Experiment-4

AIM To implement Matrix chain multiplication & analyze its time complexity

Software Used VS Code

Theory Matrix Chain Multiplication (MCM) is an optimization problem that is solved using dynamic programming. Given a sequence of matrices the goal is to find most effective & efficient way to multiply these matrices.

→ Algorithm

```
matrix chain order (p)
n = p.length - 1
let m[1..n, 1...n] & S[1...n-1, 2...n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n          // l is chain length
    for i = 1 to n - i + 1
        j = i - l + 1
        m[i, j] = $\infty$
        for k = 1 to j - 1
            $q = m[i, k] + m[k+1, j] + P_i + P_k P_j$
            if q < m[i, j]
                m[i, j] = q
                S[i, j] = k
return m & S
```

m will tell the optimisation & s will guide for parenthesization.

→ Time Complexity analysis.

1^{st} level of $A[1, n]$; $k=1$ to $n-1$

for $(n-1)$ expression,

⇒ 1^{st} level $(n-1)c$

2^{nd} level we have 2 values

$k=1$ to $n-2$

⇒ Cost $= 2 \times c\,(n-2)$

3^{rd} level we have 3 values

Cost for 1 value $= c\,(n-3)$ ($\because$ we have $(n-3)$ expressions $k=1$ to $n-3$)

