Algoritmická matematika 3

Rozděl a panuj

Petr Osička

DATA ANALYSIS AND MODELING LAB
Univerzita Palackého v Olomouci

Zimní semestr 2013

Základní idea

Pro vstupní instanci I

1. Rozděl I na k menších instancí $I_1, \ldots, I_k$ stejného problému
2. Najdi řešení instancí $I_1, \ldots, I_k$
   - Pro malé instance už známe odpověď, nebo použijeme jiný algoritmus
   - Jinak použime opět přístup rozděl a panuj
3. řešení instancí $I_1, \ldots, I_k$ zkombinuj do řesení instance I

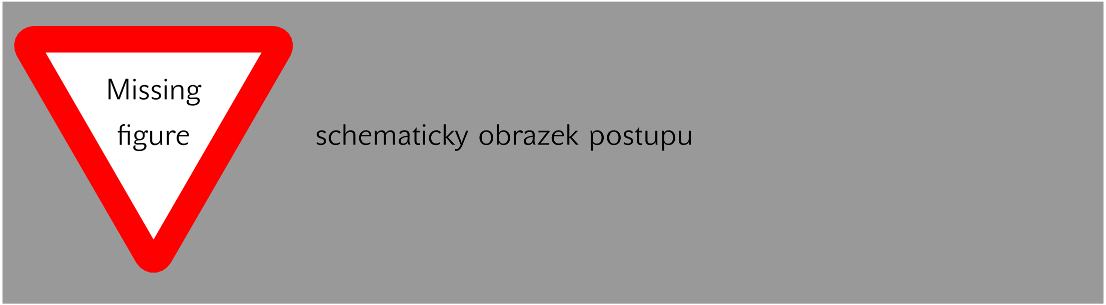


Schéma v pseudokódu

procedure DIVIDE-AND-CONQUER(I)
    if $|I| \leq c$ **then**
        return BASICALGORITM(I)
    end if
    Vytvoř instance $I_1 \dots I_k$ menší velikosti než I
    for $i \leftarrow 1$ **to** k **do**
        $x_i \leftarrow$DIVIDE-AND-CONQUER(x_i)
    end for
    Zkombinuj $x_1 \dots x_k$ do x
    return x
end procedure

Příklad

1. *Výpočet Fibonnaciho čísla.*
   n-té Fibonnaciho číslo budeme značit jako F(n), definice následuje

$$F(n) = \begin{cases} F(n-1) + F(n-2) & n \geq 2 \\ n & n \in \{0, 1\} \end{cases} \tag{1}$$

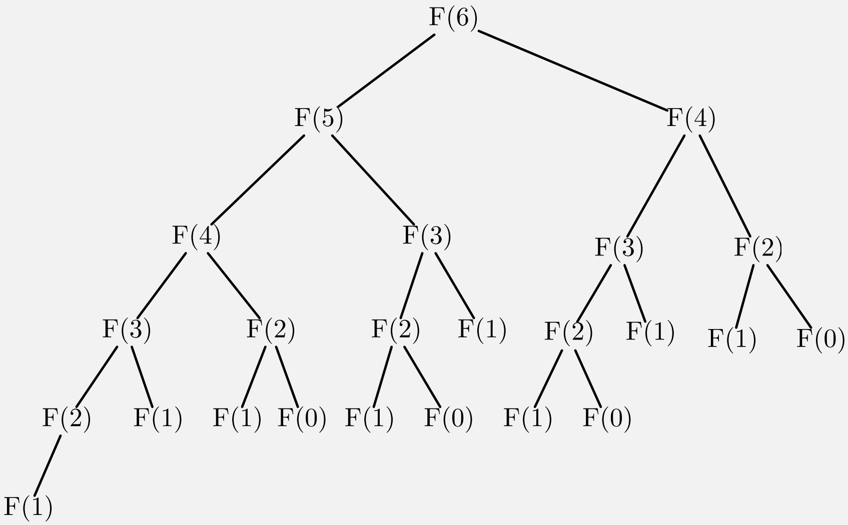
Idea algoritmu

- Pokud $n \in \{0, 1\}$, vrať n
- vstupní instanci n „rozděl" na dvě menší instance $n-1$ a $n-2$.
- rekurzivně vypočítej $F(n-1)$ a $F(n-2)$
- spočítej $F(n)$ pomocí (1)

Jak uvidíme později (přednáška *Dynamickém programování*), algoritmus založený na předchozí myšlence je neefektivní. Později si řekneme proč. Příklad je přesto užitečný pro pozdější demonstraci toho, jak se počítá složitost algoritmů fungujících na principu rozděl a panuj.

