matrix-chain.pdf (application/pdf Object) - Mozilla Firefox

http://www.columbia.edu/~cs2035/courses/csor4231.F09/matrix-chain.pdf

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.

$A_1(A_2(A_3)A_4)$

$((B_1B_2)((A_1A_2)A_3))A_1$

Oct 8-4:08 PM

matrix-chain.pdf (application/pdf Object) - Mozilla Firefox

http://www.columbia.edu/~cs2035/courses/csor4231.F09/matrix-chain.pdf

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

$$((\overset{k}{A_1A_2A_3A_4})(\overset{n-k}{A_5A_6}))$$

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.

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

This recurrence is related to the Catalan numbers, and solves to

Oct 8-4:33 PM

unique subproblems

A_1, A_2, A_3, A_4

$A_{12}, A_{13}, A_{14}, A_{23}, A_{24}, A_{34}$

$A_1 \cdots\cdots A_n$

what subproblems arise?

$A_3 A_6 A_9$ $\quad$ $A_5 A_4$

A: consecutive matrices

$A_{ij} = A_i \cdots A_j$

$\longrightarrow \Theta(n^2)$ possible unique subproblems

Oct 8-4:41 PM

A_{14} $\qquad$ $j-i = 3$

$A_{11} A_{24}$ $\quad$ $A_{12} \underline{A_{34}}$ $\quad$ $A_{13} \underline{\underline{A_{44}}}$ $\qquad$ $j-i = 0, 1, 2$

.. $A_{22} \underline{A_{34}}$ $\quad$ $A_{23} \underline{\underline{A_{44}}}$

order by $j-i$:

$j-i$ $\quad$ $0, 1$

A_1 $\quad$ A_{12} $\quad$ A_3

A_{23}

$A_{13} = \min(A_1 A_{23}, A_{12} A_3)$

Oct 8-4:50 PM

```
Matrix-Chain-Order(p)
n ← length[p] − 1
for i ← 1 to n
    do m[i, i] ← 0
for l ← 2 to n                         ▷ l is the chain length.
    do for i ← 1 to n − l + 1
           do j ← i + l − 1
              m[i, j] ← ∞
              for k ← i to j − 1
                  do q ← m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
                     if q < m[i, j]
                        then m[i, j] ← q
                             s[i, j] ← k
return m and s
```

compute $m[i,j]$

Oct 8-4:57 PM


```
Matrix-Chain-Order(p)
n ← length[p] − 1
for i ← 1 to n
    do m[i, i] ← 0
for l ← 2 to n                         ▷ l is the chain length.
    do for i ← 1 to n − l + 1
           do j ← i + l − 1
              m[i, j] ← ∞
              for k ← i to j − 1
                  do q ← m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
                     if q < m[i, j]
                        then m[i, j] ← q
                             s[i, j] ← k
return m and s
```


$O(n^3)$

Oct 8-4:59 PM

matrix-chain.pdf (application/pdf Object) - Mozilla Firefox

- 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
- $A_1 A_2 A_3 A_4$ is a 10 by 1 matrix

$A_{12} = 5000$

$A_{23} = 25000$

$A_{34} = 250$

$A_{13} = \min(A_1 A_{23}, A_{12} A_3)$

7500

$25000 + 50000 = 75000$ $\quad$ $5000 + 2500 = 7500$

$$A_{14} = \min\begin{pmatrix} A_1 A_{24} \\ A_{12} A_{34} \\ A_{13} A_4 \end{pmatrix}$$

$A_{24} = \min(A_2 A_{34}, A_{23} A_4)$

Oct 8-5:03 PM

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

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

$(75000 \to 1750$ saves 73000 step$)$

Oct 8-5:03 PM

Thought:

Figuring out the order of the computation was non-trivial for MCM. Why not automate it?

Keep track of what you have compute & don't recompute.

Memoization

Oct 8-5:06 PM

Mozilla Firefox

http://www.columbia.edu/~cs2035/courses/csor4231.F09/fib.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;
```

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

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

Oct 8-5:11 PM

Memoization:

- gives the same O( ) running time as DP.
- may or may not be easier to code
- constant factors better w/ DP
- w/ DP, one can sometimes explicitly save space, impossible w/ memoization.

Oct 8-5:17 PM

Mozilla Firefox

File Edit View History Bookmarks Tools Help

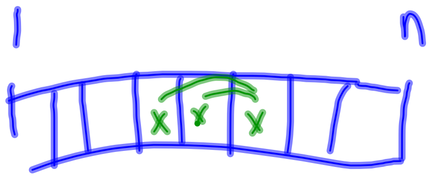
http://www.columbia.edu/~cs2035/courses/csor4231.F09/fib.c

```
/* Main demonstration program */

main()
{

int  n;

  printf("enter n - We will compute the nth fibonacci number. \n");
  scanf("%d", &n);


  printf("\n \nHit a key to begin the dynamic programming computation");
  getchar();getchar();
  printf("\n\n %d th fibonacci number is %lld\n",n,fib_dp(n));

  printf("\n \nHit a key to begin the memoized computation");
  getchar();
  printf("\n\n %d th fibonacci number is %lld\n",n,fib_memoize(n));

  printf("\n \nHit a key to begin the recursive computation");
  getchar();
    printf("\n\n %d th fibonacci number is %lld\n",n,fib_recursive(n));
```

Oct 8-5:18 PM

Space

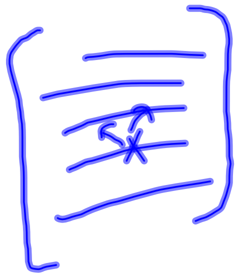
Fib $O(n)$ space



can use $O(1)$ space

for many apps.

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



Oct 8-5:20 PM