# Dynamic Programming
## (Based on [Cormen et al. 2009])

Yih-Kuen Tsay

Department of Information Management
National Taiwan University

# Design Methods

- Greedy
  - Huffman's encoding algorithm, Dijkstra's algorithm, Prim's algorithm, etc.

# Design Methods

- Greedy
  - Huffman's encoding algorithm, Dijkstra's algorithm, Prim's algorithm, etc.
- Divide-and-Conquer
  - Binary search, merge sort, quick sort, etc.

# Design Methods

- Greedy
  - Huffman's encoding algorithm, Dijkstra's algorithm, Prim's algorithm, etc.
- Divide-and-Conquer
  - Binary search, merge sort, quick sort, etc.
- **Dynamic Programming**

# Design Methods

- Greedy
  - Huffman's encoding algorithm, Dijkstra's algorithm, Prim's algorithm, etc.
- Divide-and-Conquer
  - Binary search, merge sort, quick sort, etc.
- **Dynamic Programming**
- Branch-and-Bound
- ...

# Principles of Dynamic Programming



- Property of Optimal Substructure (Principle of Optimality):
  *An optimal solution to a problem contains optimal solutions to its subproblems.*
- A particular subproblem or subsubproblem typically recurs while one tries different decompositions of the original problem.
- To reduce running time, optimal solutions to subproblems are computed only once and stored (in an array) for subsequent uses.

# Development by Dynamic Programming

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 in a bottom-up fashion.
4. Construct an optimal solution from computed information.

# Matrix-Chain Multiplication

## Problem

*Given a chain $A_1, A_2, \cdots, A_n$ of matrices where $A_i$, $1 \leq i \leq n$, has dimension $p_{i-1} \times p_i$, fully parenthesize (i.e., find a way to evaluate) the product $A_1 A_2 \cdots A_n$ such that the number of scalar multiplications is minimum.*

- Why is dynamic programming a feasible approach?

# Matrix-Chain Multiplication

## Problem

*Given a chain $A_1, A_2, \cdots, A_n$ of matrices where $A_i$, $1 \leq i \leq n$, has dimension $p_{i-1} \times p_i$, fully parenthesize (i.e., find a way to evaluate) the product $A_1 A_2 \cdots A_n$ such that the number of scalar multiplications is minimum.*

- Why is dynamic programming a feasible approach?
- To evaluate $A_1 A_2 \cdots A_n$, one first has to evaluate $A_1 A_2 \cdots A_k$ and $A_{k+1} A_{k+2} \cdots A_n$ for some $k$ and then multiply the two resulting matrices.
- An optimal way for evaluating $A_1 A_2 \cdots A_n$ must contain optimal ways for evaluating $A_1 A_2 \cdots A_k$ and $A_{k+1} A_{k+2} \cdots A_n$ for some $k$.

# Matrix-Chain Multiplication (cont.)

Let $m[i,j]$ be the minimum number of scalar multiplications needed to compute $A_i A_{i+1} \cdots A_j$, where $1 \leq i \leq j \leq n$.

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \leq k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j \end{cases}$$

# Matrix-Chain Multiplication (cont.)

**Algorithm Matrix_Chain_Order**$(n, p)$;
**begin**
    **for** $i := 1$ to $n$ **do**
        $m[i, i] := 0$;
    **for** $l := 2$ to $n$ **do** $\{$ $l$ is the chain length $\}$
        **for** $i := 1$ to $(n - l + 1)$ **do**
            $j := i + l - 1$;
            $m[i, j] := \infty$;
            **for** $k := i$ to $(j - 1)$ **do**
                **if** $m[i, k] + m[k + 1, j] + p[i - 1]p[k]p[j] < m[i, j]$ **then**
                    $m[i, j] := m[i, k] + m[k + 1, j] + p[i - 1]p[k]p[j]$
**end**

# Recursive Implementation

**Algorithm Recursive_Matrix_Chain**$(p, i, j)$;
**begin**
    **if** $i = j$ **then return** $0$;
    $m[i, j] := \infty$;
    **for** $k := i$ to $(j - 1)$ **do**
        $q := Recursive\_Matrix\_Chain(p, i, k) +$
            $Recursive\_Matrix\_Chain(p, k + 1, j) + p[i - 1]p[k]p[j]$;
        **if** $q < m[i, j]$ **then**
            $m[i, j] := q$;
    **return** $m[i, j]$
**end**

# Recursive Implementation (cont.)



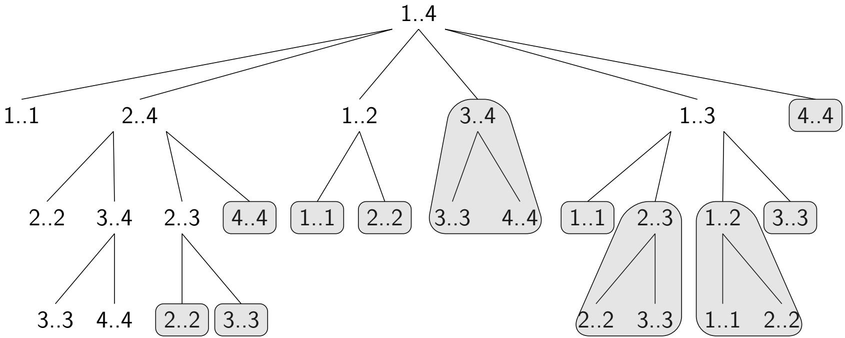


Figure: The recursion tree for the computation of **Recursive_Matrix_Chain**$(p, 1, 4)$. The computations performed in a shaded subtree are replaced by a table lookup.

