۳. کافی است هر نقطه را تنها با ۷ نقطه‌ی بعدی این دنباله‌ی مرتب چک کنیم و با کمینه‌گیری نزدیک‌ترین زوج-نقطه‌ای را به‌دست آوریم که یک سرش در سمت چپ و سر دیگرش در سمت راست میانه باشد.

۱-۵-۳ اثبات درستی و تحلیل

شکل ۱-۱۸ بخشی از نوار Δ را نشان می‌دهد که به مربع‌های کوچکی به‌اندازه‌ی $d/۲ \times d/۲$ تقسیم شده است. روشن است که در هر مربع بیش از یک نقطه نمی‌تواند باشد (و گرنه فاصله‌ی این نقاط که در یک سمت میانه هستند از d کم‌تر می‌شد و این ممکن نیست). بنابراین در بررسی نقاط Δ به ترتیب نزولی، کافی است که هر نقطه را با ۷ نقطه‌ی بعدی‌اش مورد بررسی قرار دهیم تا (p_δ, q_δ) را به‌دست آوریم.

برای تحلیل، زمان $\mathcal{O}(n \lg n)$ برای مرتب‌سازی اولیه صرف می‌شود و داریم: $T(n) = ۲T(n/۲) + \mathcal{O}(n) = \mathcal{O}(n \lg n)$ و این بهینه است.

۱-۶ ضرب دو چندجمله‌ای

دو چندجمله‌ای $P(n)$ و $Q(m)$ برحسب x به‌ترتیب از درجه‌های n و m داده شده‌اند:

$$P_n(x) = a_n x^n + a_{n-۱} x^{n-۱} + \cdots + a_۰$$

$$Q_m(x) = b_m x^m + b_{m-۱} x^{m-۱} + \cdots + b_۰$$

می خواهیم با یک الگوریتم کارا حاصل‌ضرب $R(n+m) = P(n) \times Q(m) = \sum_{i=۰}^{n+m} c_i x^i$ را محاسبه کنیم. پس در این مسئله، ورودی و خروجی به‌صورت زیرند:

ورودی: $n, m, a_n, a_{n-۱}, \ldots, a_۰, b_m, b_{m-۱}, \ldots, b_۰$

خروجی: $c_{n+m}, c_{۲n-۱}, \ldots, c_۰$

برای پیاده‌سازی می‌توانیم ورودی را آرایه‌ی $n+۱$ عنصری p و آرایه‌ی $m+۱$ عنصری q را در نظر بگیریم و خروجی را در آرایه‌ی $n+m$ عضوی r بنویسیم (شکل ؟؟.)

راه‌حل کند

برای $0 \le k \le n+m$ داریم، $r_k = \sum_{i+j=k} a_i b_j$. به‌این ترتیب، هر r_i را می‌توان با $m+1$ عمل ضرب و همین تعداد عمل جمع به‌دست آورد. Multiply-Poly-1 این الگوریتم را نشان می‌دهد.

```
MULTIPLY-POLY-1 (A, B)
for i ← 0 to n + m
   do r[i] ← 0 for i ← 0 to n
   do for j ← 0 to m
          do r[i + j] ← r[i + j] + p[i] * q[j]
```

این عمل همان پیچش یا کانولوژن[۸] دو بردار p و q است.

راه‌حل بر اساس تقسیم‌وحل

با استفاده از روش تقسیم‌وحل، می‌توانیم الگوریتمی کاراتری برای حل این مسئله بیابیم.

