8. Fourierova transformace *(K.Jakubec, M.Polák a G.Ocsovszky)*

Násobení polynomů může mnohým připadat jako poměrně (algoritmicky) snadný problém. Asi každého hned napadne „hloupý“ algoritmus – vezmeme koeficienty prvního polynomu a vynásobíme každý se všemi koeficienty druhého polynomu a příslušně u toho sečteme i exponenty (stejně jako to děláme, když násobíme polynomy na papíře). Pokud stupeň prvního polynomu je n a druhého m, strávíme tím čas $\Omega(mn)$. Pro $m = n$ je to kvadraticky pomalé. Na první pohled se může zdát, že rychleji to prostě nejde (přeci musíme vždy vynásobit „každý s každým“). Ve skutečnosti to ale rychleji fungovat může, ale k tomu je potřeba znát trochu tajemný algoritmus FFT neboli *Fast Fourier Transform*.

Trochu algebry na začátek:

Libovolný polynom P stupně n může být reprezentován dvěma různými způsoby:

- svými koeficienty, čili čísly $p_0, p_1, \ldots, p_n$, nebo
- svými hodnotami v n různých bodech $x_0, x_1, \ldots, x_n$, čili čísly $P(x_0)$, $P(x_1)$, $\ldots$, $P(x_n)$.

Konvence:

Celé polynomy označujeme velkými písmeny, jednotlivé členy polynomů pak příslušnými malými písmeny. (Př.: Polynom W stupně n má koeficienty $w_0, w_1, w_2, \ldots, w_n$.)

Povšimněme si jedné skutečnosti – máme-li dva polynomy A a B stupně n a body $x_0, \ldots, x_k$, dále polynom $C = A \cdot B$ (stupně $2n$), pak platí $C(x_k) = A(x_k) \cdot B(x_k), k = 0, 1, 2, \ldots, n$. Toto činí tento druhý způsob reprezentace polynomu velice atraktivním pro násobení. Problémem je, že typicky máme polynom zadaný koeficienty a ne hodnotami v bodech. Tím pádem potřebujeme nějaký hodně rychlý algorimtus (tj. rychlejší než kvadratický, jinak bychom si nepomohli oproti hloupému algoritmu) na převod polynomu z jedné reprezentace do druhé a zase zpět.

Dále bychom si měli uvědomit, že stupeň našeho výsledného polynomu C bude $\leq 2n$ (kde n je stupeň výchozích polynomů). Pokud chceme polynom C reprezentovat pomocí jeho hodnot v bodech, musíme tedy vzít alespoň $2n$ bodů. Tímto končí malá algebraická vsuvka.

Idea, jak by měl algoritmus pracovat:

1. Vybereme $2n$ bodů $x_0, x_1, \ldots, x_{2n-1}$.
2. V těchto bodech vyhodnotíme polynomy A a B.
3. Nyní již v lineárním čase získáme hodnoty polynomu C v těchto bodech (viz výše).
4. Převedeme hodnoty polynomu C na jeho koeficienty.

Je asi vidět, že klíčové jsou kroky 2 a 4. Vybrání bodů jistě stihneme pohodlně v lineárním čase a vynásobení samotných hodnot též (máme $2n$ bodů a $C(x_k) = A(x_k) \cdot B(x_k), k = 0, 1, 2, \ldots, 2n-1$, takže na to nepotřebujeme více než $2n$ násobení).

Celý trik spočívá v chytrém vybrání oněch bodů, ve kterých budeme polynomy vyhodnocovat. Je na to potřeba vědět pár zajímavostí o komplexních číslech, na webové stránce přednášky jsou k dispozici slajdy, zde to bude zapsáno o trochu stručněji.

Vyhodnocení polynomu metodou Rozděl a panuj (algoritmus FFT):

Mějme polynom P řádu n a chtějme jej vyhodnotit v n bodech. Vybereme si body tak, aby byly spárované, čili $\pm x_0, \pm x_1, \ldots, \pm x_{n/2-1}$. To nám výpočet urychlí, protože pak se druhé mocniny x_j shodují s druhými mocninami $-x_j$.

Polynom P rozložíme na dvě části, první obsahuje členy se sudými exponenty, druhá s lichými: $P(x) = (p_0x^0+p_2x^2+\ldots+p_{n-2}x^{n-2})+(p_1x^1+p_3x^3+\ldots+p_{n-1}x^{n-1})$.

$S(x^2) = p_0x^0 + p_2x^2 + \ldots + p_{n-2}x^{n-2}$, $L(x^2) = p_1x^1 + p_3x^3 + \ldots + p_{n-1}x^{n-1}$

Takže obecně $P(x) = S(x^2) + xL(x^2)$ a $P(-x) = S(x^2) - xL(x^2)$. Jinak řečeno, vyhodnocování P v n bodech se nám smrskne na vyhodnocení $S(x)$ a $L(x)$ (oba jsou polynomy stupně $n/2$ a vyhodnocujeme je nyní v x^2) v $n/2$ bodech (protože $(x_i)^2 = (-x_i)^2$).

