Name: Atharva Bilonikar

Uid no: 2021700011

CSE DS D1

DAA EXP 5

Aim:

Matrix Chain multiplication using Dynammic Programming

Algorithm:

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i = j
2      then print "A"_i
3      else print "("
4           PRINT-OPTIMAL-PARENS (s, i, s[i, j])
5           PRINT-OPTIMAL-PARENS (s, s[i, j] + 1, j)
6           print ")"
```

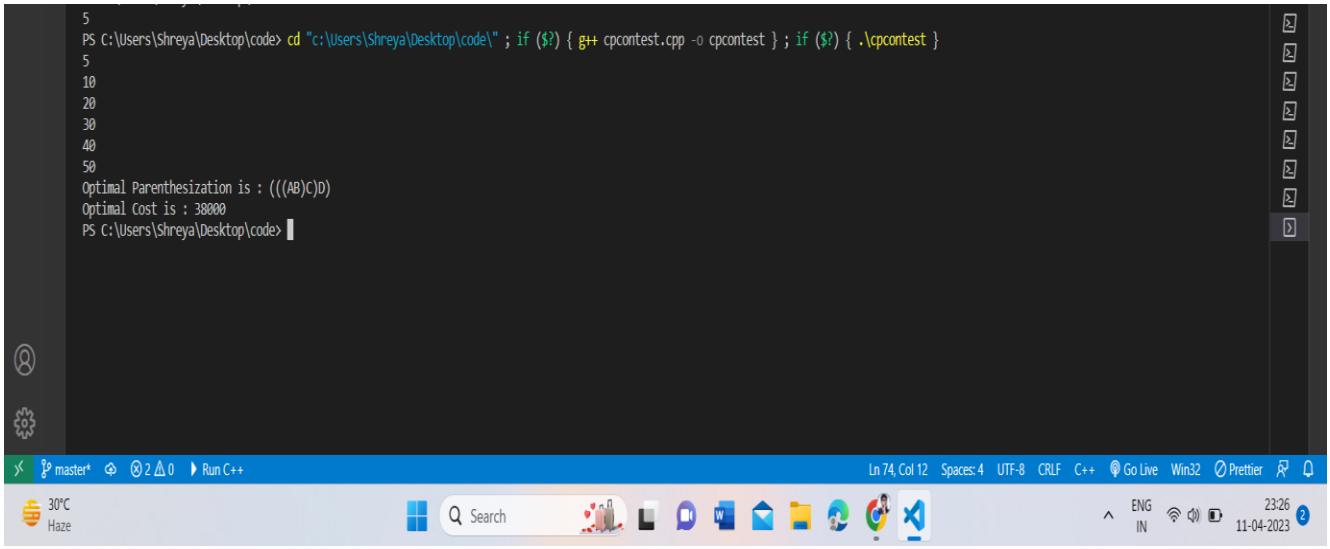
In the example of Figure 15.3, the call PRINT-OPTIMAL-PARENS$(s, 1, 6)$ prints the parenthesization $((A_1(A_2A_3))((A_4A_5)A_6))$.

Code:

```cpp
#include <bits/stdc++.h>
using namespace std;


void printParenthesis(int i, int j, int n, int* bracket,
                      char& name)
{

    if (i == j) {
        cout << name++;
        return;
    }

    cout << "(";


    printParenthesis(i, *((bracket + i * n) + j), n,
                     bracket, name);


    printParenthesis(*((bracket + i * n) + j) + 1, j, n,
                     bracket, name);
    cout << ")";
}


void matrixChainOrder(int p[], int n)
{

    int m[n][n];


    int bracket[n][n];

    for (int i = 1; i < n; i++)
        m[i][i] = 0;

    // L is chain length.
    for (int L = 2; L < n; L++)
    {
        for (int i = 1; i < n - L + 1; i++)
        {
            int j = i + L - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k <= j - 1; k++)
            {
```

```
                int q = m[i][k] + m[k + 1][j]
                        + p[i - 1] * p[k] * p[j];
                if (q < m[i][j])
                {
                    m[i][j] = q;


                    bracket[i][j] = k;
                }
            }
        }
    }

    char name = 'A';

    cout << "Optimal Parenthesization is : ";
    printParenthesis(1, n - 1, n, (int*)bracket, name);
    cout << "\nOptimal Cost is : " << m[1][n - 1];
}


int main()
{
    int N;
    cin>>N;
    int arr[N];
    for(int i=0;i<N;i++)cin>>arr[i];

    matrixChainOrder(arr, N);
    return 0;
}
```

Output:



Conclusion:

In this experiment, I understood how to implement MCM algorithm using dynamic programing technique.