9. Fourierova transformace

(K.Jakubec, M.Polák a G.Ocsovszky, V.Tůma, M.Kozák)

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

Celé polynomy označujeme velkými písmeny, jednotlivé členy polynomů příslušnými malými písmeny (př.: polynom W stupně d má koeficienty $w_0, w_1, w_2, \ldots, w_d$).

Libovolný polynom P stupně (nejvýše) d lze reprezentovat jednak jeho koeficienty, tedy čísly $p_0, p_1, \ldots, p_d$, druhak i pomocí hodnot:

Lemma: Polynom stupně nejvýše d je jednoznačně určeň svými hodnotami v $d+1$ různých bodech.

Důkaz: Polynom stupně d má maximálně d kořenů (indukcí – je-li k kořenem P, pak lze P napsat jako $(x-k)Q$ kde Q je polynom stupně o jedna menší, přitom polynom stupně 1 má jediný kořen); uvážíme-li dva různé polynomy P a Q stupně d nabývající v daných bodech stejných hodnot, tak $P-Q$ je polynom stupně maximálně d, každé z $x_0 \ldots x_d$ je kořenem tohoto polynomu $\Rightarrow$ spor, polynom stupně d má $d+1$ kořenů $\Rightarrow P-Q$ musí být nulový polynom $\Rightarrow P = Q$. ♡

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

Idea, jak by měl algoritmus pracovat:

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

Je vidět, že klíčové jsou kroky 2 a 4. Celý trik spočívá v chytrém vybrání oněch

bodů, ve kterých budeme polynomy vyhodnocovat – zvolí-li se obecná x_j, tak se to rychle neumí, pro speciální x_j ale ukážeme, že to rychle jde.

Vyhodnocení polynomu metodou Rozděl a panuj (algoritmus FFT):

Mějme polynom P stupně $\leq d$ 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:

$$P(x) = (p_0x^0 + p_2x^2 + \ldots + p_{d-2}x^{d-2}) + (p_1x^1 + p_3x^3 + \ldots + p_{d-1}x^{d-1})$$

se zavedením značení:

$$P_s(t) = p_0t^0 + p_2t^1 + \ldots + p_{d-2}t^{\frac{d-2}{2}}$$

$$P_l(t) = p_1t^0 + p_3t^1 + \ldots + p_{d-1}t^{\frac{d-2}{2}}$$

bude $P(x) = P_s(x^2) + xP_l(x^2)$ a $P(-x) = P_s(x^2) - xP_l(x^2)$. Jinak řečeno, vyhodnocování polynomu P v n bodech se nám smrskne na vyhodnocení P_s a P_l v $n/2$ bodech – oba jsou polynomy stupně nejvýše $d/2$ a vyhodnocujeme je v x^2 (využíváme rovnosti $(x_i)^2 = (-x_i)^2$).

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 mocnina 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.

Komplexní intermezzo

Základní operace

- Definice: $\mathbb{C} = \{a + b\mathrm{i} \mid a, b \in \mathbb{R}\}$
- Sčítání: $(a + b\mathrm{i}) \pm (p + q\mathrm{i}) = (a \pm p) + (b \pm q)\mathrm{i}$.
  Pro $\alpha \in \mathbb{R}$ je $\alpha(a + b\mathrm{i}) = \alpha a + \alpha b\mathrm{i}$.
- Komplexní sdružení: $\overline{a + b\mathrm{i}} = a - b\mathrm{i}$.
  $\overline{\overline{x}} = x$, $\overline{x \pm y} = \overline{x} \pm \overline{y}$, $\overline{x \cdot y} = \overline{x} \cdot \overline{y}$, $x \cdot \overline{x} \in \mathbb{R}$.
- Absolutní hodnota: $|x| = \sqrt{x \cdot \overline{x}}$, takže $|a + b\mathrm{i}| = \sqrt{a^2 + b^2}$.
  Také $|\alpha x| = |\alpha| \cdot |x|$.
- Dělení: $x/y = (x \cdot \overline{y})/(y \cdot \overline{y})$.

