Matrix Chain Multiplication

Problem

We are given n matrices, we have to multiply them in such a way that total number of operations are minimum.

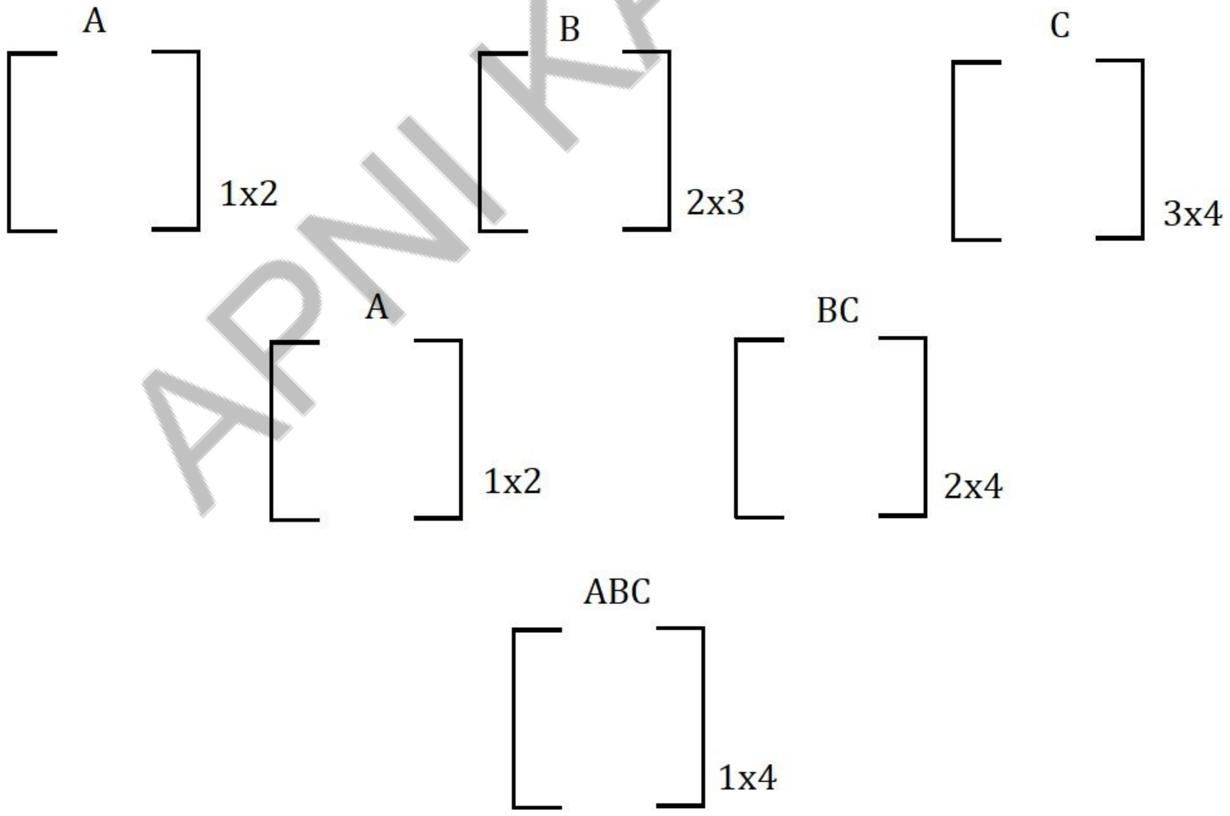
Example

$[\,]_{1x2}$ $[\,]_{2x3}$ $[\,]_{3x4}$

A B C

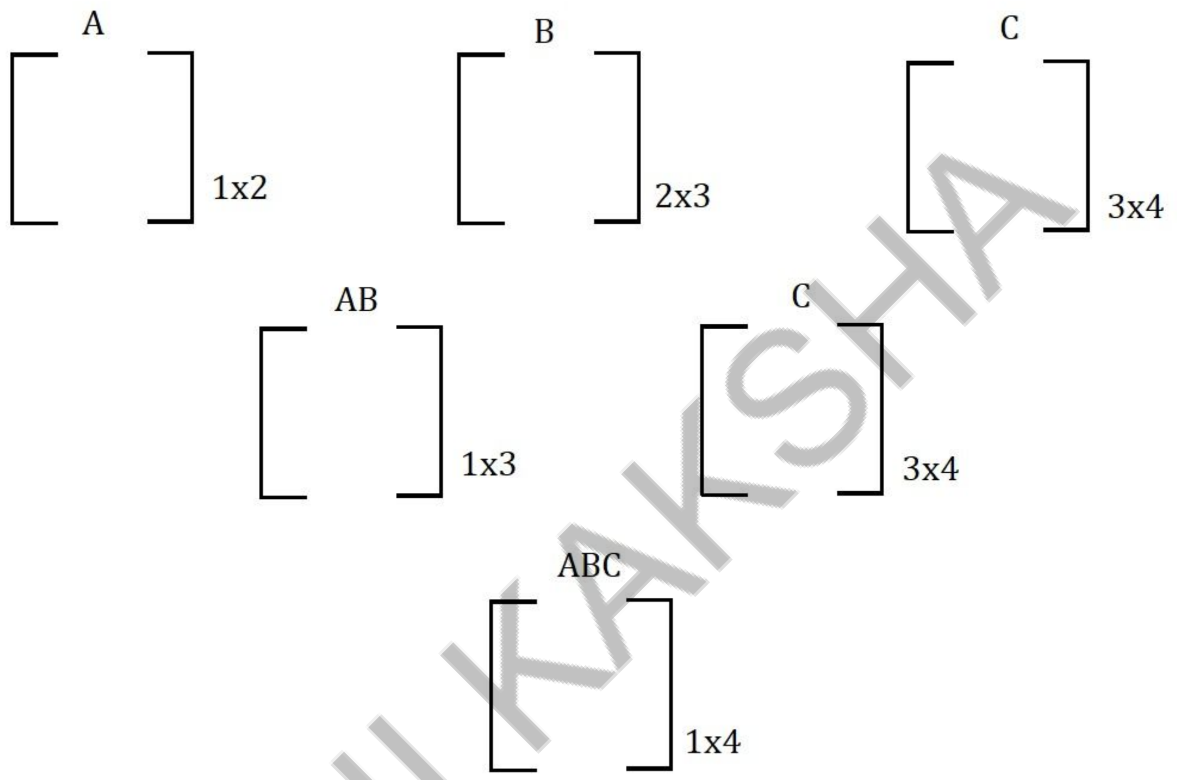
Since we know multiplication of matrices is associative, hence
A(BC) = (AB)C

Operations in A(BC)



$(\text{Total operations})_{A(BC)}$ = 2x3x4 + 1x2x4 = 32 operations

<u>Operations in (AB)C</u>

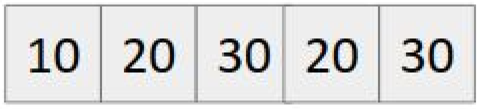


$(\text{Total Operations})_{A(BC)}$ = 1X2X3 + 1X3X4 = 18 operations

Therefore, (AB)C is more efficient than A(BC).

Dimensions of matrices will be given in the form of an array.

Example



10	20	30	20	30

The Dimension of i^{th} matrix is a[i-1] x a[i].

Example

M_1 -> a[0] x a[1] = 10 x 20

M_2 -> a[1] x a[2] = 20 x 30

M_3 -> a[2] x a[3] = 30 x 20

M_4 -> a[3] x a[4] = 20 x 30

Therefore dimension of matrix multiplication from

M_i to M_j -> a[i-1] x a[j]

Example: M_1 M_2 M_3 -> a[0] x a[3] = 10 x 20

Our Recurrence Relation becomes

$$f(M_1M_2....M_N) = min(f(M_1....M_k) + f(M_{k+1}....M_N) + a[0] \times a[1] \times a[N])$$

$$where\ 1 <= k <= N-1$$

Let us take 4 matrices A, B, C, D.

