***Strassen Algorithm For Matrix Multiplication***

Strassen Algorithm is one of the usage of Divide and Conquer.

***Definition:***

If you have kept contact with Strassen Algorithm before, you may know how to run Matrix Multiplication.

For A = (aij) and B = (bij) are all n \* n Matrix, then for i, j = 1, 2, ..., n, here define cij as the element of Multiplication C = A \* B:

Cij = Sum ( aik \* bkj ) ( k belongs from 1 to n. )

Therefore we need to calculate n^2 number of matrix elements, and each of which is Sum of n values.

***Pseudo Code - Simpler & Slower Version ( No Divide and Conquer Thinking )***

***Pre - Condition:***

* Multiplication of Matrix A and B:
* A, B are all N \* N Matrix.
* Assign C with N \* N Matrix.

**Square - Matrix - Multiplication (A, B):**

*int n = A.rows;*

*For ( int i = 0; i <= n; i ++ )*

*{*

*For ( int j = 0; j <= n; j ++ )*

*{*

*C[ i ] [ j ] = 0;*

*For ( k = 0; k <= n; k ++ )*

*{*

*C [ i ] [ j ] += A [ i ] [ k ] \* B [ k ] [ j ];*

*}*

*}*

*}*

*RETURN C;*

***Cost:***

O( N ^ 3 )

***Supplement:***

You may probably think that any Matrix Multiplication need to spend O ( N ^ 3 ) cost, but actually, we have the Divide and Conquer Algorithm which can help us speed up. It only cost O ( n ^ lg7 ).

***The Simpler Divide and Conquer Version:***

When we try to calculate Matrix Multiplication C = A \* B by using Divide and Conquer Algorithm, then we need to assume that three Matrix are all n \* n, also n is Power of 2. Here we assume that in each division steps, then n \* n matrix are divided into 4 Sub - Matrix with size of n/2 \* n/2. Also, the size of n needs to be bigger than or equal to 2.

***Basics:***

When n equals to 2, it means that the size of Matrix A and B are the size of 2,

|  |  |
| --- | --- |
| a11 | a12 |
| a21 | a22 |

|  |  |
| --- | --- |
| b11 | b12 |
| b21 | b22 |

A = B =

|  |  |
| --- | --- |
| c11 | c12 |
| c21 | c22 |

C =

Matrix A can be divided into four sub - matrix, and each matrix only contains one element:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| c11 | c12 | = | a11 | a12 | \* | b11 | b12 |
| c21 | c22 | a21 | a22 | b21 | b22 |

Convert the Equation into four Sub - Equations, which is just as below:

c11 = a11 \* b11 + a12 \* b21;

c12 = a11 \* b12 + a12 \* b22;

c21 = a21 \* b11 + a22 \* b21;

c22 = a21 \* b12 + a22 \* b22;

***Generalize:***

|  |  |
| --- | --- |
| A11 | A12 |
| A21 | A22 |

|  |  |
| --- | --- |
| B11 | B12 |
| B21 | B22 |

A = B =

|  |  |
| --- | --- |
| C11 | C12 |
| C21 | C22 |

C =

Therefore, we can modify the final Expression, which is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C11 | C12 | = | A11 | A12 | \* | B11 | B12 |
| C21 | C22 | A21 | A22 | B21 | B22 |

Converted the Equation into four Sub - Equations, just as below:

C11 = A11 \* B11 + A12 \* B21

C12 = A11 \* B12 + A12 \* B22

C21 = A21 \* B11 + A21 \* B21

C22 = A21 \* B12 + A21 \* B22

***Pseudo Code - With Divide and Conquer Thinking***

***Square - Matrix - Multiplication - Recursive (A, B):***

*N = A.rows*

*Let C to be a new n \* n matrix.*

*IF N == 1*

*RETURN c11 = a11 \* b11;*

*ELSE Partition A, B and C as in Generalize*

*C11 = Square - Matrix - Multiplication - Recursive (A11, B11)*

*+ Square - Matrix - Multiplication - Recursive (A12, B21);*

*C12 = Square - Matrix - Multiplication - Recursive (A11, B12)*

*+ Square - Matrix - Multiplication - Recursive (A12, B22);*

*C21 = Square - Matrix - Multiplication - Recursive (A21, B11)*

*+ Square - Matrix - Multiplication - Recursive (A21, B21);*

*C22 = Square - Matrix - Multiplication - Recursive (A21, B12)*

*+ Square - Matrix - Multiplication - Recursive (A21, B22);*

*RETURN C;*

***Generalize Recursion Cost for Equation Above:***

*T ( 1 ) = Theta( 1 ) ( n = 1 )*

*T ( n ) = 8 \* T( n / 2 ) + Theta( n ^ 2 ) ( n > 1 )*

***Explanation:***

*Here, 8 can not be eliminated, since 8 decide how many sub - child in the Main Tree of every node, then it decide how many contribution it makes for each Level of the Calling Tree. If we neglect the coefficient, then this Equation turns to Linear Structure, but not the ‘Bushy’ Tree, under this circumstance, each level will only contribute one item for each level.*

***Strassen Method:***