Příklad (pokračování)

Strom rekurze odpovídající výpočtu $F(6)$



Příklad (pokračování)

2. *Třídění sléváním*

```
1: procedure MergeSort(A, l, p)
2:     if p < l then
3:         q ← ⌊(l + p)/2⌋
4:         MergeSort(A, l, q)
5:         MergeSort(A, q + 1, p)
6:         Merge(A, l, q, p)
7:     end if
8: end procedure
```

- **fáze rozděl** (ř. 3-5): rozděl vstupní pole na poloviny a ty setřiď
- **fáze panuj** (ř. 6): slej setřízená pole do jednoho (proceduru Merge znáte z ALM1, pseudokód je v handoutech)

Příklad (pokračování)

Missing figure

schematicky obrazek prubehu mergesortu na malem poli

Analýza složitosti algoritmů rozděl a panuj

1: **procedure** DIVIDE-AND-CONQUER(I)
2: **if** $|I| \leq c$ **then**
3: **return** BASICALGORITM(I)
4: **end if**
5: Vytvoř instance $I_1 \dots I_k$ menší velikosti než I
6: **for** $i \leftarrow 1$ **to** k **do**
7: $x_i \leftarrow$DIVIDE-AND-CONQUER(x_i)
8: **end for**
9: Zkombinuj $x_1 \dots x_k$ do x
10: **return** x
11: **end procedure**

- složitost označíme $T(n)$
- pokud je $|I| \leq c$, pak je složitost konstanta: $T(|I|) = O(1)$ pro $|I| \leq c$
- jinak je složitost sumou složitostí rekurzivních zavolání (ř. 6-7) a $f(n)$ zachycující sumu složitostí ř. 5 a 9, tedy

$$T(|I|) = \sum_{i=1}^{k} T(|I_i|) + f(|I|).$$

Příklad

1: **procedure** MergeSort(A, l, p)
2: **if** $p < l$ **then**
3: $q \leftarrow \lfloor (l+p)/2 \rfloor$
4: MergeSort(A, l, q)
5: MergeSort$(A, q+1, p)$
6: Merge(A, l, q, p)
7: **end if**
8: **end procedure**

Spočítáme počet provedených porovnání prvků z A

- velikost instance je $n = p - l + 1$
- pro $l = p$, třídíme 1 prvek, tedy se neprovede porovnání, složitost je tedy T(1) = 0
- ř. 3 neprovádí žádné porovnání, procedura merge na ř. 6 provede nejvýše n porovnání (za porovnání počítáme i zjištění, že už není s čím porovnávat)
- celková složitost tedy je $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$

Rekurence

Co to je?
Výrazy typu $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$, kdy hodnotu funkce na levé straně získáme za pomoci jejích hodnot (pro jiné body) na pravé straně.

Řešení rekurence
Nalezení **uzavřené formy** dané rekurence. Tj. takové vyjádření $T(n)$, které neobsahuje volání sama sebe a obsahuje aplikaci pouze konečného množství známých operací.

Odhad rekurence
Co nejpřesnější určení funkce $f(n)$ tak, že $T(n) \in O(f(n))$ a/nebo $T(n) \in \Omega(f(n))$.

Příklad
Fibonnaciho čísla jsou definována pomocí rekurence. Nalezení jejího řešení znamená nalezení vzorečku pro n-té Fibonnaciho číslo. Uzavřená forma je $\frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}$, kde $\varphi = \frac{1+\sqrt{5}}{2}$. Jako odhad by postačovalo $O(2^n)$.

Metody řešení rekurencí

1. **Substituční metoda**
   Funguje ve dvou krocích. První krok je **odhad** (= uhodnutí) uzavřené formy. Druhý krok je **důkaz správnosti uzavřené formy pomocí indukce**.