We can see that answer of ABCD depends on

1. (A)(BCD)
2. (AB)(CD)
3. (ABC)(D)

Whichever from 1. , 2. Or 3 gives minimum operations, that is the answer.

In other words, we can say that 3 cuts are possible,
(i) A | B C D
(ii) A B | C D
(iii) A B C | D

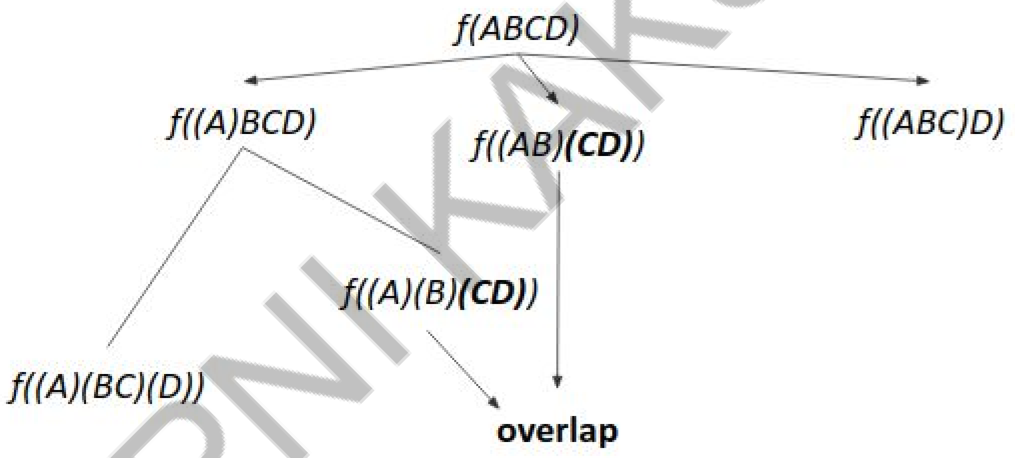
We can write its recurrence as

f(ABCD) = min(f(A|BCD), f(AB|CD), f(ABC|D))

Since it has a recurrence relation, therefore it follows optimal substructure property.

Checking whether it has overlapping subproblem property also?

Making recursion tree



We can see that computation of *f(CD)* is repeated, hence it possesses overlapping subproblem property.

Hence it can be solved using dynamic programming.

Approach 1

1. Write the recursive solution.
2. Memoize it.

Approach 2 (Tabulation (Bottom Up))

