1. Fourierova transformace

V této kapitole se budeme zabývat násobením polynomů. Tento na první pohled triviální algebraický problém má překvapivě efektivní řešení, které nás dovede až k Fourierově transformaci.

1.1. Násobení polynomů

Značení: *Polynomy* jsou výrazy typu

$$P(x) = \sum_{i=0}^{n-1} p_i \cdot x^i,$$

kde x je proměnná a p_0 až p_{n-1} jsou čísla, kterým říkáme *koeficienty* polynomu. Obecně budeme značit polynomy velkými písmeny a jejich koeficienty příslušnými malými písmeny s indexy.

V algoritmech polynomy obvykle reprezentujeme pomocí *vektoru koeficientů* $(p_0, \ldots, p_{n-1})$; oproti zvyklostem lineární algebry budeme složky vektorů v celé této kapitole indexovat od 0. Počtu koeficientů n budeme říkat *velikost polynomu* $|P|$. Časovou složitost algoritmu budeme vyjadřovat vzhledem k velikostem polynomů na jeho vstupu.

Pokud přidáme nový koeficient $p_n = 0$, hodnota polynomu se pro žádné x nezmění. Stejně tak je-li nejvyšší koeficient p_{n-1} nulový, můžeme ho vynechat. Takto můžeme každý polynom zmenšit na *normální tvar,* v němž má buďto nenulový nejvyšší koeficient, nebo nemá vůbec žádné koeficienty – to je takzvaný *nulový polynom,* který pro každé x roven nule. Nejvyšší mocnině s nenulovým koeficientem se říká *stupeň polynomu* $\deg R$, nulovému polynomu přiřazujeme stupeň -1.

Násobení polynomů: Polynomy násobíme jako výrazy:

$$P(x) \cdot Q(x) = \left(\sum_{i=0}^{n-1} p_i \cdot x^i\right) \cdot \left(\sum_{j=0}^{m-1} q_j \cdot x^j\right).$$

Po roznásobení můžeme tento součin zapsat jako polynom $R(x)$, jehož koeficient u x^k je roven $r_k = p_0 q_k + p_1 q_{k-1} + \ldots + p_k q_0$.

Snadno nahlédneme, že polynom R má stupeň $\deg P + \deg Q$ a velikost $|P| + |Q| - 1$.

Algoritmus, který počítá součin dvou polynomů velikosti n přímo podle definice, proto spotřebuje čas $\Theta(n)$ na výpočet každého koeficientu, celkem tedy $\Theta(n^2)$. Pokusíme se nalézt efektivnější způsob.

Rovnost polynomů: Odbočme na chvíli a uvažujme, kdy dva polynomy považujeme za stejné. Na to se dá nahlížet více způsoby. Buďto se na polynomy můžeme dívat

jako na výrazy a porovnávat jejich symbolické zápisy. Pak jsou si dva polynomy rovny právě tehdy, mají-li po normalizaci stejné vektory koeficientů. Tomu se říká *identická rovnost* polynomů a obvykle se značí $P \equiv Q$.

Druhá možnost je porovnávat polynomy jako reálné funkce. Polynomy P a Q si tedy budou rovny ($P = Q$) právě tehdy, je-li $P(x) = Q(x)$ pro všechna $x \in \mathbb{R}$. Identicky rovné polynomy si jsou rovny i jako funkce, ale musí to platit i naopak? Následující věta ukáže, že ano a že dokonce stačí rovnost pro konečný počet x.

Věta: Buďte P a Q polynomy stupně nejvýše d. Pokud platí $P(x_i) = Q(x_i)$ pro navzájem různá čísla $x_0, \ldots, x_d$, pak jsou P a Q identické.

Důkaz: Připomeňme nejprve následující standardní lemma o kořenech polynomů:

> **Lemma:** Polynom R stupně $t \geq 0$ má nejvýše t kořenů (čísel α, pro něž je $P(\alpha) = 0$).
>
> *Důkaz:* Z algebry víme, že je-li číslo α kořenem polynomu $R(x)$, můžeme $R(x)$ beze zbytku vydělit výrazem $x - \alpha$. To znamená, že $R(x) \equiv (x - \alpha) \cdot R'(x)$ pro nějaký polynom $R'(x)$ stupně $t - 1$. Dosazením ověříme, že kořeny polynomu R' jsou přesně tytéž jako kořeny polynomu R, s možnou výjimkou kořene α.
>
> Budeme-li tento postup opakovat t-krát, buďto nám v průběhu dojdou kořeny (a pak lemma jistě platí), nebo dostaneme rovnost $R(x) \equiv (x - \alpha_1) \cdot \ldots \cdot (x - \alpha_t) \cdot R''(x)$, kde R'' je polynom nulového stupně. Takové polynomy ovšem nemohou mít žádný kořen, a tím pádem nemůže mít žádné další kořeny ani R. □