Gaußova rovina a goniometrický tvar

- Komplexním číslům přiřadíme body v $\mathbb{R}^2$: $a + b\mathrm{i} \leftrightarrow (a, b)$.
- $|x|$ je vzdálenost od bodu $(0, 0)$.
- $|x| = 1$ pro čísla ležící na jednotkové kružnici (*komplexní jednotky*).
  Pak platí $x = \cos\varphi + \mathrm{i}\sin\varphi$ pro nějaké $\varphi \in [0, 2\pi)$.

- Pro libovolné $x \in \mathbb{C}$: $x = |x| \cdot (\cos\varphi(x) + \mathrm{i}\sin\varphi(x))$.
  Číslu $\varphi(x) \in [0, 2\pi)$ říkáme *argument* čísla x, někdy značíme $\arg x$.
- Navíc $\varphi(\overline{x}) = -\varphi(x)$.

Exponenciální tvar

- Eulerova formule: $e^{i\varphi} = \cos\varphi + \mathrm{i}\sin\varphi$.
- Každé $x \in \mathbb{C}$ lze tedy zapsat jako $|x| \cdot e^{\mathrm{i}\cdot\varphi(x)}$.
- Násobení: $xy = \left(|x| \cdot e^{\mathrm{i}\cdot\varphi(x)}\right) \cdot \left(|y| \cdot e^{\mathrm{i}\cdot\varphi(y)}\right) = |x| \cdot |y| \cdot e^{\mathrm{i}\cdot(\varphi(x)+\varphi(y))}$.
  (absolutní hodnoty se násobí, argumenty sčítají)
- Umocňování: $x^\alpha = \left(|x| \cdot e^{\mathrm{i}\cdot\varphi(x)}\right)^\alpha = |x|^\alpha \cdot e^{\mathrm{i}\alpha\varphi(x)}$.
- Odmocňování: $\sqrt[n]{x} = |x|^{1/n} \cdot e^{\mathrm{i}\cdot\varphi(x)/n}$.
  Pozor – odmocnina není jednoznačná: $1^4 = (-1)^4 = \mathrm{i}^4 = (-\mathrm{i})^4 = 1$.

Odmocniny z jedničky

- Je-li nějaké $x \in \mathbb{C}$ n-tou odmocninou z jedničky, musí platit: $|x| = 1$, takže $x = e^{\mathrm{i}\varphi}$ pro nějaké φ. Proto $x^n = e^{\mathrm{i}\varphi n} = \cos\varphi n + \mathrm{i}\sin\varphi n = 1$. Platí tedy $\varphi n = 2k\pi$ pro nějaké $k \in \mathbb{Z}$.
- Z toho plyne: $\varphi = 2k\pi/n$
  (pro $k = 0, \ldots, n-1$ dostáváme různé n-té odmocniny).
- Obecné odmocňování: $\sqrt[n]{x} = |x|^{1/n} \cdot e^{\mathrm{i}\varphi(x)/n} \cdot u$, kde $u = \sqrt[n]{1}$.
- Je-li x odmocninou z 1, pak $\overline{x} = x^{-1}$ – je totiž $1 = |x \cdot \overline{x}| = x \cdot \overline{x}$.

Primitivní odmocniny

Definice: x je *primitivní* k-tá odmocnina z $1 \equiv x^k = 1 \wedge \forall j : 0 < j < k \Rightarrow x^j \neq 1$.

Tuto definici splňují například čísla $\omega = e^{2\pi\mathrm{i}/k}$ a $\overline{\omega} = e^{-2\pi\mathrm{i}/k}$. Platí totiž, že $\omega^j = e^{2\pi\mathrm{i}j/k}$, což je rovno 1 právě tehdy, je-li j násobkem k (jednotlivé mocniny čísla ω postupně obíhají jednotkovou kružnici). Analogicky pro $\overline{\omega}$.

