1. Fourierova transformace

Co má společného násobení polynomů s kompresí zvuku? Nebo třeba s rozpoznáváním obrazu? V této kapitole ukážeme, že na pozadí všech těchto otázek je společná algebraická struktura, kterou matematici znají pod názvem *diskrétní Fourierova transformace.* Odvodíme efektivní algoritmus pro výpočet této transformace a ukážeme některé jeho zajímavé důsledky.

1.1. Polynomy a jejich násobení

Nejprve stručně připomeňme, jak se pracuje s polynomy.

Definice: *Polynom* je výraz 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. Zatím budeme předpokládat, že všechna čísla jsou reálná, v obecnosti by to mohly být prvky libovolného komutativního okruhu.

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 zadaných na vstupu. Budeme předpokládat, že s reálnými čísly umíme pracovat v konstantním čase na operaci.

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* $\deg P$, nulovému polynomu přiřazujeme stupeň -1.

S polynomy zacházíme jako s výrazy. Sčítání a odečítání je přímočaré, ale podívejme se, co se děje při *násobení:*

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

Po roznásobení můžeme tento součin zapsat jako polynom $R(x)$, jehož koeficient u x^k je roven $r_k = p_0 q_k + p_1 q_{k-1} + \ldots + p_k q_0$. 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, takže celkem $\Theta(n^2)$. Podobně jako u násobení čísel, i zde se budeme snažit najít efektivnější způsob.

Grafy 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. Tehdy říkáme, že jsou *identické* a obvykle to značíme $P \equiv Q$.

Druhá možnost je porovnávat polynomy jako reálné funkce. Polynomy P a Q jsou 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. Takový polynom ovšem nemůže 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 $P \equiv Q$. □

Díky předchozí větě můžeme polynomy 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 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 musíme polynomy vyhodnocovat v dostatečně mnoha bodech.

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 $R \equiv P \cdot Q$ 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.

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

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

Cvičení

1. Odvoďte dělení polynomů se zbytkem: Jsou-li P a Q polynomy a $\deg Q > 0$, pak existují polynomy R a S takové, že $P \equiv QR + S$ a $\deg S < \deg Q$. Zkuste pro toto dělení nalézt co nejefektivnější algoritmus.
2. Převod grafu na polynom v obecném případě: Hledáme polynom stupně nejvýše n, který prochází body $(x_0, y_0), \ldots, (x_n, y_n)$ pro x_i navzájem různá. Pomůže *Lagrangeova interpolace:* definujme polynomy

$$A_j(x) = \prod_{k \neq j} (x - x_j), \quad B_j(x) = \frac{A_j(x)}{\prod_{k \neq j}(x_k - x_j)}, \quad P(x) = \sum_j y_j B_j(x).$$

   Dokažte, že $P(x_j) = y_j$ pro všechna j. K tomu pomůže rozmyslet si, jak vyjde $A_j(x_k)$ a $B_j(x_k)$.
3. Sestrojte co nejrychlejší algoritmus pro Lagrangeovu interpolaci z předchozího cvičení.
4. Jiný pohled na interpolaci polynomů: Hledáme-li polynom $P(x) = \sum_{k=0}^{n} p_k x^k$ procházející body $(x_0, y_0), \ldots, (x_n, y_n)$, řešíme vlastně soustavu rovnic tvaru $\sum_k p_k x_j^k = y_j$ pro $j = 0, \ldots, n$. Tyto rovnice jsou lineární, takže hledáme vektor $\mathbf{x}$ splňující $\mathbf{V}\mathbf{x} = \mathbf{y}$, kde $\mathbf{V}$ je takzvaná *Vandermondova matice* s $\mathbf{V}_{jk} = x_j^k$. Dokažte, že pro x_j navzájem různá je matice $\mathbf{V}$ regulární, takže soustava má právě jedno řešení.

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 z oddílu 1.1.

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 voláme na problém téhož druhu, který je správně spárovaný.

Náš plán použít metodu Rozděl a panuj tedy nakonec 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. $\quad\quad y_j \leftarrow s_j + \omega^j \cdot \ell_j$. *(Mocninu ω^j průběžně přepočítáváme.)*
8. $\quad\quad y_{j+n/2} \leftarrow s_j - \omega^j \cdot \ell_j$.