Abychom dokázali větu, stačí uvážit polynom $R(x) \equiv P(x) - Q(x)$. Tento polynom má stupeň nejvýše d, ovšem každé z čísel $x_0, \ldots, x_d$ je jeho kořenem. Podle lemmatu musí tedy být identicky nulový, a proto $P \equiv Q$. □

Zpět k násobení: Věta, již jsme právě dokázali, vlastně říká, že polynomy můžeme reprezentovat nejen vektorem koeficientů, ale také *vektorem funkčních hodnot* v nějakých smluvených bodech – tomuto vektoru budeme říkat *graf polynomu.* Pokud zvolíme dostatečně mnoho bodů, je polynom svým grafem jednoznačně určen.

V této reprezentaci je přitom násobení polynomů triviální: Součin polynomů P a Q má v bodě x hodnotu $P(x) \cdot Q(x)$. Stačí tedy grafy vynásobit po složkách, což zvládneme v lineárním čase. Jen je potřeba dát pozor na to, že součin má vyšší stupeň než jednotliví činitelé, takže si potřebujeme pořídit dostatečný počet bodů.

Algoritmus NásobeníPolynomů

1. Jsou dány polynomy P a Q velikosti n, určené svými koeficienty. Bez újmy na obecnosti předpokládejme, že horních $n/2$ koeficientů je u obou polynomů nulových, takže součin $R = P \cdot Q$ bude také polynom velikosti n.
2. Zvolíme navzájem různá čísla $x_0, \ldots, x_{n-1}$.
3. Spočítáme grafy polynomů P a Q, čili vektory $(P(x_0), \ldots, P(x_{n-1}))$ a $(Q(x_0), \ldots, Q(x_{n-1}))$.

4. Z toho vypočteme graf součinu R vynásobením po složkách: $R(x_i) = P(x_i) \cdot Q(x_i)$.
5. Nalezneme koeficienty polynomu R tak, aby odpovídaly grafu.

Krok 4 trvá $\Theta(n)$, takže rychlost celého algoritmu stojí a padá s efektivitou převodů mezi koeficientovou a hodnotovou reprezentací polynomů. To obecně neumíme v lepším než kvadratickém čase, ale zde máme možnost volby bodů $x_0, \ldots, x_{n-1}$, takže si je zvolíme tak šikovně, aby převod šel provést rychle.

Pokus o vyhodnocení polynomu metodou Rozděl a panuj

Uvažujme polynom P velikosti n, který chceme vyhodnotit v n bodech. Body si zvolíme tak, aby byly *spárované,* tedy aby tvořily dvojice lišící se pouze znaménkem: $\pm x_0, \pm x_1, \ldots, \pm x_{n/2-1}$.

Polynom P můžeme rozložit na členy se sudými exponenty a ty s lichými:

$$P(x) = (p_0 x^0 + p_2 x^2 + \ldots + p_{n-2} x^{n-2}) + (p_1 x^1 + p_3 x^3 + \ldots + p_{n-1} x^{n-1}).$$

Navíc můžeme z druhé závorky vytknout x:

$$P(x) = (p_0 x^0 + p_2 x^2 + \ldots + p_{n-2} x^{n-2}) + x \cdot (p_1 x^0 + p_3 x^2 + \ldots + p_{n-1} x^{n-2}).$$

V obou závorkách se nyní vyskytují pouze sudé mocniny x. Proto můžeme každou závorku považovat za vyhodnocení nějakého polynomu velikosti $n/2$ v bodě x^2, tedy:

$$P(x) = P_s(x^2) + x \cdot P_\ell(x^2),$$

kde:

$$P_s(t) = p_0 t^0 + p_2 t^1 + \ldots + p_{n-2} t^{\frac{n-2}{2}},$$
$$P_\ell(t) = p_1 t^0 + p_3 t^1 + \ldots + p_{n-1} t^{\frac{n-2}{2}}.$$

Navíc pokud podobným způsobem dosadíme do P hodnotu $-x$, dostaneme:

$$P(-x) = P_s(x^2) - x \cdot P_\ell(x^2).$$

Vyhodnocení polynomu P v bodech $\pm x_0, \ldots, \pm x_{n-1}$ tedy můžeme převést na vyhodnocení polynomů P_s a P_ℓ poloviční velikosti v bodech $x_0^2, \ldots, x_{n-1}^2$.

