Parenthesization  A product of matrices is fully parenthesized if it is either
- a single matrix, or
- a product of two fully parenthesized matrices, surrounded by parentheses

Each parenthesization defines a set of \( n-1 \) matrix multiplications. We just need to pick the parenthesization that corresponds to the best ordering.

How many parenthesizations are there?

not fully paren.

\[
A_1(A_2[A_3A_4])A_5
\]

\[
(B_1B_2)(A_1A_2A_3)A_4
\]

Each parenthesization defines a set of \( n-1 \) matrix multiplications. We just need to pick the parenthesization that corresponds to the best ordering.

How many parenthesizations are there?

Let \( P(n) \) be the number of ways to parenthesize \( n \) matrices.

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

This recurrence is related to the Catalan numbers, and solves to
unique subproblems
$A_1, A_2, A_3, A_4$
$A_{12}, A_{13}, A_{14}, A_{23}, A_{24}, A_{34}$

$A_1 \ldots A_n$
what subproblems arise?
$A_3 A_6 A_9 A_5 A_4$

$A_i$: consecutive matrices
$A_{ij} = A_i \ldots A_j$
$ightarrow \Theta(n^2)$ possible unique subproblems

$A_{14}$

$A_{12} A_{34}$

$A_{13} A_{44}$

$A_{21} A_{24}$

$A_{23} A_{34}$

$A_{13} = \min(A_1 A_{23}, A_2 A_{34})$

$J - i = 3$
$J - i = 0, 1, 2$

order by $J - i$:
$J - i \geq 0, 1$
Matrix-Chain-Order(p)

1. $n \leftarrow \text{length}[p] - 1$
2. for $i \leftarrow 1$ to $n$
3.     do $m[i, i] \leftarrow 0$
4. for $l \leftarrow 2$ to $n$  \hspace{1cm} \triangleright \text{is the chain length.}$
5.     do for $i \leftarrow 1$ to $n - l + 1$
6.         do $j \leftarrow i + l - 1$
7.             $m[i, j] \leftarrow \infty$
8.             for $k \leftarrow i$ to $j - 1$
9.                 do $q \leftarrow m[i, k] + m[k + 1, j] + p_i p_k p_j$
10.                if $q < m[i, j]$
11.                   then $m[i, j] \leftarrow q$
12.                   $s[i, j] \leftarrow k$
3. return $m$ and $s$

\[O(n^3)\]
Matrix Multiplication is **associative**, so I can do the multiplication in several different orders.

**Example:**
- $A_1$ is 10 by 100 matrix
- $A_2$ is 100 by 5 matrix
- $A_3$ is 5 by 50 matrix
- $A_4$ is 50 by 1 matrix
- $A_1A_2A_3A_4$ is a 10 by 1 matrix

\[
\begin{align*}
A_{12} &= 5000 \\
A_{23} &= 25000 \\
A_{34} &= 250 \\
A_{13} &= \min(A_{12}, A_{34}) \Rightarrow 7500 \\
A_{14} &= \min(A_{12}, A_{34}, A_{13}, A_{14}) \\
A_{24} &= \min(A_{23}, A_{34}, A_{13}, A_{24}) \\
\end{align*}
\]

**Time for DP is $O(n^3)$ for** $A_1, \ldots, A_n$

$n = 4$  $4^3 = 64$ steps to determine the right order.

(25000 → 1750)

Saves 73000 step.
Thought:

Figuring out the order of the computation was non-trivial for MCM. Why not automate it? Keep track of what you have computed and don't recompute. Memoization

```c
/* Demonstration of recursion, dynamic programming and memoization on Fibonacci numbers
   Cliff Stein 10/16/05 */

/* Simple recursive program */
long long fib_recursive(int n) {
    if ((n==1) || (n==2))
        return 1;
    else if (n==0)
        return 0;
    else
        return (fib_recursive(n-1) + fib_recursive(n-2));
}

/* Dynamic programming solution. Fill in the table in order */
long long fib_dp(int n) {
    long long F[n+1];
    int i;
    F[0] = 0;
    F[1] = 1;
    for (i = 2; i <= n; i++)
        F[i] = F[i-1] + F[i-2];
    return F[n];
}
```

1, 1, 2, 3, 5, 8, 13, 21

F_n = F_{n-1} + F_{n-2}
Memorization:
- gives the same O(1) running time as DP.
- may or may not be easier to code
- constant factors better with DP
- with DP, one can sometimes explicitly save space, impossible without memorization.

/* Main demonstration program */
main()
{
    int n;
    printf("enter n - We will compute the nth fibonacci number. \n");
    scanf("%d", &n);
    printf("\nHit a key to begin the dynamic programming computation\n");
    getchar(); getchar();
    printf("\n\n%8d th fibonacci number is %lld\n", n, fib_dp(n));
    printf("\nHit a key to begin the memoized computation\n");
    getchar();
    printf("\n\n%8d th fibonacci number is %lld\n", n, fib_memoize(n));
    printf("\nHit a key to begin the recursive computation\n");
    getchar();
    printf("\n\n%8d th fibonacci number is %lld\n", n, fib_recursive(n));
Space

Fib \( O(n) \) space

\[
\begin{array}{c}
\vdots \\
1 \\
\vdots \\
1 \\
\vdots \\
1
\end{array}
\]

can use \( O(1) \) space
for many apps.

\( O(n^2) \) to \( O(n) \) space

Oct 8-5:20 PM