   Pro jednoduché rekurence lze najít uzavřenou formu. Pro složitější rekurence lze metodu použít pro nalezení odhadu a důkaz jeho správnosti.

2. **Odhad pomocí stromu rekurze**
   Pomocí této metody lze uzavřenou formu (nebo její odhad) nalézt. Postup je nutno dokončit pomocí Substituční metody. Uzavřená forma je

3. **Master theorem**
   Kombinace předchozích dvou typů fungující pro specifický typ rekurencí.

Substituční metoda

Příklad

Uvažme rekurenci

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$
$$T(1) = 1$$

Postup řešení

1. Odhadneme, že řešením je lineární funkce, tedy $T(n) \in O(n)$.
2. Nyní musíme dokázat, že pro jednu funkci $f(n) \in O(n)$ ukázat, že $T(n) \leq f(n)$. Učiníme tak indukcí. Vybereme funkci $cn - b$ a snažíme se najít konstanty b a c tak, aby důkaz indukcí fungoval, tj. takové, že $T(n) \leq cn - b$.

   Indukční předpoklad je

   $$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor - b$$
   $$T(\lceil n/2 \rceil) \leq c\lceil n/2 \rceil - b.$$

Příklad (pokračování)

Po dosazení do rekurence dostaneme

$$T(n) \leq (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1 = cn - 2b + 1$$

Výraz $cn - 2b + 1$ ted rozdělíme na sumu výrazu, který chceme obdržet a zbytku, tzv. residentu

$$cn - 2b + 1 = cn - b + (-b + 1).$$

Odtud vidíme, že $b = 1$.

Po dosazení do okrajové podmínky dostaneme nerovnost

$$T(1) = 1 \leq c - 1,$$

která platí pro $c \geq 2$.

Substituční metoda

Poznámky

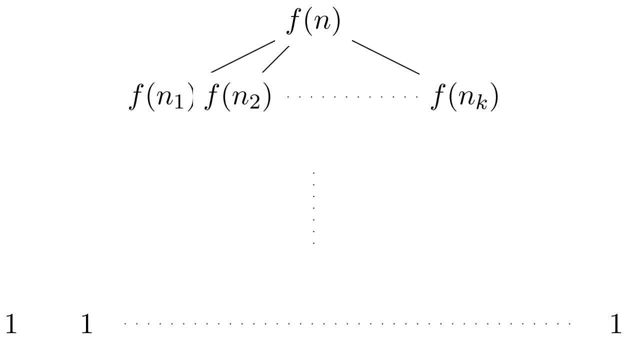
- předchozí příklad byl triviální
- obvykle je potřeba více triků (dosazování, změna okrajových podmínek, zkoušení různých reprezentantů)
- je potřeba experimentovat a zkoušet různé postupy
- získat počáteční odhad lze buď pomocí zkušeností (např. znáte algoritmus, který vede na podobnou rekurenci), nebo pomocí metody stromu
- více na cvičeních, v handoutech nebo v literatuře.

Metoda rekurzivního stromu

Rekurentní vztah $T(n) = \sum_{i=1}^{k} T(n_i) + f(n)$ si **představíme jako strom.**

- Uzly stromu odpovídají jednotlivým rekurzivním voláním T
- Každému uzlu přiřadíme množství práce, kterou pomyslný algoritmus při daném volání provede (funkce f zavolaná na příslušný argument).
- Listové uzly tak budou mít přiřazenu konstantu, která odpovídá okrajovým podmínkám rekurentního vztahu.
- *Sečteme-li práci přiřazenou všem uzlům, dostaneme řešení rekurence.*
- Součet provedeme ve dvou krocích.
  1. pro každou úroveň stromu sečteme práci provedenou v uzlech na oné úrovni,
  2. sečteme sumy z jednotlivých vrstev.

