

CSC 373 H 1 Y — Summer 2006

University of Toronto — St. George Campus

Lecture Summary for Week 6

This summary is not a replacement for the lecture. If you miss a class, please arrange with a friend to take note for you.

Tutorial this week: Counting Inversion [Section 5.3]

Matrix Chain Multiplication Problem (cont'd) [CLRS 15.2]

To find the best way of computing $A_1 \times A_2 \times \dots \times A_n$ we find the best point to “break” the sequence, i.e.

$$A_1 \times A_2 \times \dots \times A_n = (A_1 \times \dots \times A_i) \times (A_{i+1} \times \dots \times A_n)$$

Thus, in general we need to find the best way of computing $A_i \times \dots \times A_j$, for $i \leq j$. Notice that $(A_i \times \dots \times A_j)$ is the matrix with dimension $d_i \times d_{j+1}$.

The design of the dynamic-programming algorithm is as follows.

1. **Array:** $N[i, j]$ is the optimal cost of multiplying $A_i \times \dots \times A_j$, for $i \leq j$.
2. **Initialization & Recurrence:** Notice that the size of the subproblem is decided by the length of the sequence A_i, \dots, A_j . So the initialization is

$$N[i, i] = 0 \quad \text{for } 1 \leq i \leq n$$

The recurrence:

$$N[i, j] = \min\{N[i, k] + N[k + 1, j] + d_i d_{k+1} d_{j+1} : \text{for } i \leq k < j\}$$

Here $d_i d_{k+1} d_{j+1}$ is the cost of multiplying the two products $(A_i \times \dots \times A_k)$ and $(A_{k+1} \times \dots \times A_j)$.

3. **Program:** We will compute the best break point $B[i, j]$ for the sequence A_i, \dots, A_j , as well as the best cost $N[i, j]$.

```
1. For i from 1 to n do n[i, i] ← 0 End For
2. For ℓ from 1 to n - 1 do
3.   For i from 1 to n - ℓ do % compute A_i × ... × A_{i+ℓ}
4.     j ← i + ℓ
5.     N[i, j] ← ∞
6.     For k from i to j - 1 do
7.       v ← N[i, k] + N[k + 1, j] + d_i d_{k+1} d_{j+1}
8.       If v < N[i, j] do
9.         N[i, j] ← v
10.        B[i, j] ← k % break point
11.      End If
12.    End For
13.  End For
14. End For
```

Running time: $\Theta(n^3)$.

4. **Computing the solution:** Here is a way of computing $A_1 \times \dots \times A_n$ using the array B . We call $\text{Mult}(A, B, 1, n)$, where Mult is the program:

Mult(A, B, i, j):

1. If $i = j$ return A_i
2. Else
3. $k \leftarrow B[i, j]$
4. $M_1 \leftarrow Mult(A, B, i, k)$
5. $M_2 \leftarrow Mult(A, B, k + 1, j)$
6. return $M_1 \times M_2$
7. End If

Divide-and-Conquer:

Solving the problem recursively by breaking it into non-overlapping subproblems.

Mergesort [Section 5.1]

Input A sequence of number a_1, \dots, a_n

Output A permutation in increasing order:

$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

The idea is to (i) divide the sequence into 2 halves, then (ii) sort each half separately, and then (iii) merge two sorted halves

The following program merges the two sorted arrays:

Merge(A,B) % $A[1 \dots n], B[1 \dots m]$ are sorted

1. C : array of length $(n + m)$
2. $i \leftarrow 1, j \leftarrow 1, k \leftarrow 1$
3. While $k \leq (n + m)$ do
4. If $i > n$ then $C[k] \leftarrow B[j]; j \leftarrow j + 1$
5. Else If $j > m$ then $C[k] \leftarrow A[i]; i \leftarrow i + 1$
6. Else If $A[i] < B[j]$ then $C[k] \leftarrow A[i]; i \leftarrow i + 1$
7. Else $C[k] \leftarrow B[j]; j \leftarrow j + 1$
8. End If
9. $k \leftarrow k + 1$
10. End While
11. return C

Mergesort(A, p, r):

1. If $p = r$ return
2. Else If $r = p + 1$ compare $A[p], A[p + 1]$
3. Else
4. $q \leftarrow \lceil \frac{p+r}{2} \rceil$ % midpoint
5. Mergesort(A, p, q)
6. Mergesort($A, q + 1, r$)
7. Merge'(A, p, q, r)

Here $\text{Merge}'(A, p, q, r)$ is similar to Merge.

