***Matrix Chain Multiplication***

This Chapter is to solve Matrix Chain Multiplication issue by using Dynamic Programming Algorithm.

***Description:***

Given one Sequence of n Matrix <A1, A2, ... , An>, here we want to calculate their Multiplication:

A1 \* A2 \* A3 ... An

In order to calculate the final Expression, we need to first use the parenthesis to clarify Calculation Sequence, then we can use the standard Multiplication to calculate. The Matrix Multiplication satisfies the Associative Property, so each method that add Parenthesis would get the same calculation result.

***Definition:***

Fully Parenthesized Matrix Multiplication with Property that it is the single Matrix or two fully parenthesized Matrix Chain Multiplication with Parenthesis.

***Example:***

Matrix Chain <A1, A2, A3, A4>, and there are five Fully Parenthesized Matrix Multiplication Chain:

* ( A1 \* ( A2 \* ( A3 \* A4 ) ) )
* ( A1 \* ( ( A2 \* A3 ) \* A4 ) )
* ( ( A1 \* A2 ) \* ( A3 \* A4 ) )
* ( ( A1 \* ( A2 \* A3 ) ) \* A4 )
* ( ( ( A1 \* A2 ) \* A3 ) \* A4)

It would generate big influence for the Cost of Matrix Multiplication by adding Parenthesis for Matrix. Let’s consider the cost for Matrix Multiplication. Below gave the cost of two Matrix Multiplication.

*( The attributes rows and columns are the row and columns of Matrix. )*

***Matrix\_Multiply(A, B):***

*IF ( A.columns() != B.rows() )*

*{*

*Error ‘Incompatible Dimensions.’*

*}*

*FOR ( int i = 0; i < A.rows(); i ++ )*

*{*

*FOR ( int j = 0; j < B.columns(); j ++ )*

*{*

*Cij = 0;*

*FOR ( int k = 0; k < A.columns(); k ++ )*

*{*

*Cij += Aik \* Bkj;*

*}*

*}*

*}*

Since two Matrix need to be compatible, which means the column of Matrix A equals to the rows of Matrix B, then they could do Multiplication. If Matrix A is p \* q, and Matrix B is q \* r, then the Multiplication Matrix C would be p \* r.

***For Example:***

Take <A1, A2, A3> Matrix Chain Multiplication as example, explain that adding different Parenthesis would cause totally different calculation cost.

1. *Single Size of Relation:*

* A1 Size: 10 \* 100
* A2 Size: 100 \* 5
* A3 Size: 5 \* 50

1. *Multiplication Relation Size of Double Matrix:*

* A1 \* A2 Size: 10 \* 5
* A2 \* A3 Size: 100 \* 50

1. *Multiplication Size of Three Matrix:*

* ( ( A1, A2 ), A3 ) Size: 10 \* 50
* ( A1, ( A2, A3 ) ) Size: 10 \* 5

*According to the Code Piece described above, it tells Cost:*

( ( A1, A2 ), A3 ) = 10 \* 100 \* 5 + 10 \* 5 \* 50 = 7, 500

( A1, ( A2, A3 ) ) = 100 \* 5 \* 50 + 10 \* 100 \* 50 = 25, 000 + 50, 000 = 75, 000

Calculation Speed of the first one is 10 times faster than the second one.

Using m[ i, j ] to represent the least multiplication times for Matrix A i, j.

***Question Description:***

***Matrix - Chain - Multiplication - Problem:***

Given n matrix chain <A1, A2, A3, ..., An>, the size of Matrix Ai equals to pi-1 \* pi ( 1 <= i <= n ), we need to get the Final Fully Parenthesis schema, make the final multiplication times of A1 \* A2 \* A3...An the least.

***Attention:***

Here we do not need to multiple Matrix for real, we only need to make sure the least amount of Multiplication Matrix. Make sure that the favored calculation sequence would save much more times than other Matrix Multiplication.

***Calculation:***

*The Schema Number of Adding Different Parenthesis:*

Before we solve Matrix Chain Multiplication by using Dynamic Programming, we need to configure out that exhausting all methods can not help generate an effective algorithm. For the Matrix Chain with n matrix, assume that P(n) represents the Parenthesis Schema number.

*Generalization:*

When n = 1, there has only one matrix and only one Fully Parenthesis Schema.

When n >= 2, then the Fully Parenthesis Schema of Matrix Multiplication can be described as two Fully Parenthesis Schema of Partial Matrix Multiplication and the division is between the kth Matrix and the kth + 1 Matrix, k is the random value of 1, 2, 3, ... n - 1. Therefore we can get the following recursion:

* *P(n) = 1 ( n = 1 )*
* *P(n) = Sum( P ( k ) \* P ( n - k ) ) ( n >= 2 )*

*Apply Dynamic Programming:*

*Four Steps to get the final solution of Matrix Chain Multiplication:*

1. *Construct one Best Solution Structure.*
2. *Define the value of Best Solution recursively.*
3. *Calculate the value of Best Solution, normally by Bottom to Up Method.*
4. *Construct one Best Solution by using Calculated Information.*

***Step One - Construct Structure Feature for Best Parenthesis Schema:***

* The first step is to find the Best Structure Feature, then we can use this kind of Structure Feature to construct the Best Solution.
* For convenience, we use Ai..j ( i <= j ) to represent the Matrix Multiplication of Ai \* Ai+1 \*...\* Aj. We can know that if i < j, then we can parenthesized Ai \* Ai+1 \*...\* Aj, then we need to use integer to divide the Matrix Chain by integer k, and ensure that i =< k < j.
* For integer k, we need to divide Matrix into Ai..k and Ak+1..j, then calculate Multiplication to get the final Ai..j. *( The Cost of this Schema equals to the Cost of Matrix Ai..k, the Cost of Matrix Ak+1..j and the Cost of Matrix Multiplication of both Matrix Ai..k and Ak+1..j. )*

*Conclusion:*

Let’s consider how to construct the Best Solution by using the Best Sub - Structure from the Best Solution of Sub - Question.

