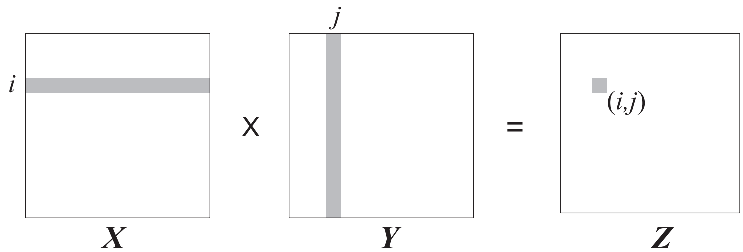
3. Rozděl a panuj II *(zapsali Lukáš Hermann, Vincent Kríž, Oto Petřík)*

Násobení matic n×n a Strassenův algoritmus

Nejdříve si připomeneme definici násobení dvou čtvercových matic typu $n \times n$. Platí, že prvek v i-tém řádku a j-tém sloupci výsledné matice Z se rovná standardnímu skalárnímu součinu i-tého řádku první matice X a j-tého sloupce druhé matice Y. Formálně zapsáno:

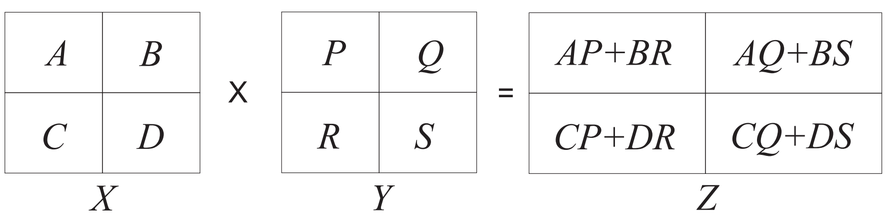
$$Z_{ij} = \sum_{k=1}^{n} X_{ik} \cdot Y_{kj}$$



Násobení matic

Algoritmus, který by násobil matice podle této definice, by měl časovou složitost $\Theta(n^3)$, protože počet prvků ve výsledné matici je n^2 a jeden skalární součin vektorů dimenze n vyžaduje lineární počet operací.

My se s touto časovou složitostí ovšem nespokojíme a budeme postupovat podobně jako při vylepšování algoritmu na násobení velkých čísel. Bez újmy na obecnosti předpokládejme, že budeme násobit dvě matice typu $n \times n$, kde $n = 2^k, k \in \mathbb{N}$. Obě tyto matice rozdělíme na čtvrtiny a tyto části postupně označíme u matice X - A, B, C a D, u matice Y - P, Q, R a S. Z definice násobení matic zjistíme, že čtvrtiny výsledné matice Z můžeme zapsat pomocí částí násobených matic. Levá horní čtvrtina bude odpovídat výsledku operací $AP+BR$, pravá horní čtvrtina bude $AQ+BS$, levá dolní $CP+DR$ a zbylá $CQ+DS$ (viz obrázek).



Násobení rozčtvrcených matic

Převedli jsme tedy problém násobení čtvercových matic řádu n na násobení čtvercových matic řádu $n/2$. Tímto rozdělováním bychom mohli pokračovat, dokud bychom se nedostali na matice řádu 1, jejichž vynásobení je triviální. Dostali jsme tedy klasický algoritmus typu *rozděl a panuj*. Pomohli jsme si ale nějak ? V každém kroku provádíme 8 násobení matic polovičního řádu a navíc konstantní počet operací na n^2 prvcích. Dostáváme tedy rekurentní zápis časové složitosti:

$$T(n) = 8\ T\left(\frac{n}{2}\right) + \mathcal{O}(n^2).$$

Po aplikaci Master Theoremu pak lehce dojdeme k závěru, že složitost je stále $\Theta(n^3)$, tedy stejná jako při násobení matic z definice. Zdánlivě jsme si tedy nepomohli, ale stejně jako tomu bylo u násobení velkých čísel, i teď můžeme zredukovat počet násobení matic polovičního řádu, které nejvíce ovlivňuje časovou složitost algoritmu. Není to bohužel nic triviálního, a proto si raději rovnou řekneme správné řešení. Jedná se o Strassenův algoritmus, který redukuje potřebný počet násobení na 7, a ještě před tím, než si ukážeme, jak funguje, dokážeme si, jak nám to s časovou složitostí vlastně pomůže:

$$T(n) = 7\ T\left(\frac{n}{2}\right) + \mathcal{O}(n^2) \Longrightarrow \mathcal{O}(n^{log_2 7}) = \mathcal{O}(n^{2.808}).$$

Výsledná složitost Strassenova algoritmu je tedy přibližně $\mathcal{O}(n^{2.808})$, což je sice malé, ale pro velké matice znatelné zlepšení oproti algoritmu vycházejícího přímo z definice.