Exercise Give the program Merge'.

Running time

$$T(n) = 2T(n/2) + \Theta(n)$$

so by the Master Theorem below, $T(n) = \Theta(n \log(n))$.

The Master Theorem [see also Section 5.2]

If for some $a, b, d > 0$:

$$T(n) = aT(n/b) + \mathcal{O}(n^d)$$

then

$$T(n) = \begin{cases} \mathcal{O}(n^d) & \text{if } a < b^d \\ \mathcal{O}(n^d \log(n)) & \text{if } a = b^d \\ \mathcal{O}(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

The Matrix Multiplication Problem [CLRS 28.2]

Input A, B : $n \times n$ matrices.

Output $A \times B$.

If we compute the product $A \times B$ using the formula

$$c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j}$$

then the running time is $\Theta(n^3)$.

For the divide-and-conquer approach, we write

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \quad B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \quad C = A \times B = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix}$$

Computing C_1, C_2, C_3, C_4 :

$$\begin{aligned} C_1 &= A_1 B_1 + A_2 B_3 & C_2 &= A_1 B_2 + A_2 B_4 \\ C_3 &= A_3 B_1 + A_4 B_3 & C_4 &= A_3 B_2 + A_4 B_4 \end{aligned}$$

Using these formulas, the first attempt will be

MMult(A, B)

1. Compute $A_1, B_1, \dots, A_4, B_4$ % by computing $m = n/2$
2. $C_1 \leftarrow \text{MMult}(A_1, B_1) + \text{MMult}(A_2, B_3)$
3. $C_2 \leftarrow \text{MMult}(A_1, B_2) + \text{MMult}(A_2, B_4)$
4. $C_3 \leftarrow \text{MMult}(A_3, B_1) + \text{MMult}(A_4, B_3)$
5. $C_4 \leftarrow \text{MMult}(A_3, B_2) + \text{MMult}(A_4, B_4)$
6. Output C

The running time $T(n)$ of MMult when A and B have size $n \times n$ satisfies

$$T(n) = 8T(n) + \Theta(n^2)$$

So $T(n) = \Theta(n^3)$. This is NOT an improvement over the straightforward $\Theta(n^3)$ algorithm.

Strassen's algorithm

Strassens algorithm for the Matrix Multiplication is based on the following observation:

$$\begin{aligned}C_1 &= P_5 + P_4 - P_2 + P_6 \\C_2 &= P_1 + P_2 \\C_3 &= P_3 + P_4 \\C_4 &= P_1 + P_5 - P_3 - P_7\end{aligned}$$

where

$$\begin{aligned}P_1 &= A_1(B_2 - B_4) \\P_2 &= (A_1 + A_2)B_4 \\P_3 &= (A_3 + A_4)B_1 \\P_4 &= A_4(B_3 - B_1) \\P_5 &= (A_1 + A_4)(B_1 + B_4) \\P_6 &= (A_2 - A_4)(B_3 + B_4) \\P_7 &= (A_1 - A_3)(B_1 + B_2)\end{aligned}$$

Exercise Verify that C_1, \dots, C_4 can be computed as above.

Exercise Using the above formulas, give the divide-and-conquer program for Matrix Multiplication.

The running time for this approach is computed using the Master Theorem as follows:

$$T(n) = 7T(n) + \Theta(n^2)$$

so $T(n) = \Theta(n^{\log_2 7})$ (around $\Theta(n^{2.81})$)