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 $\mathbf{y}$, daný přepisem

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

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

Jak to souvisí s naším algoritmem? Pokud označíme $\mathbf{p}$ vektor koeficientů 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 ověříme snadno dosazením do definice. Algoritmus tedy počítá diskrétní Fourierovu transformaci v čase $\Theta(n \log n)$. Proto se mu říká FFT – Fast Fourier Transform.

Také si všimněme, že DFT je lineární zobrazení. Jde 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 hledanou inverzní maticí není $\overline{\mathbf{\Omega}}$.

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

Důkaz: Dosazením do definice a elementárními úpravami:

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

To je ovšem geometrická řada. Pokud $j = k$, 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 číslo $\overline{\omega} = \omega^{-1}$ je také primitivní n-tou odmocninou z jedničky, takže až na faktor $1/n$ se jedná opět o Fourierovu transformaci a můžeme ji spočítat stejným algoritmem FFT. Shrňme, co jsme zjistili, do následujících vět:

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

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

Důkaz: Nejprve pomocí DFT v čase $\Theta(n \log n)$ převedeme oba polynomy na grafy, pak v $\Theta(n)$ vynásobíme grafy po složkách a výsledný graf pomocí inverzní DFT v čase $\Theta(n \log n)$ převedeme zpět na koeficienty polynomu. □

Fourierova transformace se kromě násobení polynomů hodí i na ledacos jiného. Své uplatnění nachází v dalších algebraických algoritmech, ale také ve fyzikálních aplikacích – odpovídá totiž spektrálnímu rozkladu signálu na siny a cosiny o různých frekvencích. Na tom jsou založeny například algoritmy pro filtrování zvuku, pro kompresi zvuku a obrazu (MP3, JPEG), nebo třeba rozpoznávání řeči. Něco z toho naznačíme ve zbytku této kapitoly.

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 $\mathbb{C}^n$:
   - $(x, \ldots, x)$
   - $(1, -1, 1, -1, \ldots, 1, -1)$
   - $(1, 0, 1, 0, 1, 0, 1, 0)$
   - $(\omega^0, \omega^1, \omega^2, \ldots, \omega^{n-1})$
   - $(\omega^0, \omega^2, \omega^4, \ldots, \omega^{2n-2})$
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.* Co vypovídá o vektoru $(n/4)$-tý koeficient jeho Fourierova obrazu?

5. Mějme vektor $\mathbf{y}$, který vznikl rotací vektoru $\mathbf{x}$ o k pozic ($\mathbf{y}_j = \mathbf{x}_{(j+k) \bmod n}$). Jak spolu souvisí $\mathcal{F}(\mathbf{x})$ a $\mathcal{F}(\mathbf{y})$?
6. Fourierova báze: Uvažujme systém vektorů $\mathbf{b}^0, \ldots, \mathbf{b}^{n-1}$ se složkami $\mathbf{b}^j_k = \overline{\omega}^{jk}/\sqrt{n}$. Dokažte, že tyto vektory tvoří ortonormální bázi prostoru $\mathbb{C}^n$, pokud použijeme standardní skalární součin nad $\mathbb{C}$, totiž $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_j \mathbf{x}_j \overline{\mathbf{y}_j}$. Složky vektoru $\mathbf{x}$ vzhledem k této bázi pak jsou $\langle \mathbf{x}, \mathbf{b}^0 \rangle, \ldots, \langle \mathbf{x}, \mathbf{b}^{n-1} \rangle$ (to platí pro libovolnou ortonormální bázi). Uvědomte si, že tyto skalární součiny odpovídají definici DFT, tedy až na konstantu $1/\sqrt{n}$.

1.4.* Spektrální rozklad

Ukážeme, jak FFT souvisí s digitálním zpracováním signálu – pro jednoduchost jednorozměrného, tedy třeba zvuku.