Příklad: $3 + 4x + 6x^2 + 2x^3 + x^4 + 10x^5 = (3 + 6x^2 + x^4) + x(4 + 2x^2 + 10x^4)$.

Teď nám ovšem vyvstane problém s oním párováním – druhá mocina přece nemůže být záporná a tím pádem už v druhé úrovni rekurze body spárované nebudou. Z tohoto důvodu musíme použít komplexní čísla – tam druhé mocniny záporné býti mohou. Jako $x_0, \ldots, x_{n-1}$ si zvolíme mocniny n-té primitivní odmocniny z jedné (označíme si ji jako ω). Máme n n-tých primitivních odmocnin z jedničky, rovnoměrně rozesetých po jednotkové kružnici, BÚNO $n = 2^k, k \in N$ (jinak viz slajdy Martina Mareše). Jednotlivé mocniny vypadají takto: $1, \omega, \omega^2, \ldots, \omega^{n-1}$, kde $\omega = e^{2\pi i/n}$.

Dvě poznámky:

- primitivní n-té odmocniny z jedničky jsou spárované, čili $\omega^j = -\omega^{n/2+j}$,
- umocníme-li všechny na druhou, vznikne nám $n/2$ $n/2$-tých odmocnin z jedné, které jsou i nadále spárované.

Celý algoritmus bude vypadat takto:

FFT(P, ω)

Vstup: $p_0, \ldots, p_{n-1}$, koeficienty polynomu P, a ω, $n-$tá odmocina z jedné.

Výstup: Hodnoty polynomu v bodech $1, \omega, \omega^2, \ldots, \omega^{n-1}$, čili čísla $P(1), P(\omega), P(\omega^2)$, $\ldots, P(\omega^{n-1})$.

1. Pokud $n = 1$, vrátíme P_0 a skončíme.
2. Jinak rozdělíme P na sudé a liché koeficienty a rekurzivně zavoláme FFT(S, ω^2) a FFT(L, ω^2).
3. Pro $j = 0, \ldots, n/2 - 1$ spočítáme:

$P(\omega^j) = S(\omega^{2j}) + \omega^j \cdot L(\omega^{2j})$.
$P(\omega^{j+n/2}) = S(\omega^{2j}) - \omega^j \cdot L(\omega^{2j})$.
j je z intervalu $[0, \frac{n}{2} - 1]$

Časová složitost: $T(n) = 2T(n/2) + \mathcal{O}(n) \Rightarrow$ složitost $\mathcal{O}(n \log n)$, stejně jako MergeSort.

Máme tedy algoritmus, který převede koeficienty polynomu na hodnoty tohoto polynomu v různých bodech . Potřebujeme ale také algoritmus, který dokáže reprezentaci polynomu pomocí hodnot převést zpět na koeficienty polynomu. K tomu nám pomůže podívat se na náš algoritmus trochu obecněji.

Definice: *Diskretní Fourierova transformace* (DFT) je funkce $f : \mathbb{C}^n \to \mathbb{C}^n$, kde $y = f(x) \equiv \forall j \; y_j = \sum_{k=0}^{n-1} x_k \cdot \omega^k$.

Jak najít inverzní matici? Víme, že $\Omega = \Omega^T$ protože $\omega^{jk} = \omega^{kj}$.

Využijeme následující lemma:

Lemma:

$$\Omega_j \cdot \Omega_k = \begin{cases} 0 \dots j \neq k \\ 1 \dots j = k \end{cases}.$$

Poznámka: $\Omega_j \cdot \Omega_k$ myslíme skalární součin. Jsou-li $x = (x_0, \dots, x_n)$ a $y = (y_0, \dots, y_n)$ dva komplexní vektory, pak jejich skalární součin definujeme jako: $x^* y = \sum_{i=0}^{n} \overline{x_i} y_i$.

V této definici $\overline{x}$ označuje číslo komplexně sdružené k číslu x.

Důkaz: Součin

$$\Omega_j \Omega_k = \sum_{l=0}^{n-1} \Omega_{jl} \overline{\Omega_{kl}} = \sum_l \omega^{jl} \overline{\omega^{kl}} = \sum_l \omega^{jl} \omega^{-kl} = \sum_l \omega^{(j-k)l} = \sum_{l=0}^{n-1} (\omega^{j-k})^l,$$

protože $\overline{\omega^{kl}} = \overline{\omega}^{kl} = (\frac{1}{\omega})^{kl} = \omega^{-kl}$.