Lemma: Strassenův algoritmus

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} P & Q \\ R & S \end{pmatrix} = \begin{pmatrix} T_1 + T_4 - T_5 + T_7 & T_3 + T_5 \\ T_2 + T_4 & T_1 - T_2 + T_3 + T_6 \end{pmatrix},$$

kde:

$$\begin{aligned} T_1 &= (A + D) \cdot (P + S) & T_5 &= (A + B) \cdot S \\ T_2 &= (C + D) \cdot P & T_6 &= (C - A) \cdot (P + Q) \\ T_3 &= A \cdot (Q - S) & T_7 &= (B - D) \cdot (R + S) \\ T_4 &= D \cdot (R - P) \end{aligned}$$

Důkaz: Do čtverců 4×4 si napíšeme znaky $+$ nebo $-$ podle toho, jestli se při výpočtu dané matice přičítá nebo odečítá příslušný součin dvou matic. Řádky znamenají matice A, B, C a D a sloupce značí matice P, Q, R a S. Pokud se tedy v prvním řádku a prvním sloupci vyskytuje znak $+$, znamená to že přičteme součin matic

A a P. Nejdříve si spočítáme pomocné matice T_1 až T_7 a z nich pak dopočítáme, co bude na příslušných místech ve výsledné matici.

$$T_1 = \boxed{\begin{array}{cccc} + & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & + \end{array}} \qquad T_2 = \boxed{\begin{array}{cccc} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \end{array}} \qquad T_3 = \boxed{\begin{array}{cccc} \cdot & + & \cdot & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array}} \qquad T_4 = \boxed{\begin{array}{cccc} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & + & \cdot \end{array}}$$

$$T_5 = \boxed{\begin{array}{cccc} \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array}} \qquad T_6 = \boxed{\begin{array}{cccc} - & - & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array}} \qquad T_7 = \boxed{\begin{array}{cccc} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & - & - \end{array}}$$

$$T_1 + T_4 - T_5 + T_7 = \boxed{\begin{array}{cccc} + & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array}} = AP + BR$$

$$T_3 + T_5 = \boxed{\begin{array}{cccc} \cdot & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array}} = AQ + BS$$

$$T_2 + T_4 = \boxed{\begin{array}{cccc} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & \cdot \end{array}} = CP + DR$$

$$T_1 - T_2 + T_3 + T_6 = \boxed{\begin{array}{cccc} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \end{array}} = CQ + DS$$

Jak je vidět, výsledná matice je tvořena stejnými částmi jako při obyčejném násobení. Touto kapitolou jsme tedy dokázali následující větu:

Věta: Násobení matic $n \times n$ má časovou složitost v nejhorším případě $\mathcal{O}(n^{2.808})$. ♡

Poznámka: Zatím nejlepší dokázaný algoritmus má časovou složitost $\mathcal{O}(n^{2.376})$, ovšem s velkou multiplikativní konstantou.

Hledání k-tého nejmenšího prvku (mediánu)

V tomto oddílu se budeme zabývat tím, jak co nejrychleji najít v jakékoli posloupnosti n čísel k-tý nejmenší prvek popřípadě medián. Pro ty, kdo medián neznají, tu máme definici:

Definice: Medián posloupnosti $a_1, a_2, \ldots, a_n$ je takové $m = a_i$, kde méně než $n/2$ prvků je menších než m a méně než $n/2$ prvků je větších než m.

Nejjednodušším řešením by určitě bylo celou posloupnost nejdříve setřídit a pak už jednoduše vyhmátnout požadovaný prvek. To bychom dokázali v celkem slušném čase $\mathcal{O}(n \log n)$, ale už teď můžeme prozradit, že to jde v čase $\mathcal{O}(n)$. Jak?

