Tuesday March 31, 2015

Knapsack problem:

Given n objects with:

integer weights $w_1, \ldots, w_n$

& integer values $v_1, \ldots, v_n$

& total capacity B

Goal: find subset S of objects which

maximizes $\sum_{i \in S} v_i$

subject to $\sum_{i \in S} w_i \leq B$

Last class: $O(nB)$ time algorithm for version:

at most one copy of each object

Today: unlimited supply of each object.

Defining subproblems:

First try: $K(i)$ = max value achievable using subset of objects $1, \ldots, i$

But like last class we need to know how much weight is available.

Try like last class:

$K(i,b)$ = max value achievable using a subset of objects $1,\dots,i$ & capacity b

Now we can't just decide whether to include object i or not.

we need to decide how many copies to include.

for capacity b, can include $\leq \frac{b}{w_i}$ copies.

Try $l = 0 \to \lfloor \frac{b}{w_i} \rfloor$:

- if we add l copies of object i

then: gain value $l v_i$

& then have $b - l w_i$ weight available

hence, $l v_i + K(i-1, b-l w_i)$

Therefore,

$$K(i,b) = \max_{0 \leq l \leq \lfloor b/w_i \rfloor} \{ K(i-1, b-l w_i), + l v_i \}$$

But too slow – for big b, too many l to try.

Simpler solution?

Unlimited supply so why remember which objects we've used so far.

Drop index i.

Let $K(b)$ = max value achievable with capacity b (all objects $1, \ldots, n$ are allowed)

To get the recurrence for $K(b)$,

try all possibilities for last object added.

Let l be the last object.

Need that $w_l \le b$.

If so, we gain v_l in value

& we have weight $b - w_l$ left.

hence, $K(b-w_l) + v_l$

Therefore,

$$K(b) = \max_{l} \{ K(b-w_l) + v_l : 1 \le l \le n, w_l \le b \}$$

In words, $K(b)$ equals the max of $K(b-w_l)$ where we maximize l but l is constrained to $1 \le l \le n$ & $w_l \le b$.

K is now a 1-dimensional table

$K =$

0	1	. . . .	B

fill from $b = 0 \rightarrow B$

KnapsackwithRepeat$(w_1, \ldots, w_n, v_1, \ldots, v_n, B)$

for $b = 0 \rightarrow B$

  $K(b) = 0$

  for $\ell = 1 \rightarrow n$

    if $w_\ell \leq b$ & $K(b) < K(b - w_\ell) + v_\ell$

      then $K(b) = K(b - w_\ell) + v_\ell$

Return$(K(B))$

Running time: $O(n \cdot B)$

Chain Matrix Multiply:

Example: for 4 matrices A, B, C, D
we want to compute $A \times B \times C \times D$ most efficiently.

Say A is 50×20 size,
B is 20×1,
C is 1×10,
D is 10×100.

Since matrix multiplication is associative we can compute:

$((A \times B) \times C) \times D)$ — this is standard way

or $(A \times B) \times (C \times D)$

or $(A \times (B \times C)) \times D$

or $A \times (B \times (C \times D))$

or ...

Which is best?

What's the cost?

Take W of size $a \times b$ & Y of size $b \times c$

$Z = WY$ is of size $a \times c$.

$$a\overset{b}{\begin{bmatrix} W \end{bmatrix}} \; b\overset{c}{\begin{bmatrix} \;Y\; \end{bmatrix}} = a\overset{c}{\begin{bmatrix} \;Z\; \end{bmatrix}}$$

$$Z_{ij} = \sum_{k=1}^{b} W_{ik} Y_{kj}$$ takes b multiplications & $b-1$ additions

Z has ac entries so abc multiplications (& roughly same adds)

So say <u>cost</u> of multiplying XY is <u>abc.</u>

For earlier example:

$$((A \times B) \times C) \times D \text{ costs} = \underbrace{(50)(20)(1)}_{A\times B} + \underbrace{(50)(1)(10)}_{(A\times B)\times C} + \underbrace{(50)(10)(100)}_{(A\times B\times C)\times D}$$
$$= 51{,}500$$

$$(A \times B) \times (C \times D) \text{ costs} = \underbrace{(50)(20)(1)}_{(A\times B)} + \underbrace{(1)(10)(100)}_{(C\times D)} + (50)(1)(100)$$
$$= 7{,}000$$

$$(A \times (B \times C)) \times D \text{ costs} = (20)(1)(10) + (50)(20)(10) + (50)(10)(100)$$
$$= 60{,}200$$

General problem:

For n matrices $A_1, A_2, \ldots, A_n$

where A_i is of size $m_{i-1} \times m_i$

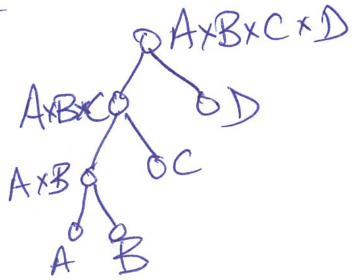
What is the min cost for computing $A_1 \times A_2 \times \cdots \times A_n$?