Metoda rekurzivního stromu



$f(n)$

$+\ \sum_{i=1}^{k} f(n_k)$

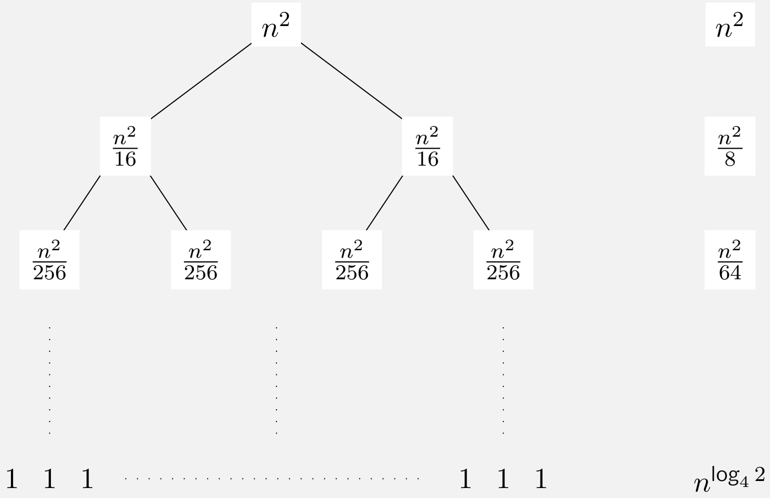
⋮

+ počet listů

= výsledek

Příklad

Uvažujme rekurenci $T(n) = 2T(n/4) + n^2$. Odpovídá jí strom



Příklad

Můžeme si všimnout, že

- Práce, kterou provedeme v jednom uzlu v i-té vrstvě (kořen je v nulté vrstvě) odpovídá $(n/4^i)^2$, počet uzlů v i-té vrstvě je 2^i, suma přes všechny uzly v této vrstvě je tedy $2^i(n/4^i)^2 = n^2/8^i$.
- Listy se nacházejí ve vrstvě $\log_4 n$, jejich počet je tedy $2^{\log_4 n} = n^{\log_4 2} = n^{1/2}$.

Pokud nyní sečteme všechny vrstvy, dostaneme

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n - 1} (n^2/8^i) + n^{1/2} \\
&= n^2 \sum_{i=0}^{\log_4 n - 1} (1/8^i) + n^{1/2}
\end{aligned}
$$

Příklad (pokračování)

Abychom se zbavili závislosti na n v sumě, nahradíme ji sumou celé geometrické posloupnosti. Dostaneme tedy

$$
\begin{aligned}
T(n) &= n^2 \sum_{i=0}^{\log_4 n - 1} (1/8^i) + n^{1/2} \\
&\leq n^2 \sum_{i=0}^{\infty} (1/8^i) + n^{1/2} \\
&= n^2 \frac{1}{1 - 1/8} + n^{1/2}
\end{aligned}
$$

Odhad řešení rekurence je tedy $O(n^2)$.

Master Theorem

Věta

Nechť $a \geq 1$ *a* $b > 1$ *jsou konstanty a* $f(n)$ *je funkce a* $T(n)$ *je definovaná na nezáporných celých číslech pomocí rekurence*

$$T(n) = aT(n/b) + f(n),$$

přičemž n/b *interpretujeme jako* $\lfloor n/b \rfloor$ *nebo* $\lceil n/b \rceil$*. Pak můžeme* $T(n)$ *následovně asymptoticky omezit.*

1. *Pokud* $f(n) = O(n^{\log_b a - \epsilon})$ *pro* $\epsilon \geq 0$, *pak* $T(n) = \Theta(n^{\log_b a})$.
2. *Pokud* $f(n) = \Theta(n^{\log_b a})$ *pak* $T(n) = \Theta(n^{\log_b a} \log n)$.
3. *Pokud* $f(n) = \Omega(n^{\log_b a + \epsilon})$ *pro* $\epsilon \geq 0$ *a pokud* $af(n/b) \leq cf(n)$ *pro konstantu* $c < 1$ *a všechna dostatečně velká* n, *pak* $T(n) = \Theta(f(n))$.

Příklad

Uvažujme rekurenci $T(n) = 2T(n/4) + n^2$.

Vidíme, že $a = 2$, $b = 4$, $f(n) = n^2$ a $\log_b a = 1/2$

Případ 1:
$n^2 \in O(n^{1/2-\epsilon})$? Takové $\epsilon > 0$ neexistuje.

Případ 2:
$n^2 \in O(n^{1/2})$? Ne.

Případ 3:
$n^2 \in O(n^{1/2+\epsilon})$? Ano, pro $0 < \epsilon < 3/2$. Navíc $2(n^2/4) = n^2/8 \leq cn^2$ platí pro $1/8 < c < 1$.

