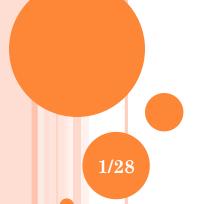
ALGORITHMS AND DATA STRUCTURES II



Divide and Conquer Algorithm Design, Matrix multiplication (MM), Strassen Algorithm for MM.

Lecturer: K. Markov

markov@u-aizu.ac.jp



DIVIDE AND CONQUER

- Recursive algorithms solve a given problem by calling themselves recursively. They follow a divide-andconquer approach:
 - break the problem into several subproblems that are similar to the original problem but smaller in size,
 - solve the sub-problems recursively,
 - combine these solutions to create a solution to the original problem.



DIVIDE AND CONQUER

- The divide-and-conquer paradigm has three steps at each level of the recursion:
- 1. Divide the problem into several subproblems.
- 2. Conquer the sub-problems by solving them recursively. If the sub-problem sizes are small enough, then solve the sub-problem straightforwardly.
- 3. Combine the solutions to the sub-problems into the solution for the original problem. 3/28

DIVIDE AND CONQUER

 The merge sort algorithm is an example of divide-and-conquer approach:

- 1. Divide: Divide an n element sequence to be sorted into two subsequences of n/2 elements each.
- 2. Conquer: Sort the two subsequences recursively using merge sort.
- 3. Combine: Merge the two sorted subsequences to get the answer.



The DIVIDE step

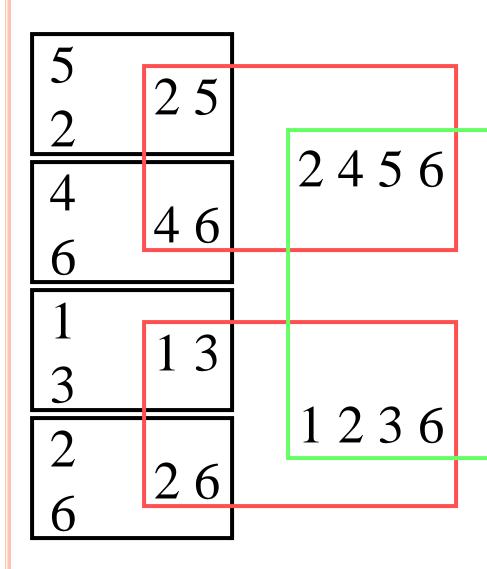
52 46 5246 1326 13 26 13

5 2

The Conquer step: sort the two subsequences recursively using merge sort. Recursion goes on until our subsequences come down to length one. Then they are sorted and we have nothing to do.

6/28

The COMBINE step



1 2 2 3 4 5 6 6

 In Lecture 1, we analyzed the merge sort algorithm and found that the time complexity is:

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ 2T(n/2) + \Theta(n), & \text{if } n > 1 \end{cases}$$

which we said that can be solved and gives:

$$T(n) = O(n \log n)$$

At the end of this lecture we will prove it using the so called Master theorem.

• Let A and B be two $n \times n$ matrices. The product of A and B is defined as C = AB, where for $1 \le i, j \le n$:

$$C[i,j] = \sum_{k=1}^{n} A[i,k] \times B[k,j]$$

• If n is a power of 2, we can partition each of A and B into four $n/2 \times n/2$ matrices and express the product of A and B in terms of these $n/2 \times n/2$ matrices as:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

• If we treat A and B as 2×2 matrices, whose elements are $n/2 \times n/2$ matrices, then the C can be expressed in terms of sums and products of these $n/2 \times n/2$ matrices:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

 $C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$
 $C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$
 $C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$

11/28

Recursive algorithm for matrix multiplication:

```
def MAT-MULT (A, B):
   n = A.rows
   C = \text{new (n} \times \text{n) matrix}
   if n == 1: C_{11} = A_{11} \cdot B_{11}
   else: // partition A, B and C
       C_{11}=MAT-MULT (A_{11}, B_{11}) + MAT-MULT (A_{12}, B_{21})
      C_{12}=MAT-MULT (A_{11}, B_{12}) + MAT-MULT (A_{12}, B_{22})
      C_{21}=MAT-MULT (A_{21}, B_{11}) + MAT-MULT (A_{22}, B_{21})
      C_{22}=MAT-MULT (A_{21}, B_{12}) + MAT-MULT (A_{22}, B_{22})
   return C
```

12/28

- Analysis of the recursive algorithm for matrix multiplication.
 - If n = 1, we do only one scalar multiplication -> $T(1) = \Theta(1)$
 - For n > 1, each recursive call multiplies two $n/2 \times n/2$ matrices contributing T(n/2) time. There are 8 such recursive calls -> 8T(n/2).
 - Four matrix additions take $\Theta(4(n/2)^2) = \Theta(n^2)$

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1\\ 8T(n/2) + \Theta(n^2), & \text{if } n > 1 \end{cases}$$



- In 1969, Strassen proposed an algorithm which is faster than the recursive matrix multiplication. It has four steps:
 - Step 1. Divide input matrices A and B into $n/2 \times n/2$ matrices.
 - Step 2. Create 10 matrices $S_1, S_2, \dots S_{10}$, each of which is sum or difference of the matrices created at Step 1.
 - Step 3. Using matrices from Step 1 and Step 2 compute 7 matrices $P_1, P_2, \dots P_7$.
 - Step 4. Compute C_{11} , C_{12} , C_{21} , C_{22} by adding and subtracting various combinations of P_i matrices.

