⛓️ Dynamic Programming: Matrix Chain Multiplication

Problem Setup:

Matrices: A1 × A2 × A3 × A4

Goal: Find the parenthesization that minimizes scalar multiplications

Example: Should we compute (A1×A2)×(A3×A4) or A1×((A2×A3)×A4)?

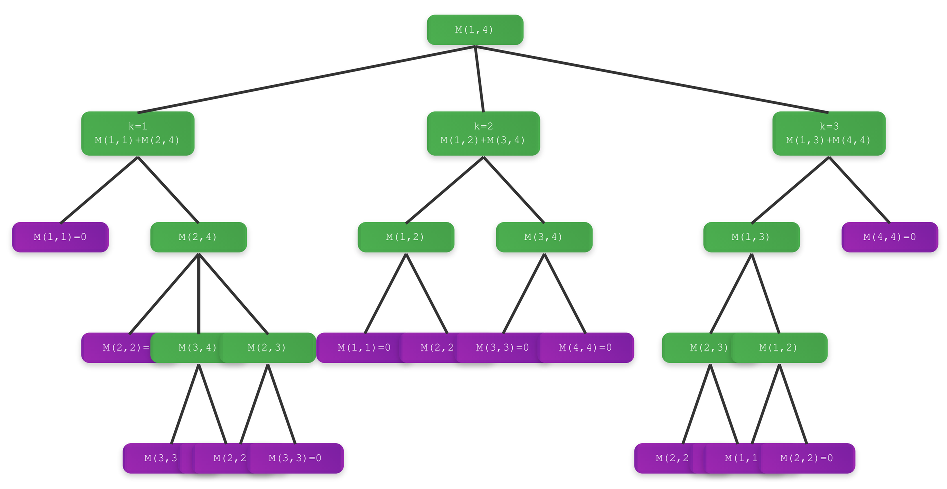
Matrix	A1	A2	A3	A4
Dimensions	2×3	3×4	4×2	2×5

Cost of multiplying matrices: For matrices of size p×q and q×r, cost = p×q×r scalar multiplications

1️⃣ Recursive Tree Structure

This shows the complete recursive call tree for solving M(1,4). Each node M(i,j) represents the minimum cost to multiply matrices Ai through Aj.

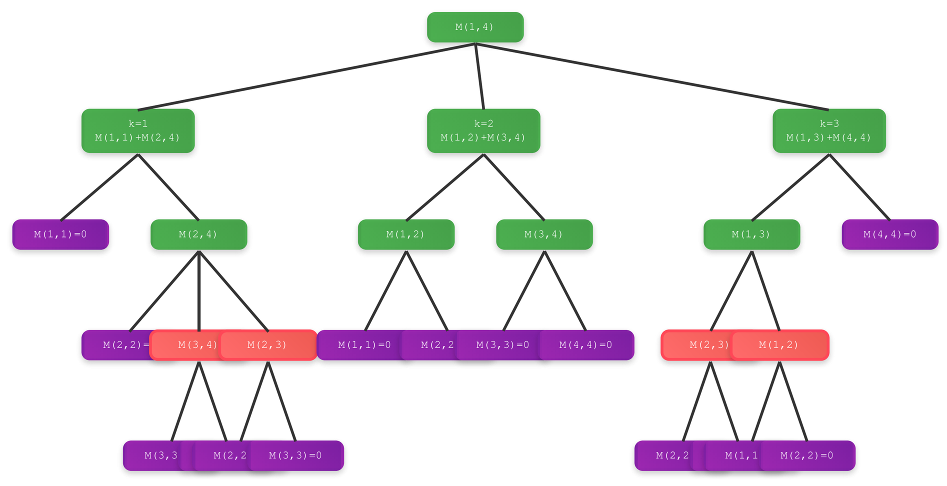
Regular Subproblems   Base Cases (i=j)



2️⃣ Overlapping Subproblems Highlighted

Red nodes show subproblems that appear multiple times in the recursion tree. Notice how M(2,3) appears in multiple branches!

