دانشکده‌ی علوم ریاضی

آنالیزالگوریتم‌ها ۲۹ بهمن ۱۳۹۱

جلسه‌ی ۴: ضرب چندجمله‌ای‌ها

مدرّس: دکتر شهرام خزائی *نگارنده: ساینا خلیلی و علی کوچک زاده*

۱ مقدمه

ما در این جلسه به دنبال یافتن الگوریتم‌هایی برای ارزیابی یک چندجمله‌ای و محاسبه حاصل‌ضرب و حاصل‌جمع دو چندجمله‌ای هستیم. یک چندجمله‌ای را به سه طریق می‌توان نمایش داد.

۱. **نمایش با ضرایب**: ساده‌ترین راه برای نمایش چندجمله‌ای $A(x) = \sum_{j=0}^{n} a_j x^j$ استفاده از بردار ضرایب آن است. به عبارت دیگر، چندجمله‌ای $A(x)$ با آرایه $(a_0, a_1, ..., a_n)$ معرفی می‌شود. ارزیابی یک چندجمله‌ای با استفاده از ضرایب آن در $O(n)$ قابل انجام است. برای این منظور از رابطه زیر که به قانون هورنر[۱] معروف است

$$A(x) = a_0 + (a_1 + (a_2 + (\cdots + (a_{n-1} + a_n x)x) \cdots)x)x$$

برای طراحی الگوریتم زیر استفاده می‌کنیم:

```
function EVALUATE(A, x)
    [assumes A = (a_0, ..., a_n)]
    y ← 0
    for j = n  to 0 do
        y ← a_j + y · x
    return y
```

همچنین با استفاده از نمایش ضرایب دو چندجمله‌ای A و B از درجه n، حاصل‌جمع آنها را می‌توان در $O(n)$ محاسبه نمود زیرا

$$A(x) + B(x) = \sum_{j=0}^{n} a_j x^j + \sum_{j=0}^{n} b_j x^j = \sum_{j=0}^{n} (a_j + b_j) x^j$$

که حاصل باز هم یک آرایه n-تایی است که ضرایب چندجمله‌ای حاصل‌جمع را مشخص می‌کند.

محاسبه مستقیم چندجمله‌ای حاصل‌ضرب $A(x) \cdot B(x)$ با استفاده از رابطه‌ی کانولوشن[۲] در زمان $O(n^2)$ قابل انجام است:

[۱] Horner

[۲] convolution

$$A(x)B(x) = \sum_{i=0}^{n} a_i x^i \sum_{j=0}^{n} b_j x^j = \sum_{j=0}^{2n} (\sum_{i=0}^{j} a_i b_{j-i}) x^j$$

با استفاده از ایده کاروتسوبا است که برای محاسبه حاصل‌ضرب اعداد استفاده می‌شود، می‌توان چندجمله‌ای حاصل‌ضرب را در زمان سریع‌تر $O(n^{\log_2 3})$ محاسبه نمود. برای این کار با استفاده از رهیافت تقسیم و حل، چندجمله‌ای‌های A و B به صورت $A(x) = A_1(x) + x^{n/2} A_2(x)$ و $A(x) = B_1(x) + x^{n/2} B_2(x)$ در نظر گرفته می‌شوند و پس از محاسبه چندجمله‌ای‌های

$$\begin{aligned} P_1(x) &= A_1(x) \cdot B_1(x) \, , \\ P_2(x) &= A_2(x) \cdot B_2(x) \, , \\ P_3(x) &= (A_1(x) + A_2(x)) \cdot (B_1(x) + B_2(x)) \, , \end{aligned}$$

به صورت بازگشتی، چندجمله‌ای حاصل‌ضرب بدین صورت محاسبه می‌شود:

$$A(x) \cdot B(x) \quad = \quad P_1(x) + x^{n/2} \big(P_3(x) - P_1(x) - P_2(x) \big) x^{n/2} + x^n P_2(x)$$

هدف نهایی ما بسط الگوریتمی با پیچیدگی زمانی $O(n \log n)$ برای محاسبه چندجمله‌ای حاصل‌ضرب است.