برای این کار ابتدا $P(n)$ را به دو چندجمله‌ای $P_A(n_a)$ و $P_B(n_b)$ به‌ترتیب با درجه‌های نزدیک به مساوی n_a و n_b تقسیم می‌کنیم. به‌صورت دقیق‌تر، $P(n)$ را به‌صورت زیر می‌نویسیم ($n = n_a + n_b + 1$).

$$P(n) = P_A(n_a) + P_B(n_b)x^{n_a+1}$$

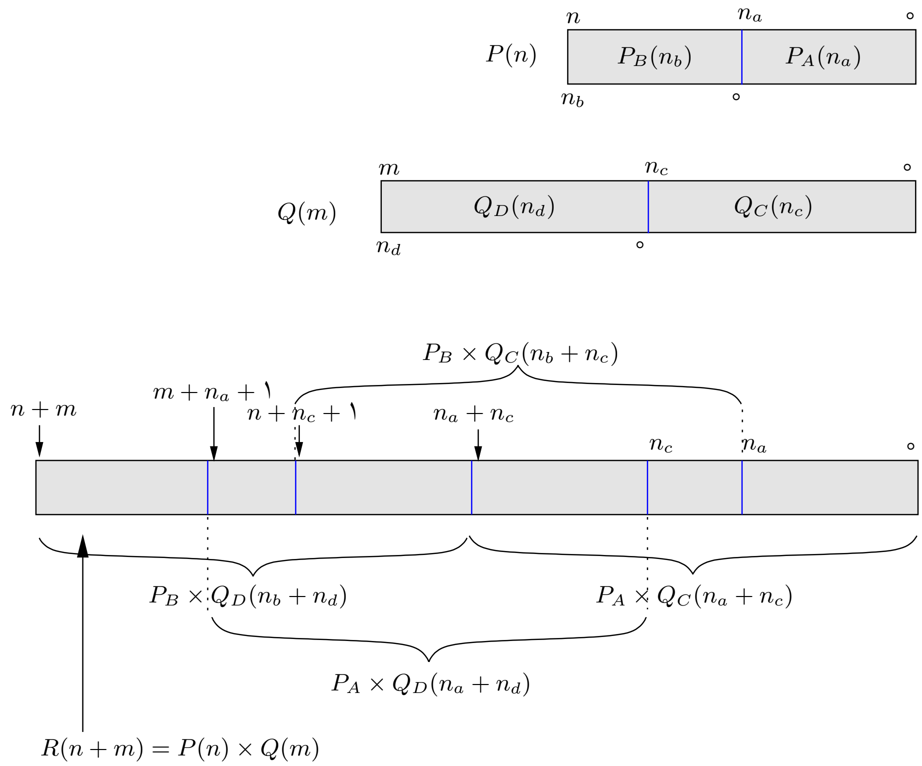
هم‌چنین $Q(m)$ را به دو چندجمله‌ای $Q_C(n_c)$ و $Q_D(n_d)$ با درجه‌های نشان‌داده‌شده تقسیم می‌کنیم ($m = n_d + n_c + 1$).

$$Q(m) = Q_D(n_d) + Q_C(n_c)x^{n_c+1}$$

[۸] convolution

حال، چنان‌که در شکل ؟؟ نشان داده شده است، حاصل‌ضرب $R(n+m) = P(n) \times Q(m)$ برابر است با،

$$\begin{aligned} R(n+m) &= P_A(n_a) \times Q_C(n_c) + P_A(n_a) \times Q_D(n_d) \times x^{n_c+1} + \\ & P_B(n_b) \times Q_C(n_c) \times x^{n_a+1} + P_B(n_b) \times Q_D(n_d) \times x^{n_a+n_c+2} \end{aligned}$$



شکل ۱-۱۹ نمایشی از مسئله‌ی ضرب دو چندجمله‌ای.

