**Matrix Chain Multiplication**

**Hard**Accuracy: 59.72% Submissions: 21296 Points: 8

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The efficient way is the one that involves the least number of multiplications.

The dimensions of the matrices are given in an array **arr[]** of size **N** (such that N = number of matrices + 1) where the **ith** matrix has the dimensions **(arr[i-1] x arr[i])**.

**Example 1:**

**Input:** N = 5

arr = {40, 20, 30, 10, 30}

**Output:** 26000

**Explaination:** There are 4 matrices of dimension

40x20, 20x30, 30x10, 10x30. Say the matrices are

named as A, B, C, D. Out of all possible combinations,

the most efficient way is (A\*(B\*C))\*D.

The number of operations are -

20\*30\*10 + 40\*20\*10 + 40\*10\*30 = 26000.

**Example 2:**

**Input:** N = 4

arr = {10, 30, 5, 60}

**Output:** 4500

**Explaination:** The matrices have dimensions

10\*30, 30\*5, 5\*60. Say the matrices are A, B

and C. Out of all possible combinations,the

most efficient way is (A\*B)\*C. The

number of multiplications are -

10\*30\*5 + 10\*5\*60 = 4500.

**Your Task:**  
You do not need to take input or print anything. Your task is to complete the function **matrixMultiplication()** which takes the value **N** and the array **arr[]** as input parameters and returns the minimum number of multiplication operations needed to be performed.

**Expected Time Complexity:** O(N3)  
**Expected Auxiliary Space:** O(N2)

**Constraints:**   
2 ≤ N ≤ 100  
1 ≤ arr[i] ≤ 500

class Solution{

public:

    int dp[101][101];

    int solve(int arr[], int i, int j) {

        if (i>=j) return 0;

        if (dp[i][j]!=-1) return dp[i][j];

        int ans=INT\_MAX;

        for (int k=i; k<j; k++) {

            int sub=solve(arr, i, k)+solve(arr, k+1, j)+arr[i-1]\*arr[k]\*arr[j];

            ans=min(ans, sub);

        }

        return dp[i][j]=ans;

    }

    int matrixMultiplication(int N, int arr[]) {

        // code here

        memset(dp, -1, sizeof(dp));

        return solve(arr, 1, N-1);

    }

};