Uvažujme reálnou funkci f definovanou na intervalu $[0, 1)$. Pokud její hodnoty *navzorkujeme* v n pravidelně rozmístěných bodech, získáme vektor $\mathbf{f} \in \mathbb{R}^n$ o složkách $\mathbf{f}_j = f(j/n)$. Co o funkci f vypovídá Fourierův obraz vektoru $\mathbf{f}$?

Lemma R: *(DFT reálného vektoru)* Je-li $\mathbf{x}$ reálný vektor z $\mathbb{R}^n$, jeho Fourierův obraz $\mathbf{y} = \mathcal{F}(\mathbf{x})$ je *antisymetrický:* $\mathbf{y}_j = \overline{\mathbf{y}_{n-j}}$ pro všechna j.

Důkaz: Z definice DFT víme, že

$$\mathbf{y}_{n-j} = \sum_k \mathbf{x}_k \omega^{(n-j)k} = \sum_k \mathbf{x}_k \omega^{nk-jk} = \sum_k \mathbf{x}_k \omega^{-jk} = \sum_k \mathbf{x}_k \overline{\omega}^{jk}.$$

Jelikož komplexní sdružení lze distribuovat přes aritmetické operace, platí $\overline{\mathbf{y}_{n-j}} = \sum_k \overline{\mathbf{x}_k} \cdot \omega^{jk}$, což je pro reálné $\mathbf{x}$ rovno $\sum_k \mathbf{x}_k \omega^{jk} = \mathbf{y}_j$. □

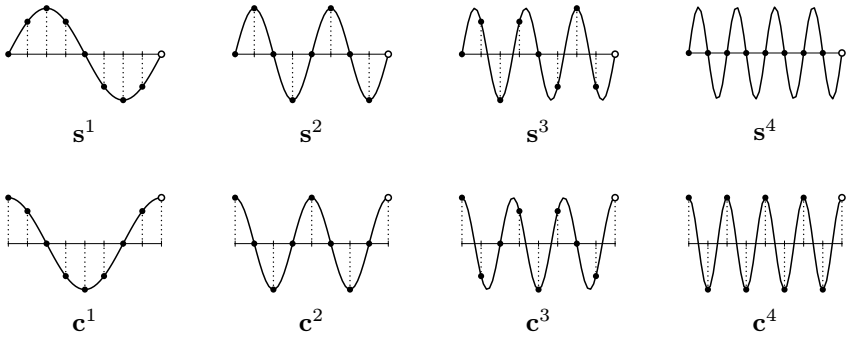
Důsledek: Speciálně $\mathbf{y}_0 = \overline{\mathbf{y}_0}$ a $\mathbf{y}_{n/2} = \overline{\mathbf{y}_{n/2}}$, takže obě tyto hodnoty jsou reálné.

Lemma A: *(O antisymetrických vektorech)* Antisymetrické vektory v $\mathbb{C}^n$ tvoří vektorový prostor dimenze n nad tělesem reálných čísel.

Důkaz: Ověříme axiomy vektorového prostoru. (To, že prostor budujeme nad $\mathbb{R}$, a nikoliv nad $\mathbb{C}$, je důležité: násobení vektoru komplexním skalárem obecně nezachovává antisymetrii.)

Co se dimenze týče: V antisymetrickém vektoru $\mathbf{y}$ jsou složky $\mathbf{y}_0$ a $\mathbf{y}_{n/2}$ reálné, u složek $\mathbf{y}_1, \ldots, \mathbf{y}_{n/2-1}$ můžeme volit jak reálnou, tak imaginární část. Ostatní složky tím jsou už jednoznačně dány. Vektor je tedy určen n nezávislými reálnými parametry. □

Definice: V dalším textu zvolme pevné n a $\omega = \mathrm{e}^{2\pi\mathbf{i}/n}$. Označíme $\mathbf{e}^k$, $\mathbf{s}^k$ a $\mathbf{c}^k$ vektory získané navzorkováním funkcí $\mathrm{e}^{2k\pi\mathbf{i}x}$, $\sin 2k\pi x$ a $\cos 2k\pi x$ (komplexní exponenciála, sinus a cosinus s frekvencí k) v n bodech.