۲. **نمایش نقطه‌ای**: روش دیگر نمایش یک چندجمله‌ای استفاده از تعدادی نقطه است که چندجمله‌ای را به‌طور یکتا مشخص کنند. می‌دانیم که یک چندجمله‌ای از درجه‌ی $n-1$ با n نقطه‌ی متمایز به طور یکتا مشخص می‌شود. با استفاده از روش درون‌یابی لاگرانژ، چندجمله‌ای یکتای از درجه حداکثر $n-1$ که از مجموعه نقاط $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ می‌گذرد (که x_i همگی متمایزند)، به صورت زیر می‌باشد.

$$A(x) = \sum_{k=1}^{n} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

با استفاده از نمایش نقطه‌ای یک چندجمله‌ای، اگر مقادیر

$$t_k = \prod_{j \neq k} \frac{1}{(x_k - x_j)}$$

از قبل محاسبه شده باشند، ارزیابی $A(x)$ در هر نقطه x (که $x \neq x_k$ برای $k = 1, \ldots, n$) در زمان $O(n)$ قابل انجام است، زیرا داریم:

$$A(x) = \prod_{j=1}^{n} (x - x_j) \sum_{k=1}^{n} \frac{y_k t_k}{x - x_k}$$

اگر $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ و $\{(x_1, y'_1), \ldots, (x_n, y'_n)\}$ نمایش‌های نقطه‌ای چندجمله‌ای‌های $A(x)$ و $A'(x)$ (با درجه‌های حداکثر $n-1$) باشند، به وضوح $\{(x_1, y_1 + y'_1), \ldots, (x_n, y_n + y'_n)\}$ یک

نمایش برای چندجمله‌ای مجموع، $A(x) + A'(x)$، است. بنابراین نمایش نقطه‌ای چندجمله‌ای حاصل‌جمع در زمان $O(n)$ قابل محاسبه است.

با این وجود $\{(x_۱, y_۱y'_۱), \ldots, (x_n, y_ny'_n)\}$ تنها در صورتی یک نمایش صحیح برای چندجمله‌ای حاصل‌ضرب، یعنی $A(x)A'(x)$، است که درجه چندجمله‌ای حاصل‌ضرب حداکثر $n - ۱$ باشد. اگر هر دو چندجمله‌ای از درجه دقیقاً $n - ۱$ باشند، در اینصورت $A(x)A'(x)$ با حداقل $۲n - ۱$ نقطه قابل نمایش است.

۳. **نمایش ریشه‌ای**: روش دیگر نمایش چندجمله‌ای‌ها با استفاده از ریشه‌های چندجمله‌ای به همراه ضریب ثابت بزرگترین درجه آن است. طبق قضیه اساسی جبر، هر چندجمله‌ای درجه n را می‌توان به‌صورت

$$a\prod_{i=۱}^{n}(x - r_i)$$

نمایش داد. بنابراین $(a, r_۱, \cdots, r_n)$ می‌تواند به عنوان نمایشی برای چندجمله‌ای بکار رود.

با استفاده از نمایش ریشه‌ای، حاصل‌ضرب و ارزیابی در زمان $O(n)$ انجام می‌شوند. ولی نمایش نقطه‌ای نمایش مناسبی برای محاسبه حاصل‌جمع نیست، زیرا برای محاسبه ریشه‌های چندجمله‌ای حاصل‌جمع مجبوریم به روش‌های عددی که غالباً پیچیدگی بالایی دارند متوسل شویم. ما بحث نمایش ریشه‌ای را بیشتر از این بررسی نخواهیم کرد.

۲ تبدیل فوریه

به طور خلاصه می‌توان جدول زیر را برای مرتبه‌زمانی روش‌هایی که تا کنون ذکر شده متصور شد. در این جدول ردیف اول، نمایش با ضرایب و ردیف دوم، نمایش با نقطه‌ها و ردیف آخر، نمایش با ریشه‌های چندجمله‌ای می‌باشد. پس جدول مرتبه‌های زمانی زیر را داریم:

جدول ۱: جدول مرتبه‌زمانی‌ها

نوع نمایش	مرتبه‌زمانی حاصل‌جمع	مرتبه‌زمانی ارزیابی	مرتبه‌زمانی حاصل‌ضرب
ضرایب	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^{۱/۵۹})$
نقطه‌ای	$\mathcal{O}(n)$	$\mathcal{O}(n^۲)$	$\mathcal{O}(n)$
ریشه‌ای	زیاد	$\mathcal{O}(n)$	$\mathcal{O}(n)$

بنابراین به معرفی روش‌های سریع‌تری برای تبدیلات از نقاط به چندجمله‌ای‌ها و از چندجمله‌ای‌ها به نقاط که یکی ازمهمترین آنها تبدیل فوریه[۳] است، می‌پردازیم.