We already knew that the solution needs to divide the Matrix Chain and each solution consists of Best Solution of Sub - Question.

*( In order to construct the Best Solution of Matrix Multiplication Chain, we can divide the question into two Sub - Question ( The Best Parenthesis Question of Ai \* Ai+1 \*...Ak and Ak+1 \* Ak+2\* ... Aj. ), and get the Best Solution of Sub - Question Instances, and combine all solutions of Sub - Question. )*

*Attention:*

We need to assure that when we divide all Matrix, those division points of all possible Sub - Questions are all under Estimation, so there will have no other possible solution left.

***Step Two - One Recursive Solution Schema***

For Matrix Chain Multiplication Problem, we need to use all 1 <= i <= j <= n to make sure the Least Cost of Schema of Ai \* Ai+1 \* ... Aj as Sub - Question. We assume that *m [ i , j ] as the least times that the Multiplication times the Matrix Ai...j needs. The Best Solution of Original Question that calculates the least cost of A1...n is m[ 1, n ].*

At first, we assume that for i = 1, 2, ..., n, then m[ i , i ] = 0; If i < j, then we calculate m [ i, j ] by using the Best Sub - Structure in step one.

*Analysis:*

The Division Pointer of Matrix Chain Ai\*Ai+1\*...Aj is among Ak and Ak+1, of which i <= k < j.

*m [ i, j ] = Cost(Ai..k) + Cost(Ak+1..j) + Cost(Ai..k \* Ak+1..j)*

Here, the size of Matrix Ai = pi-1 \* pi, therefore Ai..k \* Ak+1..j = p ( i - 1 ) \* pk \* pj. We need to mention that Cost(Ai..k) = m [ i, k ] and Cost(Ak+1..j) = m [ k+1, j ] and Cost(Ai..k \* Ak+1..j) = p ( i - 1 ) \* pk \* pj.

*Conclusion:*

The Least Cost Parenthesis Schema is to get the best value k, and k is the possible value of i, i + 1, ..., j - 1. We need to check all possible situations and try to find the best situation. So, the Least Cost Parenthesis Schema of Ai \* Ai+1 \*...\* Aj is:

*m [ i, j ] = 0 ( i = j )*

*m [ i, j ] = min ( m [ i, k ] + m [ k+1, j ] + p ( i - 1 ) \* pk \* pj ) ( i < j and i -1 < k < j )*

*Supplement:*

Of course, in these two equations, there have not provide enough information to construct the Best Solution. So, here we use the s[ i, j ] to save the Best Division Location k for Ai \* Ai+1...Aj which is to say that *Division Location k makes m[ i, j ] = m[ i, k ] + m[ k+1, j ] + p(i - 1) \* pk \* pj*.

***Step Three - Calculate the Best Cost***

*Feature:*

The Recursion Algorithm would meet the same sub - Question in the Recursive Calling Tree. *The Feature of Application Dynamic Programming is the overlapping of Sub - Question. (Another Feature of Application Dynamic Programming is the Best Sub-Structure.)*

*Data Structure - used during Calculation Process:*

The size of Matrix Ai is pi-1 \* pi (Here, i = 1, 2, 3, ..., n), p = <p0, p1, ..., pn> and the length of p.length = n + 1.

*During the Calculation Process, the auxiliary table m[ 1...n, 1...n ] is used to save m [ i, j ] while another auxiliary table s [ 1...n - 1, 2...n ] is used to save the best division k. Then we can use the table s to construct the Best Solution.*

*Example - A1 \* A2 \* A3:*

*Explanation:*

*Here, index 1, 2, and 3 means the multiplication from Matrix A1 to A3 and 3 > 1.*

* *The size of A1 equals to p0, p1;*
* *The size of A2 equals to p1, p2;*
* *The size of A3 equals to p2, p3;*

*Cost of A1 \* A2 = p0 \* p1 \* p2;*

*Size of A1 \* A2 equals to p0 \* p2;*

*Cost of ( A1 \* A2 ) \* A3 = p0 \* p2 \* p3 + p0 \* p1 \* p2;*

*Cost of A2 \* A3 = p1 \* p2 \* p3;*

*The size of A2 \* A3 equals to p1 \* p3;*

*Cost of A1 \* ( A2 \* A3 ) = p0 \* p1 \* p3 + p1 \* p2 \* p3;*

|  |  |  |  |
| --- | --- | --- | --- |
| m | 1 | 2 | 3 |
| 1 | 0 | * m[1, 2] = min(m[1,1] + m[2, 2] + p0 \* p1 \* p2 ) = p0 \* p1 \* p2 = 10 \* 100 \* 5 = 5000 | * m[1, 3] = m[1, 2] + m[3, 3] + p0 \* p2 \* p3 = 10 \* 5 \* 50 + 5000 = 7500 * m[1, 3] = m[1, 1] + m[2, 3] + p0 \* p1 \* p3 = 10 \* 100 \* 50 + 25000 = 75,000 * min (m[1, 3]) = 7500 |
| 2 |  | 0 | m[2, 3] = min( m[2, 2] + m[3, 3] + p1 \* p2 \* p3 ) = 0 + 0 + p1 \* p2 \* p3 = 100 \* 5 \* 50 = 25,000 |
| 3 |  |  | 0 |

*Best Solution for Division k:*

|  |  |  |
| --- | --- | --- |
| S | 2 | 3 |
| 1 | 1 | 1 |
| 2 |  | 2 |

***Step Four - Construct the Best Solution***