Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281

ผู้สอน: ตอน 1 ผศ. เบญจมาศ ปัญญางาม

ตอน 2 ผศ. ดร. จักริน ชวชาติ

บทที่ 5

อัลกอริทึมแบ่งแยกและเอาชนะ

(Divide and Conquer algorithms Part2)

Matrix multiplication

Matrix multiplication. Given two n-by-n matrices A and B, compute C = AB , $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

Grade-school. $\Theta(n^3)$ arithmetic operations.

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Is "grade-school" matrix multiplication algorithm asymptotically optimal?

Matrix multiplication in sub-quadratic time : Brute Force

Matrix multiplication. Given two n-by-n matrices A and B, compute C = AB , $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

Grade-school. $\Theta(n^3)$ arithmetic operations.

Block matrix multiplication

C_{11} A_{11} A_{12} B_{11}

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

B_{21}

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Block matrix multiplication: warmup

- To multiply two n-by-n matrices A and B:
  - Divide: partition A and B into ½n-by-½n blocks.
  - Conquer: multiply 8 pairs of ½n-by-½n matrices, recursively.
  - Combine: add appropriate products using 4 matrix additions.

n-by-n matrices

$$C = A \times B$$

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

½n-by-½n matrices

8 matrix multiplications (of ½n-by-½n matrices)

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

4 matrix additions (of ½n-by-½n matrices)

Block matrix multiplication: warmup

- Block matrix multiplication
  - 8 Matrix multiplication of n/2*n/2 matirces
  - 4 Matrix Addition of n/2*n/2 matirces

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}}$$

Running time = ?

Strassen's trick

- Key idea. Can multiply two 2-by-2 matrices via 7 scalar multiplications (plus 11 additions and 7 subtractions).

scalars

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

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

7 scalar multiplications

Strassen's trick

- Key idea. Can multiply two n-by-n matrices via 1/2n-by-1/2n matrix scalar multiplications (plus 11 additions and 7 subtractions).

½n-by-½n matrices

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

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

7 matrix multiplications (of ½n-by-½n matrices)

Strassen's algorithm

assume n is a power of 2

STRASSEN(n, A, B)

IF ($n = 1$) RETURN $A \times B$.

Partition A and B into ½n-by-½n blocks.

$P_1 \leftarrow$ STRASSEN($n / 2, A_{11}, (B_{12} - B_{22})$).

$P_2 \leftarrow$ STRASSEN($n / 2, (A_{11} + A_{12}), B_{22}$).

$P_3 \leftarrow$ STRASSEN($n / 2, (A_{21} + A_{22}), B_{11}$).

$P_4 \leftarrow$ STRASSEN($n / 2, A_{22}, (B_{21} - B_{11})$).

$P_5 \leftarrow$ STRASSEN($n / 2, (A_{11} + A_{22}), (B_{11} + B_{22})$).

$P_6 \leftarrow$ STRASSEN($n / 2, (A_{12} - A_{22}), (B_{21} + B_{22})$).

$P_7 \leftarrow$ STRASSEN($n / 2, (A_{11} - A_{21}), (B_{11} + B_{12})$).

← $7\,T(n/2) + \Theta(n^2)$

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

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

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

← $\Theta(n^2)$

RETURN C.

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

Analysis of Strassen's algorithm

- Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n-by-n matrices.

Gaussian Elimination is not Optimal

VOLKER STRASSEN*

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices A and B of order n from the coefficients of A and B with less than $4.7 \cdot n^{\log 7}$ arithmetical operations (all logarithms in this paper are for base 2, thus $\log 7 \approx 2.8$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order n, solving a system of n linear equations in n unknowns, computing a determinant of order n etc. all requiring less than const $n^{\log 7}$ arithmetical operations.

Analysis of Strassen's algorithm

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n-by-n matrices.

- Pf.
- When n is a power of 2, apply Case 1 of the master theorem:

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

- When n is not a power of 2, pad matrices with zeros to be n'-by-n',where n ≤ n' < 2n and n' is a power of 2.