In General case, we have
$(n-1)$ levels

for $(n-1)^{th}$ level

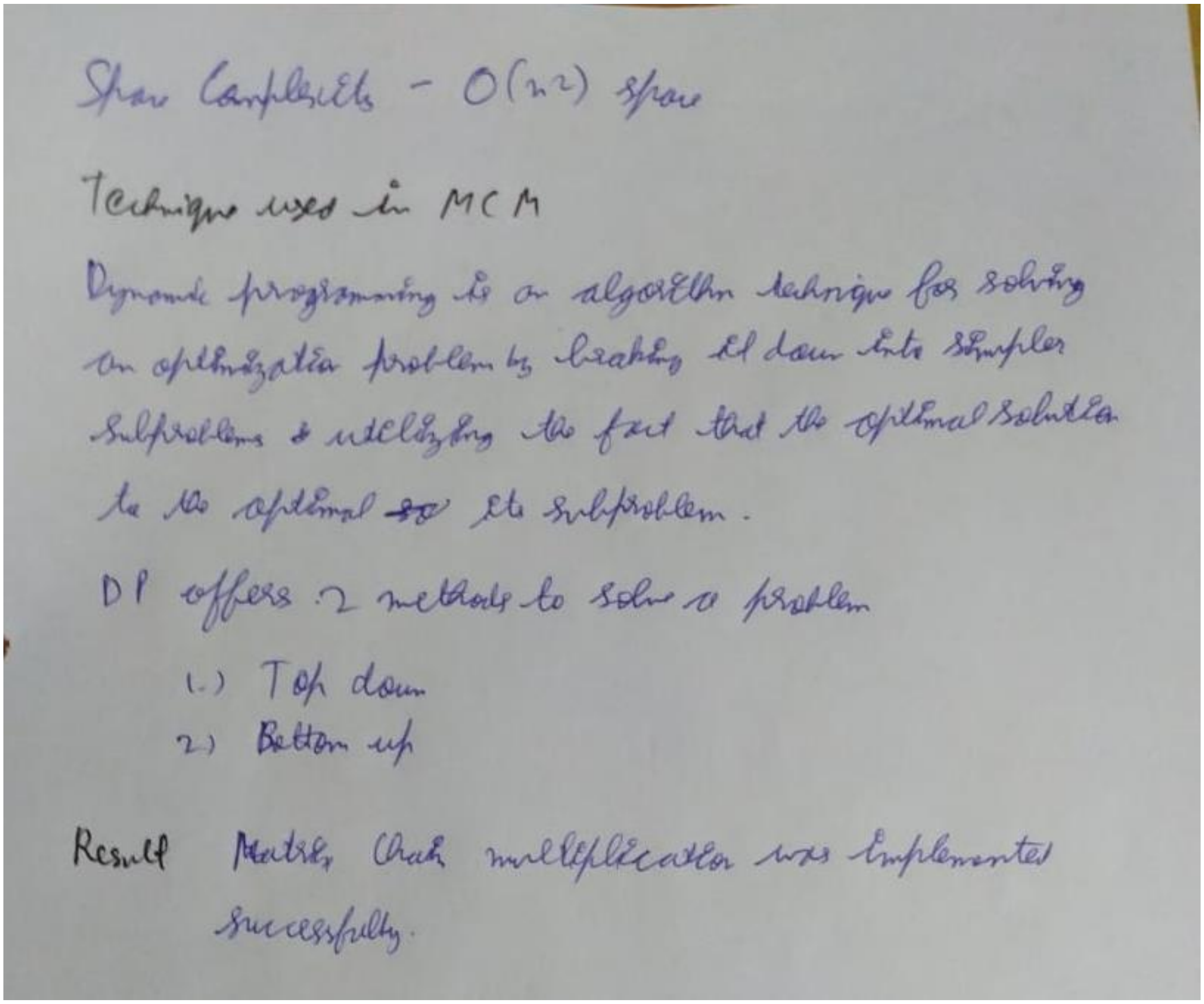
Cost $= (n-1)c\,(n-(n-1))$

⇒ Time Complexity

$$1 \cdot c(n-1) + 2c(n-2) + 3c(n-3) + \dots + (n-1) \cdot c(n-(n-1))$$

$$= c\left[(n + 2n + 3n + \dots (n-1)n) - (1 \times 1 - 2 \times 2 \dots (n-1)(n-1)\right]$$

$$= \frac{n^3}{6} \times c \Rightarrow O(n3)$$



Space Complexity - O(n2) space

Technique used in MCM

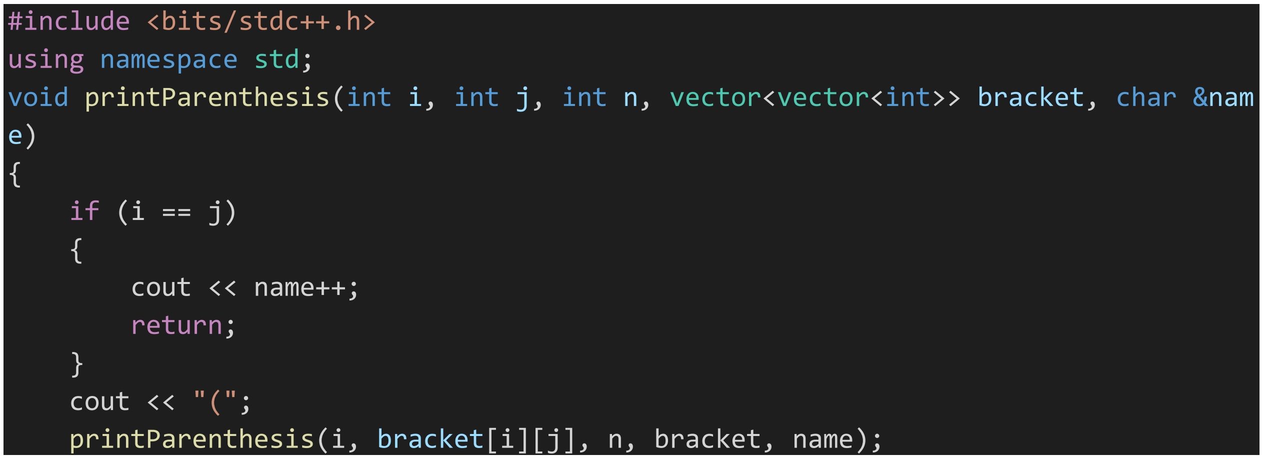
Dynamic programming is an algorithm technique for solving an optimization problem by breaking it down into simpler subproblems & utilizing the fact that the optimal solution to the optimal ~~so~~ its subproblem.