To naznačuje algoritmus s časovou složitostí $T(n) = 2T(n/2) + \Theta(n)$ a z Kuchařkové věty víme, že tato rekurence má řešení $T(n) = \Theta(n \log n)$. Jediný problém je, že tento algoritmus nefunguje: druhé mocniny, které předáme rekurzivnímu volání, jsou vždy nezáporné, takže už nemohou být správně spárované. Ouvej.

Tedy … alespoň dokud počítáme s reálnými čísly. Ukážeme, že v oboru komplexních čísel už můžeme zvolit body, které budou správně spárované i po několikerém umocnění na druhou.

1.2. Malé intermezzo o komplexních číslech

Základní operace

- Definice: $\mathbb{C} = \{a + b\mathbf{i} \mid a, b \in \mathbb{R}\}$, $\mathbf{i}^2 = -1$.
- Sčítání: $(a + b\mathbf{i}) \pm (p + q\mathbf{i}) = (a \pm p) + (b \pm q)\mathbf{i}$.
- Násobení: $(a+b\mathbf{i})(p+q\mathbf{i}) = ap+aq\mathbf{i}+bp\mathbf{i}+bq\mathbf{i}^2 = (ap-bq)+(aq+bp)\mathbf{i}$.
  Pro $\alpha \in \mathbb{R}$ je $\alpha(a + b\mathbf{i}) = \alpha a + \alpha b\mathbf{i}$.
- Komplexní sdružení: $\overline{a + b\mathbf{i}} = a - b\mathbf{i}$.
  $\overline{\overline{x}} = x$, $\overline{x \pm y} = \overline{x} \pm \overline{y}$, $\overline{x \cdot y} = \overline{x} \cdot \overline{y}$, $x \cdot \overline{x} = (a+b\mathbf{i})(a-b\mathbf{i}) = a^2+b^2 \in \mathbb{R}$.
- Absolutní hodnota: $|x| = \sqrt{x \cdot \overline{x}}$, takže $|a + b\mathbf{i}| = \sqrt{a^2 + b^2}$.
  Také $|\alpha x| = |\alpha| \cdot |x|$.
- Dělení: $x/y = (x \cdot \overline{y})/(y \cdot \overline{y})$. Takto upravený jmenovatel je reálný, takže můžeme vydělit každou složku zvlášť.

Gaußova rovina a goniometrický tvar

- Komplexním číslům přiřadíme body v $\mathbb{R}^2$: $a + b\mathbf{i} \leftrightarrow (a, b)$.
- $|x|$ je vzdálenost od bodu $(0, 0)$.
- $|x| = 1$ pro čísla ležící na jednotkové kružnici (*komplexní jednotky*).
  Pak platí $x = \cos\varphi + \mathbf{i}\sin\varphi$ pro nějaké $\varphi \in [0, 2\pi)$.
- Pro libovolné $x \in \mathbb{C}$: $x = |x| \cdot (\cos\varphi(x) + \mathbf{i}\sin\varphi(x))$.
  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: $\mathrm{e}^{\mathbf{i}\varphi} = \cos\varphi + \mathbf{i}\sin\varphi$.
- Každé $x \in \mathbb{C}$ lze tedy zapsat jako $|x| \cdot \mathrm{e}^{\mathbf{i}\varphi(x)}$.
- Násobení: $xy = \left(|x| \cdot \mathrm{e}^{\mathbf{i}\varphi(x)}\right) \cdot \left(|y| \cdot \mathrm{e}^{\mathbf{i}\varphi(y)}\right) = |x| \cdot |y| \cdot \mathrm{e}^{\mathbf{i}(\varphi(x)+\varphi(y))}$
  (absolutní hodnoty se násobí, argumenty sčítají).
- Umocňování: $x^\alpha = \left(|x| \cdot \mathrm{e}^{\mathbf{i}\varphi(x)}\right)^\alpha = |x|^\alpha \cdot \mathrm{e}^{\mathbf{i}\alpha\varphi(x)}$.

