CMSC 451 - Algorithm Design

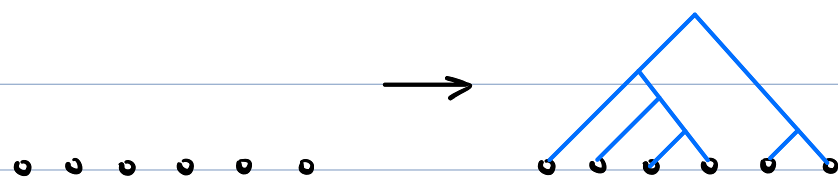
Lecture 10 - DP: Chain Matrix Multiplication

Dynamic programming for tree structures -

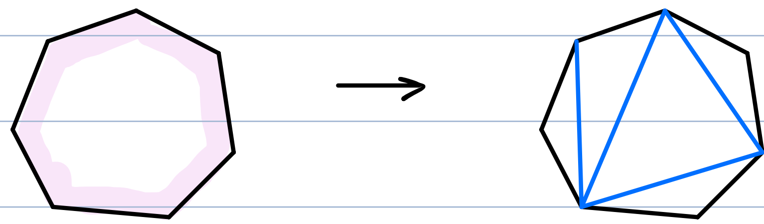
- Many optimization problems produce a tree structure as output

Optimal:

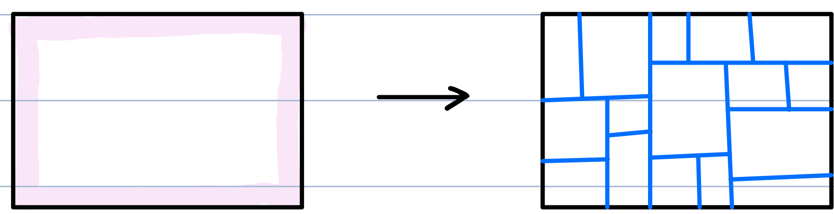
Binary search tree



Polygon triangulation



Quadtree or k-d tree

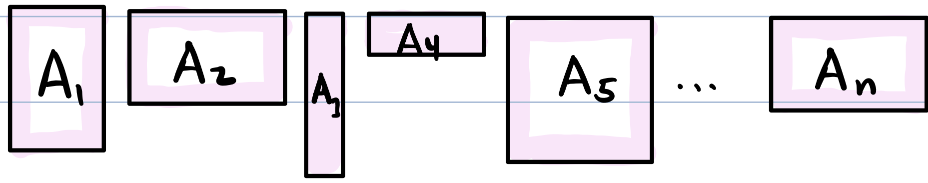


Chain Matrix Mutiplication -

A simple example of tree-based DP solutions.

Problem: Given a sequence of matrices (of various sizes), what is the best evaluation order?

$$A_1 \cdot A_2 \cdot \cdots \cdot A_n$$



Facts:

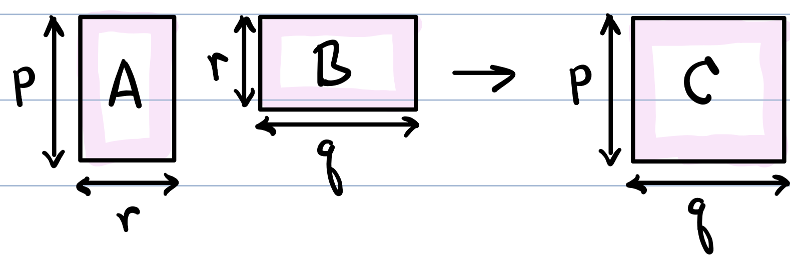
- Matrix mult is associative, but not commutative
  - $\Rightarrow$ Can parenthesize as you like, but cannot change order

- To multiply: $A \cdot B \rightarrow C$

  A is $p \times r$ (p rows, r columns)

