**MATRIX CHAIN MULTIPLICATION**

**PRATEEK MOHANTY**

**20BCE1482**

* **DYANMIC PROGRAMMING**

**CODE**

#include <bits/stdc++.h>

using namespace std;

int dp[100][100];

// Function for matrix chain multiplication

int matrixChainMemoised(int \*p, int i, int j)

{

    if (i == j)

    {

        return 0;

    }

    if (dp[i][j] != -1)

    {

        return dp[i][j];

    }

    dp[i][j] = INT\_MAX;

    for (int k = i; k < j; k++)

    {

        dp[i][j] = min(

            dp[i][j], matrixChainMemoised(p, i, k) + matrixChainMemoised(p, k + 1, j) + p[i - 1] \* p[k] \* p[j]);

    }

    return dp[i][j];

}

int MatrixChainOrder(int \*p, int n)

{

    int i = 1, j = n - 1;

    return matrixChainMemoised(p, i, j);

}

// Driver Code

int main()

{

    int arr[] = {1, 2, 3, 4};

    int n = sizeof(arr) / sizeof(arr[0]);

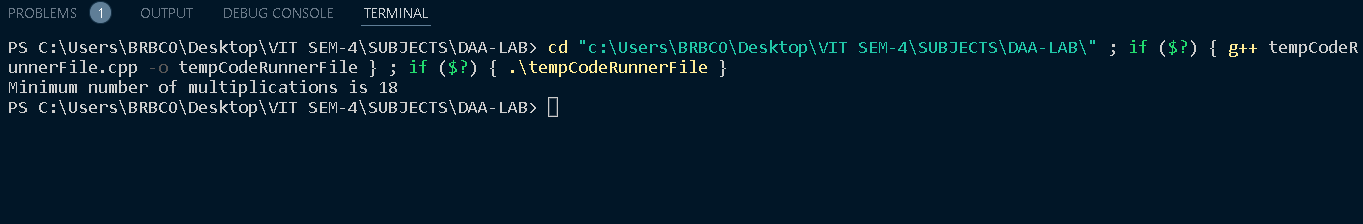
    memset(dp, -1, sizeof dp);

    cout << "Minimum number of multiplications is "

         << MatrixChainOrder(arr, n);

}

**OUTPUT**

****

* **ITERATIVE METHOD**

**CODE**

#include <bits/stdc++.h>

using namespace std;

// Matrix Ai has dimension p[i-1] x p[i]

// for i = 1..n

int MatrixChainOrder(int p[], int i, int j)

{

    if (i == j)

        return 0;

    int k;

    int min = INT\_MAX;

    int count;

    // place parenthesis at different places

    // between first and last matrix, recursively

    // calculate count of multiplications for

    // each parenthesis placement and return the

    // minimum count

    for (k = i; k < j; k++)

    {

        count = MatrixChainOrder(p, i, k) + MatrixChainOrder(p, k + 1, j) + p[i - 1] \* p[k] \* p[j];

        if (count < min)

            min = count;

    }

    // Return minimum count

    return min;

}

// Driver Code

int main()

{

    int arr[] = {1, 2, 3, 4};

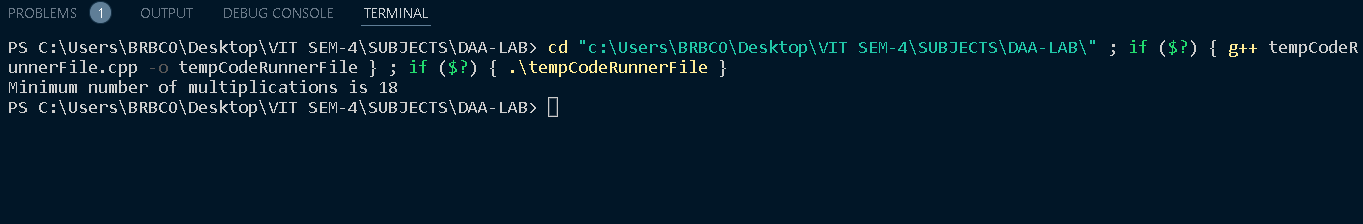
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Minimum number of multiplications is "

         << MatrixChainOrder(arr, 1, n - 1);

}

**OUTPUT**

****