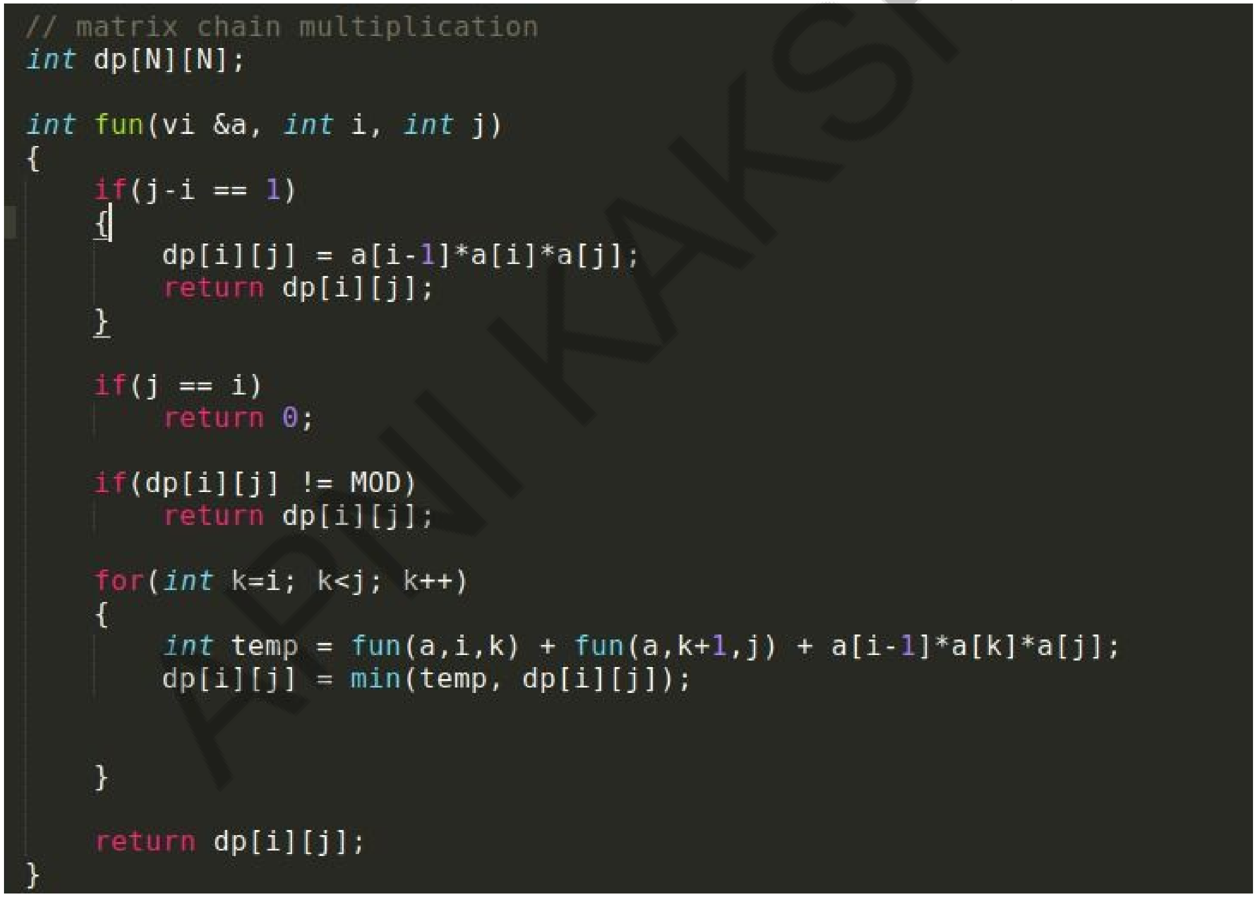
1. Build from base.
2. For each gap=0 to gap=n-2, compute all submatrix multiplication and their results.
3. Build the answer using,

for every k=i to k=j-1

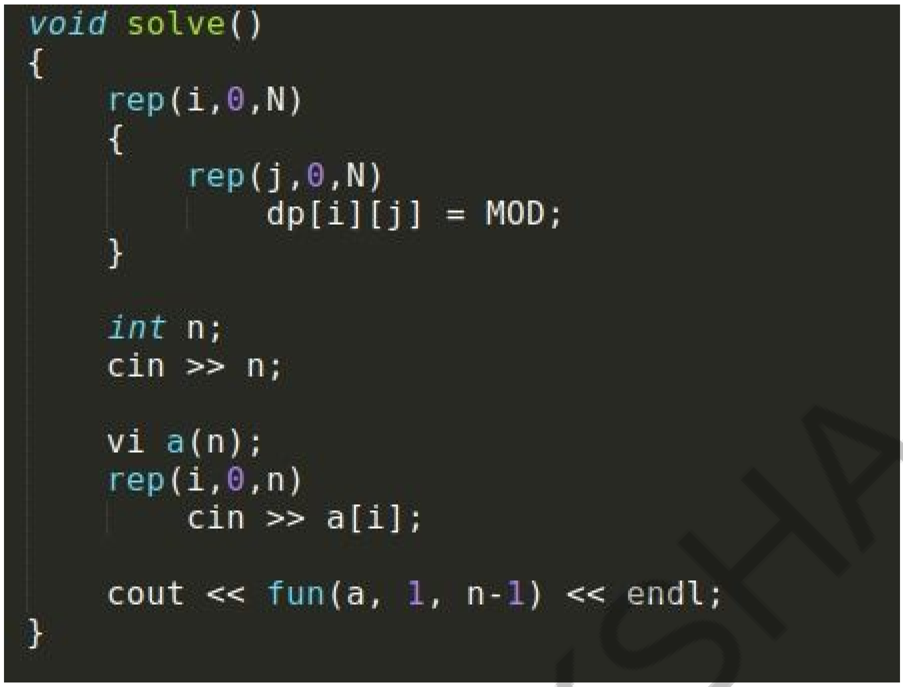
dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] + a[i-1] x a[k] x a[j])