Platí případ 3 a odhadem $T(n)$ je $\Theta(n^2)$.

Nalezení dvojice nejbližších bodů na ploše

- **vstup**: množina bodů $P = \{p_1, p_2, \dots\}$, $p_i = \langle x_i, y_i \rangle$.
- $p_i, p_j \in P$, vzdálenost $d(p_i, p_j)$ je definována

$$d(p_i, p_j) = \sqrt{|x_i - x_j|^2 - |y_i - y_j|^2}.$$

- **cíl:** je nalézt dvojici různých bodů $p_i, p_j \in P$, jejichž vzdálenost je nejmenší mezi všemi dvojicemi bodů z P.
- naivní algoritmus má složitost $O(n^2)$.

Nalezení dvojice nejbližších bodů na ploše

Algoritmus je založen na následující myšlence. Pro množinu bodů P

1. Rozdělíme body vertikální čarou na poloviny (v případě lichého počtu bodů má levá polovina o jeden bod více).
2. Rekurentně nalezneme dvojici nejbližších bodů pro levou i pravou polovinu. Rekurze končí v případě, že množina obsahuje pouze 3 body, to nalezneme dvojici nejbližších bodů hrubou silou.
3. Dvojicí nejbližších bodů v P je pak buď lepší z dvojic nejbližších bodů v levé a pravé polovině, nebo dvojice s jedním bodem z levé a s jedním bodem z pravé poloviny. Existenci takové dvojice lze efektivně ověřit.

Nalezení dvojice nejbližších bodů na ploše

Pro množinu bodů P:

- P_x = seznam bodů z P uspořádaný vzestupně podle x-ové souřadnice.
- P_y = seznam bodů z P uspořádaných vzestupně podle y-ové souřadnice.
- Q = množina prvních $\lceil |P|/2 \rceil$ bodů ze seznamu P_x
- R zbývající body jako
- dvojice nejbližších bodů ve Q je q_1^* a q_2^*,
- dvojice nebližších bodů v R je r_1^* a r_2^*.
- δ = kratší ze vdáleností $d(q_1^*, q_2^*)$ a $d(r_1^*, r_2^*)$.

Pokud v P existuje dvojice bodů $q \in Q$ a $r \in R$ taková, že $d(q, r) < \delta$, lze tuto dvojici efektivně najít.

Nalezení dvojice nejbližších bodů na ploše

Theorem

Nechť x^* *je* x*-ová souřadnice nejpravějšího bodu v* Q *a nechť* L *je svislá čára daná rovnicí* $x = x^*$*. Pokud existují body* $q \in Q$ *a* $r \in R$ *takové, že* $d(q, r) < \delta$*, pak tyto body leží maximálně ve zdálenosti* δ *od* L*.*

Důkaz.

Označme $q = \langle q_x, q_y \rangle$ a $r = \langle r_x, r_y \rangle$. Z definice x^* plyne, že $q_x \leq x^* \leq r_x$. Odtud máme, že platí

$$x^* - q_x \leq r_x - q_x \leq d(q, r) \leq \delta,$$

a

$$r_x - x^* \leq r_x - q_x \leq d(q, r) \leq \delta,$$

z čehož už trvzení plyne. □

Nalezení dvojice nejbližších bodů na ploše

Při hledání q a r můžeme omezit na body ležící ve vzdálenosti maximálně δ od L. Označme si množinu takových bodů jako S.

Theorem

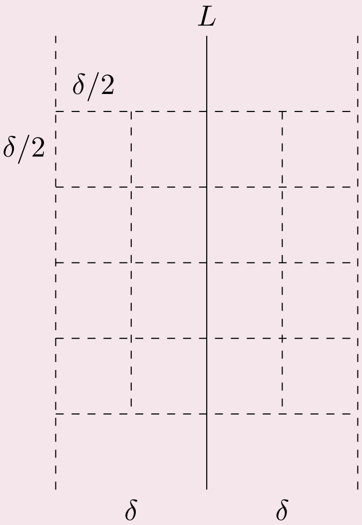
Pro všechny body $s', s \in S$ platí, že pokud $d(s', s) < \delta$, pak s' a s jsou od sebe v setřízeném seznamu S_y vzdáleny maximálně 15 míst.

