec intervalu a už budu rekurzit vždy jednou
   - 2-d intervalové stromy
     - v každém vrcholu $x$-stromu je $y$-strom pro daný podstrom
     - každý vrchol je v $\mathcal{O}(\log n)$ sekundárních stromech (jeden za každý vrchol na cestě do kořene) – paměť $\mathcal{O}(n \log n)$
     - vyhledávání:
       - pro každý nalezený vrchol z $x$-stromu se ptám daného $y$-stromu – $\mathcal{O}(n \log^2 n)$
     - dynamizace - weight-balanced přestavení celého nevyváženého podstromu $\mathcal{O}(n \log^2 n)$
   - $d$-d intervalové stromy
     - v každém vrcholu primárního stromu je $(d-1)$-d strom
     - paměť $\mathcal{O}\big(n \log^{d-1} n\big)$
     - vyhledávání v $\mathcal{O}\big(n \log^{d} n\big)$
   - zlomkové kaskádování ve 2-d
     - místo $y$-stromů udržuji seřazená pole a odkazy do podseznamů pod každým z $x$-potomků, pro pokračování vyhledávání
     - $\mathcal{O}(\log n)$
7. Definujte sufixové pole a LCP pole. Popište a analyzujte algoritmy na jejich konstrukci (pro sufixové pole stačí ve skoro lineárním čase). Popište příklad úlohy, kterou umí tato pole efektivně řešit.
8. Popište zámky a atomické operace CAS a LL/SC. Navrhněte a analyzujte bezzámkovou implementaci zásobníku. Vysvětlete problém ABA a navrhněte jeho řešení. Porovnejte paralelizaci datových struktur za použití zámků a za použití atomikých operací (tzv. bezzámkové datové struktury), přičemž v obou případech vysvětlete, jaké mohou nastat problémy.

# Malé otázky

1. Popište dynamické pole, tedy „nafukovací pole„ se zvětšováním a zmenšováním. Analyzujte jeho amortizovanou složitost.
   - když dojde místo, realokuju na dvojnásobek
   - po každé realokuji 2× tolik prvků, kolik jsem od poslední realokace přidal
2. Popište vyhledávací stromy s líným vyvažováním (BB[α]-stromy). Analyzujte jejich amortizovanou složitost. Uveďte příklad jejich použití.
   - začínám s plně vyváženým stromem, v každém vrcholu si počítám množství vrcholů v podstromu
   - za vyvážený ho považuji, pokud $\frac{1}{3}m(v) \leq m(\ell(v)) \leq \frac{2}{3}m(v)$
   - to udrží log hloubku (délka cesty je omezená $\log_{\frac{3}{2}}(n)$)
   - postavit nový podstrom trvá $n$ času, ale od jeho poslední přestavby z tohoto vrcholu jsem musel přidat $\frac{1}{3}n$ vrcholů $\Longrightarrow$ amort
     - potenciál je součet přes všechny vrcholy $|m(\ell(v)) - m(p(v))|$
     - pokud je strom vyvážený, je potenciál 0, ne 1

- použití: k-d stromy, k-d intervaláče? prostě BVS?

3. Navrhněte operace Find, Insert a Delete na Splay stromu. Analyzujte jejich amortizovanou složitost. Větu o složitosti operace Splay stačí vyslovit, nemusíte ji dokazovat.
   - **složitost Splay**: $3 \cdot (r'(v) - r(v)) + 1$, kde $r(v)$ je log mohutnosti vrcholu $v$ před Splay a $r'(v)$ je po Splay
     - mohutnosti jsou $\mathcal{O}(\log n)$
     - všechny operace můžu naúčtovat Splayi, který je $\mathcal{O}(\log n)$
   - **Find**: Najdu jako v BVS a vysplayuju do kořene
   - **Insert**: Insetrnu jako v BVS a vysplayuju do kořene
     - přidáním listu zvýším rank celé cesty a tím i potenciál, ale jen o $\mathcal{O}(\log n)$
   - **Delete**: Vysplayuju do kořene, deletenu, vysplayuju pravého syna levého podstromu
4. Analyzujte hloubku (a,b)-stromů.
   - pro hloubku $h$ platí $h \in \Theta(\log n)$
5. Analyzujte k-cestný Mergesort v cache-aware modelu. Jaká je optimální volba k?
   - $\log_k n = \frac{\log n}{\log k}$ průchodů
   - každá vrstva je $\mathcal{O}(n \log k)$ (používáme haldu)
   - složitost $\Theta\left(\frac{n \cdot \log k \cdot \log n}{\log k}\right) = \Theta(n \log n)$
   - potřebujeme $K$ bloků na jednotlivé průchody a dalších $K$ na udržení haldy, takže $M \geq 2BK$, $K \in \mathcal{O}\left(\frac{M}{B}\right)$
   - IO složitost $\mathcal{O}\left(\frac{n}{B} \cdot \frac{\log n}{\log k}\right) = \mathcal{O}\left(\frac{n}{B} \cdot \frac{\log n}{\log\left(\frac{M}{B}\right)}\right)$
6. Formulujte cache-oblivious algoritmus pro transpozici čtvercové matice. Rozeberte časovou složitost a I/O složitost.
   - rekurzivní, rozdělíme transpozici na dvě transpozice a dvě kombinované transpozice a prohození
   - transpozice a prohození se dále rozděluje na čtyři transpozice a prohození
   - k reálnému swapování dochází pouze na nejnižší úrovni rekurze, každý prvek se swapne jednou, $\mathcal{O}(n^2)$
   - IO složitost, v určitou chvíli swapujeme a transponujeme dva bloky velikosti $B \times B$, pokud je keš velká alespoň $2 \cdot B^2$ (tall cache assumption, pak se celé takové bločky vejdou do keše a transponuje se dobře, každý blok načteme jen jednou, $\mathcal{O}\left(\frac{N^2}{B}\right)$
7. Popište systém hešovacích funkcí odvozený ze skalárního součinu. Dokažte, že je to 1--univerzální systém ze $\mathbb{Z}_p^k$ do $\mathbb{Z}_p$.
   - $h_v(x) = xv$, vektor $v$ je náhodně generovaný a parametrizuje hashovací funkci
   - $c$-universálnost: $P_{h \in \mathcal{H}}[h(x) = h(y)] \leq \frac{c}{p}$
   - nechť se x a y liší v $d$-tém prvku, $P[xv = yv] = P[(x - y)v = 0]$, bude platit pro právě jednu volbu $v_d \Longrightarrow \frac{1}{p}$
8. Popište systém hešovacích funkcí založených na lineární kongruenci. Dokažte, že je to 2--nezávislý systém ze $\mathbb{Z}_p$ do $[m]$ (můžete využít lemma o modulení, které zformulujte, ale nemusíte dokazovat).
   - $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$
   - $k, c$-nezávislost: $P_{h \in \mathcal{H}}[h(x_1) = y_1 \wedge \ldots \wedge h(h_k) = y_k] = \frac{c}{m^k}$
   - **lemma o modulení**: pokud $\mathcal{H}$ je $k, c$-nezávislý systém z $\mathcal{U}$ do $[r]$, pak pro $2km \leq r$, $\mathcal{H} \bmod m$ je $k, 2c$-nezávislý
   - výběr