- Pokud $j \neq k$, použijeme vzoreček pro součet geometrické posloupnosti, kde $a_1 = 1$ a $q = \omega^{(j-k)}$ a dostaneme $\frac{\omega^{(j-k)n} - 1}{\omega^{(j-k)} - 1} = \frac{1-1}{r-1} = \frac{0}{\neq 0} = 0$. Kde r je číslo různé od jedničky.
- Pokud $j = k$, pak $\sum_{l=0}^{n-1} (\omega^0)^l = n$.

♡

Důsledek: $\Omega \cdot \overline{\Omega} = nE$.

Jedná se o násobení matic, čili prvek na pozici ij je 0 nebo n. $\Rightarrow \Omega^{-1} = \frac{1}{n} \overline{\Omega}$.

Našli jsme inverzi:

$\Omega(\frac{1}{n}\overline{\Omega}) = \frac{1}{n}\Omega \cdot \overline{\Omega} = E, \quad \Omega^{-1}_{jk} = \frac{1}{n}\overline{\omega^{jk}} = \frac{1}{n}\omega^{-jk} = \frac{1}{n}(\omega^{-1})^{jk},$ kde ω^{-1} je $\overline{\omega}$ a ω_n je n-tá primitivní odmocnina z jedničky.

Náš algoritmus počítá tedy i inverzní transformaci, pouze místo ω_n použijeme komplexně sdružené $\overline{\omega_n}$ a matici vynásobíme $(1/n)$. Což je skvělé – stačí znát pouze jeden algoritmus u kterého stačí v jednom případě použít transformovanou matici a vydělit n.

Výsledek: Pro $n = 2^k$ lze DFT na $\mathbb{C}^n$ spočítat v čase $\mathcal{O}(n \log n)$ a DFT^{-1} taktéž.

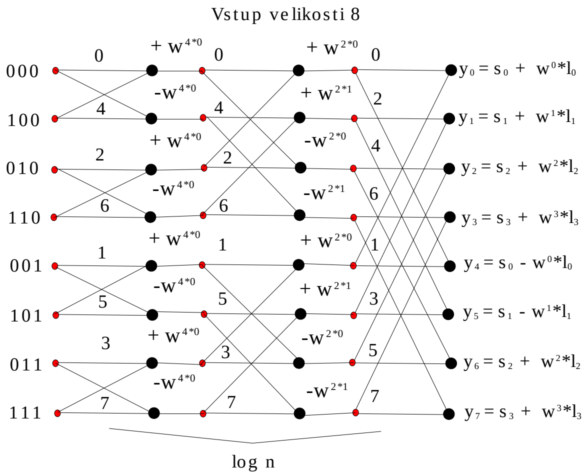
Důsledek:

Polynomy stupně n lze násobit v čase $\mathcal{O}(n \log n)$: $\mathcal{O}(n \log n)$ pro vyhodnocení, $\mathcal{O}(n)$ pro vynásobení a $\mathcal{O}(n \log n)$ pro převedení zpět.

Použití:

- Zpracování signálu – rozklad na siny a cosiny o různých frekvencích $\Rightarrow$ spektrální rozklad.
- komprese dat – například formát JPEG.
- Násobení dlouhých čísel v čase $\mathcal{O}(n \log n)$.

Hardwarová implementace FFT



Příklad průběhu algoritmu na vstupu velikosti 8

Obrázek ukazuje zapojení kombinačního obvodu pro DFT pro vstup velikosti 8. Hladin bude vždy $\log_2 n$, tj. v našem případě $\log_2 8 = 3$ hladiny.

Podívejme se na pravou část obrázku, tedy výstup celého obvodu. Černá kolečka představují podobvody, rovnice vedle nich operaci, kterou provádějí. Hodnoty y_j značí hodnotu polynomu P v bodě ω^j kde ω^j je $j - t\acute{a}$ mocnina primitivní n-té odmocniny z jedničky. K jejímu spočtení ale potřebujeme znát hodnoty s_k a l_k kde k je z intervalu $[0, n/2 - 1]$ a s_k a l_k jsou hodnoty polynomu stupně $n/2$ v bodě ω^{2k}. V polynomu s jsou sudé koeficienty a v polynomu l liché koeficienty polynomu P. Vidíme ze se jedná přesně o náš rekurzivní algoritmus pro počítání FFT a tímto způsobem postavíme celou síť. Tímto obvodem jsme tedy získali nerekurzivní algoritmus pro počítání FFT. Všimeme si pořadí vstupních hodnot (koeficientů). Čísla jsou v binárním tvaru 0–7 přečtená pozpátku. Pro představu jaké koeficienty polynomu P se objevují v různých hladinách, na obrázku jsou naznačena jejich čísla spolu s příslušnými mocninami primitivní n-té odmocniny z jedničky.

Z toho:

- Kombinační obvod pro DFT s $\mathcal{O}(\log n)$ hladinami a $\mathcal{O}(n)$ hradly na hladině.
- Nerekurzivní algoritmus (postupujeme zleva) v čase $\mathcal{O}(n \log n)$.