در این صورت حل مسئله‌ی اصلی به حل ۴ زیر مسئله‌ی کوچک‌تر تبدیل می‌شود. آرایه‌ی نهایی r را در ابتدا برابر صفر قرار می‌دهیم و جواب هر زیرمسئله را با انتقال (شیفت) نشان‌داده شده در آرایه r می‌نویسیم، یعنی مقدار جدید هر درایه را با مقدار قبلی‌اش جمع می‌کنیم. حال اگر $n = m$ و $n_a = n_c = \lfloor \frac{n}{2} \rfloor$ و در نتیجه $n_b = n_d = \lceil \frac{n}{2} \rceil + 1$ باشد، در آن‌صورت حل مسئله‌ی اصلی به‌اندازه‌ی n نیاز به حل ۴ زیر مسئله به اندازه‌ی تقریبی $\lceil \frac{n}{2} \rceil$ خواهد داشت. به‌طور دقیق، اندازه‌ی یک زیرمسئله‌ها $\lfloor \frac{n}{2} \rfloor$ $(P_A \times Q_C)$ و سه زیرمسئله به اندازه‌های $\lceil \frac{n}{2} \rceil + 1$ $(P_A \times Q_D$ ،$P_B \times Q_D$ و $P_B \times Q_D)$ خواهد بود. در این محاسبه اندازه‌ی زیرمسئله‌ی حاصل‌ضرب $P_B \times Q_C$ و $P_A \times Q_D$ برابر $\lceil \frac{n}{2} \rceil + 1$ در نظر گرفته شده است.

به‌این ترتیب اگر $T(n)$ هزینه‌ی حل مسئله باشد، داریم:

$$T(n) \leq T(\lfloor \frac{n}{2} \rfloor) + 3T(\lceil \frac{n}{2} \rceil + 1) \leq 4T(\lceil \frac{n}{2} \rceil + 1)$$

با استقرا می‌توان نشان داد که $T(n) = \Theta(n^2)$. بنابراین درجه‌ی پیچیدگی این الگوریتم، با وجود پیچیدگی پیاده‌سازی مانند الگوریتم ساده‌ی ارایه‌شده است.

راه‌حل کارا

ما می‌توانیم روش تقسیم‌وحل بالا را بهتر کنیم. با توجه به این‌که حاصل حل زیرمسئله‌های $P_B \times P_C$ و $P_A \times P_D$ در آرایه‌ی نهایی به‌دقت به یک مقدار شیفت داده می‌شوند (به‌اندازه‌ی $n_a + 1$) و درایه‌های متناظر آن‌ها با هم جمع زده می‌شوند، این دو زیرمسئله را می‌توان به‌صورت $P_A \times P_D + P_B \times P_C$ نوشت. ولی می‌دانیم که

$$ad + bc = (a - b)(d - c) + ac + bd$$

یعنی به‌جای محاسبه‌ی مستقیم $P_A \times P_D + P_B \times P_C$ می‌توان آرایه‌ی متناظر با P_B را از آرایه‌ی P_A کم کرد و چندجمله‌ای $P_{A-B}(\lceil \frac{n}{2} \rceil + 1)$ و مشابه‌ی آن چندجمله‌ای $P_{D-C}(\lceil \frac{n}{2} \rceil + 1)$ را به‌دست آورد و سپس با عملیات زیر حاصل را به‌دست آورد.

$$P_A \times P_D + P_B \times P_C = P_{A-B} \times P_{D-C} + P_A \times P_C + P_B \times P_D$$

با توجه با این‌که $P_A \times P_C$ و $P_B \times P_D$ از پیش محاسبه شده‌اند، در مجموع فقط به ۳ عدد ضرب دو تا چندجمله‌ای از درجه‌ی حداکثر $\lceil \frac{n}{2} \rceil + 1$ نیاز داریم. در این صورت خواهیم داشت:

$$T(n) \leq 3T(\lceil \frac{n}{2} \rceil + 1) + \Theta(n)$$

و با استفاده از قضیه‌ی اصلی و استقرا می‌توان ثابت کرد که $T(n) = \Theta(n^{\log_2 3})$.

کران پایین پیچیدگی راه‌حل این مسئله $\mathcal{O}(n \lg n)$ است و از روش تبدیل فوریه‌ی سریع[9] به‌دست می‌آید. (علاقه‌مندان به فصل ؟ از کتاب CLRS مراجعه نمایند.)

