6. CVIČENÍ Z DATOVÝCH STRUKTUR 1, ZS24/25

Kešujeme

1. *Z minula: Násobení matic.* Chceme spočíst matici $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$, kde $\mathbf{A}$ a $\mathbf{B}$ jsou zadané čtvercové matice $n \times n$. Pro následující přístupy určete I/O složitost (můžete předpokládat, že $n > M$):

a) Nejprve předpokládejte, že $\mathbf{A}$ je uložena po řádcích a $\mathbf{B}$ po sloupcích, a použijte přímočarý algoritmus pro násobení dle definice.

b) Nyní každou z matic rozdělíme na 4 bloky, které vynásobíme rekurzivně. (Pro jednoduchost předpokládejme, že n je mocnina dvojky.) Pro analýzu předpokládejte, že cache je „štíhlá“, tedy $M \geq c \cdot B^2$ pro nějakou konstantu c (tzv. tall cache assumption).

2. *Analýza algoritmu dle starověkého „Rozděl a panuj!“* Proveďte analýzu počtu přenesených bloků, tedy I/O složitost, pro algoritmus QuickSelect pro nalezení k-tého nejmenšího prvku s náhodným výběrem pivota. Jelikož je pravděpodobnostní, určete střední hodnotu.

(Pro jednoduchost můžete nejprve odstranit náhodnost a předpokládat, že v každém kroku vybereme pivota, který je *pseudomediánem*, tedy leží v prostředních dvou kvartilech.)

3. *Binární vyhledávání optimálně.* Binární vyhledávání v uspořádaném poli má I/O složitost $\Theta(\log N - \log B + 1)$. Navrhněte způsob uložení prvků do pole tak, abychom mohli vyhledávat s I/O složitostí $O(\log_B N + 1) = O(\log N / \log B + 1)$ (tato složitost je dokonce optimální). Hint: sestrojte nejprve dokonale vyvážený BVS.

4. *TransposeAndSwap naivně.* Připomeňte si cache-oblivious transpozici matic. Poté ukažte, že pokud TransposeAndSwap upravíme tak, že nejprve transponujeme obě zadané matice (rekurzivně) a teprve poté je prohazujeme (průchodem po řádcích), tak dostaneme horší časovou složitost i počet přenosů.

5. *Kompetitivnost LRU v problému online správy cache.* Určete nejhorší možný poměr počtu výpadků (cache misses) algoritmu LRU (Least Recently Used) oproti optimálnímu algoritmu, mají-li tyto algoritmy k dispozici stejně velkou velikost cache. Srovnejte se Sleator-Tarjanovou větou.

6. *Bonus: Optimální správa cache, která není online.* Ukažte optimalitu algoritmu Longest Forward Distance (LFD), který vyhodí z cache blok, jež bude potřeba nejpozději v budoucnu. (Hint: spor.)