Input: $m_0, m_1, \ldots, m_n$

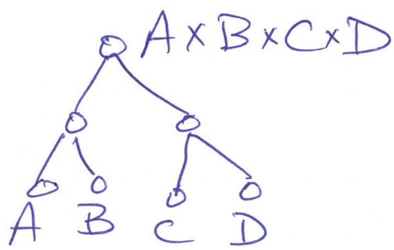
Goal: Determine parenthesization of min cost for computing $A_1 \times A_2 \times \cdots \times A_n$

Graphical view:

$((A \times B) \times C) \times D$



$(A \times B) \times (C \times D)$



Leaves correspond to $A_1, \ldots, A_n$

internal nodes are intermediate computations.

root represents $A_1 \times \cdots \times A_n$

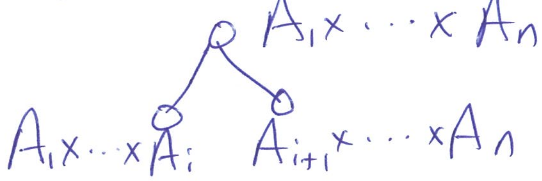
Try prefixes for subproblems:

$C(i)$ = min cost for computing $A_1 \times A_2 \times \cdots \times A_i$

But for the root,

left child is a prefix

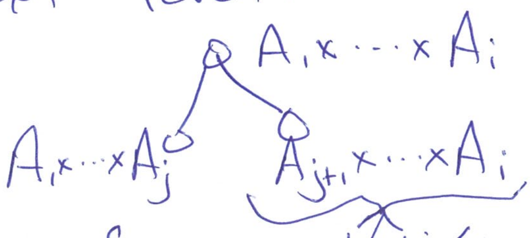
but right child is a suffix



for some $1 \le i \le n-1$

So need suffixes too.

Then for next level:



for some $1 \le j \le i-1$

need <u>substrings</u>

<u>Definition of subproblems:</u>

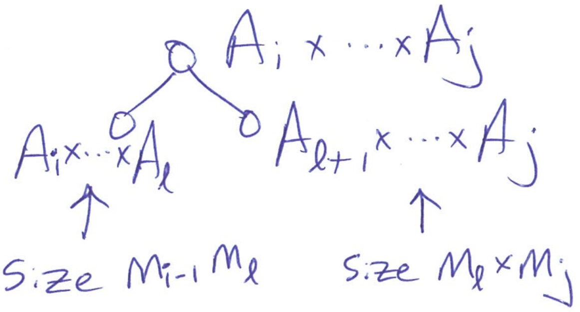
For $1 \le i \le j \le n$,

let $C(i,j)$ = min cost of computing $A_i \times \cdots \times A_j$

Computing $C(i,j)$:

Base case: $C(i,i)=0$

For $i<j$ try all ℓ for the split where $i \le \ell \le j-1$



So root costs $m_{i-1} m_\ell m_j$

best left subtree costs $C(i,\ell)$

best right subtree costs $C(\ell+1,j)$

total $= m_{i-1} m_\ell m_j + C(i,\ell) + C(\ell+1,j)$

Therefore,

$$C(i,j) = \min_{\ell} \{ C(i,\ell) + C(\ell+1,j) + m_{i-1} m_\ell m_j \}$$

ℓ ← over ℓ such that $i \le \ell \le j-1$

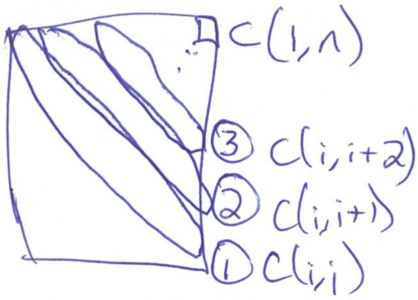
Let $s = j - i$.

Base case is $s=0$.

To fill for $C(i,j)$ with $s = j - i$

Use smaller $s' < s$ for $C(i',j')$ with $s' = j' - i'$

C = 



ChainMatrixMultiply($m_0, m_1, \ldots, m_n$)

For $i = 1 \to n$, $C(i,i) = 0$

For $s = 1 \to n-1$,

  For $i = 1 \to n-s$,

    Let $j = i+s$

    $C(i,j) = \infty$

    For $l = i \to j-1$

      $cur = m_{i-1} m_k m_j + C(i,l) + C(l+1,j)$

      if $C(i,j) > cur$

        then $C(i,j) = cur$

Return($C(1,n)$)

Running time: $O(n^3)$