Odmocniny z jedničky

Odmocňování v komplexních číslech obecně není jednoznačné: jestliže třeba budeme hledat čtvrtou odmocninu z jedničky, totiž řešit rovnici $x^4 = 1$, nalezneme hned čtyři řešení: 1, -1, $\mathbf{i}$ a $-\mathbf{i}$.

Prozkoumejme nyní obecněji, jak se chovají n-té odmocniny z jedničky, tedy komplexní kořeny rovnice $x^n = 1$:

- Jelikož $|x^n| = |x|^n$, musí být $|x| = 1$. Proto $x = \mathrm{e}^{\mathbf{i}\varphi}$ pro nějaké φ.
- Má platit $1 = x^n = \mathrm{e}^{\mathbf{i}\varphi n} = \cos\varphi n + \mathbf{i}\sin\varphi n$. To nastane, kdykoliv $\varphi n = 2k\pi$ pro nějaké $k \in \mathbb{Z}$.

Dostáváme tedy n různých n-tých odmocnin z 1, totiž $\mathrm{e}^{2k\pi\mathbf{i}/n}$ pro $k = 0, \ldots, n-1$. Některé z těchto odmocnin jsou ovšem speciální:

Definice: Komplexní číslo x je *primitivní* n-tá odmocnina z 1, pokud $x^n = 1$ a žádné z čísel $x^1, x^2, \ldots, x^{n-1}$ není rovno 1.

Příklad: Ze čtyř zmíněných čtvrtých odmocnin z 1 jsou $\mathbf{i}$ a $-\mathbf{i}$ primitivní a druhé dvě nikoliv (ověřte sami dosazením). Pro obecné $n > 2$ vždy existují alespoň dvě primitivní odmocniny, totiž čísla $\omega = \mathrm{e}^{2\pi\mathbf{i}/n}$ a $\overline{\omega} = \mathrm{e}^{-2\pi\mathbf{i}/n}$. Platí totiž, že $\omega^j = \mathrm{e}^{2\pi\mathbf{i}j/n}$, a to je rovno 1 právě tehdy, je-li j násobkem n (jednotlivé mocniny čísla ω postupně obíhají jednotkovou kružnici). Analogicky pro $\overline{\omega}$.

Pozorování: Pro sudé n a libovolné číslo ω, které je primitivní n-tou odmocninou z jedničky, platí:

- $\omega^j \neq \omega^k$, kdykoliv $0 \leq j < k < n$. Stačí se podívat na podíl $\omega^k/\omega^j = \omega^{k-j}$. Ten nemůže být roven jedné, protože $0 < k - j < n$ a ω je primitivní.
- Pro sudé n je $\omega^{n/2} = -1$. Platí totiž $(\omega^{n/2})^2 = \omega^n = 1$, takže $\omega^{n/2}$ je druhá odmocnina z 1. Takové odmocniny jsou jenom dvě: 1 a -1, ovšem 1 to být nemůže, protože ω je primitivní.

1.3. Rychlá Fourierova transformace

Ukážeme, že primitivních odmocnin lze využít k záchraně našeho párovacího algoritmu na vyhodnocování polynomů.

Nejprve polynomy doplníme nulami tak, aby jejich velikost n byla mocninou dvojky. Poté zvolíme nějakou primitivní n-tou odmocninu z jedničky ω a budeme polynom vyhodnocovat v bodech $\omega^0, \omega^1, \ldots, \omega^{n-1}$. To jsou navzájem různá komplexní čísla, která jsou správně spárovaná – hodnoty $\omega^{n/2}, \ldots, \omega^{n-1}$ se od $\omega^0, \ldots, \omega^{n/2-1}$ liší pouze znaménkem. To snadno ověříme: pro $0 \leq j < n/2$ je $\omega^{n/2+j} = \omega^{n/2}\omega^j = -\omega^j$. Navíc ω^2 je primitivní $(n/2)$-tá odmocnina z jedničky, takže se rekurzivně opět voláme na problém téhož druhu, a ten je opět správně spárovaný.

Náš plán použít metodu Rozděl a panuj tedy vyšel, opravdu máme algoritmus o složitosti $\Theta(n \log n)$ pro vyhodnocení polynomu. Ještě ho upravíme tak, aby místo s polynomy pracoval s vektory jejich koeficientů či hodnot. Tomuto algoritmu se říká FFT, vzápětí prozradíme, proč.

