Dynamic Programming

Divide-and-Conquer

Suppose there are independent, share subproblems.

Divide-and-Conquer would solve common subproblems.

Dynamic Programming would solve each subproblem once and save.

Counterexample:

Chapter 16 Section 1

Introduction to Dynamic Programming

How to develop a Dynamic Programming algorithm

1. Construct optimal solution from computed information
2. Recursively define value of optimal solution
3. Compute value of optimal solution in a bottom-up manner
4. Construct optimal solution from computed information

Counterexample:

Chapter 16 Section 1

Introduction to Dynamic Programming

How to develop a Dynamic Programming algorithm

1. Construct optimal solution from computed information
2. Recursively define value of optimal solution
3. Compute value of optimal solution in a bottom-up manner
4. Construct optimal solution from computed information

Counterexample:

Chapter 16 Section 1

Introduction to Dynamic Programming

How to develop a Dynamic Programming algorithm

1. Construct optimal solution from computed information
2. Recursively define value of optimal solution
3. Compute value of optimal solution in a bottom-up manner
4. Construct optimal solution from computed information
### Matrix Multiplication Example

Consider a sequence of $A_1, A_2, \ldots, A_n$ matrices.

#### Outline

1. **Matrix Multiplication**
   - Number of scalar multiplications
   - Formulas for scalar multiplication
2. **Matrix Multiplication**
   - Number of scalar multiplications
   - Formulas for scalar multiplication
3. **Matrix Multiplication**
   - Number of scalar multiplications
   - Formulas for scalar multiplication
4. **Matrix Multiplication**
   - Number of scalar multiplications
   - Formulas for scalar multiplication
5. **Matrix Multiplication**
   - Number of scalar multiplications
   - Formulas for scalar multiplication
6. **Matrix Multiplication**
   - Number of scalar multiplications
   - Formulas for scalar multiplication
7. **Matrix Multiplication**
   - Number of scalar multiplications
   - Formulas for scalar multiplication
8. **Matrix Multiplication**
   - Number of scalar multiplications
   - Formulas for scalar multiplication

### Scalar Multiplication

- **Operator:** $c \cdot A$
- **Matrix:** $A$
- **Scalar:** $c$
- **Result:** $c \cdot A$

### Illustration

- **Diagram:** Matrix multiplication
- **Legend:** Dimensions, operations, etc.

### Notes

- **Comment:** Use of Java syntax. To find an optimal algorithm for a given problem, consider the following steps:
  1. Identify the key components in the algorithm.
  2. Optimize each component individually.
  3. Combine the optimized components to form the final algorithm.

- **Example:**

```java
int c = 8;
int d = 6;
int e = 5;
int f = 4;
int g = 3;
int h = 2;
int i = 1;
int j = 100;
int k = 100;
int m = 30;
int n = 20;
int p = 50;
```

### Table

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar multiplication</td>
<td>$c \cdot A$</td>
</tr>
<tr>
<td>Matrix multiplication</td>
<td>$A \cdot B$</td>
</tr>
<tr>
<td>Diagonal multiplication</td>
<td>$D \cdot D$</td>
</tr>
</tbody>
</table>

### Conclusion

Matrix multiplication is a fundamental operation in computer science, requiring careful consideration of algorithmic efficiency.
The primary objective of the multiplication problem is to minimize the number of scalar multiplications. The task is to find a sequence of matrices that minimizes the overall number of multiplications.

**Step 1:** Choose an arbitrary sequence of matrices.

**Step 2:** Check whether the chosen sequence contains optimal solutions.

**Observation:** The number of scalar multiplications is minimized when the product is calculated in a way that avoids unnecessary intermediate products.

**In this reversible solution:**

1. Compute the product by multiplying the two matrices $A_i$ and $A_{i+1}$.
2. Split the sum over the positions $A_i$ and $A_{i+1}$.
3. Compute each of the positions $v^i_n$ and $v^{i+1}_n$ with $n > 1$.

An optimal solution:

- Portrait: Two of $A_i$ and $A_{i+1}$, the matrix $A_i$.
There is to compute \( f(n) \) in total

\[
(\tau, v) = \frac{\tau}{(\tau + v)} = \frac{\tau}{\tau + v} \quad \text{(for all possible subproblems)}
\]

Then \( (\tau, v) \) is possible and \( \tau \geq \frac{v}{n} \).

How many subproblems start at \( 1 \)?

For \( A^{+} \cdot \cdot \cdot A^{+} \cdot \cdot \cdot A^{+} \cdot \cdot \cdot A^{+} \), \( n \) possible subproblems

**Problem:** Number of possible subproblems

**Problem:** Recurrence tree example: \([1..4]\)
Matrix Chain Multiplication

**Problem:**
Given a sequence of matrices, determine the most efficient way to multiply them together. The cost of multiplying two matrices A and B is given by the product of their dimensions: \( \text{cost}(A,B) = \text{dim}(A) \times \text{dim}(B) \).

**Algorithm:**
Matrix Chain Multiplication (bottom-up algorithm)

1. **Compute all \( m[i,j] \) values:**
   - For each \( i,j \) in the range of matrices, compute the minimum cost of multiplying the chain of matrices from \( A_i \) to \( A_j \).
   - \( m[1,n] = 0 \) (no cost to multiply one matrix).
2. **Optimal parentheses:**
   - Determine the position of the optimal split for the chain from \( A_1 \) to \( A_n \).

**Example:**
Given matrices:

\[
\begin{align*}
A_{10} & : 10 \times 20 \\
A_{20} & : 20 \times 5 \\
A_{30} & : 30 \times 15 \\
A_{40} & : 40 \times 20
\end{align*}
\]

**Solution:**
The optimal sequence is:

\[
A_{10} \times A_{20} \times A_{30} \times A_{40}
\]

**Cost:**
\( 5 \times 20 \times 5 + 10 \times 5 \times 15 + 15 \times 10 \times 20 = 7000 \)

**Implementation:**
- Dynamic programming approach to find the optimal sequence.
- Use a table to store intermediate results to avoid recomputation.
Optimal solution is computed with matrix-chain multiplication.

Even with matrix multiplication can be computed recursively:

1. 
2. 
3. 

where terms are used to compute terms recursively.

Initial situation: optimal solution

Step 4: Compute optimal solution using the matrix-chain multiplication order.

Does not apply to direct matrix multiplication order.

Constructing optimal solution: Example.
OVERLAPPING SUBPROBLEMS

Optimal Substructure

Notice that no overlapping subproblems were solved. To avoid solving the same subproblem multiple times, we can:

- Choose a subproblem to solve and determine optimal solutions.
- Store the solution to a subproblem for later use.
- Recursively define a larger problem by solving subproblems.

Space of subproblems must be small.

In other words, where $c(n)$ is the number of matches in column $n$.

$\text{Max}$ problem solved above by:

- Bellman's equation.
- Programmatic approach.
- A top-down (recursive) approach.

$\text{Max}$ problem solved above by:

- Bellman's equation.
- Programmatic approach.