DP offers 2 methods to solve a problem

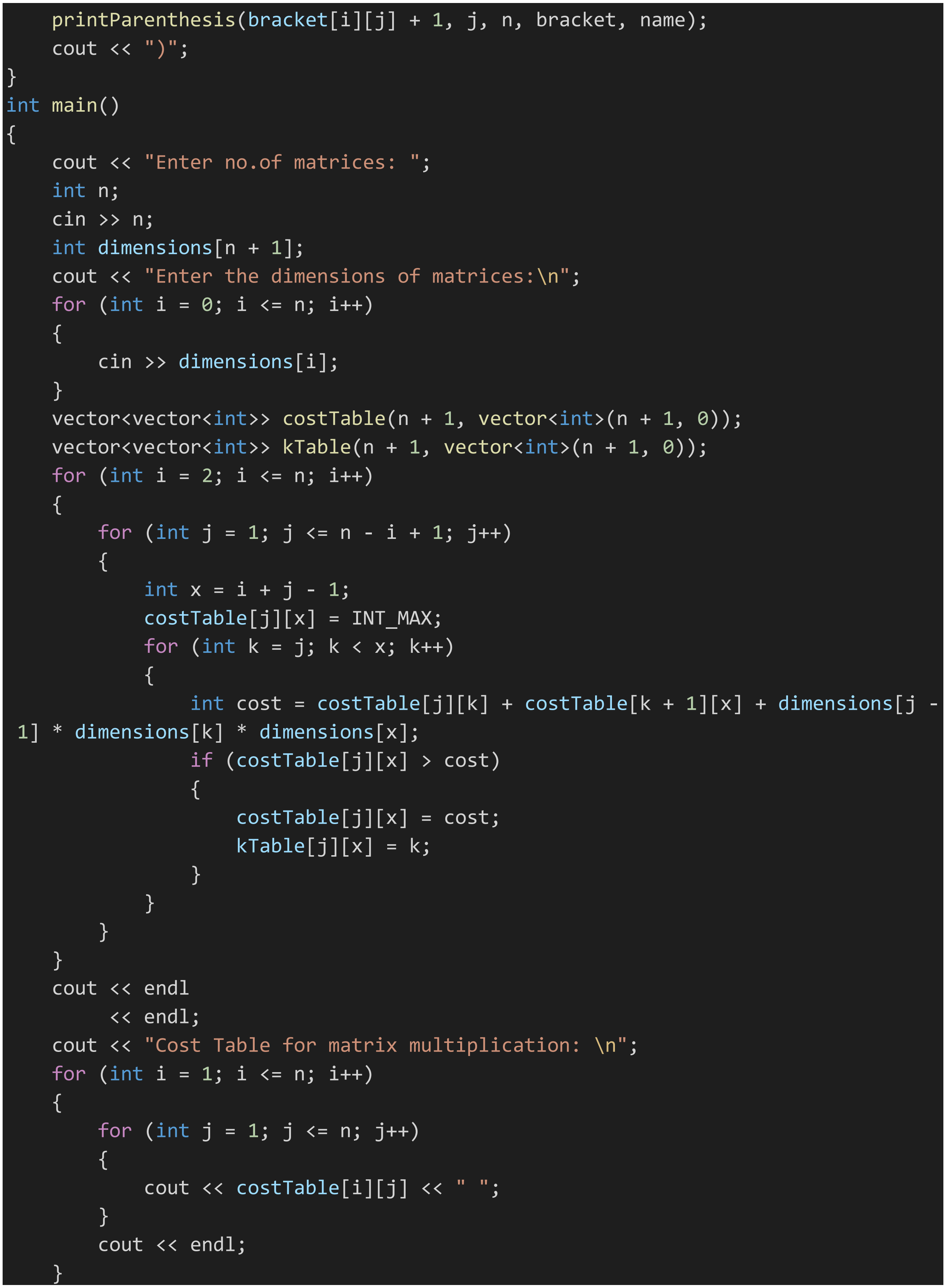
1.) Top down
2.) Bottom up

Result Matrix Chain multiplication was implemented successfully.