Důkaz.

Větu dokážeme následující geometrickou konstrukcí. Představíme si plochu obsahující všechny prvky S, a tuto plochu rozdělíme na čtverce o velikosti stran $\delta/2$. Všimněme si, že jedna řada se skládá ze 4 čtverců.

$\square$

pokračování.



$\square$

Nalezení dvojice nejbližších bodů na ploše

pokračování.

Nyní dokážeme, že každý takový čtverec může obsahovat pouze jeden bod z S. Důkaz provedeme sporem. Předpokládejme, že dva body z S leží ve stejném čtverci. Pak buď oba dva leží v Q nebo oba dva leží v R. Díky tomu, že strana čtverce je $\delta/2$, je maximální vzdálenost mezi těmito body rovna $\delta \cdot \sqrt{2}/2 \leq \delta$, což je spor s tím, že nejmenší vzdálenost mezi body uvnitř Q nebo uvnitř R je δ.

Vezměme nyní $s, s' \in S$ takové, že jsou od sebe v seznamu S_y vzdáleny 16 míst. Předpokládejme, že $s_y < s'_y$. Díky tomu, že v každém čtverci může být maximálně jeden bod, musí mezi s a s' ležet minimálně tři řady čtverců (v každé řadě jsou 4 čtverce). Tedy vzdálenost mezi s a s' je minimálně $3\delta/2$. □

Algoritmus

procedure ClosestPair(P)
    sestav P_x a P_y
    return ClosestPairHelp(P_x, P_y)
end procedure

procedure ClosestPairHelp(P_x, P_y)
    if $|P_x| \leq 3$ **then**
        najdi (p_0^*, p_1^*) hrubou silou
        return (p_0^*, p_1^*)
    end if
    sestav Q_x, Q_y, R_x, R_y
    $(q_0^*, q_1^*) \leftarrow$ClosestPairHelp($Q_x, Q_y$)
    $(r_0^*, r_1^*) \leftarrow$ClosestPairHelp($R_x, R_y$)
    if $d(q_0^*, q_1^*) \leq d(r_0^*, r_1^*)$ **then**
        $\delta \leftarrow d(q_0^*, q_1^*)$
        $(b_0, b_1) \leftarrow (q_0^*, q_1^*)$
    else
        $\delta \leftarrow d(r_0^*, r_1^*)$
        $(b_0, b_1) \leftarrow (r_0^*, r_1^*)$
    end if
    $x^* \leftarrow \max\{q_x \mid q \in Q\}$
    $S \leftarrow \{p \in P \mid |p_x - x^*| \leq \delta\}$
    sestav S_y
    for $s \in S$ **do**
        for $s' \in S$, $S_y[s'] - S_y[s] \leq 15$ **do**
            if $d(b_0, b_1) > d(s, s')$ **then**
                $(b_0, b_1) \leftarrow (s, s')$
            end if
        end for
    end for
    return (b_0, b_1)
end procedure

Nalezení dvojice nejbližších bodů na ploše

Složitost

- řádky 11 a 23 provést v lineárním čase, stejně tak cyklus na řádcích 24–30 a řádek 21.
- řádky 7–10 a podmínku na řádcích 14-20 lze provést v konstantním čase.
- složitost ClosestPairHelp – rekurence

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n),$$

  jejímž řešením je $O(n \log n)$.
- Třídění v proceduře ClosestPair má složitost $O(n \log n)$,
- celkově je tedy složitost $O(n \log n)$, kde n je počet prvků v P.

Násobení polynomů a FFT

Pro polynomy

$$A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_dx^d$$
$$B(x) = b_0 + b_1x + b_2x^2 + \cdots + b_dx^d$$

je jejich součinem polynom definovaný jako

$$AB(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{2d}x^{2d},$$

kde

$$c_k = a_0b_k + a_1b_{k-1} + \cdots + a_kb_0 = \sum_{i=0}^{k} a_ib_{k-i}.$$

Že tomu tak je snadno ověříme **jednoduchým roznásobením** polynomů A a B. Složitost takového násobení je $O(d^2)$ (násobíme „každý s každým“).