Ukažme si několik pozorování fungujících pro libovolné číslo ω, které je primitivní k-tou odmocninou z jedničky (někdy budeme potřebovat, aby navíc k bylo sudé):

- Pro $0 \leq j < l < k$ je $\omega^j \neq \omega^l$, neboť $\omega^l/\omega^j = \omega^{l-j} \neq 1$, protože $l - j < k$ a ω je primitivní.
- $\omega^{k/2} = -1$, protože $(\omega^{k/2})^2 = 1$, a tedy $\omega^{k/2}$ je druhá odmocnina z 1. Takové odmocniny jsou dvě: 1 a -1, ovšem 1 to být nemůže, protože ω je primitivní.
- $\omega^j = -\omega^{k/2+j}$ – přímý důsledek předchozího bodu, pro nás ale velice zajímavý: $\omega^0, \omega^1, \ldots, \omega^{k-1}$ jsou po dvou spárované.
- ω^2 je $k/2$-tá primitivní odmocnina z 1 – dosazením.

Konec intermezza

Vraťme se nyní k algoritmu. Z poslední části komplexního intermezza se zdá, že by nemusel být špatný nápad zkusit vyhodnocovat polynom v mocninách n-té primitivní odmocniny z jedné (tedy za $x_0, x_1, \ldots, x_{n-1}$ z původního algoritmu

zvolíme $\omega^0, \omega^1, \ldots, \omega^{n-1}$). Aby nám vše vycházelo pěkně, zvolíme n jako mocninu dvojky.

Celý algoritmus bude vypadat takto:

FFT(P, ω)

Vstup: $p_0, \ldots, p_{n-1}$, koeficienty polynomu P stupně nejvýše $n-1$, a ω, n-tá primitivní 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 členy se sudými a lichými exponenty (jako v původní myšlence) a rekurzivně zavoláme FFT(P_s, ω^2) a FFT(P_l, ω^2) – P_l i P_s jsou stupně max. $n/2-1$, ω^2 je $n/2$-tá primitivní odmocnina, a mocniny ω^2 jsou stále po dvou spárované (n je mocnina dvojky, a tedy i $n/2$ je sudé; popř. $n = 2$ a je to zřejmé).
3. Pro $j = 0, \ldots, n/2 - 1$ spočítáme:
4. $\quad P(\omega^j) = P_s(\omega^{2j}) + \omega^j \cdot P_l(\omega^{2j})$.
5. $\quad P(\omega^{j+n/2}) = P_s(\omega^{2j}) - \omega^j \cdot P_l(\omega^{2j})$.

Časová složitost: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow$ složitost $\Theta(n \log n)$, 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: *Diskrétní Fourierova transformace* (DFT) je zobrazení $f : \mathbb{C}^n \to \mathbb{C}^n$, kde

$$y = f(x) \equiv \forall j \; y_j = \sum_{k=0}^{n-1} x_k \cdot \omega^{jk}$$

(DFT si lze mimo jiné představit jako funkci vyhodnocující polynom s koeficienty x_k v bodech ω^j). Takovéto zobrazení je lineární a tedy popsatelné maticí Ω s prvky $\Omega_{jk} = \omega^{jk}$. Chceme-li umět převádět z hodnot polynomu na koeficienty, zajímá nás inverze této matice.

Jak najít inverzní matici?

Značme $\overline{\Omega}$ matici, jejíž prvky jsou komplexně sdružené odpovídajícím prvkům Ω, a využijme následující lemma:

Lemma:

$$\Omega \cdot \overline{\Omega} = n \cdot E.$$

Důkaz:

$$(\Omega \cdot \overline{\Omega})_{jk} = \sum_{l=0}^{n-1} \omega^{jl} \cdot \overline{\omega^{lk}} = \sum \omega^{jl} \cdot \overline{\omega}^{lk} = \sum \omega^{jl} \cdot \omega^{-lk} = \sum \omega^{l(j-k)}.$$

- Pokud $j = k$, pak $\sum_{l=0}^{n-1} (\omega^0)^l = n$.
- Pokud $j \neq k$, použijeme vzoreček pro součet geometrické řady s kvocientem $\omega^{(j-k)}$ a dostaneme $\frac{\omega^{(j-k)n}-1}{\omega^{(j-k)}-1} = \frac{1-1}{\neq 0} = 0$ ($\omega^{j-k} - 1$ je jistě $\neq 0$, neboť ω je n-tá primitivní odmoncina a $j - k < n$).