Algoritmus FFT
Vstup: Číslo $n = 2^k$, primitivní n-tá odmocnina z jedničky ω a vektor $(p_0, \ldots, p_{n-1})$ koeficientů polynomu P.

1. Pokud $n = 1$, položíme $y_0 \leftarrow p_0$ a skončíme.
2. Jinak se rekurzivně zavoláme na sudou a lichou část koeficientů:
3. $\quad (s_0, \ldots, s_{n/2-1}) \leftarrow \mathrm{FFT}(n/2, \omega^2, (p_0, p_2, p_4, \ldots, p_{n-2}))$.
4. $\quad (\ell_0, \ldots, \ell_{n/2-1}) \leftarrow \mathrm{FFT}(n/2, \omega^2, (p_1, p_3, p_5, \ldots, p_{n-1}))$.
5. Z grafů obou částí poskládáme graf celého polynomu:
6. $\quad$ Pro $j = 0, \ldots, n/2 - 1$:
7. $\quad\quad y_j \leftarrow s_j + \omega^j \cdot \ell_j$.

8. $y_{j+n/2} \leftarrow s_j - \omega^j \cdot \ell_j$.

Výstup: Graf polynomu P, tedy vektor $(y_0, \ldots, y_{n-1})$, kde $y_j = P(\omega^j)$.

Vyhodnotit polynom v mocninách čísla ω umíme, ale ještě nejsme v cíli. Potřebujeme umět provést dostatečně rychle i opačný převod – z hodnot na koeficienty. K tomu nám pomůže podívat se na vyhodnocování polynomu trochu abstraktněji jako na nějaké zobrazení, které jednomu vektoru komplexních čísel přiřadí jiný vektor. Toto zobrazení matematici v mnoha různých kontextech potkávají už několik staletí a nazývají ho Fourierovou transformací.

Definice: *Diskrétní Fourierova transformace (DFT)* je zobrazení $\mathcal{F} : \mathbb{C}^n \to \mathbb{C}^n$, které vektoru $\mathbf{x}$ přiřadí vektor $\mathbf{y}$, jehož složky jsou dány předpisem

$$y_j = \sum_{k=0}^{n-1} x_k \cdot \omega^{jk},$$

kde ω je nějaká pevně zvolená primitivní n-tá odmocnina z jedné.

Pozorování: Pokud označíme $\mathbf{p}$ vektor koeficientů nějakého polynomu P, pak jeho Fourierova transformace $\mathcal{F}(\mathbf{p})$ není nic jiného než graf tohoto polynomu v bodech $\omega^0, \ldots, \omega^{n-1}$. To snadno ověříme dosazením do definice.

Náš algoritmus tedy počítá diskrétní Fourierovu transformaci v čase $\Theta(n \log n)$. Odtud pramení jeho název FFT – Fast Fourier Transform.

Také si všimněme, že DFT je lineární zobrazení. Můžeme ho proto zapsat jako násobení nějakou maticí $\mathbf{\Omega}$, kde $\mathbf{\Omega}_{jk} = \omega^{jk}$. Pro převod grafu na koeficienty tedy potřebujeme najít inverzní zobrazení určené inverzní maticí $\mathbf{\Omega}^{-1}$.

Jelikož $\omega^{-1} = \overline{\omega}$, pojďme zkusit, zda $\overline{\mathbf{\Omega}}$ není hledanou inverzní maticí.

Lemma: $\mathbf{\Omega} \cdot \overline{\mathbf{\Omega}} = n \cdot \mathbf{E}$, kde $\mathbf{E}$ je jednotková matice.

Důkaz: Dosazením do definice a elementárními úpravami:

$$\begin{aligned}(\mathbf{\Omega} \cdot \overline{\mathbf{\Omega}})_{jk} &= \sum_{\ell=0}^{n-1} \mathbf{\Omega}_{j\ell} \cdot \overline{\mathbf{\Omega}}_{\ell k} = \sum_{\ell=0}^{n-1} \omega^{j\ell} \cdot \overline{\omega^{\ell k}} = \sum_{\ell=0}^{n-1} \omega^{j\ell} \cdot \overline{\omega}^{\ell k} \\ &= \sum_{\ell=0}^{n-1} \omega^{j\ell} \cdot (\omega^{-1})^{\ell k} = \sum_{\ell=0}^{n-1} \omega^{j\ell} \cdot \omega^{-\ell k} = \sum_{\ell=0}^{n-1} \omega^{(j-k)\ell}.\end{aligned}$$

