Matrix Chain Multiplication | (DP-48)

Problem Statement:

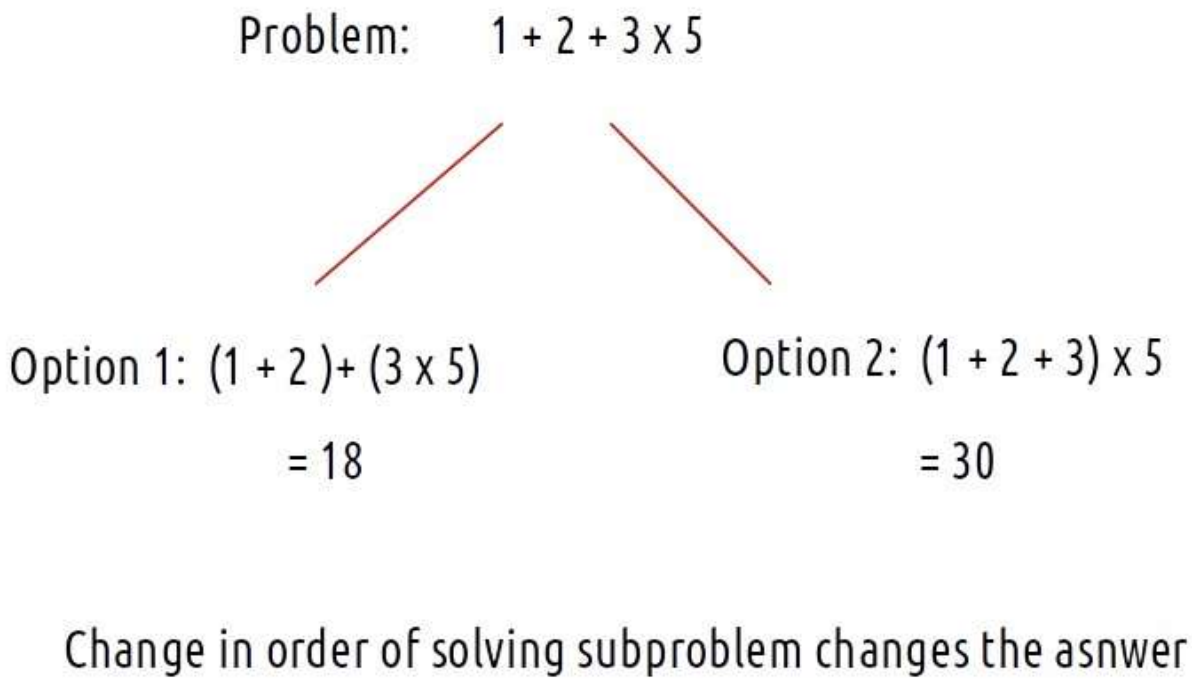
Matrix Chain Multiplication | Partition DP Starts

In the coming articles, we will discuss problems related to a new pattern called "**Partition DP"**.Before proceeding further, let us understand how to identify whether a problem can be solved using this pattern.

Pattern Identification:

Whenever we need to find the answer to a large problem such that the problem can be broken into subproblems and the final answer varies due to the **order** in which the subproblems are solved, we can think in terms of partition DP.

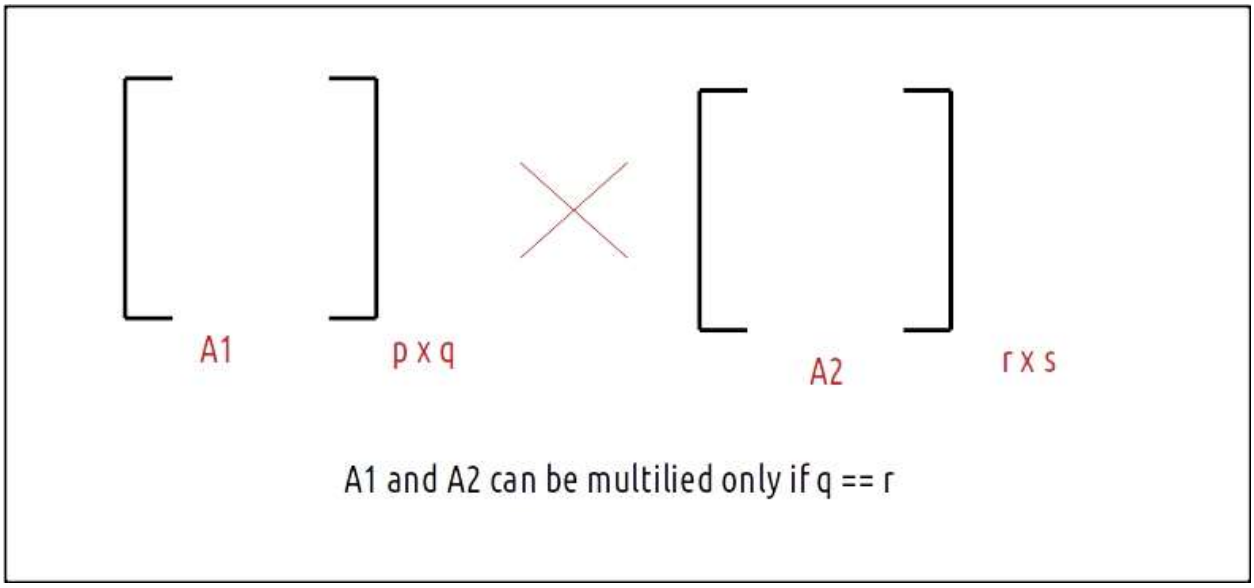
For example:



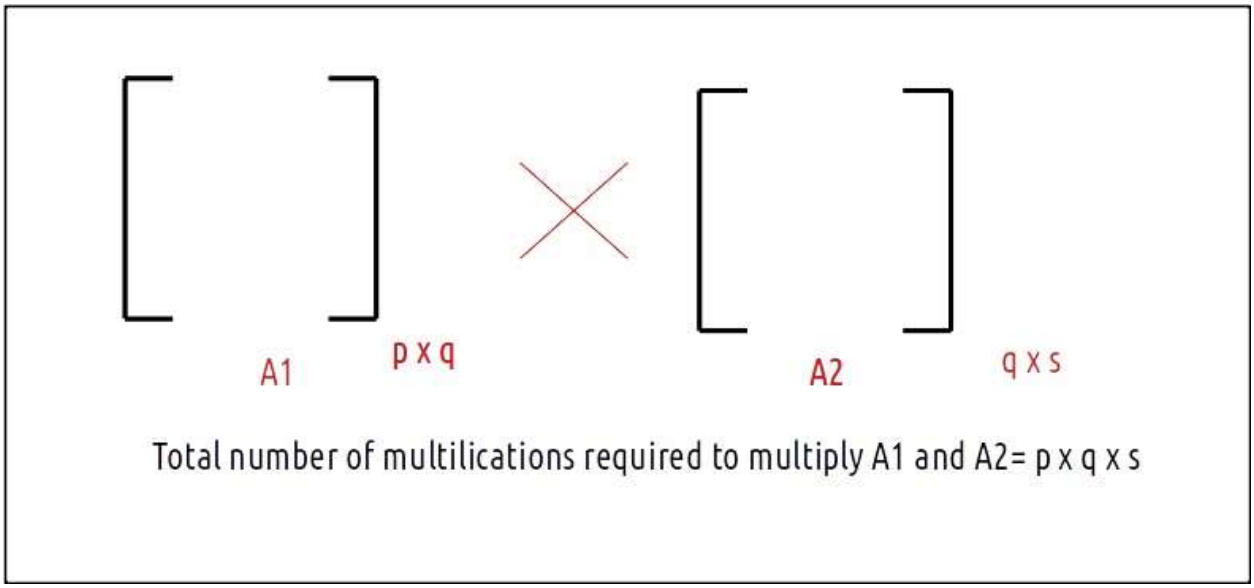
Matrix Chain Multiplication:

Let us first understand the problem of matrix chain multiplication. In order to understand the problem we need to first understand the rules of matrix multiplication:

- Two matrices A1 and A2 of dimensions [p x q] and [r x s] can only be multiplied if and only if q=r.