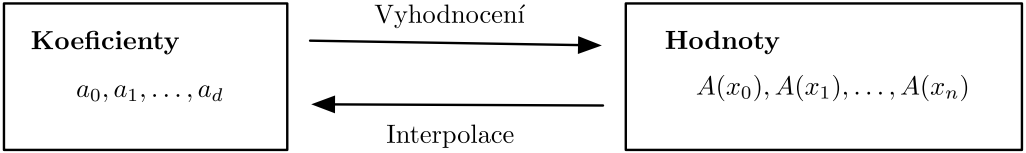
Násobení polynomů a FFT

zefektivnění = změna reprezentace polynomu

Věta

Polynom stupně d je jednoznačně reprezentován svými hodnotami v alespoň $d+1$ různých bodech.

- místo koeficienty polynom reprezentujeme pomocí hodnot polynomu ve stejném počtu bodů, násobení pak má linerání složitost
- mezi oběma reprezentacemi lze přecházet



Násobení polynomů a FFT

Pro rychlý výpočet hodnot polynomu i interpolaci lze použít **Rychlou Fourierovu Transformaci (FFT).**

- **vstup**: polynom reprezentovaný n koeficienty, kde $n = 2^k$ (proč to lze?)
- **výstup**: reprezentace polynomu hodnotami v n bodech (ve kterých, je dáno algoritmem)
- funguje na základě principu rozděl a panuj

Naivní algoritmus

- dosadíme n bodu do n výskytů proměnné v polynomu
- složitost je tedy $\Theta(n^2)$
- aby se FFT vyplatila, musí mít složitost lepší

Idea: druhé mocniny opačných bodů se rovnají

Uvažujme body $\pm x_0, \pm x_1, \cdots \pm x_{n/2-1}$. Polynom rozdělíme na sudé a liché mocniny

$$A(x) = (a_0 + a_2x^2 + \dots) + x(a_1 + a_3x^3 + \dots)$$

- **polynom v levé závorce**:

$$A_S(z) = (a_0 + a_2z + \dots) \text{ a tedy } (a_0 + a_2x^2 + \dots) = A_S(x^2)$$

- **polynom v pravé závorce**:

$$A_L(z) = (a_1 + a_3z + \dots) \text{ a tedy } (a_1 + a_3x^3 + \dots) = A_L(x^2).$$

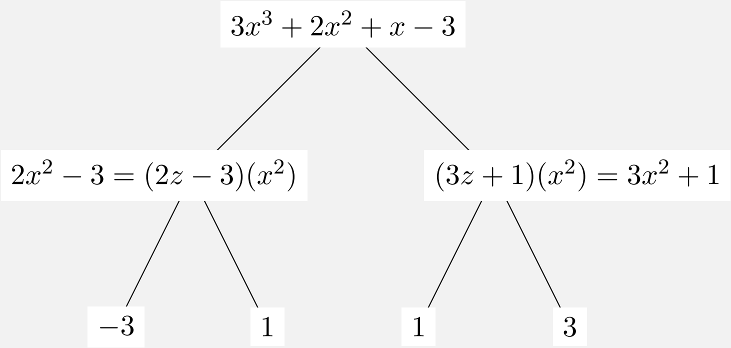
- **Hodnoty pro** $\pm x_i$

$$A(x_i) = A_s(x_i^2) + x_i \cdot A_l(x_i^2),$$
$$A(-x_i) = A_s(x_i^2) - x_i \cdot A_l(x_i^2).$$

FFT

- K výpočtu hodnot polynomu A stupně n v n bodech tedy potřebujeme vypočítat hodnoty polynomů A_S a A_L stupně $n/2$ v $n/2$ bodech.
- K výpočtu A_S a A_L použijeme stejný trik.

Příklad



FFT

Problém: Pro výpočet A_S a A_L potřebujeme opačné body, ale máme druhé mocniny.