To je ovšem geometrická řada. Pokud je $j = k$, jsou všechny členy řady jedničky, takže se sečtou na n. Pro $j \neq k$ použijeme známý vztah pro součet geometrické řady s kvocientem $q = \omega^{j-k}$:

$$\sum_{\ell=0}^{n-1} q^\ell = \frac{q^n - 1}{q - 1} = \frac{\omega^{(j-k)n} - 1}{\omega^{j-k} - 1} = 0.$$

Poslední rovnost platí díky tomu, že $\omega^{(j-k)n} = (\omega^n)^{j-k} = 1^{j-k} = 1$, takže čitatel zlomku je nulový; naopak jmenovatel určitě nulový není, jelikož ω je primitivní a $0 < |j - k| < n$. □

Důsledek: $\mathbf{\Omega}^{-1} = (1/n) \cdot \overline{\mathbf{\Omega}}$.

Matice $\mathbf{\Omega}$ tedy je regulární a její inverze se kromě vydělení n liší pouze komplexním sdružením. Navíc $\overline{\omega} = \omega^{-1}$ je také primitivní n-tou odmocninou z jedničky, takže až na faktor $1/n$ se jedná opět o Fourierovu transformaci, kterou můžeme spočítat stejným algoritmem FFT. Shrňme, co jsme zjistili, do následující věty:

Věta: Je-li n mocnina dvojky, lze v čase $\Theta(n \log n)$ spočítat diskrétní Fourierovu transformaci v $\mathbb{C}^n$ i její inverzi.

Tím jsme také doplnili poslední část algoritmu na násobení polynomů:

Věta: Polynomy velikosti n nad tělesem $\mathbb{C}$ lze násobit v čase $\Theta(n \log n)$.

V obou větách přitom činíme předpoklad, že základní operace s komplexními čísly umíme provést v konstantním čase. Pokud tomu tak není, stačí složitost vynásobit složitostí jedné operace.

Poznámka: *(Další použití FFT)* Dodejme ještě, že Fourierova transformace se hodí i k jiným věcem než k násobení polynomů. Své uplatnění nachází i v dalších algebraických algoritmech, třeba v násobení velkých čísel (lze se dostat až ke složitosti $\Theta(n)$). Mimo to skýtá i lecjaké fyzikální aplikace – odpovídá totiž spektrálnímu rozkladu signálu na siny a cosiny o různých frekvencích. Na tom jsou založeny například algoritmy pro filtrování zvuku, také pro kompresi zvuku a obrazu (MP3, JPEG) nebo rozpoznávání řeči.

Cvičení:

1. O jakých vlastnostech vektoru vypovídá nultý a $(n/2)$-tý koeficient jeho Fourierova obrazu (výsledku Fourierovy transformace)?
2. Spočítejte Fourierovy obrazy následujících vektorů z $\mathbb{C}^n$:
   - $(x, \ldots, x)$
   - $(1, -1, 1, -1, \ldots, 1, -1)$
   - $(\omega^0, \omega^1, \omega^2, \ldots, \omega^{n-1})$
   - $(\omega^0, \omega^2, \omega^4, \ldots, \omega^{2n-2})$
3. Rozšířením výsledků z předchozího cvičení najděte pro každé j vektor, jehož Fourierova transformace má na j-tém místě jedničku a všude jinde nuly. Z toho lze přímo sestrojit inverzní transformaci.
4. Ukažte, že je-li $\mathbf{x}$ reálný vektor z $\mathbb{R}^n$, je jeho Fourierova transformace $\mathbf{y} = \mathcal{F}(\mathbf{x})$ *antisymetrická:* $\mathbf{y}_j = \overline{\mathbf{y}_{n-j}}$ pro všechna j.
5. Podobně ukažte, že Fourierova transformace každého antisymetrického vektoru je reálná.
6.* Uvažujme reálnou funkci f definovanou na intervalu $[0, 2\pi)$. Pokud její hodnoty *navzorkujeme* v n pravidelně rozmístěných bodech, získáme vektor $\mathbf{f} \in \mathbb{R}^n$ o složkách $\mathbf{f}_j = f(2\pi j/n)$. Jak vypadá Fourierova transformace tohoto vektoru pro následující funkce?
   - $\mathrm{e}^{\mathbf{i}kx}$ pro $k \in \mathbb{N}$

- $\cos kx$
- $\sin kx$

7.* Pomocí předchozího cvičení dokažte, že libovolnou reálnou funkci na $[0, 2\pi)$ existuje lineární kombinace funkcí $\sin kx$ a $\cos kx$, která při vzorkování v n bodech není od zadané funkce rozlišitelná.