[۳] Fourier Transform

۱.۲ تبدیل فوریه

در بخش‌های قبل با نحوه‌ی نمایش چندجمله‌ای‌ها آشنا شدیم. حال با بهترین روش موجود برای ضرب چندجمله‌ای‌ها که این کار را در $O(n \log n)$ انجام می‌دهد، آشنا می‌شویم.

معادله $X^n - ۱ = ۰$ را در نظر بگیرید. در مجموعه‌ی اعداد حقیقی این معادله حداکثر دو جواب ۱ و ۱− را دارد. ولی در مجموعه اعداد مختلط n ریشه دارد.

تعریف ۱ *(ریشه‌های مختلط واحد) ریشه‌های معادله $X^n - ۱ = ۰$ در مجموعه اعداد مختلط ریشه‌های n ام واحد نامیده می‌شوند که آن‌ها را با $w_۰, w_۱, \cdots, w_{n-۱}$ نمایش می‌دهیم.*

این n ریشه‌ی مختلط به طور کاملاً یکنواخت روی دایره‌ی واحد در فضای مختلط پخش شده‌اند. بنابراین هر کدام از ریشه‌های عدد واحد را به صورت توانی از ریشه‌ی پایه (که $i = \sqrt{-۱}$)

$$w = \exp(۲i\pi/n)$$

می‌توان نوشت یعنی $w_k = w^k$.
مسئله ارزیابی هم‌زمان چندجمله $\sum_{i=۰}^{n-۱} a_i x^i$ در نقاط $x_۰, \ldots, x_{n-۱}$ را در نظر بگیرید. می‌توان این مسئله را به صورت حاصل‌ضرب ماتریسی زیر نیز نمایش داد:

$$\begin{bmatrix} y_۰ \\ \vdots \\ y_{n-۱} \end{bmatrix} = \begin{bmatrix} ۱ & x_۰ & x_۰^۲ & \cdots & x_۰^{n-۱} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ ۱ & x_{n-۱} & x_{n-۱}^۲ & \cdots & x_{n-۱}^{n-۱} \end{bmatrix} \begin{bmatrix} a_۰ \\ \vdots \\ a_{n-۱} \end{bmatrix}$$

حالت خاصی از این مسئله وقتی است که $x_k = w^k$ باشد. لذا تبدیل فوریه را به صورت زیر تعریف می‌کنیم:

تعریف ۲ *تبدیل فوریه‌ی یک چندجمله‌ای با درجه‌ی $n - ۱$ عبارت است از ارزیابی چندجمله‌ای در نقاط w^k برای $k = ۰, ۱, \ldots, n - ۱$ که در آن: $w = e^{\frac{۲\pi i}{n}}$.*

معادلاً می‌توان گفت تبدیل فوریه بردار $(a_۰, \ldots, a_{n-۱})$ را که نمایش ضرایب چندجمله‌ای $A(x) = \sum_{i=۰}^{n-۱} a_i x^i$ است به بردار $(y_۰, \ldots, y_{n-۱})$ تبدیل می‌کند.

$$\begin{bmatrix} y_۰ \\ \vdots \\ y_j \\ \vdots \\ y_{n-۱} \end{bmatrix} = \begin{bmatrix} ۱ & ۱ & ۱ & \cdots & ۱ \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ ۱ & w^j & w^{۲j} & \cdots & w^{(n-۱)j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ ۱ & w^{n-۱} & w^{۲(n-۱)} & \cdots & w^{(n-۱)(n-۱)} \end{bmatrix} \begin{bmatrix} a_۰ \\ \vdots \\ a_j \\ \vdots \\ a_{n-۱} \end{bmatrix}$$

یک راه حل بدیهی این است که به ازای هر نقطه، یک بار حاصل چندجمله‌ای را محاسبه کنیم (ضرب ماتریسی فوق را مستقیم انجام دهیم.) که این عمل از $\mathcal{O}(n^۲)$ خواهد بود. اما به روش تقسیم و حل می‌توان این عملیات را ساده‌تر کرد. اگر جملات با توان فرد و زوج را جدا کرده و چندجمله‌ای $A(x)$ را به صورت مجموع $A(x) =$