  B is $r \times q$

  C is $p \times q$

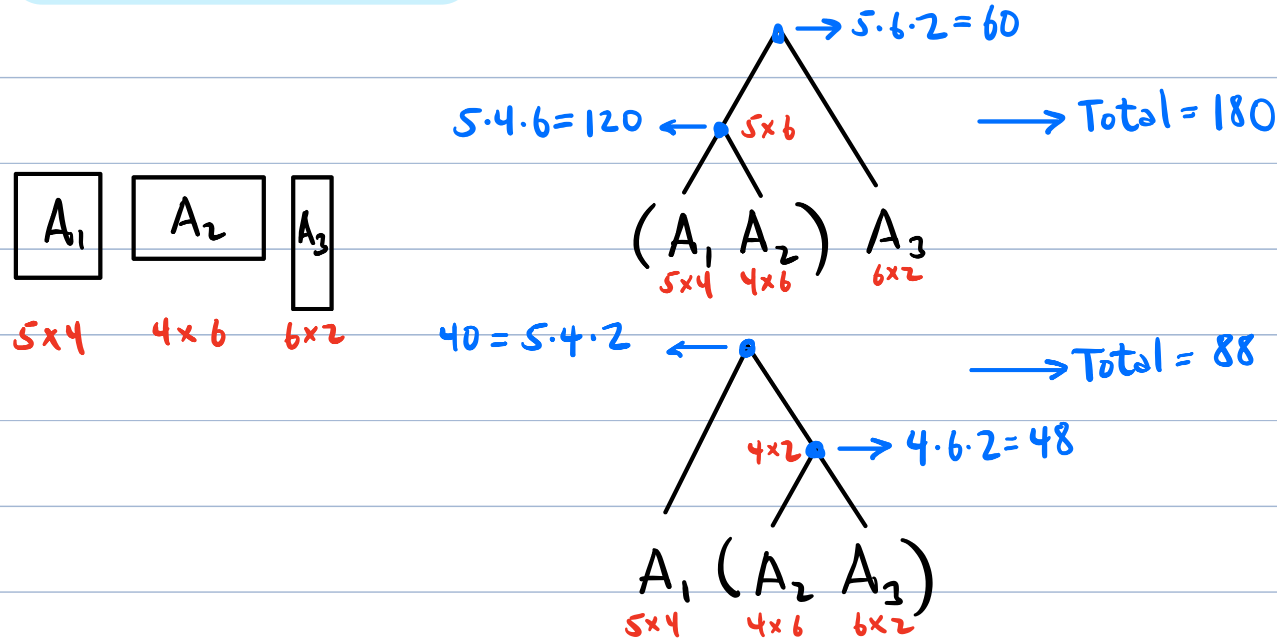


$$(p \times r)(r \times q) \rightarrow (p \times q)$$

for $1 \le i \le p$, $1 \le j \le q$, $c_{ij} = \sum_{k=1}^{r} a_{ik} \cdot b_{kj}$

- Time to multiply $\sim p \cdot q \cdot r$ (simplified)

Order matters:

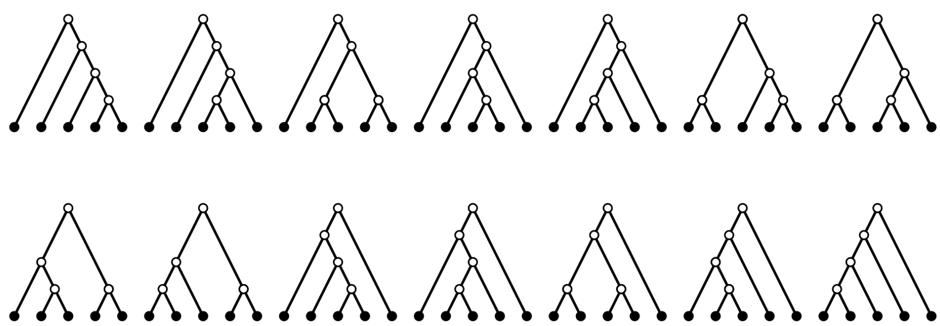


Chain Matrix Mult Problem:

Given a seq. of matrices $A_1, \dots, A_n$ presented by their dimensions $\bar{p} = \langle p_0, p_1, \dots, p_n \rangle$ where A_i is $p_{i-1} \times p_i$, determine the order of multiplications (binary tree) that minimizes num. of ops.