Obr. 1.1: Vektory $\mathbf{s}^k$ a $\mathbf{c}^k$ při vzorkování v 8 bodech

Lemma V: *(O vzorkování funkcí)* Fourierův obraz vektorů $\mathbf{e}^k$, $\mathbf{s}^k$ a $\mathbf{c}^k$ vypadá pro $0 < k < n/2$ následovně:

$$\mathcal{F}(\mathbf{e}^k) = (0, \ldots, 0, n, 0, \ldots, 0),$$
$$\mathcal{F}(\mathbf{s}^k) = (0, \ldots, 0, n/2\mathbf{i}, 0, \ldots, 0, -n/2\mathbf{i}, 0, \ldots, 0),$$
$$\mathcal{F}(\mathbf{c}^k) = (0, \ldots, 0, n/2, 0, \ldots, 0, n/2, 0, \ldots, 0),$$

přičemž první vektor má nenulu na pozici $n-k$, další dva na pozicích k a $n-k$.

Zatímco vztah pro $\mathcal{F}(\mathbf{e}^k)$ funguje i s $k = 0$ a $k = n/2$, siny a cosiny se chovají odlišně: $\mathbf{s}^0$ i $\mathbf{s}^{n/2}$ jsou nulové vektory, takže $\mathcal{F}(\mathbf{s}^0)$ a $\mathcal{F}(\mathbf{s}^{n/2})$ taktéž, $\mathbf{c}^0$ je vektor samých jedniček s $\mathcal{F}(\mathbf{c}^0) = (n, 0, \ldots, 0)$ a $\mathbf{c}^{n/2} = (1, -1, \ldots, 1, -1)$ s $\mathcal{F}(\mathbf{c}^{n/2}) = (0, \ldots, 0, n, 0, \ldots 0)$ s n na pozici $n/2$.

Důkaz: Pro $\mathbf{e}^k$ si stačí všimnout, že $\mathbf{e}^k_j = \mathrm{e}^{2k\pi\mathbf{i}\cdot j/n} = \mathrm{e}^{jk\cdot 2\pi\mathbf{i}/n} = \omega^{jk}$. Proto t-tá složka Fourierova obrazu vyjde $\sum_j \omega^{jk}\omega^{jt} = \sum_j \omega^{j(k+t)}$. To je opět geometrická řada, podobně jako u odvození inverzní FT. Pro $t = n-k$ se sečte na n, všude jinde na 0. Pro $\mathbf{s}^k$ a $\mathbf{c}^k$ necháváme jako cvičení. □

Všimněme si nyní, že reálnou lineární kombinací vektorů $\mathcal{F}(\mathbf{s}^1), \ldots, \mathcal{F}(\mathbf{s}^{n/2-1})$ a $\mathcal{F}(\mathbf{c}^0), \ldots, \mathcal{F}(\mathbf{c}^{n/2})$ můžeme získat libovolný antisymetrický vektor. Jelikož DFT je lineární, plyne z toho, že lineární kombinací $\mathbf{s}^1, \ldots, \mathbf{s}^{n/2-1}$ a $\mathbf{c}^0, \ldots, \mathbf{c}^{n/2}$ lze získat libovolný reálný vektor. Přesněji to říká následující věta:

Věta: Pro každý vektor $\mathbf{x} \in \mathbb{R}^n$ existují koeficienty $\alpha_0, \ldots, \alpha_{n/2} \in \mathbb{R}$ a $\beta_0, \ldots, \beta_{n/2} \in \mathbb{R}$ takové, že:

$$\mathbf{x} = \sum_{k=0}^{n/2} \left(\alpha_k \mathbf{c}^k + \beta_k \mathbf{s}^k\right).$$

Tyto koeficienty jdou navíc vypočíst z Fourierova obrazu

$$\mathcal{F}(\mathbf{x}) = (a_0 + b_0\mathbf{i}, \ldots, a_{n-1} + b_{n-1}\mathbf{i})$$

takto:

$$\begin{aligned}
\alpha_0 &= a_0/n, \\
\alpha_j &= 2a_j/n \quad \text{pro } j = 1, \ldots, n/2. \\
\beta_0 &= \beta_{n/2} = 0, \\
\beta_j &= -2b_j/n \quad \text{pro } j = 1, \ldots, n/2 - 1,
\end{aligned}$$

