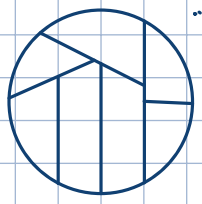


"Divide & Conquer" Max/Min-imization problems

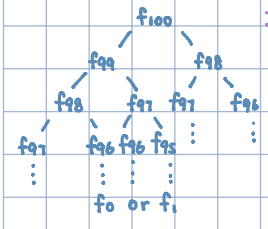
Break the big problems into small ones and combine solution together.

Two characteristics

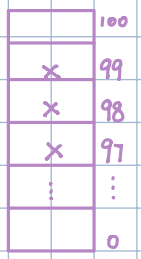
1. Optimal substructure: optimal solution to the "big problem" entails optimal solutions to similar-type subproblems (like greedy)
2. Overlapping subproblems: As you break the large problem into smaller, often you need to re-calculate same thing. Store intermediate solutions and reuse them (memorization)

Example 1: Fibonacci #s: $f_n = f_{n-1} + f_{n-2}$, where $F_0 = 0$, $F_1 = 1$, $F_{100} = ?$

Recurssion way: a lot of double calculation



Instead, save all the results in an array:



save after the 1st calculation,

fetch when needed

Matrix Parentheziation

Need multiply n-matrices $A_1, A_2, A_3 \dots A_n$ in this order.

How to parenthesize them to minimize the # of scalar multiplications?

Assume matrix A_i has dimensions $P_{i-1} \times P_i$

Motivational Example

A_1 (10×100) A_2 (100×5) A_3 (5×50)

$(A_1A_2)A_3$ (10×5): $= 10 \times 100 \times 5 + 10 \times 5 \times 50$

$= 5000 + 2500 = 7500$ multiplications

$A_1(A_2A_3)$ (100×50): $= 100 \times 5 \times 50 + 10 \times 100 \times 50$

$= 25000 + 50000 = 75000$ multiplications

Computing all parenthesizations exhaustively

$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k) = \Omega\left(\frac{4^n}{n^{3/2}}\right)$ exponential! ⇒ won't work well

key idea

An optimal parenthesization for $A_1, A_2, \dots A_n$ involves the index k that minimizes # of multiplications

$A_1 \; A_2 \dots A_k$ (# of multi) + $A_{k+1} \dots A_n$ (# of multi)

i.e: $A_1 \; A_2 \; A_3 \; [A_4((A_5 \; A_6)(A_7 \; A_8))]$

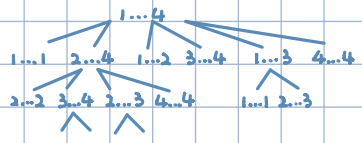
Let $m[i,j]$ = {min # of scalar multiplications to calculate $A_i \dots A_j$}

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

($P_{i-1}P_kP_j$: # of multiplications to find the product $A_{i..k} \times A_{k+1..j}$)

Priciple 1: optimal substructure problem

Naive recursive implementation of previous recurrance: (4 matrices)



We recalculate results!

Solution?

Memorization (store intermediate results)

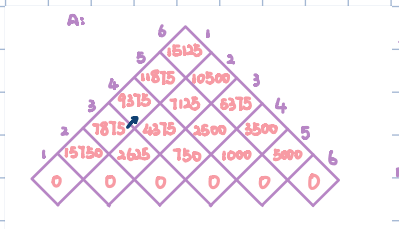
Use a 2-dimensional table to memorize

Example:

Calculate $A_1, A_2, A_3, \dots A_6$

A_1: 30×35
A_2: 35×15
A_3: 15×5
A_4: 5×10
A_5: 10×20
A_6: 20×25

Only need to use half of the two dimensional array:
Cell A[i,j] contains min multiplication for $A_{i...j}$ product $\Rightarrow$ m[i,j]
fill horizontally, down to up



arrow shows where the data came from

$m(1,2) = 30 \times 35 \times 15 = 15750$
$m(1,3) = A_1A_2A_3 = (A_1A_2)A_3 = 15750 + 30 \times 15 \times 5$ (already calculate)
or $= A_1(A_2A_3) = 2625 + 30 \times 35 \times 5 = 7875$ ✓ smaller
$m(1,4) = A_1A_2A_3A_4 = A_1(A_2A_3A_4) = 4375 + 30 \times 35 \times 10$
$= (A_1A_2)(A_3A_4) = 15750 + 750 + 30 \times 15 \times 10$
$= (A_1A_2A_3)A_4 = 7875 + 5 \times 10 \times 30 = 9375$ ✓ smaller

Run time

$O(n^2)$ squares $\times$ $O(n)$ to run index
$= O(n^3)$ times
(Naive: $\Omega(4^n/n^{3/2})$)

Longest Common Subsequence (LCS)

Problem statement

Given two strings $X = x_1 x_2 \dots x_m$ Find a subsequence (not necessarily consecutive but in the original order) which is common to both strings.
$Y = y_1 y_2 \dots y_n$

i.e: springtime / pioneer $\Rightarrow 4$ hello / eloquent $\Rightarrow 3$ find sequence in DNA

Naive: $\Theta(n2^m)$ if $m \le n$

Theorem: If $Z = z_1 \dots z_k$ is a LCS of X & Y

1. If $x_m = y_n$ then $x_m = y_n = z_k$ and Z_{k-1} is a LCS of X_{m-1} & Y_{n-1}
2. If $x_m \neq y_n$ then $z_k \neq x_m$ implies that Z is a LCS of X_{m-1} & Y
3. If $x_m \neq y_n$ then $z_k \neq y_n$ implies that Z is a LCS of X & Y_{n-1} $(y_1 y_2 \dots y_{n-1})$

Proof of 1:

① ATaC the $x_m = y_n$ but $x_m = y_n \neq z_k$ or Z_{k-1} is not a LCS of X_{m-1} & Y_{n-1}
if $x_m = y_n \neq z_k$ then add $x_m (= y_n)$ to Z and create a "longer" LCS, a contradiction as Z is a LCS

② ATaC the $x_m = y_n = z_k$ but Z_{k-1} is not a LCS of X_{m-1} & Y_{n-1}, let W be one:
$|W| \cup x_m > |Z_{k-1}| \cup x_m$
longer LCS > "Z" $\Rightarrow$ contradiction

Theorem builds the recursive solution:

c[i,j] = length of LCS of X_i & Y_j

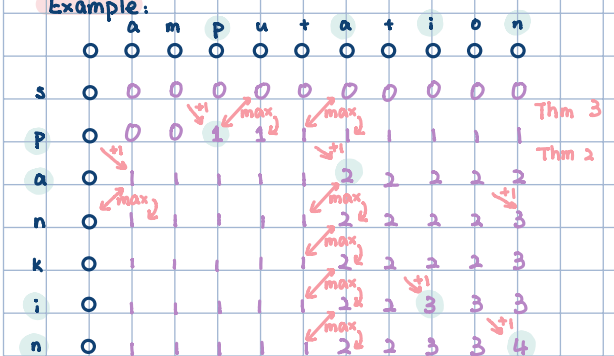
$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } x_i = y_j \quad \text{Application of Thm 1} \\ \max\{c[i-1,j], c[i,j-1]\} & \text{if } x_i \neq y_j \quad \text{Application of Thm 2,3} \end{cases}$$

	x	y
a	i-1, j-1	i-1, j
b	i, j-1	i, j

Naive implementation

Consider string "bozo" & "bat"
c[4,3]
$x_i \neq y_j$

Example:



g 0 1 1 1 1 1 2 2 3 3 4 path

Run time: $O(n \cdot m)$

Naive: $\Theta(n \cdot 2^m)$