2 STRATEGY DIVIDE-AND-CONQUER

- Divide Problem P into smaller problem $P_1, P_2, \ldots, P_k$.
- Solve problems $P_1, P_2, \ldots, P_k$ to obtain solutions $S_1, S_2, \ldots, S_k$
- Combine solution $S_1, S_2, \ldots, S_k$ to get the final solution.

Subproblems $P_1, P_2, \ldots, P_k$ are solved recursively using divide-and-conquer.

Examples: Quicksort and mergesort.

3 STRATEGY GREEDY

Solution $\leftarrow \Phi$

for $i \leftarrow 1$ to n do

SELECT the next input x.

If $\{x\} \cup$ Solution is **FEASIBLE** then

solution $\leftarrow$ **COMBINE**(Solution, x)

- **SELECT** appropriately finds the next input to be considered.
- A **FEASIBLE** solution satisfies the constraints required for the output.
- **COMBINE** enlarges the current solution to include a new input.

Examples: Max finding, Selection Sort, and Kruskal's Smallest Edge First algorithm for Minimum Spanning Tree.

4 STRATEGY DYNAMIC PROGRAMMING

- Fibonacci Numbers:

$$F_n = F_{n-1} + F_{n-2}$$

$F_1 = F_0 = 1.$

— Recursive solution requires exponential time: has **overlapping subproblems.**

— **Bottom-up** iterative solution is linear – **compute once, store, and use many times.**

4.1 Matrix Sequence Multiplication

- **eg. 1:**

  $$A_{30x1}\ X\ B_{1x40}\ X\ C_{40x10}\ X\ D_{10x25}\ X\ E_{25x1}$$

  — Left to right evaluation requires more than 12K multiplications.

  — $(A\ X\ (\ (B\ X\ C)\ X\ (D\ X\ E)\ )\ )$ needs only 690 multiplications (minimum needed).

  — **Greedy Algorithm: Largest Common Dimension First**

- **eg. 2:**

  $$A_{1x2}\ X\ B_{2x3}\ X\ C_{3x4}\ X\ D_{4x5}\ X\ E_{5x6}$$

  — Largest Common Dimension First imposes following order:

  $$(A\ X\ (\ B\ X\ (C\ X\ (D\ X\ E)\ )\ )\ )$$

  which needs 240 multiplications.

— Best order:

$$((((A\ X\ B)\ X\ C)\ X\ D)\ X\ E)$$

which needs 68 multiplications.

— **Another Greedy Algorithm: Smallest Common Dimension First** but did not work for eg. 1.

4.2 Divide and Conquer Solution

Input:

$$\begin{array}{ccccccc} A_1 & * & A_2 & \dots & * & A_n \\ d_0 * d_1 & & d_1 * d_2 & \dots & & d_{n-1} * d_n \end{array}$$

Output: A paranthesization of the input sequence resulting in minimum number of multiplications needed to multiply the n matrices.

- **Subgoal:** Ignore Structure of Output (order of parenthesization), focus on obtaining a numerical solution (minimum number of multiplications)
- Define **M[i,j]** = the minimum number of multiplications needed to compute

  $$A_i * A_{i+1} * \cdots * A_j$$

  for $i \leq j \leq n$
- Subgoal is to obtain $M[1, n]$.

e.g. For,

$$A1_{30x1}\ X\ A2_{1x40}\ X\ A3_{40x10}\ X\ A4_{10x25}\ X\ A5_{25x1}$$

$$M[1,1] = 0, M[1,2] = 1200, M[1,5] = 690$$

4.3 Recursive Formulation of $M[i, j]$

- A1 X A2 = (A1) X (A2)

  — Partition at k=1: Subproblems (A1) and (A2)

  — cost of (A1) is $M[1, 1]$ and that of (A2) is $M[2, 2]$

  — cost of combining (A1) and (A2) into one is $d_0 * d_1 * d2$.

  — $M[1, 2] = M[1, 1] + M[2, 2] + d_0 * d_1 * d2$.

  — $M[1, 2] = 0 + 0 + 1200 = 1200$.

- A2 X A3 X A4 = (A2) X (A3 X A4) *(k=2)*
  Or, = (A2 X A3) X A4 *(k=3)*.