Důkaz: Jelikož DFT má inverzi, můžeme bez obav fourierovat obě strany rovnice. Chceme, aby platilo $\mathbf{y} = \mathcal{F}(\sum_k \alpha_k \mathbf{s}^k + \beta_k \mathbf{c}^k)$. Suma na pravé straně je přitom díky linearitě $\mathcal{F}$ rovna $\sum_k (\alpha_k \mathcal{F}(\mathbf{s}^k) + \beta_k \mathcal{F}(\mathbf{c}^k))$. Označme tento vektor $\mathbf{z}$ a za vydatné pomoci lemmatu **V** vypočítejme jeho složky:

- K $\mathbf{z}_0$ přispívá pouze $\mathbf{c}^0$ (ostatní $\mathbf{s}^k$ a $\mathbf{c}^k$ mají nultou složku nulovou). Takže $\mathbf{z}_0 = \alpha_0 \mathbf{c}^0_0 = (a_0/n) \cdot n = a_0$.
- K $\mathbf{z}_j$ pro $j = 1, \ldots, n/2 - 1$ přispívají pouze $\mathbf{c}^j$ a $\mathbf{s}^j$: $\mathbf{z}_j = \alpha_j \mathbf{c}^j_j + \beta_j \mathbf{s}^j_j = 2a_j/n \cdot n/2 - 2b_j/n \cdot n/2\mathbf{i} = a_j + b_j\mathbf{i}$.
- K $\mathbf{z}_{n/2}$ přispívá pouze $\mathbf{c}^{n/2}$, takže analogicky vyjde $\mathbf{z}_{n/2} = 2a_{n/2}/n \cdot n/2 = a_{n/2}$.

Vektory $\mathbf{z}$ a $\mathbf{y}$ se tedy shodují v prvních $n/2 + 1$ složkách (nezapomeňte, že $b_0 = b_{n/2} = 0$). Jelikož jsou oba antisymetrické, musí se shodovat i ve zbývajících složkách. □

Důsledek: Pro libovolnou reálnou funkci f na intervalu $[0, 1)$ existuje lineární kombinace funkcí $\sin 2k\pi x$ a $\cos 2k\pi x$ pro $k = 0, \ldots, n/2$, která není při vzorkování v n bodech od dané funkce f rozlišitelná.

To je diskrétní ekvivalent známého tvrzení o spojité Fourierově transformaci, podle nejž každá „rozumně hladká“ periodická funkce jde lineárně nakombinovat ze sinů a cosinů o celočíselných frekvencích.

To se hodí například při zpracování zvuku: jelikož $\alpha \cos x + \beta \sin x = A \sin(x + \varphi)$ pro vhodné A a φ, můžeme kterýkoliv zvuk rozložit na sinusové tóny o různých frekvencích. U každého tónu získáme jeho amplitudu A a fázový posun φ, což je vlastně

(až na nějaký násobek n) absolutní hodnota a argument původního komplexního Fourierova koeficientu. Tomu se říká *spektrální rozklad* signálu a díky FFT ho můžeme z navzorkovaného signálu spočítat velmi rychle.

