پارسا محمدپور

درس الگوریتم‌های پیشرفته – ارشد

الگوریتم Karatsuba:

**توضیح:**

یک الگوریتم برای ضرب دو عدد می‌باشد که برای وقتایی که تعداد ارقام ورودی بسیار زیاد باشد، خوب جواب می‌دهد و حتی از الگوریتم grade-school هم بهتر جواب می‌دهد. ایده این عملیات ضرب بر مبنای تقسیم و حل (Divide and Conquer) می‌باشد. به این صورت که ورودی را به دو قسمت مساوی تقسیم می‌کنیم، به طوریکه تعداد رقم‌های هر قسمت برابر باشد یا درمواقعی که تعداد رقم‌های عدد فرد است، اختلاف این دو قسمت یکی باشد. یعنی:

$$X = X_1\, X_2 \ldots X_n = X_1\, X_2 \ldots X_{\left\lfloor \frac{n}{2} \right\rfloor}\ \ X_{\left\lfloor \frac{n}{2} \right\rfloor + 1} \ldots X_n \qquad m_1 = \left\lfloor \frac{n}{2} \right\rfloor, m_2 = n - m_1$$

$$Y = Y_1\, Y_2 \ldots Y_n = Y_1\, Y_2 \ldots Y_{\left\lfloor \frac{n}{2} \right\rfloor}\ \ Y_{\left\lfloor \frac{n}{2} \right\rfloor + 1} \ldots Y_n \qquad m_1 = \left\lfloor \frac{n}{2} \right\rfloor, m_2 = n - m_1$$

پس با این حساب می‌توان گفت که:

$$X = \overline{X_{High} \quad X_{Low}} = X_{High} * 10^{m_2} + X_{Low}$$

$$Y = \overline{Y_{High} \quad Y_{Low}} = Y_{High} * 10^{m_2} + Y_{Low}$$

سپس اگر حاصلضرب x و y را حساب کنیم، می‌شود:

$$X.Y = \overline{X_{High}\ X_{Low}} * \overline{Y_{High}\ Y_{Low}}$$

$$= \left(X_{High} * 10^{m_2} + X_{Low}\right) * \left(Y_{High} * 10^{m_2} + Y_{Low}\right)$$

$$= X_{High} * Y_{High} * 10^{2*m_2} + 10^{m_2} * \left(X_{High} * Y_{Low} + X_{Low} * Y_{High}\right) + X_{Low} * Y_{Low}$$

$$= X_{High} * Y_{High} * 10^{2*m_2} + 10^{m_2} * \left(X_{High} * Y_{Low} + X_{Low} * Y_{High}\right) + X_{Low} * Y_{Low} + 10^{m_2} * X_{High} * Y_{High} - 10^{m_2} * X_{High} * Y_{High} + 10^{m_2} * X_{Low} * Y_{Low} - 10^{m_2} * X_{Low} * Y_{Low}$$

$$= X_{High} * Y_{High} * 10^{2*m_2} + 10^{m_2} * \left(X_{High} * Y_{Low} + X_{Low} * Y_{High} + X_{High} * Y_{High} - X_{High} * Y_{High} + X_{Low} * Y_{Low} - X_{Low} * Y_{Low}\right) + X_{Low} * Y_{Low}$$

$$= X_{High} * Y_{High} * 10^{2*m_2} + 10^{m_2} * \left(X_{High} * \left(Y_{Low} + Y_{High}\right) + X_{Low} * (Y_{High} + Y_{Low}) - X_{High} * Y_{High} - X_{Low} * Y_{Low}\right) + X_{Low} * Y_{Low}$$

$$= X_{High} * Y_{High} * 10^{2*m_2} + 10^{m_2} * \left((X_{High} + X_{Low}) * (Y_{High} + Y_{Low}) - X_{High} * Y_{High} - X_{Low} * Y_{Low}\right) + X_{Low} * Y_{Low}$$

$$= X_{High} * Y_{High} * 10^{2*m_2} + 10^{m_2} * \left(X_{High} + X_{Low}\right) * \left(Y_{High} + Y_{Low}\right) - 10^{m_2} * X_{High} * Y_{High} - 10^{m_2} * X_{Low} * Y_{Low} + X_{Low} * Y_{Low}$$

$$= X_{High} * Y_{High} * (10^{2*m_2} - 10^{m_2}) + 10^{m_2} * \left(X_{High} + X_{Low}\right) * \left(Y_{High} + Y_{Low}\right) + (1 - 10^{m_2}) * X_{Low} * Y_{Low}$$

پس با توجه به محاسبات فوق، فرمول ضرب الگوریتم Karatsuba اثبات شد.