[9] Fast Fourier Transform (FFT)

۱-۷ الگوریتم استراسون برای ضرب ماتریس‌ها

آقای استراسون[۱۰] در سال ۱۹۶۹ روشی مشابه، اما پیچیده‌تر از تکنیک به‌کار رفته در مسئله‌ی پیشین را برای ضرب دو ماتریس به‌کار برد و موفق شد ضرب دو ماتریس $n \times n$ که در حالت معمولی به $\Theta(n^3)$ عملیات ضرب و جمع نیاز دارد را در $\Theta(n^{\lg 7}) = \mathcal{O}(n^{2.81})$ انجام دهد. این کار در آن‌زمان یک تحول مهم در محاسبات مهندسی محسوب شد که وقت زیادی صرف ضرب ماتریس‌های بزرگ می‌کردند.

اگر بخواهیم ضرب $C = A \times B$ را به‌دست آوریم، می‌توانیم هر کدام از ماتریس‌های A، B و C به اندازه‌ی $n \times n$ را به ۴ ماتریس با اندازه‌های $n/2 \times n/2$ تقسیم کنیم،

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

رابطه‌های زیر در مورد این ۱۲ ماتریس کوچک‌تر برقرارند:

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21}, \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22}, \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21}, \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned}$$

اگر با روش تقسیم‌وحل ۸ حاصل‌ضرب ماتریس‌های کوچک‌تر را به‌صورت بازگشتی حل کنیم، و نتیجه‌ها را درهم ترکیب نماییم، روشی داریم که هزینه‌ی آن از رابطه‌ی بازگشتی $T(n) = 8T(n/2) + \Theta(n^2)$ به‌دست می‌آید، که جواب آن همان $\Theta(n^3)$ است که در راه‌حل ساده‌ی معمولی داشتیم. کاری که آقای استرسون کرد این بود که تعداد ضرب ماتریس‌های کوچک را از ۸ به ۷ تقلیل داد و در آن‌صورت رابطه‌ی بازگشتی برای هزینه‌ی الگوریتم حاصل $T(n) = 7T(n/2) + \Theta(n^2)$ می‌شد که جواب آن $\Theta(n^{\lg 7}) = \mathcal{O}(n^{2.81})$ است.

[۱۰] Strassen

لم ۱–۶ ضرب دو ماتریس دل‌خواه ۲×۲ را می‌توان با ۷ عمل ضرب و ۱۸ عمل جمع و تفریق انجام داد.

اثبات: این دو ماتریس و ماتریس حاصل را به‌صورت زیر نمایش می‌دهیم:

$$\begin{bmatrix} c_{۱۱} & c_{۱۲} \\ c_{۲۱} & c_{۲۲} \end{bmatrix} = \begin{bmatrix} a_{۱۱} & a_{۱۲} \\ a_{۲۱} & a_{۲۲} \end{bmatrix} \times \begin{bmatrix} b_{۱۱} & b_{۱۲} \\ b_{۲۱} & b_{۲۲} \end{bmatrix}$$

و حاصل‌ضرب‌های زیر را به‌دست می‌آوریم،

$$\begin{aligned} m_{۱} &= (a_{۱۱} - a_{۲۲})(b_{۲۱} + b_{۲۲}), \\ m_{۲} &= (a_{۱۱} + a_{۲۲})(b_{۱۱} + b_{۲۲}), \\ m_{۳} &= (a_{۱۱} - a_{۲۱})(b_{۱۱} + b_{۱۲}), \\ m_{۴} &= (a_{۱۱} + a_{۱۲})b_{۲۲}, \\ m_{۵} &= a_{۱۱}(b_{۱۲} - b_{۲۲}), \\ m_{۶} &= a_{۲۲}(b_{۲۱} - b_{۱۱}), \\ m_{۷} &= (a_{۲۱} + a_{۲۲})(b_{۱۱}. \end{aligned}$$

حال، می‌توان دید که اعداد ماتریس حاصل به‌صورت زیر قابل محاسبه‌اند،

$$\begin{aligned} c_{۱۱} &= m_{۱} + m_{۲} + m_{۴} + m_{۶}, \\ c_{۱۲} &= m_{۴} + m_{۵}, \\ c_{۲۱} &= m_{۶} + m_{۷}, \\ c_{۲۲} &= m_{۲} - m_{۳} + m_{۵} - m_{۷}. \end{aligned}$$

که ۷ عمل ضرب و ۱۸ عمل جمع و تفریق دارد. □

قضیه‌ی ۱–۲ ضرب دو ماتریس $n \times n$ را می‌توان در $\Theta(n^{\lg ۷})$ انجام داد.