♡

Našli jsme inverzi: $\Omega(\frac{1}{n}\overline{\Omega}) = \frac{1}{n}\Omega \cdot \overline{\Omega} = E$, $\quad \Omega_{jk}^{-1} = \frac{1}{n}\overline{\omega^{jk}} = \frac{1}{n}\omega^{-jk} = \frac{1}{n}(\omega^{-1})^{jk}$, (připomínáme, ω^{-1} je $\overline{\omega}$).

Vyhodnocení polynomu lze provést vynásobením Ω, převod do původní reprezentace vynásobením Ω^{-1}. My jsme si ale všimli chytrého spárování, a vyhodnocujeme polynom rychleji než kvadraticky (proto FFT, jakože *fast*, ne jako *fuj*). Uvědomíme-li si, že $\overline{\omega} = \omega^{-1}$ je také n-tá primitivní odmocnina z 1 (má akorát úhel s opačným znaménkem), tak můžeme stejným trikem vyhodnotit i zpětný převod – nejprve vyhodnotíme A a B v ω^j, poté pronásobíme hodnoty a dostaneme tak hodnoty polynomu $C = A \cdot B$, a pustíme na ně stejný algoritmus s ω^{-1} (hodnoty C vlastně budou v algoritmu „koeficienty polynomu“). Nakonec jen získané hodnoty vydělíme n a máme chtěné koeficienty.

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

Důsledek: Polynomy stupně n lze násobit v čase $\Theta(n \log n)$: $\Theta(n \log n)$ pro vyhodnocení, $\Theta(n)$ pro vynásobení a $\Theta(n \log n)$ pro převedení zpět.

Použití FFT:

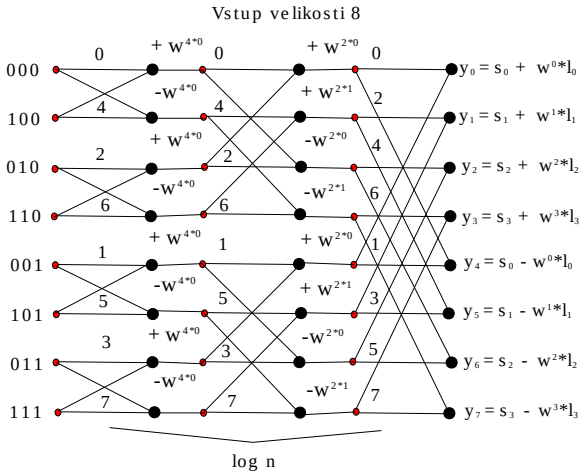
- Zpracování signálu – rozklad na siny a cosiny o různých frekvencích $\Rightarrow$ spektrální rozklad.
- Komprese dat – například formát JPEG.
- Násobení dlouhých čísel v čase $\Theta(n \log n)$.

Paralelní implementace FFT

Zkusme si průběh algoritmu FFT znázornit graficky (podobně, jako jsme kreslili hradlové sítě). Na levé straně obrázku se nachází vstupní vektor $x_0, \ldots, x_{n-1}$ (v nějakém pořadí), na pravé straně pak výstupní vektor $y_0, \ldots, y_{n-1}$. Sledujme chod algoritmu pozpátku: Výstup spočítáme z výsledků „polovičních“ transformací vektorů $x_0, x_2, \ldots, x_{n-2}$ a $x_1, x_3, \ldots, x_{n-1}$. Černé kroužky přitom odpovídají výpočtu lineární kombinace $a + \omega^k b$, kde a, b jsou vstupy kroužku a k nějaké přirozené číslo závislé na poloze kroužku. Každá z polovičních transformací se počítá analogicky z výsledků transformace velikosti $n/4$ atd. Celkově výpočet probíhá v $\log_2 n$ vrstvách po $\Theta(n)$ operacích.