Brute force Solution?

- Enumerate all binary trees with n leaves
- Exponentially many!
- $n=5$:



- Number of possible parenthesizings:

$$P(n) = \begin{cases} 1 & \text{if } n=1 \\ \sum_{k=1}^{n-1} P(k)\cdot P(n-k) & n \geq 2 \end{cases}$$

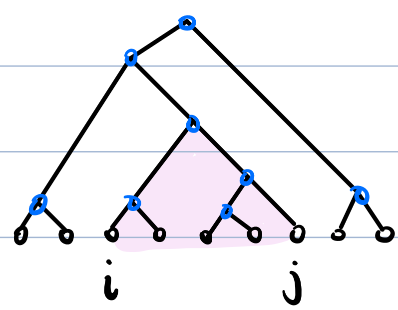
Solution is related to famous combinatorial function, Catalan Numbers. $P(n) = C(n-1)$

$$C(m) = \frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

Aside: Named for Eugène Catalan (1800's) but originally discovered in 1730's by Mongolian mathematician/astronomer Ming Antu.

DP Solution to Chain Matrix Mult:

- How to form subproblems for partioning problems?



- For $1 \leq i \leq j \leq n$, let

  $A(i,j) = A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$

  $M(i,j) =$ min. num. of ops to compute $A(i,j)$

- Goal: $M(1,n)$

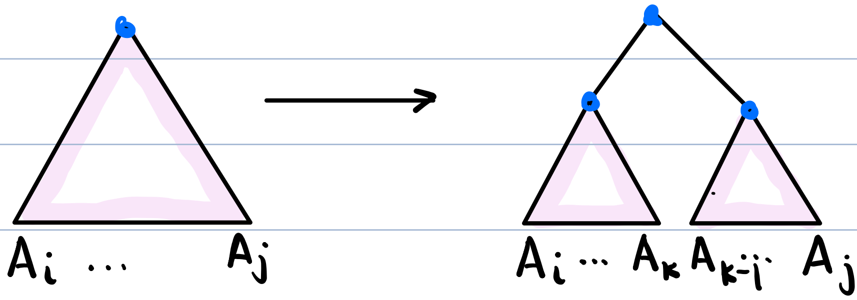
Deriving the recursive formulation:

- $A(i,i) = A_i$ — trivial (no ops. needed)
- Assume $i<j$

$$A_i \cdot A_{i+1} \cdots A_j = A(i,j)$$

$$p_{i-1}\times p_i \quad p_i\times p_{i+1} \cdots p_{j-1}\times p_j \qquad\qquad p_{i-1}\times p_j$$

- Where's the top split?



$$(A_i \cdots A_j) = (A_i \cdots A_k)(A_{k+1} \cdots A_j) \quad \text{for } i \le k \le j-1$$

$$A(i,j) = A(i,k) \cdot A(k+1,j)$$

Issues:

1. Where to split? (value of k)
2. Cost to compute $A(i,k)$ + $A(k+1,j)$?
3. How many ops for last multiplication?

Answers:

① DP Credo – Try all k. Take the best

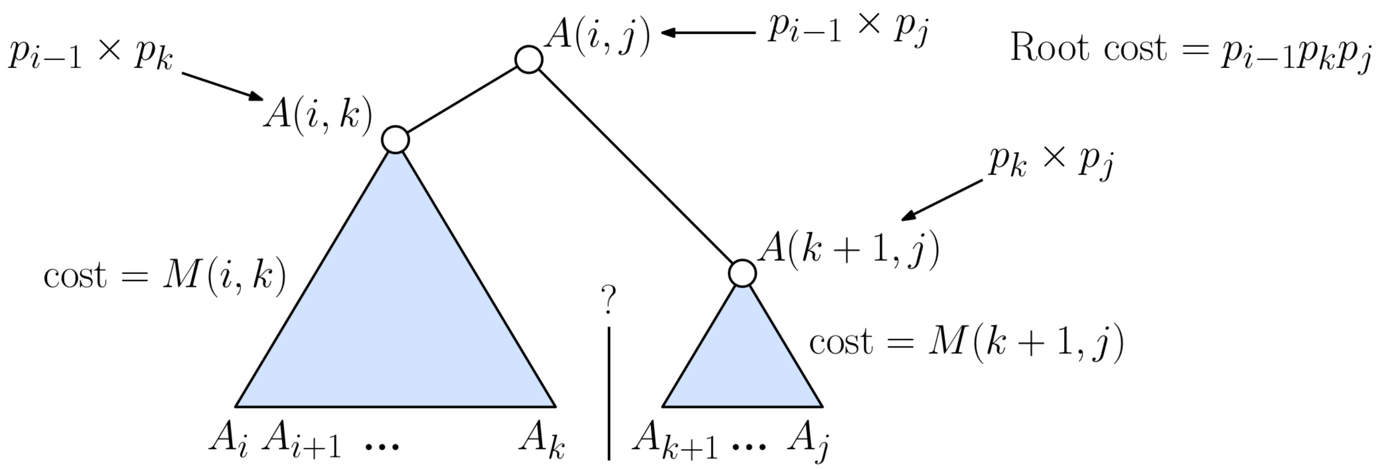
② Princip. of Optimality – Best possible

$$M(i,k) + M(k+1,j)$$

③ $A(i,k)$ is $p_{i-1} \times p_k$

$A(k+1,j)$ is $p_k \times p_j$

$A(i,k) \cdot A(k+1,j)$ takes $p_{i-1} \cdot p_k \cdot p_j$ ops



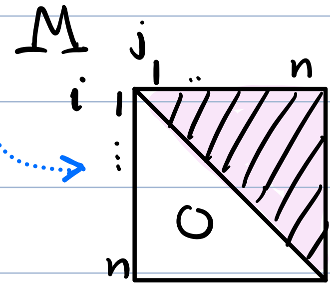
Recursive DP formulation: for $1 \le i \le j \le n$

$$\boxed{M(i,j) = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k \le j-1} \left\{ M(i,k) + M(k+1,j) + p_{i-1} \cdot p_k \cdot p_j \right\} & \text{if } i<j \end{cases}}$$

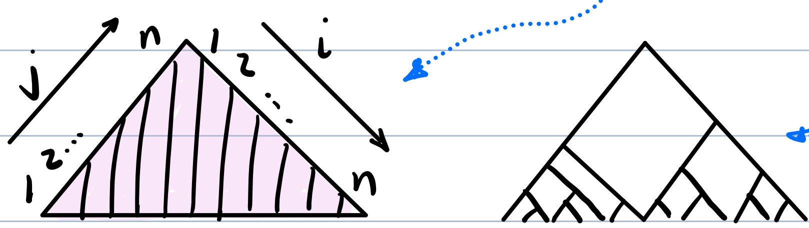
Final answer is $M(1,n)$

Memoized Implementation:

- Array $M[i,j]$, $1 \le i \le j \le n$
  - Upper triangular
  - Init: $M[i,j] = -1$ (undef.)



- In our pictures we'll rotate it:

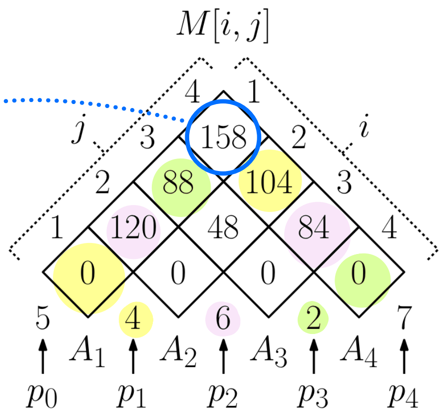


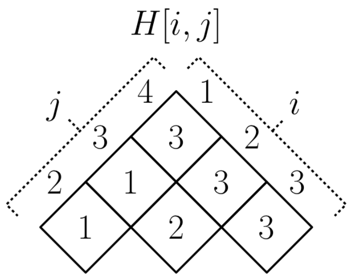
so it looks more like a tree.