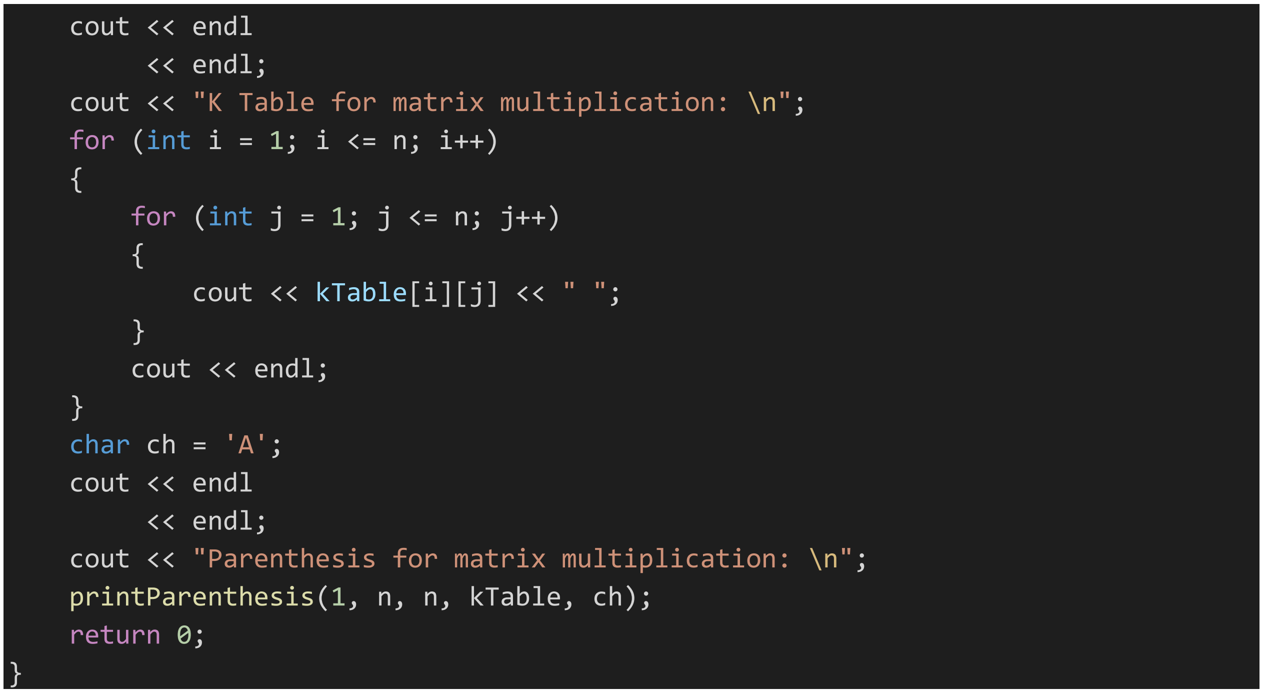
Code



```
#include <bits/stdc++.h>
using namespace std;
void printParenthesis(int i, int j, int n, vector<vector<int>> bracket, char &name)
{
    if (i == j)
    {
        cout << name++;
        return;
    }
    cout << "(";
    printParenthesis(i, bracket[i][j], n, bracket, name);
```