Používat budeme metodu *rozděl a panuj*. Nějakým způsobem si zvolíme jeden prvek posloupnosti, který nazveme *pivot*. Poté rozdělíme zadanou posloupnost na tři disjunktní množiny. Do první dáme všechny prvky menší než pivot, do druhé stejné jako pivot a do třetí větší než pivot. Tímto máme zajištěno, že prvky z první množiny jsou určitě menší než prvky z druhé a ty než prvky z třetí.

O tom, jak jsou prvky uspořádány uvnitř těchto množin, ale nic nevíme. V posledním kroku našeho algoritmu se pak rozhodneme, na kterou množinu svůj algoritmus rekurzivně zavoláme. Pokud je k menší než velikost první množiny, pokračujeme v první množině, pokud je k menší než součet velikostí první a druhé množiny, pak hledaným prvkem je právě vybraný pivot a algoritmus skončí, a nakonec pokud ani jedna podmínka splněna nebyla, pustíme se do hledání ve třetí množině, ovšem už nehledáme k-tý nejmenší prvek, ale l-tý, kde l se rovná k minus velikost prvních dvou množin. Pro větší názornost zapíšeme tento algoritmus předhledněji:

Select(k, X): (Hledání k-tého nejmenšího prvku v množine X)

1. Jestliže $|X| <= 1$, pak triviálně ošetříme.
2. Zvolíme pivota $p \in X$.
3. Rozdělíme množinu X na tři podmnožiny: $M = \{x \in X; x < p\}$, $P = \{x \in X; x = p\}, V = \{x \in X; x > p\}$.
4. Jestliže $k <= |M|$, pak vrať výsledek funkce *Select*$(k,\ M)$.
5. Jestliže $k <= |M| + |P|$, pak vrať pivota p.
6. Jinak vrať výsledek funkce *Select*$(k - |M| - |P|, V)$.

Na první pohled je vidět, že se algoritmus zastaví (vstup se vždy zmenší alespoň o 1) a že vydá vždy správný výsledek. Jak je to ovšem s časovou složitostí? Rozdělení do množin a podmínky v druhém a třetím kroku mají lineární složitost, čemuž se nevyhneme. Při nešťastné volbě pivota se nám může stát, že počet rekurencí může být až n, tedy celková složitost v nejhorším případě je $\Theta(n^2)$, čímž jsme si oproti prostému setřídění ještě pohoršili. Co s tím? Jak je vidět, velmi důležitá je volba pivota. Tu můžeme provést několika způsoby:

a) Pivot by se v setříděné posloupnosti vyskytoval uprostřed, vstup by se tedy stále půlil. Časovou složitost vypočteme z rekurentního zápisu:

$$T(n) = T\left(\frac{n}{2}\right) + \mathcal{O}(n) = \mathcal{O}\left(n + \frac{n}{2} + \frac{n}{4} + \ldots\right) = \mathcal{O}(n).$$

To by bylo sice skvělé, ale nalezení takového pivota je vlastně vyřešení úlohy hledání mediánu, o což se snažíme. Tedy jsme si vůbec nepomohli.

b) Pivot by se v setříděné posloupnosti náchazel v prostředních dvou čtvrtinách (nazvěme tento prvek "lžimedián"). Tím bychom v každém kroku určitě odstranili množinu velikosti čtvrtiny vstupu. Časová složitost tohoto řešení by byla:

$$T(n) = T\left(\frac{3}{4}n\right) + \mathcal{O}(n) = \mathcal{O}\left(n + \frac{3}{4}n + \frac{9}{16}n + \dots\right) = \mathcal{O}(n).$$

Tímto bychom tedy také dosáhli lineární časové složitosti. Ale jak vybrat pivota tak, aby se nacházel v prostředních dvou čtvrtinách, aby nám nám to nepokazilo lineární složitost?

c) Pivot bude vždy prostřední prvek zadané posloupnosti. V tomto případě je zřejmé, že existují určité vstupy, pro které bude časová složitost $\Omega(n^2)$, ale v průměrném případě, bude časová složitost $\mathcal{O}(n)$, což dokážeme v příští kapitole.

d) Pivot bude náhodně zvolený prvek zadané posloupnosti. Tímto dosáhneme něčeho podobného jako v předchozím případě, ale už nebude existovat vstup, pro který by časová složitost byla $\Omega(n^2)$, jen některé průběhy tohoto algoritmu na různých vstupech budou trvat takto dlouho. Průměrná časová složitost je tedy zase $O(n)$.