**محاسبه order زمانی:**

حال با توجه به رابطه بدست آمده برای این الگوریتم، یک بار رابطه بازگشتی را با در نظر گرفتن اینکه تعداد ارقام ورودی n تاست (در اصل برابر با log x می‌باشد که x همان عدد ورودی است) می‌نویسیم:

$$T(n) = 3 * T\left(\frac{n}{2}\right) + n$$

حال با توجه به فرم این رابطه بازگشتی و اینکه می‌دانیم $2^1 < 3$، پس طبق قضیه اصلی، پیچیدگی زمانی این الگوریتم برابر است با:

$$T(n) = \theta(n^{log_3^2})$$

**فرضیات:**

در این روش فرض می‌کنیم که ما ضرب اعداد چهار رقمی،جمع اعداد و همچنین محاسبه توان ده (که در مبنای ده در اصل همان شیفت است) را می‌توانیم انجام دهیم.

## الگوریتم استاندارد یا grade-school:

این الگوریتم در اصل همان الگوریتم ساده و آشنایی است که برای ضرب کردن استفاده می‌کنیم. در اصل بر روی ارقام هر دو ورودی ورودی iterate می‌کنیم و هر رقم را با هر رقم دیگر ضرب می‌کنیم و در توانی از ده هم ضرب می‌کنیم و سپس همه این مقادیر را با یکدیگر جمع می‌کنیم.

**پیچیدگی زمانی:**

پیچیدگی زمانی این الگوریتم با در نظر گرفتن اینکه یک ورودی n رقم و دیگری هم m رقم دارد، می‌شود O(n*m) که اگر عد بزرگتر را n درنظر بگیریم، به عبارتی $O(n^2)$ می‌شود.

**فرضیات:**

در اینجا فرض می‌کنیم ضرب اعداد یک رقمی و همچنین توان ده (که در مبنای ده در اصل همان شیفت است) را می‌توانیم انجام دهیم.

## الگوریتم repeated addition:

این الگوریتم در اصل همان مفهوم اولیه ضرب می‌باشد. به این صورت که عدد اول را به اندازه عدد دوم بار با خودش جمع می‌کنیم.

**پیچیدگی زمانی:**

پیچیدگی زمانی این الگوریتم، در اصل می‌شود ئ که همان ورودی دوم است. اما از جایی که در بقیه موارد پیچیدگی زمانی را بر حسب تعداد ارقام ورودی گفتیم و با توجه به اینکه تعداد ارقام ورودی عدد برابر است با log m، (n = log m)پس پیچیدگی زمانی این الگوریتم بر حسب تعداد ارقام ورودی، برابر $10^n$ می‌باشد.

## الگوریتم Toom-Cook:

این الگوریتم هم همانند الگوریتم Karatsuba یک الگوریتم divide and conquer می‌باشد. در این روش ورودی به قسمت‌های کوچک تقسیم می‌شود و در هم ضرب می‌شوند. اگر ورودی از یک حدی کوچک‌تر باشد، ضرب می‌شوند و در غیر این صورت خود این تابع به صورت بازگشتی با قسمت‌های کوچک‌تر عدد صدا زده می‌شود. Toom-Cook-3 یکی از این حالت‌هاست که در آن ورودی به سه قسمت تقسیم می‌شود. این الگوریتم برای مقدارهای خیلی بزرگ خوب عمل می‌کند همچنین در عددهای بسیار بالا، از یه حدی به بعد، بهتر از الگوریتم Karatsuba هم عمل می‌کند.