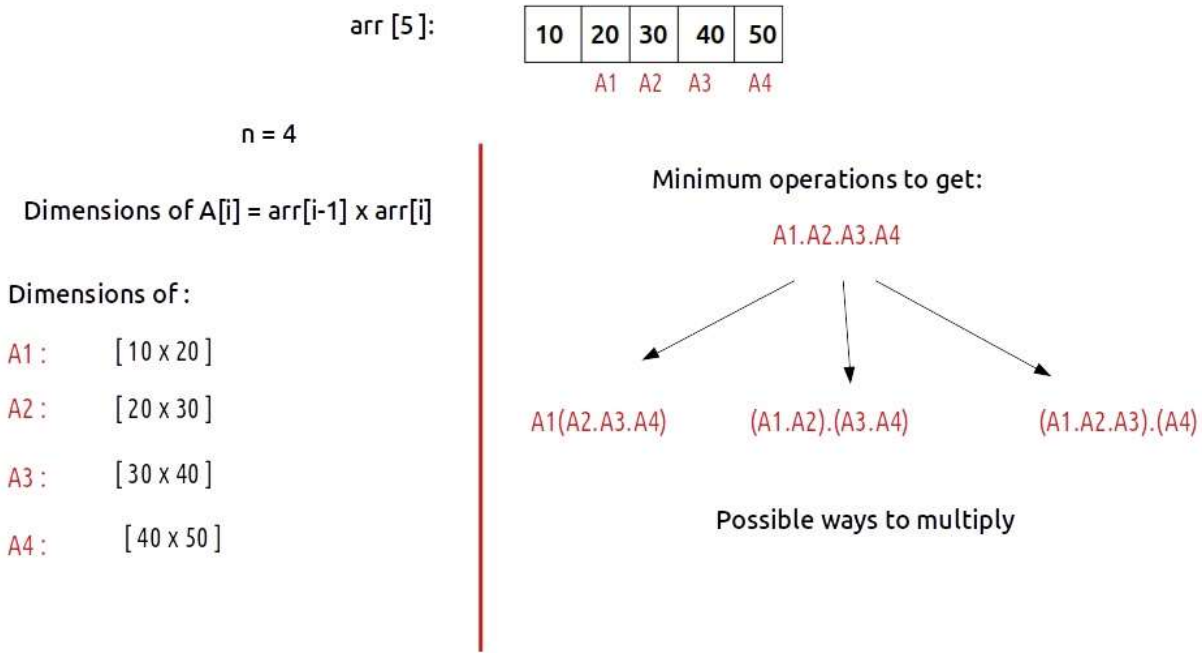


- The total number of multiplications required to multiply A1 and A2 are: p*q*s



Problem statement:

Given a chain of matrices A1,..., An denoted by an array of size n+1, find out the minimum number of operations to multiply these n matrices.



Approach:

As this problem of matrix multiplication solves one big problem (minimum operations to get A1.A2....An) and the answer varies in the order the subproblems are being solved, we can identify this problem of pattern partition DP.

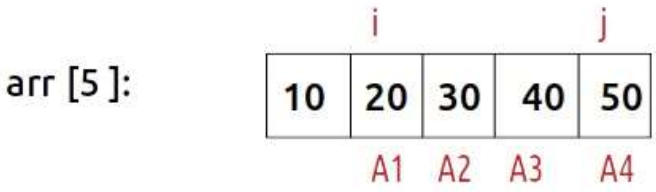
Rules to solve the problem of partition DP:

1. Start with the entire block/array and mark it with i,j. We need to find the value of f(i,j).
2. Try all partitions.
3. Run the best possible answer of the two partitions (answer that comes after dividing the problem into two subproblems and solving them recursively).

Now let us go through these rules and apply them one by one to the MCM problem.

- **Start with the entire block/array and mark it with i,j**

We are given the following array, which can be represented by i and j as shown:

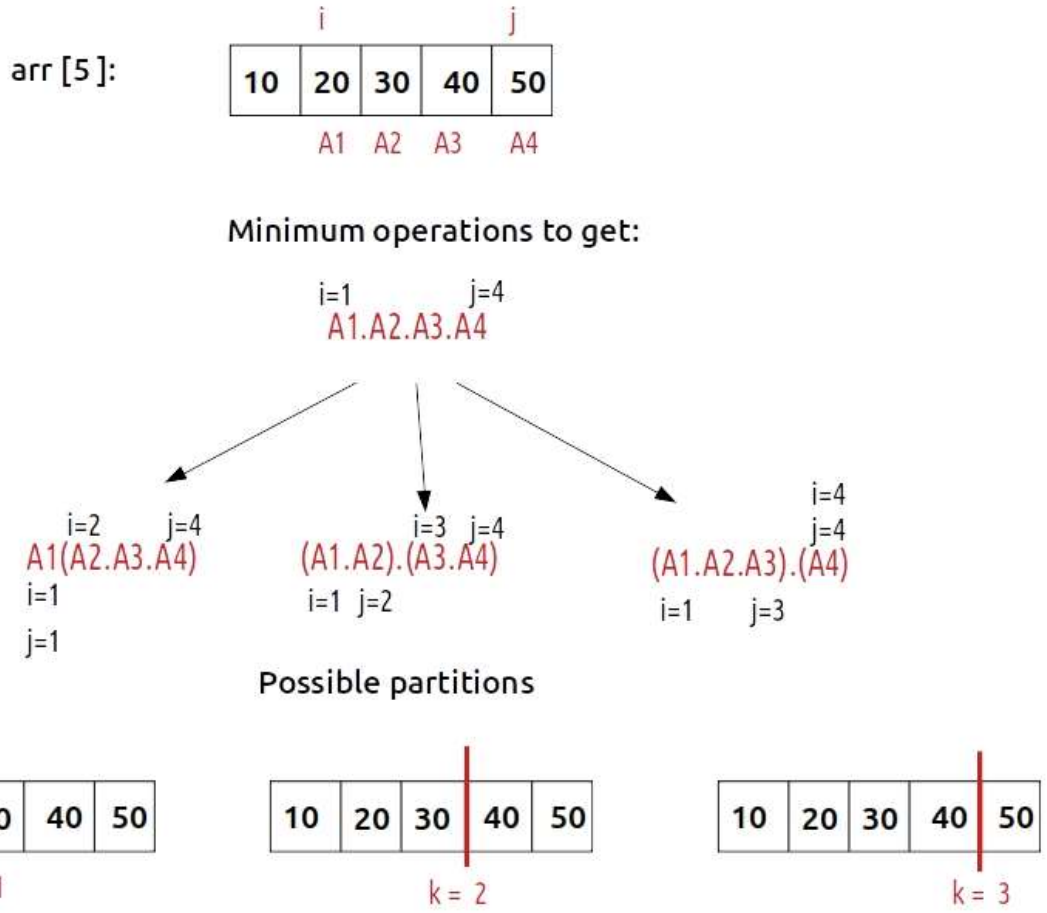


f(i,j) → Minimum operations to get the product of matrices Ai...Aj

Here, i = 1 and j=4, therefore f(i,j) represents minimum operations to get the product of A1.A2.A3.A4

- **Try all partitions:**

Let us first see all the partitions that can be made with respect to the indexing:



As we can see 3 partitions are possible, to try all three partitions, we can write a for loop starting from i and ending at j-1, (1 to 3). The two partitions will be **f(i,k) and f(k+1,j).**

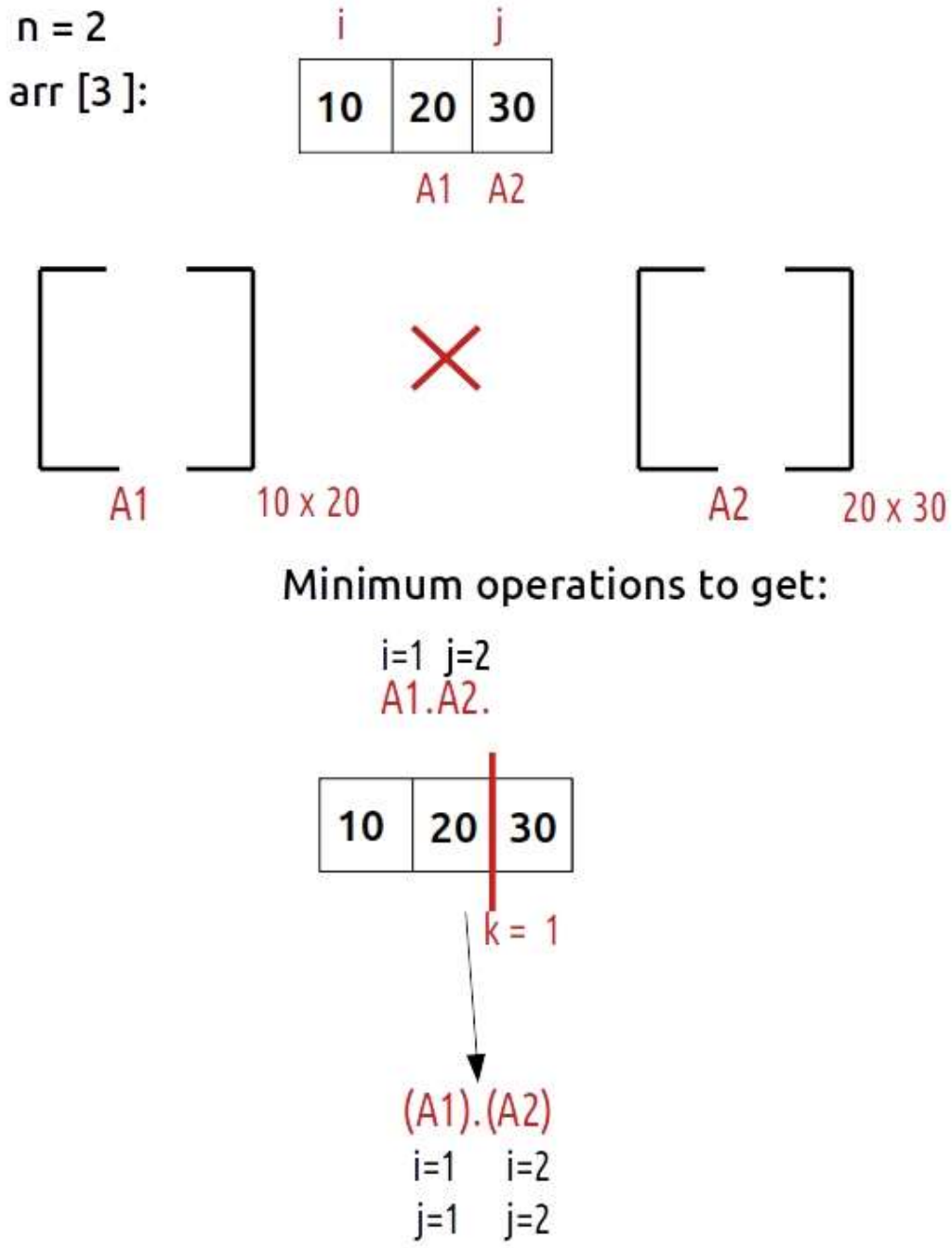
- **Run the best possible answer of the two partitions:**