Strassen algorithm matrix computation:

Step 1:
$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$
 $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$

Step 2:
$$S_1 = B_{12} - B_{22}$$
 $S_2 = A_{11} + A_{12}$ $S_3 = A_{21} + A_{22}$ $S_4 = B_{21} - B_{11}$ $S_5 = A_{11} + A_{22}$ $S_6 = B_{11} + B_{22}$ $S_7 = A_{12} - A_{22}$ $S_8 = B_{21} + B_{22}$ $S_9 = A_{11} - A_{21}$ $S_{10} = B_{11} + B_{12}$

15/28

Strassen algorithm matrix computation: Step 3:

$$P_1 = A_{11}S_1$$
 $P_2 = S_2B_{22}$ $P_3 = S_3B_{11}$ $P_4 = A_{22}S_4$ $P_5 = S_5S_6$ $P_6 = S_7S_8$ $P_7 = S_9S_{10}$

Step 4:

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_1 + P_5 - P_3 - P_7$$

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$C_{16/28}$$

Strassen algorithm matrix computation - check:

$$C_{12} = P_1 + P_2 = A_{11}S_1 + S_2B_{22}$$

$$= A_{11}B_{12} - A_{11}B_{22} + A_{11}B_{22} + A_{12}B_{22}$$

$$= A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = P_3 + P_4 = S_3 B_{11} + A_{22} S_4$$

= $A_{21} B_{11} + A_{22} B_{11} + A_{22} B_{21} - A_{22} B_{11}$
= $A_{21} B_{11} + A_{22} B_{21}$

- o Strassen algorithm analysis.
 - If n = 1, we do only one scalar multiplication -> $T(1) = \Theta(1)$
 - For n > 1, at step 2 each recursive call multiplies two $n/2 \times n/2$ matrices contributing T(n/2) time. There are 7 such recursive calls -> 7T(n/2). The number of additions is 18.
 - Matrix additions take $\Theta(18(n/2)^2) = \Theta(n^2)$

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2), & \text{if } n > 1 \end{cases}$$



SOLVING RECURRENCES

- A recurrence is an equation that describes a function in terms of its value in smaller inputs.
- There are three main methods for solving recurrences:
 - In the substitution method, we guess a bound and then use induction to prove it.
 - The recursion tree method converts the recurrence into a tree and uses bounding summations.
 - The master method provides bounds for recurrences of the form: T(n) = aT(n/h) + f(n)

$$T(n) = aT(n/b) + f(n)$$

 The master method depends on the following theorem:

THEOREM* (Master theorem)

Let $a \ge 1, b > 1$ and c > 1 be constants, and let T(n) be defined on the nonnegative integers by the recurrence:

$$T(n) = \begin{cases} b, & \text{if } n = 1\\ aT(n/c) + bn, & \text{if } n > 1 \end{cases}$$



^{*} Simplified version

o (theorem continuation)

Then, if n is a power of c, T(n) has the following asymptotic bounds:

$$T(n) = \begin{cases} O(n), & \text{if } a < c, \\ O(n \log n), & \text{if } a = c, \\ O(n^{\log_c a}), & \text{if } a > c, \end{cases}$$

- Lets find the solution for our algorithms:
 - MERGE SORT.

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1\\ 2T(n/2) + \Theta(n), & \text{if } n > 1 \end{cases}$$

we have a=c, therefore (according to the second row)

$$T(n) = O(n \log n)$$



- Lets find the solution for our algorithms:
 - RECURSIVE MATRIX MULTIPLICATION.

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ 8T(n/2) + \Theta(n^2), & \text{if } n > 1 \end{cases}$$

we have a=8>c=2, therefore (according to the third row)

$$T(n) = O(n^{\log_c a}) = O(n^{\log_2 8}) = O(n^3)$$



- Lets find the solution for our algorithms:
 - STRASSEN ALGORITHM.

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2), & \text{if } n > 1 \end{cases}$$

we have a=7>c=2, therefore (according to the third row)

$$T(n) = O(n^{\log_c a}) = O(n^{\log_2 7}) = O(n^{2.81})$$



VINOGRAD ALGORITHM

 The Vinograd algorithm is a variant of the Strassen algorithm which requires (the same) 7 multiplications, but only 15 additions/subtractions.

 Vinograd algorithm complexity is the same, but the reduced number of additions/subtractions has practical significance.



DISCUSSION

- There are two key issues when efficiently applying Strassen algorithm to arbitrary matrices.
 - First the constraint that the matrix size be a power of 2 must be handled.
 - One solution zero padding.
 - The second key issue for efficiency of Strassen algorithm is controlling the depth of recursion.
 - \circ For small n, Strassen algorithm is actually slower!

DISCUSSION

- Matrix multiplication is a fundamental operation and is critical when attempting to speed up scientific computations.
- The performance of matrix multiplication is dependent on two elements:
- √the operation count and
- ✓ the memory reference count.
- Minimizing both of these factors will produce an optimal algorithm.

THAT'S ALL FOR TODAY!