Time complexity: O(n^3)

Code (Recursive)

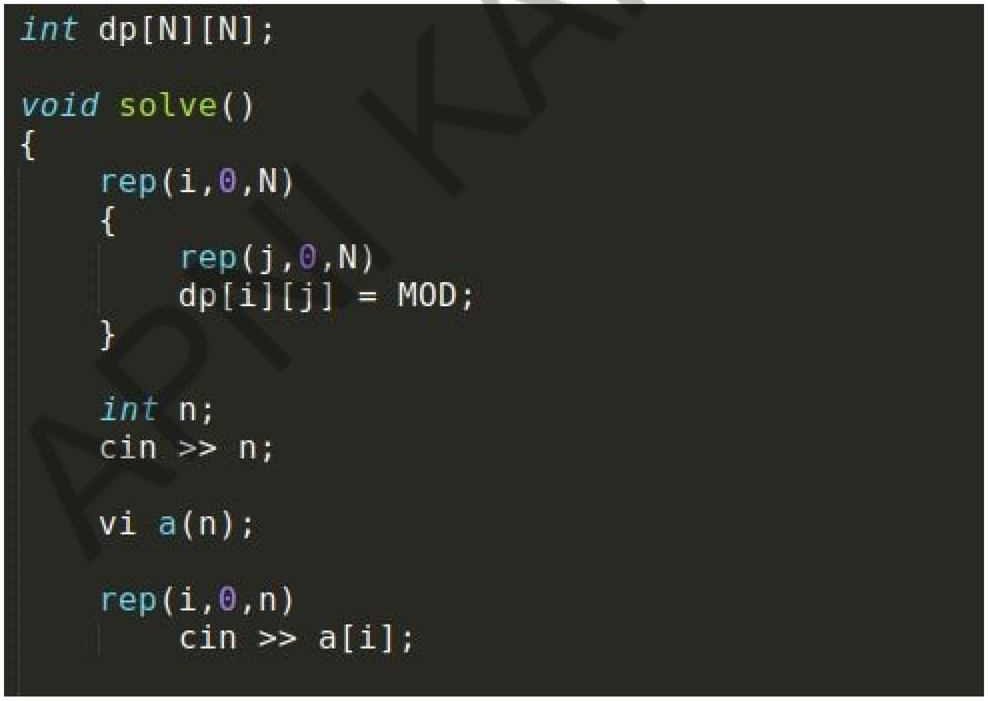


```
// matrix chain multiplication
int dp[N][N];

int fun(vi &a, int i, int j)
{
    if(j-i == 1)
    {
        dp[i][j] = a[i-1]*a[i]*a[j];
        return dp[i][j];
    }

    if(j == i)
        return 0;

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

    for(int k=i; k<j; k++)
    {
        int temp = fun(a,i,k) + fun(a,k+1,j) + a[i-1]*a[k]*a[j];
        dp[i][j] = min(temp, dp[i][j]);


    }

    return dp[i][j];
}
```