  — $k = 2$: cost$= M[2, 2] + M[3, 4] + d1 * d2 * d4$

  — $k = 3$: cost$= M[2, 3] + M[4, 4] + d1 * d3 * d4$

  — $M[2, 4] = min(M[2, 2] + M[3, 4] + d1 * d2 * d4, M[2, 3] + M[4, 4] + d1 * d3 * d4)$

In short, $M[2,4] = \min_{2 \leq k \leq 3}(M[2,k] + M[k,4] + d_1 d_k d_4)$

- A2 X A3 X A4 X A5
  = (A2) X (A3 X A4 X A5) *(k=2)*
  Or, = (A2 X A3) X (A4 X A5) *(k=3)*
  Or, = (A2 X A3 X A4) X (A5) *(k=4)*

  $M[2,5] = (M[2,2]+M[3,5]+d_1 d_2 d_5, M[2,3]+M[4,5]+d_1 d_3 d_5, M[2,4]+M[5,5]+d_1 d_4 d_5)$

- In general, by factoring $(A_i * A_{i+1} * \cdots * A_j)$ at kth index position into $(A_i * A_{i+1} * \cdots * A_k)$ and $(A_{k+1} * \cdots * A_j)$ need $M[i,k] + M[k+1,j]$ multiplications and creates matrices of dimensions $d_{i-1} * d_k$ and $d_k * d_j$. These two matrices need additional $d_{i-1} * d_k * d_j$ multiplications to combine.

4.4 Recursive Formula and Time Taken

- Recursively,
  $M[i,j] =$
  $$\min_{i \le k \le j-1} \Big(M[i,k] + M[k+1,j] + d_{i-1} d_k d_j \Big)$$
  $$M(i,i) = 0$$
- **Optimal Substructure**

- We can recursively solve for
$M[1,n] =$
$\min_{1 \le k \le n-1} (M[1,k] + M[k+1,n] + d_0 d_k d_n)$
$= min[M[1,1] + M[2,n] + d_0 d_1 d_n,$
$M[1,2] + M[3,n] + d_0 d_2 d_n,$
$M[1,3] + M[4,n] + d_0 d_3 d_n,$
$\vdots$
$M[1,n-1] + M[n,n] + d_0 d_{n-1} d_n$

- Time Complexity:

$$
\begin{aligned}
T_n = n \; & + T_1 + T_{n-1} \\
& + T_2 + T_{n-2} \\
& + T_3 + T_{n-3} \\
& \vdots \\
& + T_{n-2} + T_2 \\
& + T_{n-1} + T_1
\end{aligned}
$$

$$\begin{aligned} T_n &= n + 2T_1 + 2T_2 + \cdots + 2T_{n-1} \quad (1) \\ T_{n-1} &= n - 1 + 2T_1 + 2T_2 + \cdots + 2T_{n-2} \quad (2) \end{aligned}$$

Subtracting (I)-(II) yields

$$\begin{aligned} T_n - T_{n-1} &= 1 + 2T_{n-1} \\ T_n &= 1 + 3T_{n-1} \\ &= 1 + 3(1 + 3T_{n-2}) \\ T_n &= 1 + 3 + 3^2 + 3^3 + \cdots + 3^{n-1}T_1 \\ &= 1 + 3 + 3^2 + 3^3 + \cdots + 3^{n-2} \end{aligned}$$

- Recursive Solution is exponential time $\Omega(3^{n-2})$
- Space $O(n)$ stack depth.
- **Overlapping subproblems:** e.g. Recursion tree for $M[1,4]$.
  26 recursive calls for just 10 subproblems
  $M[1,1], M[2,2], M[3,3], M4,4], M[1,2], M[2,3], M$

So we turn to dynamic Programming,

- the same formulation
- approach the problem bottom-to-top
- find a suitable table to store the sub-solutions.

How many sub-solutions do we have?

$$
\begin{array}{ll}
M[1,1], M[2,2], M[3,3] \cdots, M[n,n] & n \\
M[1,2], M[2,3], \cdots, M[n-1,n] & n-1 \\
M[1,3], M[2,4], \cdots, M[n-2,n] & n-2 \\
\vdots & \\
M[1,n-1], M[2,n] & 2 \\
M[1,n] & 1 \\
 & \frac{n(n-1)}{2}
\end{array}
$$

