Algorithms analysis	Section	02
	Student number	21600635
Homework 2 – MCM by DP	Name	Jung, Bo Moon

□ Explanation of MCM algorithm.

- Dynamic Programming

DP is to solve one big problem by dividing it into several subproblems, do memoization, and combine them to solve the big problem optimally. Here, memoization is the concept of storing and reusing a subproblem once calculated. Therefore, the DP algorithm must have the same correct answer every time when the same problem is solved, and it can be used efficiently when sub-problems occur repeatedly. There are two implementation methods of the DP algorithm, which are'Bottom-up' and'Top-down'. The former method is a method of obtaining memoization from the sub-problem in order, the latter method is a lazy way to solve the subproblem only when the subproblem has not yet been solved when solving the big problem.

- MCM Algorithm

The MCM algorithm is based on DP's 'Bottom-up'. What we want to find using the MCM algorithm is to find the optimal cost of multiplying the entire matrix when there is a matrix of $A_i \dots A_j$. Therefore, it is to find the optimal index(k), divide it into two subproblems like $A_i \dots A_k$ and $A_{k+1} \dots A_j$ to find the optimal cost, and solve the upper problem after memoization. Also note the value of 'k'. The following is a step-by-step description of this process.

1) Find the optimal substructure
2) A recursive solution
3) Computing the optimal costs
4) Constructing an optimal solution.

- My code algorithm

To reduce unnecessary computation, I use two matrices to record the values we need. The first matrix is cost matrix representing cost. The other matrix is s matrix representing the position of k are declared as dynamic allocation. The reason is that the length of the matrix is different for each problem. In addition, the matrix representing cost is initialized to 0 for cost[i][i] and INT_MAX for the rest, so that the value can be replaced when more efficient cost comes out. And the matrix representing the position of k is initialized to 0 so that it can be replaced with the value of k representing the most efficient cost.

Next, since the first index of the array is 0, for convenience, the length of the two matrices is 1, such as A_iA_{i+1} and 2, such as $A_iA_{i+1}A_{i+2}$ and so on.

If so, now let's find the cost matrix and the s matrix.

1) In first for statement, when n = 'length of matrix', it increases from l = 1 to l = n-1, and allows you to calculate cost for each length of matrix.
2) In second for statement, cost[i][j] is incremented from i = 0 to I = n – l – 1 to assign a value.
3) In last for statement, the value of k in $A_i \dots A_k \dots A_j$ should be specified to determine the optimal cost. In this process, cost is calculated for each value of k, then the more efficient cost is assigned to cost[i][j] and k is assigned in s[i][j].

Next, Print the cost matrix and s matrix created by the above process to check, and print the expression of multiplication using the result() function. Finally, the most efficient cost value is also printed.

The Result() function displays the multiplication expression by reflecting the values of the s matrix as recursion.

Finally, it allows you to save memory space by freeing dynamic allocations.

□ More than 10 MCM Problems and solutions

Index	Problem	Solution	Equation
1	{1,2.3.4.5}	38	(((A1A2)A3)A4)
2	{4,3,2,1}	18	(A1(A2A3))
3	{2,4,5,1,6}	40	((A1(A2A3))A4)
4	{5,7,2,4,5}	160	((A1A2)(A3A4))
5	{9,1,5,6,4}	90	(A1((A2A3)A4))
6	{3,6,5,2}	96	(A1(A2A3))
7	{7,1,8,4}	60	(A1(A2A3))
8	{10,2,4,3}	84	(A1(A2A3))
9	{6,1,9,3}	45	(A1(A2A3))
10	{30,35,15,5,10,20,25}	15125	((A1(A2A3))((A4A5)A6))

□ Screenshots of your program running

1)

```
int mat1[] = {1,2,3,4,5};
```

```
cost matrix
          0          6         18         38
 2147483647          0         24         64
 2147483647          0         60
 2147483647          0

k matrix
          0          1          2          3
          0          0          2          3
          0          0          0          3
          0          0          0          0
(((A1A2)A3)A4)
value : 38
```