Before understanding the approach, let us first understand the base condition:

```
If (i ==j), we return 0.
```

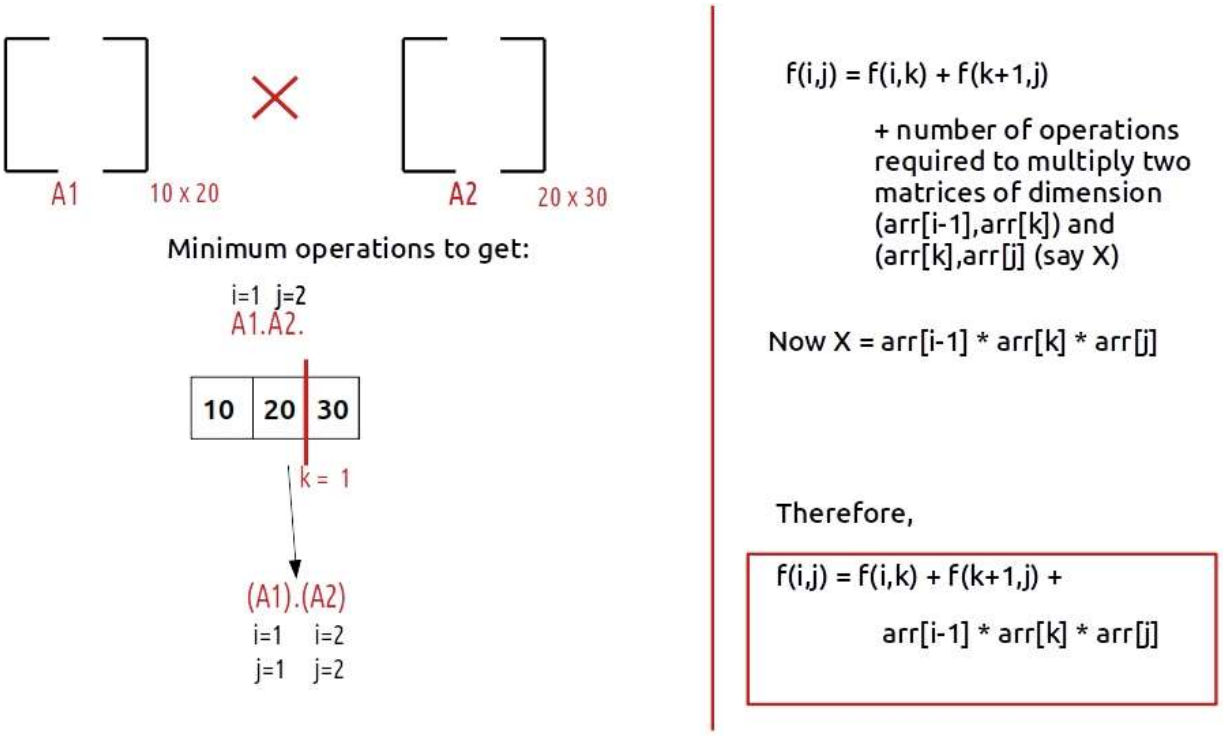
If i==j, it means we are interested in a single matrix and no operation is required to multiply it so we return 0.