     $$r = (ax + b) \bmod p$$
     $$s = (ay + b) \bmod p$$

     je bijektivní na výběr $a, b$, tedy je i 2-nezávislý

9. Sestrojte $k$-nezávislý systém hešovacích funkcí ze $\mathbb{Z}_p$ do $[m]$. Zdůvodněte k-nezávislost (můžete využít lemma o modulení, které zformulujte, ale nemusíte dokazovat).
   - $\mathcal{P}_k = \{h_t \mid t \in \mathbb{Z}_p^k\}, h_t = \sum_{i=0}^{k-1} t_i x^i$
   - existuje právě jeden polynom takový, že pro $x_1 ... x_k$ a $y_1 ... y_k$ tž $h(x_i) = y_i$ pro každé $i$, tzn. pravděpodobnost je $\frac{1}{p^k}$
   - přes lemma o modulení, pokud $p \geq 2km$, máme po vymodulení $k$, 2-nezávislý systém
10. Sestrojte 2-nezávislý systém hešující řetězce délky nejvýše $L$ nad abecedou $[a]$ do $[m]$ založený na polynomech, tedy „rolling hash“. Popište výhodu použití tohoto systému oproti jiným hešovacím funkcím.
   - $\mathcal{R} = \{h_a \mid a \in \mathbb{Z}_p\}, h_{a(x)} = \sum_{i=0}^{L-1} x_{i+1} a^i$
   - dá se posouvat v konstantním čase – vynásobím $a$, přičtu příchozí znak, odečtu odchozí znak $\cdot$ $a^L$ (musím předpočítat $a^L$)
11. Popište a analyzujte Bloomův filtr. Uveďte příklad jeho praktického použití.
   - pomocí hashe indexujeme v poli bitů a flipujeme bity nahoru pro prvky, které jsme viděli
   - pak se můžeme ptát „viděli jsme tenhle prvek?“, na což dostaneme buď korektní NE, nebo possibly flase positive ANO
   - typicky $k$ hashovacích funkcí
   - také lze mít všechny hashe v jednom poli
   - s počítadly lze i delete
12. Definujte k-d stromy a ukažte, že 2-d intervalové dotazy trvají Ω(√n).
   - pro data z $\mathbb{R}^k$ vytvářím strom, kde na $l$-té úrovni mám medián daných podstromů v dimenzi $l \bmod k$
   - pro $\{0\} \times \mathbb{R}$ to suckuje, pro $x$ jdu furt do leva, ale pro $y$ furt rekurzím do všech
13. Ukažte, jak dynamizovat dvou dimenzionální intervalové stromy (tedy Range trees), stačí Insert.
   - pomocí BB[α] stromů, $\mathcal{O}(\log^2 n)$
14. Ukažte, jak použít sufixové pole a LCP pole na nalezení nejdelšího společného podřetězce dvou řetězců.
   - concatnu za sebe s pomocí oddělovacího znaku, najdu max v LCP t.ž. jeden je před a jeden je za oddělovačem
15. Ukažte, jak paralelizovat (a,b)-strom pomocí zámků.
   - pomocí top-down strategie — preemptivně rozdělujeme/spojujeme už při cestě dolů, abychom vždy mohli udělat Insert nebo Delete
   - zamykáme vždy dva vrcholy za sebou
   - při deletu může dávat smysl vždy zamykat nejdřív levého sourozence — pro případ, že s ním budeme slučovat nebo půjčovat
   - pokud odstraňujeme klíč, který není v nejnižší vrstvě, je možné, že budeme muset dlouho hledat následníka, kdyžtak se dá vrchol označit hrobečkem