2)

```
int mat2[] = {4,3,2,1};
```

```
cost matrix
          0         24         18
 2147483647          0          6
 2147483647          0

k matrix
          0          1          1
          0          0          2
          0          0          0
(A1(A2A3))
value : 18
```

3)

```
int mat3[] = {2,4,5,1,6};
```

```
cost matrix
          0         40         28         40
 2147483647          0         20         44
 2147483647          0         30
 2147483647          0

k matrix
          0          1          1          3
          0          0          2          3
          0          0          0          3
          0          0          0          0
((A1(A2A3))A4)
value : 40
```

4)

```
int mat4[] = {5,7,2,4,5};
```

```
cost matrix
          0         70        110        160
 2147483647          0         56        110
 2147483647          0         40
 2147483647          0

k matrix
          0          1          2          2
          0          0          2          2
          0          0          0          3
          0          0          0          0
((A1A2)(A3A4))
value : 160
```

5)

```
int mat5[] = {9,1,5,6,4};
cost matrix
          0          45          84          90
 2147483647           0          30          54
 2147483647           0         120
 2147483647          0
 2147483647

k matrix
          0           1           1           1
          0           0           2           3
          0           0           0           3
          0           0           0           0
(A1((A2A3)A4))
value : 90
```

6)

```
int mat6[] = {3,6,5,2};
cost matrix
          0          90          96
 2147483647           0          60
 2147483647          0

k matrix
          0           1           1
          0           0           2
          0           0           0
(A1(A2A3))
value : 96
```

7)

```
int mat7[] = {7,1,8,4};
cost matrix
          0          56          60
 2147483647           0          32
 2147483647          0

k matrix
          0           1           1
          0           0           2
          0           0           0
(A1(A2A3))
value : 60
```

8)

```
int mat8[] = {10,2,4,3};
cost matrix
          0          80          84
 2147483647           0          24
 2147483647          0

k matrix
          0           1           1
          0           0           2
          0           0           0
(A1(A2A3))
value : 84
```

9)

```
int mat9[] = {6,1,9,3};
cost matrix
          0          54          45
 2147483647           0          27
 2147483647          0

k matrix
          0           1           1
          0           0           2
          0           0           0
(A1(A2A3))
value : 45
```

10)

```
int mat10[] = {30,35,15,5,10,20,25};
cost matrix
          0       15750        7875        9375       11875       15125
 2147483647           0        2625        4375        7125       10500
 2147483647           0         750        2500        5375
 2147483647           0        1000        3500
 2147483647           0        5000
 2147483647          0

k matrix
          0           1           1           3           3           3
          0           0           2           3           3           3
          0           0           0           3           3           3
          0           0           0           0           4           5
          0           0           0           0           0           5
          0           0           0           0           0           0
((A1(A2A3))((A4A5)A6))
value : 15125
```

□ Discussion about the results

- Compare with non-optimal solution.

index	MCM cost	Non MCM cost (multiply just in order)
1	38	38
2	18	32
3	40	62
4	160	210
5	90	531
6	96	120
7	60	280
8	84	200
9	45	216
10	15125	40500

As can be seen from the above results, it can be seen that the mcm algorithm gives a much more efficient cost than the case of multiplying the matrices sequentially.

- Time complexity

Time complex of brute – force algorithm = $\Omega(2^n)$
Time complex of mcm algorithm = $O(n^3)$

- Any possible way to improve the algorithm.

In this project, I think many students would have used the cost matrix and the s matrix to have a specific size. However, if it is possible to save memory space, I think that the algorithm will be able to perform more efficient processing not only in terms of cost but also in terms of multi-process. Furthermore, if the memory space is allocated like a staircase through dynamic allocation when declaring the matrix, a much more efficient algorithm is likely to be completed.

- What I learn from this task.