**پیچیدگی زمانی:**

پیچیدگی زمانی این الگوریتم برای حالت k = 3 (که در آن ورودی هر بار به سه قسمت تقسیم می‌شود یا همان Toom-Cook-3) برابر با $O(n^{1.46}) = O\left(n^{log_3^5}\right)$ می‌باشد.

## الگوریتم Schönhage–Strassen:

یک الگوریتم بسیار سریع برای ضرا اعداد خیلی بزرگ می‌باشد. در این الگوریتم از recursive Fast Fourier Transforms ها استفاده می‌شود. در عمل این الگوریتم برای اعداد بزرگتر از $2^{15}$ و $2^{17}$ الگوریتم‌های قبلی را خیلی خیلی بهبود می‌بخشد یا به اصطلاح، outperform می‌کند.

**پیچیدگی زمانی:**

پیچیدگی زمانی این الگوریتم O(n * log n * log log n) می‌باشد.

## الگوریتم‌های ضرب ماتریس‌ها:

- الگوریتم چرخشی (iterative)
- الگوریتم divide and conquer
- الگوریتم sub-cubic
- الگوریتم موازی و توزیع شده (parallel and distributed)

### الگوریتم iterative:

این در اصل همان الگوریتم عادی ایست که برای ضرب دو تا ماتریس استفاده می‌شود. در اصل میایم و سطرهای ماتریس اول و ستون‌های ماتریس دوم را پیمایش می‌کنیم و درایه‌های را با هم ضرب می‌کنیم.(سه تا حلقه تودرتو)

```
input A and B, both n by n matrices
initialize C to be an n by n matrix of all zeros
for i from 1 to n:
    for j from 1 to n:
        for k from 1 to n:
            C[i][j] = C[i][j] + A[i][k]*B[k][j]
output C (as A*B)
```

**پیچیدگی زمانی:**

این الگوریتم دارای پیچیدگی زمانی $O(n^3)$ می‌باشد. (فرض می‌کنیم که سطرها و ستون‌های ماتریس‌ها از اردر n باشند یا n بزرگترینشان باشد)

## الگوریتم *Strassen’s*:

این الگوریتم از نوع الگوریتم‌های *divide and conquer* است. نکته کلیدی در این الگوریتم این است که در ضرب دو ماتریس دو در دو، به جای هشت تا عمل ضرب که در حالت عادی (حالت قبل) داشتیم، می‌توانیم تنها با هفت عمل ضرب هم این کار را انجام دهیم. البته در این روش باید یازده تا عمل جمع و یک عمل تفریق هم انجام دهیم. برای این کار باید ورودی *n*n* را به عنوان بلاک‌های ماتریس دو در دو، در نظر بگیریم. پس بنابراین عمل ضرب ماتریس *n*n* در اصل به هفت تا زیر مسئله ضرب ماتریس‌های $\frac{n}{2} * \frac{n}{2}$ تبدیل می‌شود. این الگوریتم همچنان با استفاده از موازی‌سازی عملکردش بهبود می‌یابد.

## پیچیدگی زمانی:

پیاده‌سازی عمل گفته شده به صورت بازگشتی، باعث می‌شود تا پیچیدگی زمانی ما به $O\left(n^{log_2^7}\right) = O(n^{2.807})$ می‌باشد.

## بهترین الگوریتم برای ضرب ماتریس‌ها:

در ژانوریه سال *2024* هم تعدادی از دانشمندان به نام‌های *Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou* یک الگوریتم معرفی کده‌اند که پیچیدگی زمانی آن در $O\left(n^{log_2^7}\right) = O(n^{2.371552})$ می‌باشد. اینکه آیا ضرب ماتریس‌ها در اردر $O(n^2)$ قابل انجام است یا نه را نمیدانیم. ولی اگر این مقدار بدست بیاید، قطعا بهترین مقدار است، چون به هر حال باید برای ضرب دو ماتریس تمام درایه‌های هر دو ماتریس را پیمایش کرد.

فایل نوت‌بوک در کنار این فایل و همچنین در *google colab* در آدرس زیر موجود می‌باشد:

*https://colab.research.google.com/drive/1w6GQATNWlTWWuLvFV1j9QwwYTMKhYrAq?usp=sharing*