Cvičení

1. Dokažte „inverzní“ lemma **R**: DFT antisymetrického vektoru je vždy reálná.
2. Dokažte zbytek lemmatu **V**: Jak vypadá $\mathcal{F}(\mathbf{s}^k)$ a $\mathcal{F}(\mathbf{c}^k)$?
3.* Analogií DFT pro reálné vektory je *diskrétní cosinová transformace (DCT).* Z DFT v $\mathbb{C}^n$ odvodíme DCT v $\mathbb{R}^{n/2+1}$. Vektor $(x_0, \ldots, x_{n/2})$ doplníme jeho zrcadlovou kopií na $\mathbf{x} = (x_0, x_1, \ldots, x_{n/2}, x_{n/2-1}, \ldots, x_1)$. To je reálný a antisymetrický vektor, takže jeho Fourierův obraz $\mathbf{y} = \mathcal{F}(\mathbf{x})$ musí být podle lemmatu **R** a cvičení 1 také reálný a antisymetrický: $\mathbf{y} = (y_0, y_1, \ldots, y_{n/2}, y_{n/2-1}, \ldots, y_1)$. Vektor $(y_0, \ldots, y_{n/2})$ prohlásíme za výsledek DCT. Rozepsáním $\mathcal{F}^{-1}(\mathbf{y})$ podle definice dokažte, že tento výsledek popisuje, jak zapsat vektor $\mathbf{x}$ jako lineární kombinaci cosinových vektorů $\mathbf{c}^0, \ldots, \mathbf{c}^{n/2}$. Oproti DFT tedy používáme pouze cosiny, zato však o dvojnásobném rozsahu frekvencí.
4. *Konvoluce* vektorů $\mathbf{x}$ a $\mathbf{y}$ je vektor $\mathbf{z} = \mathbf{x} * \mathbf{y}$ takový, že $\mathbf{z}_j = \sum_k \mathbf{x}_k \mathbf{y}_{j-k}$, přičemž indexujeme modulo n. Tuto sumu si můžeme představit jako skalární součin vektoru $\mathbf{x}$ s vektorem $\mathbf{y}$ napsaným pozpátku a zrotovaným o j pozic. Konvoluce nám tedy řekne, jak tyto „přetočené skalární součiny“ vypadají pro všechna j. Dokažte následující vlastnosti:
   a) $\mathbf{x} * \mathbf{y} = \mathbf{y} * \mathbf{x}$ *(komutativita)*
   b) $\mathbf{x} * (\mathbf{y} * \mathbf{z}) = (\mathbf{x} * \mathbf{y}) * \mathbf{z}$ *(asociativita)*
   c) $\mathbf{x} * (\alpha\mathbf{y} + \beta\mathbf{z}) = \alpha\mathbf{x} * \mathbf{y} + \beta\mathbf{x} * \mathbf{z}$ *(bilinearita)*
   d) $\mathcal{F}(\mathbf{x}*\mathbf{y}) = \mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{y})$, kde $\odot$ je součin vektorů po složkách. To nám dává algoritmus pro výpočet konvoluce v čase $\Theta(n \log n)$.
5. Vyhlazování signálu: Mějme vektor $\mathbf{x}$ naměřených dat. Obvyklý způsob, jak je vyčistit od šumu, je transformace typu $\mathbf{y}_j = \frac{1}{4}\mathbf{x}_{j-1} + \frac{1}{2}\mathbf{x}_j + \frac{1}{4}\mathbf{x}_{j+1}$. Ta „obrousí špičky“ tím, že každou hodnotu zprůměruje s okolními. Pokud budeme $\mathbf{x}$ indexovat cyklicky, jedná se o konvoluci $\mathbf{x}*\mathbf{z}$, kde $\mathbf{z}$ je maska tvaru $(\frac{1}{2}, \frac{1}{4}, 0, \ldots, 0, \frac{1}{4})$. Fourierův obraz $\mathcal{F}(\mathbf{z})$ nám říká, jak vyhlazování ovlivňuje spektrum. Například pro $n = 8$ vyjde $\mathcal{F}(\mathbf{z}) \approx (1, 0.854, 0.5, 0.146, 0, 0.146, 0.5, 0.854)$, takže stejnosměrná složka signálu $\mathbf{c}^0$ zůstane nezměněna, naopak nejvyšší frekvence $\mathbf{c}^4$ zcela zmizí a pro ostatní $\mathbf{c}^k$ a $\mathbf{s}^k$ platí, že čím vyšší frekvence, tím víc je tlumena. Tomu se říká *dolní propust* – nízké frekvence propustí, vysoké omezuje. Jak vypadá propust, která vynuluje $\mathbf{c}^4$ a ostatní frekvence propustí beze změny?
6. Zpět k polynomům: Uvědomte si, že to, co se děje při násobení polynomů s jejich koeficienty, je také konvoluce. Jen musíme doplnit vektory nulami, aby se neprojevila cykličnost indexování. Takže vztah $\mathcal{F}(\mathbf{x}*\mathbf{y}) = \mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{y})$ z cvičení 4 je jenom jiný zápis našeho algoritmu na rychlé násobení polynomů.
7.** Diagonalizace: Mějme vektorový prostor dimenze n a lineární zobrazení f v něm. Zvolíme-li si nějakou bázi, můžeme vektory zapisovat jako n-tice čísel a zobra-