Through this task, I realized how important the cost of the algorithm is. Although there are many algorithms, it is considered that the algorithm using DP has a great advantage in terms of cost. In addition, complex codes and algorithms used in the actual field may take several days each time they are executed. At this time, if they are implemented with inefficient cost, it is considered a huge waste of time. In addition, by unintentionally holding the concept of dynamic allocation of double pointers in this task, it was possible to save not only cost but also memory space, which became an opportunity for myself to learn a lot.

□ Codes

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <limits.h>
#include <windows.h>
using namespace std;
int n = 0;
void mcm(int* mat);
void result(int** s, int i, int j, int* num);



int main(){
    //1
    int mat1[] = {1,2,3,4,5};
    n = sizeof(mat1) / sizeof(int) - 1;
    mcm(mat1);

    //2
    int mat2[] = {4,3,2,1};
    n = sizeof(mat2) / sizeof(int) - 1;
    mcm(mat2);

    //3
    int mat3[] = {2,4,5,1,6};
    n = sizeof(mat3) / sizeof(int) - 1;
    mcm(mat3);

    //4
    int mat4[] = {5,7,2,4,5};
    n = sizeof(mat4) / sizeof(int) - 1;
    mcm(mat4);

    //5
    int mat5[] = {9,1,5,6,4};
    n = sizeof(mat5) / sizeof(int) - 1;
    mcm(mat5);

    //6
    int mat6[] = {3,6,5,2};
    n = sizeof(mat6) / sizeof(int) - 1;
    mcm(mat6);

    //7
    int mat7[] = {7,1,8,4};
    n = sizeof(mat7) / sizeof(int) - 1;
    mcm(mat7);
```

```
    //8
    int mat8[] = {10,2,4,3};
    n = sizeof(mat8) / sizeof(int) - 1;
    mcm(mat8);

    //9
    int mat9[] = {6,1,9,3};
    n = sizeof(mat9) / sizeof(int) - 1;
    mcm(mat9);

    //10
    int mat10[] = {30,35,15,5,10,20,25};
    n = sizeof(mat10) / sizeof(int) - 1;
    mcm(mat10);
}


void mcm(int* mat){
    //memory allcoate
    int **cost_mat = new int*[n];
    int **s = new int*[n];
    for (int i = 0; i < n; ++i) {
        cost_mat[i] = new int[n];
        s[i] = new int[n];
    }
    //init
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            s[i][j] = 0;
            cost_mat[i][j] = INT_MAX;
        }
    }

    for(int i = 0; i < n; i++){
        cost_mat[i][i] = 0;
    }

    //calculate cost and k
    for(int l = 1; l < n; l++){
        for(int i = 0; i < n - l; i++){
            int j = i + l;
            for(int k = i; k < j; k++){
                int cost = cost_mat[i][k] + cost_mat[k + 1][j] + mat[i] * mat[k + 1] *
mat[j + 1];
                if(cost < cost_mat[i][j]){
                    cost_mat[i][j] = cost;
                    s[i][j] = k + 1;
                }
            }
        }
    }
```

```
    //print out cost matrix and k matrix
    cout << "cost matrix\n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cout.width(10);
            cout << cost_mat[i][j];
        }
        cout << "\n";
    }
    cout << "\nk matrix\n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cout.width(10);
            cout << s[i][j];
        }
        cout << "\n";
    }
    //print out result
    int num = 1;
    result(s, 0, n - 1, &num);
    cout << "\n";
    cout << "value : " << cost_mat[0][n - 1];
    cout << "\n_____________________________\n\n";

    //free memory
    for (int i = 0; i < n; ++i) {
        delete[] cost_mat[i];
        delete[] s[i];
    }
}

void result(int** s, int i, int j, int* num){
    Sleep(50);
    if(i == j){
        cout << "A" << *num;
        *num = *num + 1;
        return;
    }
    cout << "(";
    result(s, i, s[i][j] - 1, num);
    result(s, s[i][j], j, num);
    cout << ")";
}
```