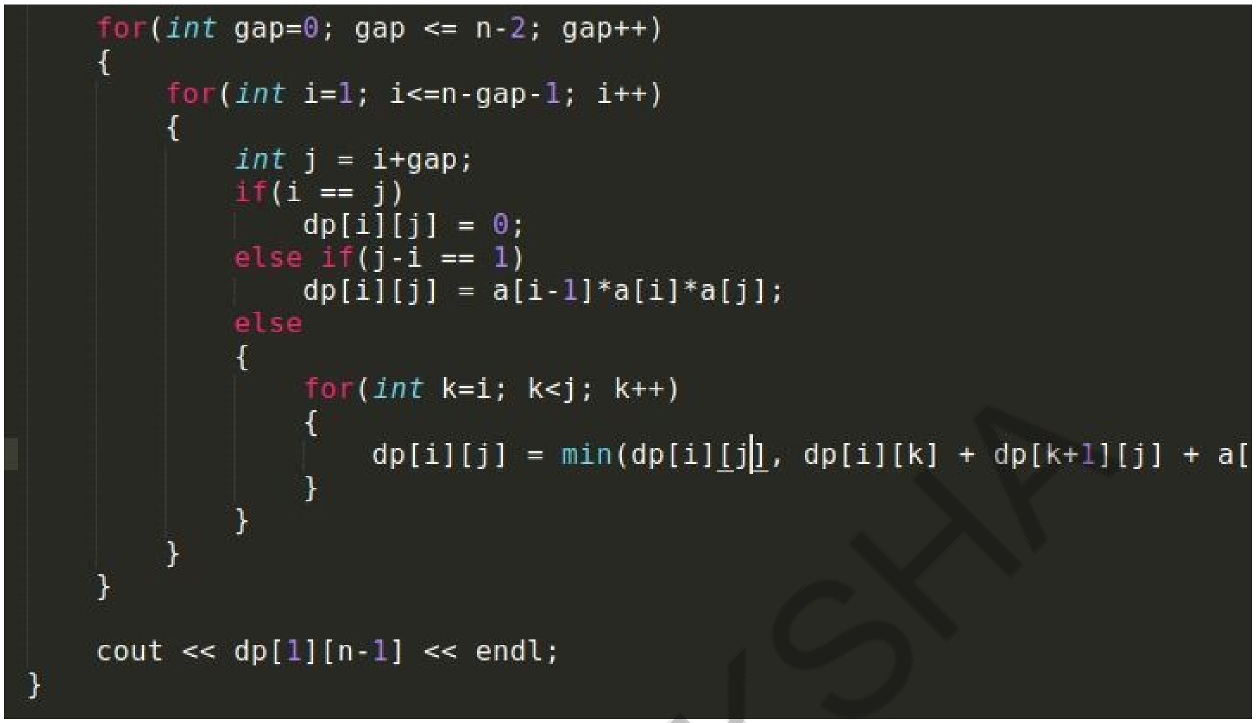


```
void solve()
{
    rep(i,0,N)
    {
        rep(j,0,N)
            dp[i][j] = MOD;
    }

    int n;
    cin >> n;

    vi a(n);
    rep(i,0,n)
        cin >> a[i];

    cout << fun(a, 1, n-1) << endl;
}
```

Code (Iterative)



```
int dp[N][N];

void solve()
{
    rep(i,0,N)
    {
        rep(j,0,N)
        dp[i][j] = MOD;
    }

    int n;
    cin >> n;

    vi a(n);

    rep(i,0,n)
        cin >> a[i];

```



```
    for(int gap=0; gap <= n-2; gap++)
    {
        for(int i=1; i<=n-gap-1; i++)
        {
            int j = i+gap;
            if(i == j)
                dp[i][j] = 0;
            else if(j-i == 1)
                dp[i][j] = a[i-1]*a[i]*a[j];
            else
            {
                for(int k=i; k<j; k++)
                {
                    dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] + a[
                }
            }
        }
    }

    cout << dp[1][n-1] << endl;
}
```