Now let us understand how to find the answer in a generic case, we will take a smaller example with only two matrices in order to illustrate:



As there are only two matrices, we can say that only one partition is possible at k=1, when we do this partition, we see that the two partitions made are f(1,1) and f(2,2) (by f(i,k) and f(k+1,j) therefore we hit the base condition in both of these subcases which return 0.

Now, we know that the subproblems/partitions give us 0 operations, but we still have some work to do. The partition f(i,k) gives us a resultant matrix of dimensions [i -1 x k] and the partition f(k+1,j) gives us a resultant matrix of dimensions [k,j]. Therefore we need to count the number of operations for our two partitions (k=1) as shown:



Now, this is at one partition, in general, we need to calculate this answer of every partition and return the minimum as the answer.

To summarize:

- Represent the entire array by two indexes i and j. In this question i =1 and j = n. We need to find f(i,j).
- Maintain a mini variable to get the minimum answer.
- Set a for loop to find the answer(variable k) from i to j-1,
- In every iteration find the answer, with the formula discussed above and compare it with the mini value.
- Return mini as the answer.

Code:

- C++ Code
- Java Code

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

int f(vector<int>& arr, int i, int j){

    // base condition
    if(i == j)
        return 0;

    int mini = INT_MAX;

    // partioning loop
    for(int k = i; k<= j-1; k++){

        int ans = f(arr,i,k) + f(arr, k+1,j) + arr[i-1]*arr[k]*arr[j];

        mini = min(mini,ans);

    }

    return mini;
}

int matrixMultiplication(vector<int>& arr, int N){

    int i =1;
    int j = N-1;

    return f(arr,i,j);

}

int main() {

        vector<int> arr = {10, 20, 30, 40, 50};

        int n = arr.size();
```

```
        cout<<"The minimum number of operations is
"<<matrixMultiplication(arr,n);

        return 0;
}
```

Output:

The minimum number of operations is 38000

Steps to memoize a recursive solution:

As there are overlapping subproblems in the recursive tree, we can memoize the recursive code to reduce the time complexity.

1. Create a dp array of size [n][n]. I and j can range from 0 to n-1 so we take the size n X n.
2. We initialize the dp array to -1.
3. Whenever we want to find the answer to particular parameters (say f(i,j)), we first check whether the answer is already calculated using the dp array(i.e dp[i][j]!= -1). If yes, simply return the value from the dp array.
4. If not, then we are finding the answer for the given value for the first time, we will use the recursive relation as usual but before returning from the function, we will set dp[i][j] to the solution we get.

Code:

- C++ Code
- Java Code

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


int f(vector<int>& arr, int i, int j, vector<vector<int>>& dp){


    // base condition
    if(i == j)
```

```
        return 0;

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

    int mini = INT_MAX;

    // partioning loop
    for(int k = i; k<= j-1; k++){

        int ans = f(arr,i,k,dp) + f(arr, k+1,j,dp) + arr[i-1]*arr[k]*arr[j];

        mini = min(mini,ans);

    }

    return mini;
}


int matrixMultiplication(vector<int>& arr, int N){

    vector<vector<int>> dp(N,vector<int>(N,-1));

    int i =1;
    int j = N-1;


    return f(arr,i,j,dp);


}

int main() {

        vector<int> arr = {10, 20, 30, 40, 50};

        int n = arr.size();

        cout<<"The minimum number of operations is
"<<matrixMultiplication(arr,n);

        return 0;
}
```

Output:

The minimum number of operations is 38000

Time Complexity: O(N*N*N)

Reason: There are N*N states and we explicitly run a loop inside the function which will run for N times, therefore at max ‘N*N*N’ new problems will be solved.

Space Complexity: O(N*N) + O(N)

Reason: We are using an auxiliary recursion stack space(O(N))and a 2D array (O(N*N)).