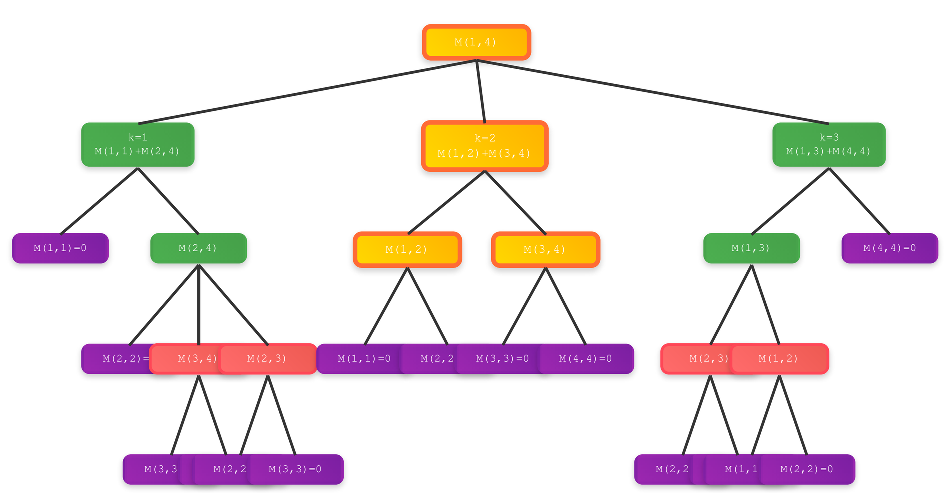
Unique Subproblems   Overlapping Subproblems   Base Cases (i=j)



3️⃣ Optimal Substructure Path

Gold nodes show one optimal path through the recursion tree. The optimal way to parenthesize A1×A2×A3×A4 uses optimal solutions for subchains.

Regular Subproblems   Optimal Path (k=2 split)   Overlapping Subproblems   Base Cases (i=j)



🔄 Overlapping Subproblems

The same subproblem M(i,j) appears multiple times in the recursive tree. For example, M(2,3) is computed in different branches when trying different split points.

Why DP helps: We can store the result of M(2,3) the first time and reuse it instead of recalculating multiple times.

🏗️ Optimal Substructure

The optimal solution contains optimal solutions to subproblems. If M(1,4) splits optimally at k=2, then M(1,2) and M(3,4) must also be optimal.

Principle of Optimality: The best way to parenthesize the entire chain uses the best ways to parenthesize the subchains.

📝 Understanding the Recursion: M(i, j)

M(i, j) = Minimum cost to multiply matrices Ai through Aj

```
Base case: M(i, i) = 0 (single matrix, no multiplication needed)
```

```
Recurrence: M(i, j) = min { M(i, k) + M(k+1, j) + p[i-1] × p[k] × p[j] }
            for all k where i ≤ k < j
```

Where p[0] = 2, p[1] = 3, p[2] = 4, p[3] = 2, p[4] = 5 (matrix dimensions)

🎯 Why Dynamic Programming Works Here:

1. Overlapping Subproblems: When computing M(1,4), we try different split points (k=1, k=2, k=3). Each split creates subproblems like M(2,3), M(1,2), etc. These same subproblems appear when computing other ranges, creating overlap.

2. Optimal Substructure: If the optimal way to compute A1×A2×A3×A4 is ((A1×A2)×(A3×A4)), then:

- The way we compute A1×A2 must be optimal for that subchain
- The way we compute A3×A4 must be optimal for that subchain

Example Calculation:

- Split at k=2: M(1,2) + M(3,4) + 2×3×2 = 24 + 40 + 12 = 76
- M(1,2) = M(1,1) + M(2,2) + 2×3×4 = 0 + 0 + 24 = 24
- M(3,4) = M(3,3) + M(4,4) + 4×2×5 = 0 + 0 + 40 = 40