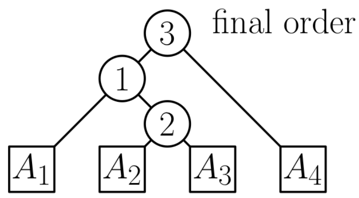
- Hooks - $H[i,j]$ stores optimal split index k

```
memo-cmm(i, j)                                  // Memoized CMM
    if (M[i,j] = -1)                            // undefined?
        minCost ← +∞
        for (k ← i to j-1)                      // try all splits
            cost ← (memo-cmm(i, k) + memo-cmm(k+1, j)
                     + p[i-1] * p[k] * p[j])    // cost
            if (cost < minCost)                 // new best cost?
                minCost ← cost                  // ... save it
                H[i,j] ← k                      // ... save split index
        M[i,j] ← minCost                        // save final cost
    return M[i,j]
```

Example:







M[1,4] = min:

(k=1) $M[1,1]+M[2,4]+p_0 \cdot p_1 \cdot p_4 = 0+104+140 = 244$

(k=2) $M[1,2]+M[3,4]+p_0 \cdot p_2 \cdot p_4 = 120+84+210 = 414$

(k=3) $M[1,3]+M[4,4]+p_0 \cdot p_3 \cdot p_4 = 88+0+70 = 158$

best split

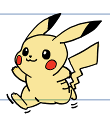
min

$$\boxed{\begin{array}{l} M[1,4] \leftarrow 158 \\ H[1,4] \leftarrow 3 \end{array}}$$

Correctness:

- Follows from correctness of DP formulation

Running Time:



- $O(n^2)$ table entries
- Each takes $O(j-i+1) \leq O(n)$ time to compute
- Total: $O(n^3)$

Extracting the Final Sequence:

- So far we only compute the opt. cost
- To extract final sequence - use hooks
- Recall - $H[i,j]$ stores best split for $A_i \cdots A_j$
- $A(i,j) = A_i A_{i+1} \cdots A_j$

$$= (A_i \cdots A_k)(A_{k+1} \cdots A_j)$$

where $k = H[i,j]$

- Continue splitting until $i = j$, $A(i,i) = A_i$

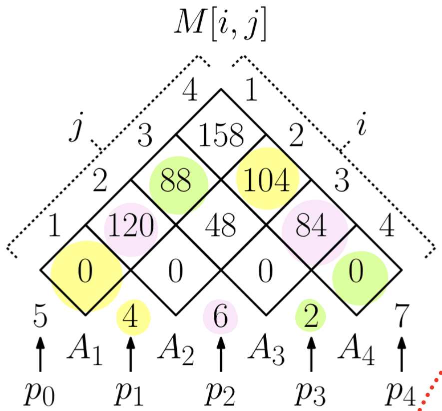
```
do-mult(i, j)                 // multiply Ai ... Aj optimally
    if (i = j) return Ai      // basis - one matrix
    else
        k ← H[i,j]            // opt split index
        X ← do-mult(i,k)      // X ← Ai ... Ak
        Y ← do-mult(k+1, j)   // Y ← Ak+1 .. Aj
        return X·Y            // final product
```

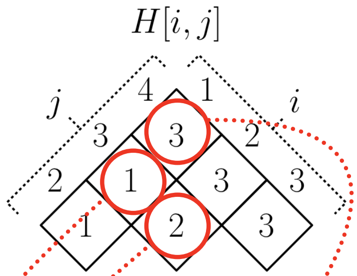
Initial call: do-mult(1,n)