بنویسیم که $A_e(x^۲) + xA_o(x^۲)$ و $A_e(x)$ هر کدام از درجه‌ی حداکثر $\frac{n}{۲}$ باشند (فرض می‌کنیم n زوج است)، می‌توان نوشت:

$$\begin{aligned} A(۱) &= A_e(۱) + A_o(۱) \\ A(-۱) &= A_e(۱) - A_o(۱) \\ A(i) &= A_e(-۱) + A_o(-۱)i \\ A(-i) &= A_e(-۱) - A_o(-۱)i \end{aligned}$$

و در حالت کلی اگر $v_۰, v_۱, \ldots, v_{\frac{n}{۲}-۱}$ ریشه‌های $\frac{n}{۲}$ ام واحد باشند، روابط بازگشتی را می‌توان نوشت:

$$\begin{aligned} A(w^k) &= A_e(v^k) + w^k A_o(v^k) \\ A(w^{k+\frac{n}{۲}}) &= A_e(v^k) - w^k A_o(v^k) \end{aligned}$$

به این نکته توجه می‌کنیم که $w^{k+\frac{n}{۲}} = -w^k$.
به کمک همین رابطه‌ی بازگشتی الگوریتم تبدیل فوریه سریع[۴] (FFT) را می‌نویسیم:

Algorithm 1 Algorithm: FFT

---

function $\textsc{FFT}(a_0, a_1 \ldots, a_{n-1})$
  [*assumes n is a power of two*]
  if $n = 1$ **then**
    return a_0
  $(e_0, e_1, ..., e_{n/2-1}) \leftarrow \textsc{FFT}((a_0, a_2, \ldots, a_{n-2})$
  $(d_0, d_1, ..., d_{n/2-1}) \leftarrow \textsc{FFT}(a_1, a_3, \ldots, a_{n-1})$
  for $k = 0$ **to** $\frac{n}{2} - 1$ **do**
    $w^k \leftarrow e^{\frac{2\pi ik}{n}}$
    $y_k \leftarrow e_k + w^k d^k$
    $y_{k+n/2} \leftarrow e_k - w^k d^k$
  return $y_0, y_1, ..., y_{n-1}$

در مورد پیچیدگی زمانی نیز با توجه به الگوریتم بازگشتی فوق واضح است که

$$T(n) = ۲T(\frac{n}{۲}) + \mathcal{O}(n) \Rightarrow T(n) = \mathcal{O}(n \log n)$$

حال نشان می‌دهیم که تبدیل معکوس فوریه را نیز به راحتی می‌توان محاسبه کرد. ابتدا ماتریس $M(w)$ را به صورت

[۴] Fast Fourier Transform

زیر تعریف می‌کنیم:

$$M(w) = \begin{bmatrix} ۱ & ۱ & ۱ & \cdots & ۱ \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ ۱ & w^j & w^{۲j} & \cdots & w^{(n-۱)j} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ ۱ & w^{n-۱} & w^{۲(n-۱)} & \cdots & w^{(n-۱)(n-۱)} \end{bmatrix}$$

این ماتریس یک ماتریس واندرموند است. برای محاسبه‌ی تبدیل معکوس کافی است تا ماتریس معکوس آن یعنی $M^{-۱}$ را محاسبه کنیم.

قضیه ۱ *معکوس ماتریس $M(w)$ برابر است با: $M(w)^{-۱} = \frac{۱}{n}M(w^{-۱})$*

برهان. *برای سادگی در نمایش $M(w)$ و $M(w^{-۱})$ را به ترتیب با M و M^* نمایش می‌دهیم. حال حاصل‌ضرب MM^* را محاسبه می‌کنیم. درایه‌ی (j,k) این ماتریس برابر است با:*

$$(MM^*)_{j,k} = ۱ + w^{j-k} + w^{۲(j-k)} + \cdots + w^{(n-۱)(j-k)} = \begin{cases} \frac{۱-w^{n(j-k)}}{۱-w^{j-k}} & j \neq k \\ n & j = k \end{cases}$$

که از رابطه‌ی مجموع سری هندسی با قدر نسبت $w^{(j-k)}$ استفاده کرده‌ایم. اگر j با k برابر نباشد، این مقدار صفر است (چون $w^n = ۱$) و اگر j با k برابر باشد، این مقدار برابر n است. لذا به طور خلاصه $MM^* = nI$ یعنی $M^{-۱} = \frac{۱}{n}M^*$. ■

لذا الگوریتم تبدیل فوریه معکوس را هم با تغییرات جزئی می‌توان نوشت:

Algorithm 2 Algorithm: FFT-INVERSE

```
function FFT-INVERSE(a_0, a_1, ..., a_{n-1})
    [assumes n is a power of two]
    if n = 1 then
        return a_0
    (e_0, e_1, ..., e_{n/2-1}) ← FFT-INVERSE(a_0, a_2, ..., a_{n-2})
    (d_0, d_1, ..., d_{n/2-1}) ← FFT-INVERSE(a_1, a_3, ..., a_{n-1})
    for k = 0 to n/2 - 1 do
        w^k ← e^{-2πik/n}
        y_k ← (1/n)(e_k + w^k d^k)
        y_{k+n/2} ← (1/n)(e_k - w^k d^k)
    return y_0, y_1, ..., y_{n-1}
```

۲.۲ محاسبه‌ی حاصل‌ضرب چندجمله‌ای‌ها به کمک تبدیل فوریه

برای محاسبه‌ی چندجمله‌ای حاصل‌ضرب $C(x) = A(x)B(x) = (\sum_{i=0}^{n-1} a_i x^i)(\sum_{i=0}^{n-1} b_i x^i)$ به این نحو عمل می‌کنیم.

- ابتدا ادامه‌ی ضرایب a_i و b_i را صفر قرار می‌دهیم تا چند جمله‌ای‌ها به صورت چندجمله‌ای‌هایی با درجه حداکثر $2n-1$ درآیند. یعنی: $a_j = b_j = 0$ برای $j = n, \ldots, 2n-1$.
- سپس به تعداد لازم ادامه‌ی ضرایب a_i و b_i را صفر قرار می‌دهیم تا تعداد ضرایب توانی از ۲ شود. یعنی اگر N کوچکترین توانی از ۲ باشد که بزرگتر از $2n-1$ است قرار می‌دهیم $a_j = b_j = 0$ برای $j = 2n, \ldots, N-1$.
- حال تبدیل فوریه سریع را برای هر دو دنباله به طول N به طور جداگانه محاسبه می‌کنیم تا $A_k = A(w^k)$ و $B_k = B(w^k)$ برای $k = 0, \ldots, N-1$ بدست آیند.
- سپس $C_k = C(w^k) = A_k B_k$ را برای $k = 0, \ldots, N-1$ محاسبه کرده و در نهایت از آن تبدیل فوریه‌ی معکوس می‌گیریم تا ضرایب $c_0, c_1, \ldots, c_{N-1}$ بدست آیند.

نهایتا $C(x) = \sum_{i=0}^{n-1} c_i x^i$ چندجمله‌ای حاصل‌ضرب است که البته می‌توان از جملات صفر انتهایی صرف‌نظر کرد.
بنابراین توانستیم با $\mathcal{O}(n \log n)$ عملیات، حاصل‌ضرب دو چندجمله‌ای را حساب کنیم.

۳ محاسبه‌ی حاصل‌ضرب اعداد صحیح به کمک تبدیل فوریه

شاید به نظر برسد که بتوان با همین ایده ضرب دو عدد n بیتی را هم ساده‌تر کرد. زیرا دو عدد n بیتی

$$a = \sum_{i=0}^{n-1} a_i 2^i, b = \sum_{i=0}^{n-1} b_i 2^i$$

که $a_i, b_i \in \{0, 1\}$ در واقع ارزیابی چندجمله‌ای های $A(x) = \sum_{i=0}^{n-1} a_i x^i$ و $B(x) = \sum_{i=0}^{n-1} b_i x^i$ در نقطه $x = 2$ هستند. یعنی، $a = A(2)$ و $b = B(2)$ و لذا

$$c = ab = A(2)B(2) = C(2)$$

که $C(x) = A(x)B(x)$ چندجمله‌ای حاصلضرب است.
اما نکته‌ای که وجود دارد این است که در الگوریتم تبدیل فوریه سریع از ضرب اعداد مختلط (که خود به ضرب اعداد حقیقی احتیاج دارد) استفاده شده است که به دلیل نیاز به استفاده از تقریب کافی برای محاسبه صحیح عملیات پیچیدگی را زیاد می‌کنند. یعنی با اینکه به $\mathcal{O}(n \log n)$ ضرب اعداد حقیقی نیاز داریم ممکن است عملیات بیتی مورد استفاده خیلی بیشتر باشد.
بهترین الگوریتمی که برای ضرب اعداد n بیتی داده شده است از مرتبه‌ی $\mathcal{O}(n \log n \log \log n)$ است که در سال ۱۹۷۱ توسط Schönhage و Strassen کشف شد.