اثبات: معادلات لم ۱-۶ را نیز می‌توان برای ضرب ماتریس‌ها به‌کار برد. به‌وضوح اگر $n = 2^k$، این فرمول‌های برای ماتریس‌های به ابعاد $n/2 \times n/2$ هم برقرارند. بنابراین داریم،

$$T(n) = 7T(\frac{n}{2}) + 18(\frac{n}{2})^2$$

که جواب آن $T(n) = \Theta(n^{\lg 7})$ است. اگر n توانی از دو نباشد، می‌توان ابعاد ماتریس را تا توان دو بعدی افزایش داد و عناصر جدید را به‌عنوان مثال ۱ فرض کرد. از حاصل‌ضرب ماتریس‌های جدید می‌توان ماتریس حاصل از ضرب ماتریس اولیه را به‌سهولت به‌دست آورد. بنابراین هزینه‌ی حاصل‌ضرب همان $\Theta(n^{\lg 7})$ خواهد بود. □

جالب است بدانید که بهترین الگوریتم موجود برای ضرب دو ماتریس $n \times n$ از مرتبه‌ی $\mathcal{O}(n^{2.376})$ است که توسط کاپراسمیت و وینوگراد[11] در سال ۱۹۸۷ ارایه شد. البته این نتیجه تنها از بُعد نظری قابل توجه است و هنوز کاربرد عملی ندارد. هم‌چنین روشن نیست که آیا بهتر از این الگوریتم وجود دارد یا خیر. البته بدیهی است که کران پایین این الگوریتم $\Theta(n^2)$ است.

۱-۸ شبکه‌های مرتب‌ساز

می‌خواهیم با استفاده از مقایسه‌کننده‌ها به‌عنوان عنصر اصلی، مداری بسازیم تا با (تا حد امکان) کمینه‌تعداد مقایسه‌کننده‌ها (یا اندازه) و نیز کمینه‌ی زمان رسیدن به جواب (یا عمق)، n عدد چند بیتی را از ورودی دریافت و در خروجی آن‌ها را مرتب نماید. شکل ۱-۲۰ شمای کلی این مدار و نیز یک مدار مرتب‌ساز که ۴ عدد ورودی را مرتب می‌کند را نشان می‌دهد. اندازه‌ی این مدار ۵ و عمق آن ۳ می‌باشد. لازم به ذکر است که این مدار بهینه است و برای مرتب‌سازی ۴ عدد، مداری با اندازه یا عمق کم‌تر نمی‌توان ساخت.

شکل ۱-۲۱ همان مدار مرتب‌سازی ۴ تایی و نحوه‌ی کار آن برای یک ورودی خاص نشان داده شده است. در این شکل به‌جای مدار مقایسه‌کننده از $\downarrow$ استفاده شده است. شکل ۱-۲۲ هم دیاگرام حالت برای مقایسه‌کننده‌ی دودویی را نشان می‌دهد. این مداد دو عدد n بیتی و m بیتی را با هم مقایسه می‌کند و عدد کوچک‌تر را به خروجی بالا عدد دیگر را به خروجی پایین منتقل می‌کند. این مدار را به‌سادگی می‌توان ساخت.

اولین نکته‌ی قابل توجه این است که چه‌گونه می‌توان اثبات کرد که یک مدار داده‌شده برای همه‌ی مقادیر ورودی درست کار می‌کند. مدار شکل ۱-۲۱ را می‌توان با بررسی

[11] [illegible]