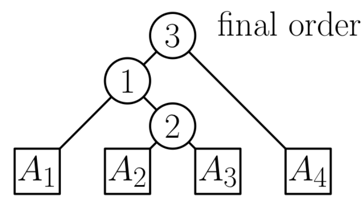
Running time:

- $O(n)$ to extract opt. evaluation tree
- $O(M[1,n])$ to perform multiplications

Example:







$$A_1 .. A_4 = \xrightarrow{H[1,4]=3} (A_1 .. A_3)(A_4)$$

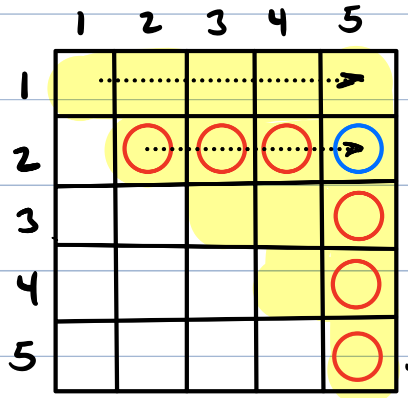
$$\xrightarrow{H[1,3]=1} ((A_1)(A_2 .. A_3))\,(A_4)$$

$$\xrightarrow{H[2,3]=2} ((A_1)((A_2)(A_3)))\,(A_4)$$

Bottom-Up Implementation:

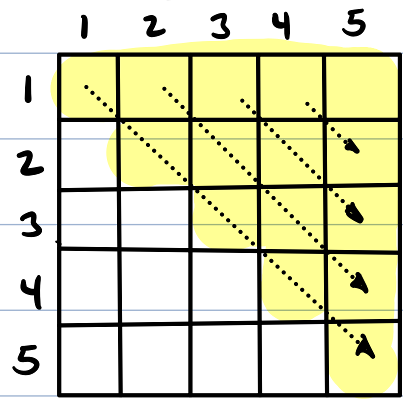
- Normally this is an easy extension.
  Just fill table row by row.
- Doesn't work!

E.g., Computing $M[2,5]$ accesses $\begin{cases} M[2,2] & M[3,5] \\ M[2,3] & M[4,5] \\ M[2,4] & M[5,5] \end{cases}$



Not yet computed!

Trick: Compute entries diagonal by diagonal, working out from main diagonal.



Coding this is a bit tricky. (Think about it)

Recall: $1 \le i \le j \le n$ (upper triangular)

Main = 1^{st} diagonal: $[1,1]\ [2,2] \cdots [n,n] \Rightarrow j-i+1=1$

2^{nd} diagonal: $[1,2]\ [2,3] \ldots [n-1,n] \Rightarrow j-i+1=2$

3^{rd} diagonal: $[1,3]\ [2,4] \ldots [n-2,n] \Rightarrow j-i+1=3$

$\vdots$

n^{th} diagonal: $[1,n] \Rightarrow j-i+1=n$

Let $\ell = j-i+1$ = which diagonal

ℓ runs from 1 (main diag) to n (corner)

$i = 1, 2, \ldots, \underbrace{n-\ell+1}$ chosen so that $j \le n$

$j = i+\ell-1$

```
bottom-up-cmm ( )                              // bottom-up CMM
    for (i ← 1 to n) M[i,i] = 0                // basis-main diag.
    for (ℓ ← 2 to n)                           // ℓ = diagonal number
        for (i ← 1 to n-ℓ+1)
            j ← i+ℓ-1                          // pick j so j-i+1 = ℓ
            minCost ← ∞
            for (k ← i to j-1)
                minCost ← min (minCost,
                    M[i,k] + M[k+1,j] + p_{i-1}·p_k·p_j)
            M[i,j] ← minCost
    return M[1,n]
```

same as before

Smaller j-i differences so entry is defined

Running Time:

$O(n^3)$ - 3 nested loops

- Each in range 1..n

Summary:

- $O(n^3)$ DP algorithm for chain matrix mult
- Explary of DP's that build hierarchies
- Example: Optimum binary search tree