zení f popsat jako násobení n-tice vhodnou maticí $\mathbf{A}$ tvaru $n \times n$. Někdy se povede najít bázi z vlastních vektorů, vzhledem k níž je matice $\mathbf{A}$ *diagonální* – má nenulová čísla pouze na diagonále. Tehdy umíme součin $\mathbf{Ax}$ spočítat v čase $\Theta(n)$. Pro jeden součin se to málokdy vyplatí, protože převod mezi bázemi bývá pomalý, ale pokud jich chceme počítat hodně, pomuže to.

Rozmyslete si, že podobně se můžeme dívat na DFT. Konvoluce je bilineární funkce na $\mathbb{C}^n$, což znamená, že je lineární v každém parametru zvlášť. Zvolíme-li bázi, můžeme každou bilineární funkci popsat trojrozměrnou tabulkou $n \times n \times n$ čísel (to už není matice, ale *tenzor třetího řádu*). Vztah $\mathcal{F}(\mathbf{x} * \mathbf{y}) = \mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{y})$ pak můžeme vyložit takto: $\mathcal{F}$ převádí vektory z kanonické báze do *Fourierovy báze* (viz cvičení 1.3.6), vzhledem k níž je tenzor konvoluce diagonální (má jedničky na „tělesové úhlopříčce“ a všude jinde nuly). Pomocí FFT pak mužeme mezi bázemi převádět v čase $\Theta(n \log n)$.

8.* Při zpracování obrazu se hodí *dvojrozměrná DFT,* která matici $\mathbf{X} \in \mathbb{C}^{n \times n}$ přiřadí matici $\mathbf{Y} \in \mathbb{C}^{n \times n}$ takto (ω je opět primitivní n-tá odmocnina z jedné):

$$\mathbf{Y}_{jk} = \sum_{uv} \mathbf{X}_{uv} \omega^{ju+kv}.$$

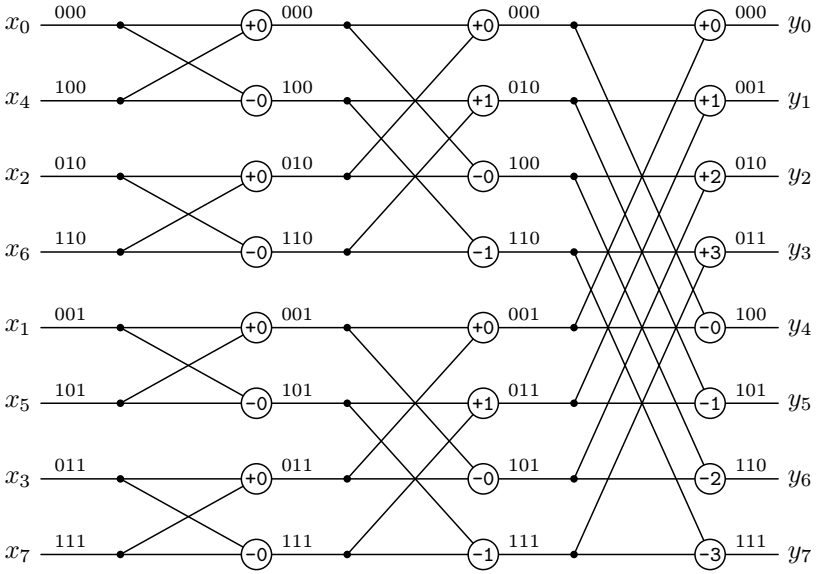
Ověřte, že i tato transformace je bijekce, a odvoďte algoritmus na její efektivní výpočet pomocí jednorozměrné FFT. Fyzikální interpretace je podobná: Fourierův obraz popisuje rozklad matice na „prostorové frekvence“. Také lze odvodit dvojrozměrnou cosinovou transformaci, na níž je založený například kompresní algoritmus JPEG.