Řešení: použijeme komplexní čísla

$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$

$$(-\tfrac{\sqrt{2}}{2} + \tfrac{\sqrt{2}}{2} \quad (\tfrac{\sqrt{2}}{2} - \tfrac{\sqrt{2}}{2} \quad (-\tfrac{\sqrt{2}}{2} - \tfrac{\sqrt{2}}{2}i) \quad (\tfrac{\sqrt{2}}{2} + \tfrac{\sqrt{2}}{2}i) \quad -i \quad i \quad -1 \quad 1$$

$$-i \qquad i \qquad -1 \qquad 1$$

$$-1 \qquad 1$$

$$1$$

FFT

FFT využívá následujících vlastností $\sqrt[n]{1}$. (pro $n = 2^k$)

- n-té odmocniny jedné jsou $\omega^0 = 1, \omega, \omega^2, \dots, \omega^{n-1}$, kde $\omega = e^{\frac{2\pi i}{n}}$.
- $\omega^{\frac{n}{2}+j} = -\omega^j$
- Množina druhých mocnin $\sqrt[n]{1}$ jsou právě $\{1, \omega^2, (\omega^2)^2, \dots, (\omega^2)^{n/2-1}\}$, tedy $n/2$-té odmocniny 1.

Poznámky

- **iterace**: spočítat ω a pak iterovat přes její mocniny.
- snadno můžeme najít opačné body
- druhé mocniny opět tvoří opačné body

procedure FFF($A[0,\ldots,n-1],\omega$) ▷ n je mocnina dvou

  if $\omega = 1$ **then** ▷ V tomto případě už $|A| = 1$

    return A

  end if

  for $i \leftarrow 0$ to $n/2 - 1$ **do**

    $A_S[i] \leftarrow A[i \cdot 2]$ ▷ Koeficienty pro sudé mocniny

    $A_L[i] \leftarrow A[i \cdot 2 + 1]$ ▷ Koeficienty pro liché mocniny

  end for

  $S \leftarrow$ FFT(A_S, ω^2)

  $L \leftarrow$ FFT(A_L, ω^2)

  $x \leftarrow 1$ ▷ Začínáme od ω^0

  for $j \leftarrow 0$ to $n/2 - 1$ **do**

    $R[j] \leftarrow S[j] + x \cdot L[j]$

    $R[j + n/2] \leftarrow S[j] - x \cdot L[j]$

    $x \leftarrow x \cdot \omega$ ▷ Další mocnina ω

  end for

  return R

end procedure

FFT

Složitost:

- velikost instance = počet koeficientů
- algoritmu jsou dvě rekurzivní volání, každému z nich předáváme instanci o velikosti $n/2$
- Zbývající část algoritmu (ř. 5 až 8 a 13 až 17) má složitost $O(n)$.
- rekurence: $T(n) = 2T(n/2) + O(n)$
- řešením je (Master theorem): $\Theta(n \log n)$

Interpolace

- lze spočítat pomocí FFT
- Pro hodnoty $AB(\omega^0), AB(\omega), AB(\omega^2), \ldots, AB(\omega^{n-1})$ dostaneme koeficienty $ab_0, ab_1, \ldots, ab_{n-1}$ pomocí

$$[ab_0, ab_1, \ldots, ab_{n-1}] = \frac{1}{n}\text{FFT}([AB(\omega^0), AB(\omega), AB(\omega^2), \ldots, AB(\omega^{n-1})], \omega^{-1}).$$

Rychlé násobení polynomů

procedure FASTPOLYMULTIPLY($A[0,\ldots,s], B[0,\ldots,t]$)
  $n \leftarrow 2^{\lceil \log_2(s+t+1) \rceil}$
  $\omega \leftarrow e^{\frac{2\pi i}{n}}$
  Doplň pomocí nul A i B na n prvků. ▷ Nuly přidávám na konec
  $V_A \leftarrow$FFT(A, ω) ▷ Hodnoty pro A
  $V_B \leftarrow$FFT(B, ω) ▷ Hodnoty pro B
  for $i \leftarrow 0$ to $n-1$ **do**
    $V_{AB}[i] \leftarrow V_A[i] \cdot V_B[i]$ ▷ Násobení polynomů
  end for
  return $\frac{1}{n}$FFT(V_{AB}, ω^{-1}) ▷ Interpolace pomocí FFT
end procedure

Složitost:

- Algoritmus třikrát volá FFT se složitostí $\Theta(n \log n)$, zbytek je v linearním čase.
- Složitost je tedy $\Theta(n \log n)$