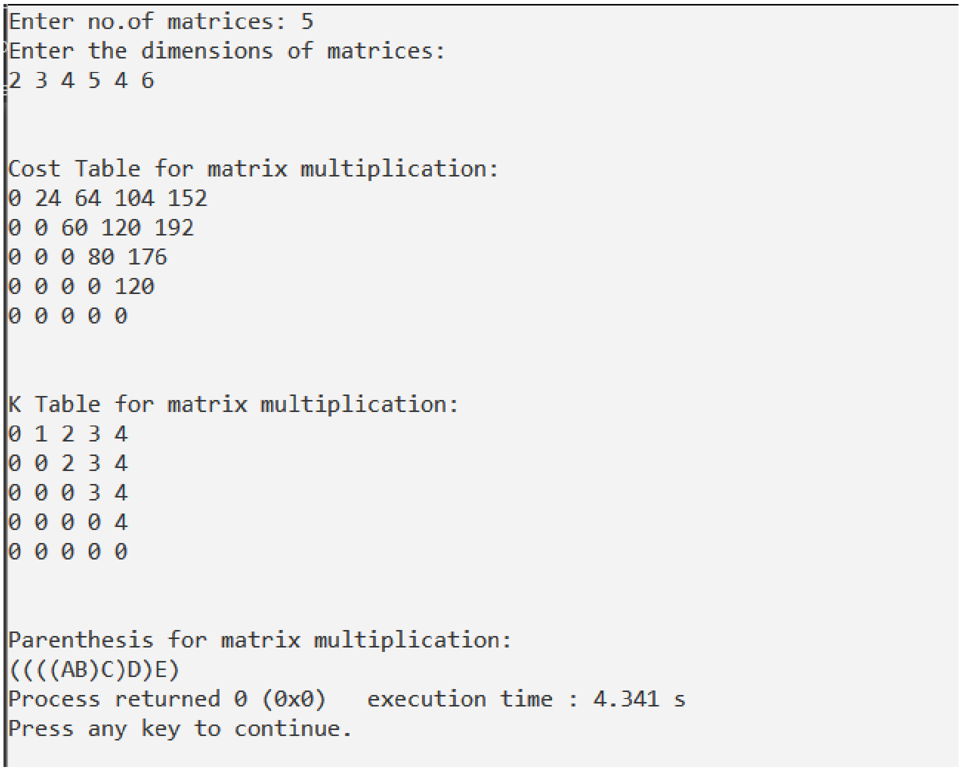


```
    printParenthesis(bracket[i][j] + 1, j, n, bracket, name);
    cout << ")";
}
int main()
{
    cout << "Enter no.of matrices: ";
    int n;
    cin >> n;
    int dimensions[n + 1];
    cout << "Enter the dimensions of matrices:\n";
    for (int i = 0; i <= n; i++)
    {
        cin >> dimensions[i];
    }
    vector<vector<int>> costTable(n + 1, vector<int>(n + 1, 0));
    vector<vector<int>> kTable(n + 1, vector<int>(n + 1, 0));
    for (int i = 2; i <= n; i++)
    {
        for (int j = 1; j <= n - i + 1; j++)
        {
            int x = i + j - 1;
            costTable[j][x] = INT_MAX;
            for (int k = j; k < x; k++)
            {
                int cost = costTable[j][k] + costTable[k + 1][x] + dimensions[j -
 1] * dimensions[k] * dimensions[x];
                if (costTable[j][x] > cost)
                {
                    costTable[j][x] = cost;
                    kTable[j][x] = k;
                }
            }
        }
    }
    cout << endl
         << endl;
    cout << "Cost Table for matrix multiplication: \n";
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            cout << costTable[i][j] << " ";
        }
        cout << endl;
    }
```



```
    cout << endl
         << endl;
    cout << "K Table for matrix multiplication: \n";
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            cout << kTable[i][j] << " ";
        }
        cout << endl;
    }
    char ch = 'A';
    cout << endl
         << endl;
    cout << "Parenthesis for matrix multiplication: \n";
    printParenthesis(1, n, n, kTable, ch);
    return 0;
}
```

Output



```
Enter no.of matrices: 5
Enter the dimensions of matrices:
2 3 4 5 4 6


Cost Table for matrix multiplication:
0 24 64 104 152
0 0 60 120 192
0 0 0 80 176
0 0 0 0 120
0 0 0 0 0


K Table for matrix multiplication:
0 1 2 3 4
0 0 2 3 4
0 0 0 3 4
0 0 0 0 4
0 0 0 0 0


Parenthesis for matrix multiplication:
((((AB)C)D)E)
Process returned 0 (0x0)   execution time : 4.341 s
Press any key to continue.
```