My se ale nespokojíme pouze s lineární průměrnou časovou složitostí. Chceme dosáhnout lineární složitosti i v nejhorším případě. Než si ale prozradíme takové řešení, rozmyslíme si, že už teď známe 3 časové složitosti, i když ne všechny ještě umíme analyzovat. Je to časová složitost v nejhorším přídadě, v průměrném případě přes všechny vstupy a v průměrném případě přes všechny běhy programu. Všimněte si hlavně rozdílu posledních dvou.

A nyní už slibované řešení s lineární nejhorší časovou složitostí. Vyjdeme z možnosti b), tedy pokusíme se najít takového pivota, který by na příští krok zaručeně omezil velikost analyzované množiny. Dosáhneme toho tímto algoritmem:

Volba pivota:

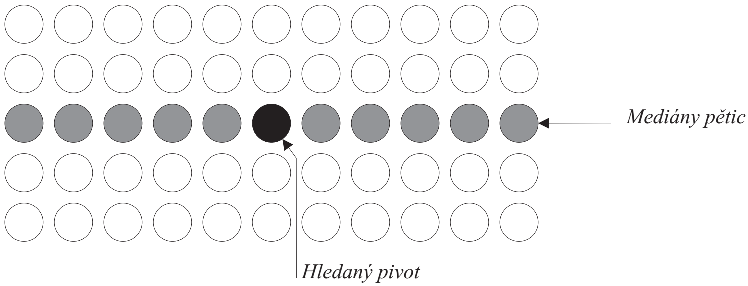
1. Rozdělíme vstup na pětice ... $\mathcal{O}(n)$
2. Spočteme medián každé pětice ... $\mathcal{O}(n)$
3. Spočteme medián mediánů pětic tak, že rekurzivně zavoláme tentýž algoritmus $\mathit{Select}(n/10$, mediány pětic) a to je pivot.

Abychom dokázali, že tento algoritmus bude mít opravdu lineární časovou složitost, musíme si nejdříve dokázat následující lemma:

Lemma: V každém kroku vypadne alespoň $3/10 \cdot n$ prvků.

Důkaz: Ten provedeme obrázkem. Představme si vybrané pětice seřazené podle velikosti od největšího prvku a zakresleme je do sloupců. Jejich mediány tedy vyplňují prostřední řadu. Tyto pětice pak seřaďme podle velikosti jejich mediánů (nejmenší vlevo). Hledaný pivot se tedy nachází (pokud předpokládáme pro jednoduchost lichý počet pětic) přesně uprostřed. O prvcích nad pivotem a napravo od něj můžeme určitě říct, že jsou větší nebo rovny pivotu, prvky pod ním a nalevo od něj jsou zase určitě menší nebo rovny pivotu.

Podle konstrukce algoritmu tedy zaručeně vypadne jedna nebo druhá skupina prvků. Obě tyto skupiny přitom obsahují, jak je vidět z obrázku, alespoň $3/10 \cdot n$ prvků. ♡



Pětice

Nyní už se tedy můžeme pustit do výpočtu časové složitosti. V každém kroku funkce zavolá sama sebe nejdříve na vstup velikosti $\frac{n}{5}$ a poté na vstup velikosti nejvýše $\frac{7}{10}n$. Ostatní operace zůstávají lineární. Časovou složitost v nejhorším případě tedy můžeme zapsat rekurentním vzorcem:

$$T(n) = \mathcal{O}(n) + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right).$$

Tento rekurentní vzorec ale zatím neumíme obecně řešit. Mohli bychom ho postupně rozepsat, ale trvalo by dlouho, než bychom z toho něco vykoukali. Lepší bude rovnou dokázat, že tento rekurentní vzorec odpovídá lineární složitosti, zkusíme tedy dosadit $T(n) = cn$:

$$cn = n + \frac{1}{5}cn + \frac{7}{10}cn \quad \iff \quad c = 10.$$

Tímto jsme tedy dokázali, že časová složitost tohoto algoritmu v nejhorším případě je $\mathcal{O}(n)$. Jelikož to rychleji určitě není možné, můžeme na závěr této kapitoly zformulovat tuto větu:

Věta: Hledání k-tého nejmenšího prvku v posloupnosti délky n má časovou složitost v nejhorším případě $\Theta(n)$. ♡