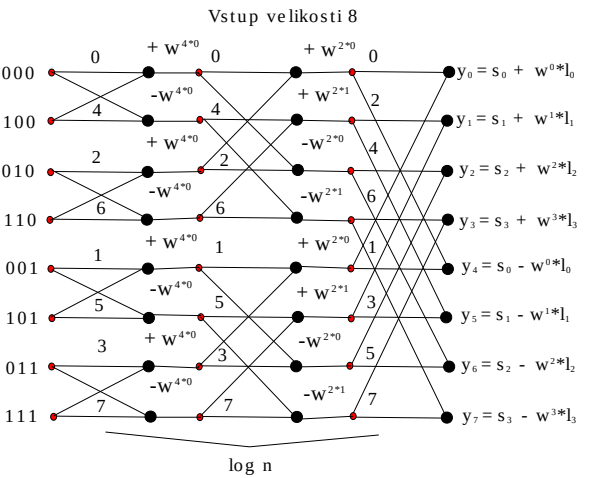
Přesněji řečeno, pro každý vektor $\mathbf{f} \in \mathbb{R}^n$ existují vektory $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ takové, že platí:

$$\mathbf{f}_j = \sum_{k=0}^{n-1} \mathbf{a}_k \sin \frac{2jk\pi}{n} + \mathbf{b}_k \cos \frac{2jk\pi}{n}.$$

Koeficienty a_k a b_k lze přitom snadno získat z Fourierova obrazu vektoru $\mathbf{f}$.

1.4. Další varianty FFT

Zkusme si ještě průběh algoritmu FFT znázornit graficky Na levé straně následujícího 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.



Obr. 1.1: Příklad průběhu algoritmu pro vstup velikosti 8

Na obrázek se také můžeme dívat jako na schéma hradlové sítě pro výpočet DFT. Kroužky jsou přitom „hradla“ pracující s komplexními čísly. Všechny operace

v jedné vrstvě jsou na sobě nezávislé, takže je síť počítá paralelně. Síť proto pracuje v čase $\Theta(\log n)$ a prostoru $\Theta(n)$, opět případně násobeno složitostí jedné operace s komplexními čísly

Cvičení: Dokažte, že permutace vektoru $x_0, \ldots, x_{n-1}$ na levé straně hradlové sítě odpovídá bitovému zrcadlení, 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 $\Theta(n)$:

Algoritmus FFT2

Vstup: $x_0, \ldots, x_{n-1}$

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

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. Číslo 2 je tedy primitivní $2k$-tá odmocnina z jedné. To se nám ovšem nehodí pro algoritmus FFT, neboť $2k$ bude málokdy mocnina dvojky.

Zachrání nás ovšem algebraická věta, která říká, že multiplikativní grupa[⟨1⟩] libovolného konečného tělesa $\mathbb{Z}_p$ je cyklická, tedy že všechny nenulové prvky tělesa lze zapsat jako mocniny nějakého čísla g (generátoru grupy). Jelikož mezi čísly $g^1, g^2, \ldots, g^{p-1}$ se každý nenulový prvek tělesa vyskytne právě jednou, je g primitivní p-tou odmocninou z jedničky. V praxi se hodí například tyto hodnoty:

- $p = 2^{16} + 1 = 65\,537$, $g = 3$, takže funguje $\omega = 3$ pro $n = 2^{16}$ (analogicky $\omega = 3^2$ pro $n = 2^{15}$ atd.),
- $p = 15 \cdot 2^{27} + 1 = 2\,013\,265\,921$, $g = 31$, takže pro $n = 2^{27}$ dostaneme $\omega = g^{15} \bmod p = 440\,564\,289$.

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

- $p = 3 \cdot 2^{30} + 1 = 3\,221\,225\,473$, $g = 5$, takže pro $n = 2^{30}$ vyjde $\omega = g^3 \bmod p = 125$.

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.

Výhodou těchto podob Fourierovy transformace je, že na rozdíl od té klasické komplexní nejsou zatíženy zaokrouhlovacími chybami (komplexní odmocniny z jedničky mají obě složky iracionální). To se hodí například ve zmiňovaných algoritmech na násobení velkých čísel.