Jelikož operace na každé vrstvě probíhají na sobě nezávisle, můžeme je počítat paralelně. Náš diagram tedy popisuje hradlovou síť pro paralelní výpočet FFT v čase $\Theta(\log n \cdot T)$ a prostoru $\mathcal{O}(n \cdot S)$, kde T a S značí časovou a prostorovou složitost výpočtu lineární kombinace dvou komplexních čísel.

Cvičení: Dokažte, že permutace vektoru $x_0, \ldots, x_{n-1}$ odpovídá bitovému zrcadlení,



Příklad průběhu algoritmu na vstupu velikosti 8

tedy že na pozici b shora se vyskytuje prvek x_d, kde d je číslo b zapsané ve dvojkové soustavě pozpátku.

Nerekurzivní FFT

Obvod z předchozího obrázku také můžeme vyhodnocovat po hladinách zleva doprava, čímž získáme elegantní nerekurzivní algoritmus pro výpočet FFT v čase $\Theta(n \log n)$ a prostoru $\Theta(n)$:

1. *Vstup:* $x_0, \ldots, x_{n-1}$
2. Pro $j = 0, \ldots, n-1$ položíme $y_k \leftarrow x_{r(k)}$, kde r je funkce bitového zrcadlení.
3. Předpočítáme tabulku $\omega^0, \omega^1, \ldots, \omega^{n-1}$.
4. $b \leftarrow 1$ *(velikost bloku)*
5. Dokud $b < n$, opakujeme:
6. Pro $j = 0, \ldots, n-1$ s krokem $2b$ opakujeme: *(začátek bloku)*
7. Pro $k = 0, \ldots, b-1$ opakujeme: *(pozice v bloku)*
8. $\alpha \leftarrow \omega^{nk/2b}$
9. $(y_{j+k}, y_{j+k+b}) \leftarrow (y_{j+k} + \alpha \cdot y_{j+k+b}, y_{j+k} - \alpha \cdot y_{j+k+b})$.
10. $b \leftarrow 2b$
11. *Výstup:* $y_0, \ldots, y_{n-1}$

FFT v konečných tělesech

Nakonec dodejme, že Fourierovu transformaci lze zavést nejen nad tělesem komplexních čísel, ale i v některých konečných tělesech, pokud zaručíme existenci primitivní n-té odmocniny z jedničky. Například v tělese $\mathbb{Z}_p$ pro prvočíslo $p = 2^k + 1$ platí $2^k = -1$, takže $2^{2k} = 1$ a $2^0, 2^1, \ldots, 2^{2k-1}$ jsou navzájem různá, takže číslo 2 je primitivní $2k$-tá odmocnina z jedné. To se nám ovšem nehodí pro algoritmus FFT, jelikož $2k$ bude málokdy mocnina dvojky.

Zachrání nás ovšem algebraická věta, která říká, že multiplikativní grupa⟨1⟩ libovolného konečného tělesa je cyklická, tedy že všechny nenulové prvky tělesa lze zapsat jako mocniny nějakého čísla g (generátoru). Například pro $p = 2^{16} + 1 = 65\,537$ je jedním takovým generátorem číslo 3. Jelikož mezi čísly $g^1, g^2, \ldots, g^{p-1}$ se musí každý nenulový prvek tělesa vyskytnout právě jednou, je g primitivní 2^k-tou odmocninou z jedničky, takže můžeme počítat FFT pro libovolný vektor, jehož velikost je mocnina dvojky menší nebo rovná 2^k.⟨2⟩

⟨1⟩ To je množina všech nenulových prvků tělesa s operací násobení.

⟨2⟩ Bližší průzkum našich úvah o FFT dokonce odhalí, že není ani potřeba těleso. Postačí libovolný komutativní okruh, ve kterém existuje příslušná primitivní odmocnina z jedničky, její multiplikativní inverze (ta ovšem existuje vždy, protože $\omega^{-1} = \omega^{n-1}$) a multiplikativní inverze čísla n. To nám poskytuje ještě daleko více volnosti než tělesa, ale není snadné takové okruhy hledat.