Source: redrawn from [Cormen *et al.* 2006, Figure 15.5].

# Recursion with Memoization

**Algorithm Memoized_Matrix_Chain**$(n, p)$;
**begin**
    **for** $i := 1$ to $n$ **do**
        **for** $j := i$ to $n$ **do**
            $m[i, j] := \infty$;
    **return** *Lookup_Matrix_Chain*$(p, i, n)$
**end**

# Recursion with Memoization (cont.)

**Function Lookup_Matrix_Chain**$(p, i, j)$;
**begin**
    **if** $m[i,j] < \infty$ **then return** $m[i,j]$;
    **if** $i = j$ **then**
        $m[i,j] := 0$;
    **else**
        **for** $k := i$ to $(j-1)$ **do**
            $q := Lookup\_Matrix\_Chain(p, i, k) +$
                $Lookup\_Matrix\_Chain(p, k+1, j) + p[i-1]p[k]p[j]$;
            **if** $q < m[i,j]$ **then**
                $m[i,j] := q$;
    **return** $m[i,j]$
**end**

# Single-Source Shortest Paths

## Problem

*Given a weighted directed graph $G = (V, E)$ with no negative-weight cycles and a vertex $v$, find (the lengths of) the shortest paths from $v$ to all other vertices.*

- Notice that edges with negative weights are permitted; so, Dijkstra's algorithm does not work here.
- A shortest path from $v$ to any other vertex $u$ contains at most $n - 1$ edges.
- A shortest path from $v$ to $u$ with at most $k$ $(> 1)$ edges is either (1) a known shortest path from $v$ to $u$ with at most $k - 1$ edges or (2) composed of a shortest path from $v$ to $u'$ with at most $k - 1$ edges and the edge from $u'$ to $u$, for some $u'$.

# Single-Source Shortest Paths (cont.)

Denote by $D^l(u)$ the length of a shortest path from $v$ to $u$ containing *at most* $l$ edges; particularly, $D^{n-1}(u)$ is the length of a shortest path from $v$ to $u$ (with no restrictions).

$$D^1(u) = \begin{cases} length(v,u) & \text{if } (v,u) \in E \\ 0 & \text{if } u = v \\ \infty & \text{otherwise} \end{cases}$$

$$\begin{aligned} D^l(u) = \; & \min\{D^{l-1}(u), \min_{(u',u)\in E}\{D^{l-1}(u') + length(u',u)\}\}, \\ & 2 \leq l \leq n-1 \end{aligned}$$

# Single-Source Shortest Paths (cont.)

**Algorithm Single_Source_Shortest_Paths**(*length*);
**begin**
    $D[v] := 0$;
    **for** all $u \neq v$ **do**
        **if** $(v, u) \in E$ **then**
            $D[u] := length(v, u)$
        **else** $D[u] := \infty$;
    **for** $k := 2$ to $n - 1$ **do**
        **for** all $u \neq v$ **do**
            **for** all $u'$ such $(u', u) \in E$ **do**
                **if** $D[u'] + length[u', u] < D[u]$ **then**
                    $D[u] := D[u'] + length[u', u]$
**end**

# All-Pairs Shortest Paths

## Problem

*Given a weighted directed graph $G = (V, E)$ with no negative-weight cycles, find (the lengths of) the shortest paths between all pairs of vertices.*

- Consider a shortest path from $v_i$ to $v_j$ and an arbitrary intermediate vertex $v_k$ (if any) on this path.
- The subpath from $v_i$ to $v_k$ must also be a shortest path from $v_i$ to $v_k$; analogously for the subpath from $v_k$ to $v_j$.

# All-Pairs Shortest Paths (cont.)

Index the vertices from 1 through $n$.
Denote by $W^k(i,j)$ the length of a shortest path from $v_i$ to $v_j$ going through no vertex of index greater than $k$, where $1 \leq i,j \leq n$ and $0 \leq k \leq n$; particularly, $W^n(i,j)$ is the length of a shortest path from $v_i$ to $v_j$.

$$W^0(i,j) = \begin{cases} \mathit{length}(i,j) & \text{if } (i,j) \in E \\ 0 & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}$$

$$W^k(i,j) = \min\{W^{k-1}(i,j),\ W^{k-1}(i,k) + W^{k-1}(k,j)\}, 1 \leq k \leq n$$

# All-Pairs Shortest Paths (cont.)

IM NTU

```
Algorithm All_Pairs_Shortest_Paths(length);
begin
    for i := 1 to n do
        for j := 1 to n do
            if (i, j) ∈ E then W[i, j] := length(i, j)
            else W[i, j] := ∞;
    for i := 1 to n do W[i, i] := 0;
    for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                if W[i, k] + W[k, j] < W[i, j] then
                    W[i, j] := W[i, k] + W[k, j]
end
```