$\Rightarrow$ we need $O(n^2)$ space

M,Factor: Matrix

for $i \leftarrow 1$ to n do $M[i,i] \leftarrow 0$
/* main diagonal*/

for diagonal $\leftarrow 1$ to $n-1$ do

for $i \leftarrow 1$ to $n-$diagonal do

$j \;=\; i +$ diagonal
$M[i,j] \;= \min_{i \leq k \leq j-1}(M[i,k]+M[k+1,j]$
$+d_{i-1}d_k d_j)$
Factor$[i,j] = k$ that gave the minimum value for $M[i,j]$.

endfor

endfor

4.6 Work out

$A1_{30x1}\ X\ A2_{1x40}\ X\ A3_{40x10}\ X\ A4_{10x25}\ X\ A5_{25x1}$

$M[1,2] = \min_{1 \leq k \leq 1}[M[i,k] + M[k+1,j] + d_{i-1}d_kd_j]$
$= min[M[1,1] + M[2,2] + d_0d_1d_2]$
$= 0 + 0 + 30 * 1 * 40$
$= 1200$

$M[1,3] = \min_{1 \leq k \leq 3-1}[M[i,k] + M[k+1,j] + d_{i-1}d_kd_j]$
$= min[M[1,1] + M[2,3] + d_0d_1d_3,$
$M[1,2] + M[3,3] + d_0d_2d_3]$
$= min[0 + 400 + 30 * 1 * 10,$
$1200 + 0 + 30 * 40 * 10]$
$= min[700, 12000 + 1200]$
$= 700$

$$M[2,4] = min[M[i,k] + M[k+1,j] + d_{i-1}d_kd_j]$$
$$2 \le k \le 3$$
$$= min[M[2,2] + M[3,4] + d_1d_2d_4,$$
$$M[2,3] + M[4,4] + d_1d_3d_4]$$
$$= min[0 + 10000 + 1 * 40 * 25, 400 + 0 + 1 * 10 * 25]$$
$$= min[10100, 650] = 650$$

$$M[3,5] = min[M[i,k] + M[k+1,j] + d_{i-1}d_kd_j]$$
$$3 \le k \le 4$$
$$= min[M[3,3] + M[4,5] + d_2d_3d_5,$$
$$M[3,4] + M[5,5] + d_2d_4d_5]$$
$$= min[0 + 250 + 40 * 10 * 1, 10000 + 0 + 40 * 25 * 1]$$
$$= min[650, -] = 650$$

$$M[1,4] = min[M[1,1] + M[2,4] + d_0d_1d_4,$$
$$M[1,2] + M[3,4] + d_0d_2d_4,$$
$$M[1,3] + M[4,4] + d_0d_3d_4]$$
$$= min[0 + 650 + 30 * 1 * 25,$$
$$1200 + 10000 + 30 * 40 * 25,$$
$$700 + 0 + 30 * 10 * 25]$$
$$= min[1400, -, -] = 1400$$

5 DYNAMIC PROGRAMMING REQUIREMENTS

Requirements: a) Optimal Substructure

b) Overlapping subproblem

Steps: 1) Characterize the structure of an optimal solution

2) Formulate a recursive solution

3) Compute the value of *an* opt. solution bottom-up. (get value rather than the structure)

4) Construct an optimal solution (structure) from computed infomation.

Memoization: Top-down, compute and store first time, reuse subsequent times.

5.1 Observation 1: Optimal Substructure

The optimal solution containings optimal subsolutions.

Recursion Tree (do not wide yet)

Depth? $\theta(m+n)$

outdegree 3 $\Rightarrow$ number of nodes $\approx$ amount of work in recursive calls is $\theta(3^{m+n})$

Observation 2: Overlapping Subproblems

- wide some repeated problems, as above.
- a few problems, but many recursive instances unlike good divide-and-conquer where problems are independent.
- LCS has an mn distinct problems.

5.3 Memorize

(to deal with overlapping problems)

- after computing solution to a subproblem, sort in a table. Subsequent call-do table lookup.
- Time $O(mn)$

  — each problem is solved once and used twice.

  — see in figure for $1, 2$ or $2, 3$

  — might not need to solve all subproblem, only those needed.

5.4 Dynamic Programming Implimentation

- Compute table bottom-up instead of starting at (m, n), start at $(1, 1)$

- Demonstrate algorithm

  — time = $\theta(m, n)$

  — space = $\theta(min(m, n))$

- Initialize top row & left column to 0

- Produce from top row, left to right, $x[i] = y[j]$ fill diagonal neighbor+1 & chaw . slre fill max of the other neighbors.