0.1 Matrix multiplication

- *Algorithm:* Naive (algo. 1 & alg:2), Strassen (algo. algo. 3)
- *Input:* Matrices
- *Complexity:* $\mathcal{O}(n^\omega)$, with $2 < \omega$
- *Data structure compatibility:* Array (2D matrix)
- *Common applications:* All kinds of calculation missions in Mathematics, Physics, Engineering, etc.

Problem. Matrix multiplication

The multiplication between matrices is a common calculation mission. A naive way to calculate it according to the definition of matrix multiplication is of high complexity. Therefore several algorithms of better complexity performances are proposed, which gradually cut the time complexity to a polynomial of lower power. The basic ideas of those algorithms will be discussed here.

Description

Firstly recall the definition of matrix multiplication. Given 2 matrices A, B of size $m \times p$, $p \times n$, their product M is a $m \times n$ matrix with elements

$$M_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}$$

for any $1 \leq i \leq m$ and $1 \leq j \leq n$. Therefore the most instinctive method is calculate the elements in M one by one.

Algorithm 1: Naive (Iteration)

```
Input  : matrices $A$ of $m \times p$ and $B$ of size $p \times n$
Output: matrix $M$ the product of $A$ and $B$
1 for i for 1 to m do
2     for j for 1 to n do
3         $M_{ij} \leftarrow 0$
4         for k for 1 to p do
5             $M_{ij} \leftarrow M_{ij} + A_{ik} B_{kj}$
6         end for
7     end for
8 end for
9 return $M$
```

It is easy to see that the Algo. 1 cost $\Theta(mpn)$. For purpose of generality, we may assume the both matrices are square matrices of size $n \times n$, and the time complexity is then $\mathcal{O}(n^3)$.

There is an *divide-and-conquer alternative* for it, based on which further improvements are possible. The basic idea here is *block partition*, to keep on splitting the original large matrices into halves so as to consecutively divide the problem into multiplications of 2×2 square matrices. Suppose $C = AB$, where

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}, A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

and we may calculate C in the way of

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

It means that we only need to calculate 4 smaller-size multiplications and n^2 additions. The complexity is then given by recurrence

$$\begin{cases} T(1) = \Theta(1) \\ T(n) = 8T(n/2) + \Theta(n^2) \end{cases}$$

We can see that the time complexity is $\mathcal{O}(n^3)$ according to Master Theorem, which the same as the iteration method. The pseudocode is given in Algo. 2

Algorithm 2: Name

```
Input  : $n \times n$ square matrices $A$, $B$
Output: $M$ the product of $A, B$
1  Function Multi($A, B$):
2      if $A, B$ is $2 \times 2$ then  return $AB$          /* 8 scalar multiplications */ ;
3      divide $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$
4      $M_1 \leftarrow Multi(A_{11}, B_{11})$
5      $M_2 \leftarrow Multi(A_{12}, B_{21})$
6      $M_3 \leftarrow Multi(A_{11}, B_{12})$
7      $M_4 \leftarrow Multi(A_{12}, B_{22})$
8      $M_5 \leftarrow Multi(A_{21}, B_{11})$
9      $M_6 \leftarrow Multi(A_{22}, B_{21})$
10     $M_7 \leftarrow Multi(A_{21}, B_{12})$
11     $M_8 \leftarrow Multi(A_{22}, B_{22})$
12     calculate $C = \begin{bmatrix} M_1 + M_2 & M_3 + M_4 \\ M_5 + M_6 & M_7 + M_8 \end{bmatrix}$
13 end
14 return $C$
```

Strassen Algorithm is named after Volker Strassen, proposed in 1969, it is one of the important tries to decrease the matrix multiplication complexity [3]. Although the improvement of Strassen Algorithm is limited, it inspires later inventions of better algorithms. The basic idea of Strassen Algorithm is similar to that of divide-and-conquer scheme, but cut the number of sub-problems in each layer of recursion. With the same background settings as in Algo. 2, the calculation is altered as in Algo. 3.

Now the recurrence is

$$\begin{cases} T(1) = \Theta(1) \\ T(n) = 7T(n/2) + \Theta(n^2) \end{cases}$$

and hence the complexity is $\mathcal{O}(n^{\log_2 7}) = \mathcal{O}(n^{2.807})$. Although the calculation process is more complicated compared to naive algorithms, Strassen Algorithm has better performances in cases where $n > 100$ [1].

Since any algorithm managing to deal with matrix multiplication has to traversal all the n^2 entries in two matrices, the time complexity of matrix multiplication algorithm has a lower bound $\mathcal{O}(n^2)$. Setting the $\mathcal{O}(n^3)$ naive algorithm has baseline, we can express the time complexity of all matrix multiplication algorithms as $\mathcal{O}^\omega$, where ω is called the exponent of matrix multiplication, which is a number between 2 and 3. The lowest known k is achieved by François Le Gall with Coppersmith–Winograd algorithm, which is 2.3728639 [2].

Algorithm 3: Strassen

```
Input  : $n \times n$ square matrices $A$, $B$
Output: $M$ the product of $A$, $B$
Function Multi($A, B$):
    if $A, B$ is $2 \times 2$ then return $AB$                /* 8 scalar multiplications */ ;
    divide $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$
    $M_1 \leftarrow Multi(A_{11} + A_{22}, B_{11} + B_{22})$
    $M_2 \leftarrow Multi(A_{21} + A_{22}, B_{11})$
    $M_3 \leftarrow Multi(A_{11}, B_{12} - B_{22})$
    $M_4 \leftarrow Multi(A_{22}, B_{21} + B_{11})$
    $M_5 \leftarrow Multi(A_{11} + A_{12}, B_{22})$
    $M_6 \leftarrow Multi(A_{21} - A_{11}, B_{11} + B_{12})$
    $M_7 \leftarrow Multi(A_{12} - A_{22}, B_{21} + B_{22})$
    calculate $C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}$
end
return $C$
```

References.

[1] Michael A Bender, Dongdong Ge, Simai He, Haodong Hu, Ron Y Pinter, Steven Skiena, and Firas Swidan. "Improved bounds on sorting by length-weighted reversals". In: *Journal of Computer and System Sciences* 74.5 (2008), pp. 744–774 (cit. on p. 2).

[2] François Le Gall. "Powers of tensors and fast matrix multiplication". In: *Proceedings of the 39th international symposium on symbolic and algebraic computation*. 2014, pp. 296–303 (cit. on p. 2).

[3] Volker Strassen. "Gaussian elimination is not optimal". In: *Numerische Mathematik* 13 (Aug. 1969), pp. 354–356 (cit. on p. 2).