1.5.* Další varianty FFT

FFT jako hradlová síť

Zkusme průběh algoritmu FFT znázornit graficky. Na levé straně obrázku 1.2 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 $x_1, x_3, \ldots, x_{n-1}$. Kroužky přitom odpovídají výpočtu lineární kombinace $a + \omega^k b$, kde a, b jsou vstupy kroužku (a přichází rovně, b šikmo) a k číslo uvnitř 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.

Čísla nad čarami prozrazují, jak jsme došli k permutaci vstupních hodnot nalevo. Ke každému podproblému jsme napsali ve dvojkové soustavě indexy prvků vstupu, ze kterých podproblém počítáme. V posledním sloupci je to celý vstup. V předposledním nejprve indexy končící 0, pak ty končící 1. O sloupec vlevo jsme každou podrozdělili podle předposlední číslice atd. Až v prvním sloupci jsou čísla



Obr. 1.2: Průběh FFT pro vstup velikosti 8

uspořádaná podle dvojkových zápisů čtených pozpátku. (Rozmyslete si, proč tomu tak je.)

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 tedy pracuje v čase $\Theta(\log n)$ a prostoru $\Theta(n)$.

Nerekurzivní FFT

Obvod z obrázku 1.2 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: Komplexní čísla $x_0, \ldots, x_{n-1}$, primitivní n-tá odmocnina z jedné ω.

1. Předpočítáme tabulku hodnot $\omega^0, \omega^1, \ldots, \omega^{n-1}$.
2. Pro $k = 0, \ldots, n-1$ položíme $y_k \leftarrow x_{r(k)}$, kde r je funkce bitového zrcadlení.
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 tvaru $2^k + 1$ platí $2^k = -1$. Proto $2^{2k} = 1$ a $2^0, 2^1, \ldots, 2^{2k-1}$ jsou navzájem různé. 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 všech $p-1$ nenulových prvků tělesa lze zapsat jako mocniny nějakého čísla g (generátoru grupy). Jelikož mezi čísly $g^0, g^1, g^2, \ldots, g^{p-2}$ se každý nenulový prvek tělesa vyskytne právě jednou, je g primitivní $(p-1)$-ní 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 $\omega = g^{15} \bmod p = 440\,564\,289$.
- $p = 3 \cdot 2^{30} + 1 = 3\,221\,225\,473$, $g = 5$, 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 v algoritmech na násobení velkých čísel – viz cvičení 1.

Cvičení

1.* Pomocí FFT lze rychle násobit čísla. Každé n-bitové číslo x můžeme rozložit na k-bitové bloky $x_0, \ldots, x_{m-1}$ (kde $m \approx n/k$). To je totéž, jako kdybychom ho zapsali v soustavě o základu $B = 2^k$: $x = \sum_j x_j B^j$. Pokud k číslu přiřadíme polynom $X(t) = \sum_j x_j t^j$, bude $X(B) = x$.

   To nám umožňuje převést násobení čísel na násobení polynomů: Chceme-li vynásobit čísla x a y, sestrojíme polynomy X a Y, pomocí FFT vypočteme jejich součin Z a pak do něj dosadíme B. Dostaneme $Z(B) = X(B) \cdot Y(B) = xy$.

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

Na RAMu přitom mužeme zvolit $k = \Theta(\log n)$, takže s čísly polynomiálně velkými vzhledem k B zvládneme počítat v konstantním čase. Proto FFT ve vhodném konečném tělese poběží v čase $\mathcal{O}(m \log m) = \mathcal{O}(n)$. Zbývá domyslet, jak vyhodnotit $Z(B)$. Spočítejte, jak velké jsou koeficienty polynomu Z a ukažte, že při vyhodnocování od nejnižších řádů jsou přenosy dostatečně malé na to, abychom vše stihli v čase $\Theta(n)$.

Tím jsme získali algoritmus na násobení n-bitových čísel v čase $\Theta(n)$.