# Chapter 3 Dynamic Programming

## Key Points

- Understand the concept of DP
- Master the basic elements of DP
    - Optimal substructure
    - Overlapping subproblems
- Master the steps of designing DP ALG
    - Characterize the structure of an optimal solution
    - Recursively define the value of an optimal solution
    - Compute the value of an optimal solution, typically in a **bottom-up** fashion
    - Construct an optimal solution from computed information
- Learn from the example

## General Thought of DP

- DP applies when sub-problems overlap
    - when sub-problems share common sub-subproblems
- The number of different sub-problems is usually polynomial order of magnitude
- A DP ALG solves each sub-subproblem just once and the saves its answer in a table
    - avoiding recomputing

## What problems should be solved by DP?

- Problems with the following features
    - Optimal substructure
    - Overlapping subproblems

## Basic Elements

- Optimal substructure
    - combine the optimal solutions of sub-problems
    - **bottom-up fashion**
    - analysis of optimal substructure: **contradiction**
- Overlapping subproblems
    - recursive ALG solve subproblems rather than generating new sub-problems
    - solve once and **save the answer in a table**
    - the magnitude of subproblems grows at polynomial rate as the input size increases
    - DP ALG is of polynomial time complexity

## Basic Steps

- Typically applied to optimization problems
    1. Characterize the structure of an optimal solution
    2. Recursively define the value of an optimal solution

3. Compute the value of an optimal solution, typically in a **bottom-up** fashion
4. Construct an optimal solution from computed information

# Problems & Solutions

## Matrix Chain Multiplication (MCM)

- 矩阵乘法次数：共同下标只乘一次
    - $A_{50\times 10}\cdot B_{10\times 30}:\ 50\times \underline{\mathbf{10}}\times 30$
- Brute Force: $\Omega(\frac{4^n}{n^{3/2}})$
    - Let the number of different parenthesizations of the product $A_1A_2\ldots A_n$ be $P(n)$
    - $A_1A_2\ldots A_n:\ A_1A_2\ldots A_k$ "+" $A_kA_{k+1}\ldots A_n$

$$P(n)=\begin{cases}1 & n=1\\ \sum_{k=1}^{n-1}P(k)P(n-k) & n>1\end{cases}\Rightarrow P(n)=\Omega(4^n/n^{3/2})$$

### Substructure of an Optimal Solution

- $A[i:j]:\ A_iA_{i+1}\ldots A_j$
- **Computation amounts of** $A[1:n]$ **= that of** $A[1:k]$ **+ that of** $A[k+1:n]$ **+ that of** $A[i:k]\times A[k+1:j]$
- Optimal parenthesization of the product $A[i:j]:$ separated by $A_k$ and $A_{k+1}$

$$(A_iA_{i+1}\cdots A_k)(A_kA_{k+1}\cdots A_j)$$

- In the optimal parenthesization of $A[1:n]$ ($A[i:j]$), the parenthesization of $A[1:k]$ ($A[i:k]$) and $A[k+1:n]$ ($A[k+1:j]$) are also optimal
    - proof by **contradiction**
    - can be solved in the same way
    - recursive
- Optimal structure MCM优化解包含了子问题的优化解
    - can be solved by DP

### Establishment of Recursive Relationship

- $m[i:j]:$ the least # of scalar multiplications for computing $A[i:j]$ the solution of MCM: $m[1,n]$
- When $i=j$, $A[i:j]=A_i$, then $m[i,j]=0$
- When $i<j$, according to its optimal substructure

$$m[i,j]=m[i,k]+m[k+1,j]+\mathbf{p_{i-1}p_kp_j}$$

- $m[i,j]$ is recursively defined as

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$

## Computation of Optimal Value

- Each ordered pair $(i, j)$, $1 \leq i \leq j \leq n$, corresponds to a subproblem. Then the number of subproblems are at most 每选两个数，顺序时确定的

$$\binom{n}{2} + n = \Theta(n^2)$$

- Common sub-problems are repeatedly solved
    - another feature that DP can be applied in MCM
- Code description

```
void MatrixChain(int *p, int n, int **m, int **s)
{
    for (int i = 1; i <= n; i++)
        m[i][i] = 0;
    for (int r = 2; r <= n; r++)
    {
        for (int i = 1; i <= n - r + 1; i++)
        {
            int j = i + r - 1;
            m[i][j] = m[i + 1][j] + p[i - 1] * p[i] * p[j];
            s[i][j] = i;
            for (int k = i + 1; k < j; k++)
            {
                int t = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (t < m[i][j])
                {
                    m[i][j] = t;
                    s[i][j] = k;
                }
            }
        }
    }
}
```

## Complexity Analysis

- The main computation of matrixChain depends on the triplet loops of r, i and k
- The computation amounts are $O(1)$ in the loop and there are $O(n^3)$ loops in total
- The time complexity is upper bounded by $O(n^3)$
- The space complexity is upper bounded by $O(n^2)$ 表格最多为$O(n^2)$级别