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

History

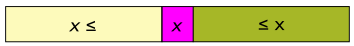
year	algorithm	arithmetic operations
1858	"grade school"	$O(n^3)$
1969	Strassen	$O(n^{2.808})$
1978	Pan	$O(n^{2.796})$
1979	Bini	$O(n^{2.780})$
1981	Schönhage	$O(n^{2.522})$
1982	Romani	$O(n^{2.517})$
1982	Coppersmith-Winograd	$O(n^{2.496})$
1986	Strassen	$O(n^{2.479})$
1989	Coppersmith-Winograd	$O(n^{2.3755})$
2010	Strother	$O(n^{2.3737})$
2011	Williams	$O(n^{2.372873})$
2014	Le Gall	$O(n^{2.372864})$
	???	$O(n^{2+\varepsilon})$

Quicksort expected running time analysis

The idea of Quicksort

- Sorts "in place" (like insertion sort)
- Based on the D&C paradigm like merge sort

- Divide: Partition the array into 2 subarrays around a pivot x such that elements in lower subarray ≤ x ≤ elements in upper subarray.



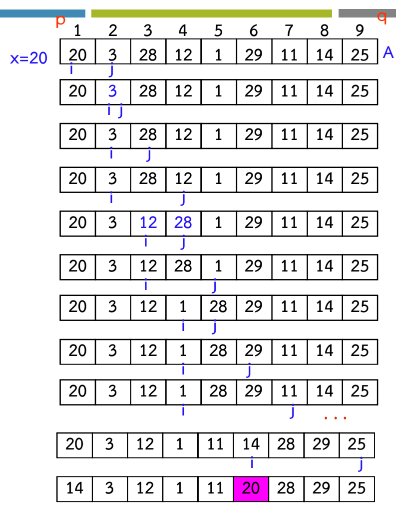
- Conquer: Recursively sort the 2 subarrays.
- Combine: No need
- Key:Linear-time partitioning subroutine.

- Input: An array A and indices p and r
  Output: An sorted array A

```
QuickSort(A, p, r)
    if p < r then
        q = Partition(A,p,r)
        QuickSort(A,p,q-1)
        QuickSort(A,q+1,r)

Initial call:QuickSort(A,1,n)
```

```
Partition(A, p, q) //A[p. . q]
    x=A[p]  //pivot-> A[p]
    i=p /* i->splitpoint*/
    for j=p+1 to q // j-> unknown
      if A[j] ≤ x then
        i=i+ 1
        swap(A[i] ,A[j])
    swap(A[p],A[i])
    return(i)
```

Quicksort expected running time analysis

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms when duplicate input elements exist.
- Best case : Occurs when the subarrays are completely balanced every time.
- Each subarray has ≤ n/2 elements.
- Let T(n)=best-case running time on an array of n elements

Quicksort expected running time analysis

- Let T(n) = worst-case running time on an array of n elements
  - Input sorted or reverse sorted.
  - Partition around min or max element.
  - One side of partition always has no elements

Quicksort expected running time analysis

- Randomizedquicksort : Randomized Algorithm
  - Partition around a random element.
- Running time is independent of the input order.
- T(n)= O(n log n)
- The worst case is determined only by the output of a random-number generator

RandomizedPartition(A, p, r)

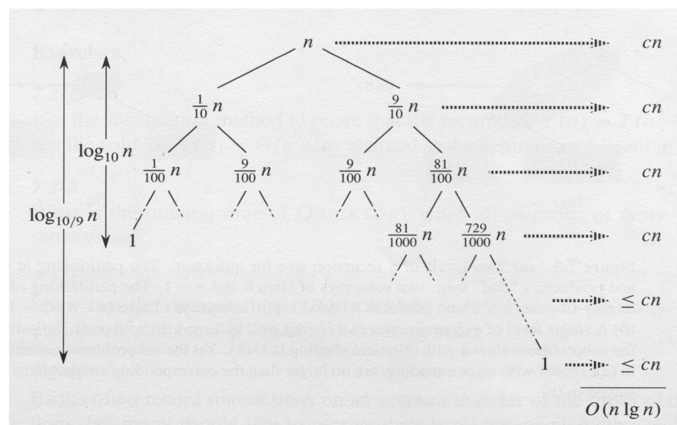
1. i = Random(p, r);
2. swap(A[p],A[i]);
3. Partition(A, p, r)

Quicksort expected running time analysis

Balanced partitioning

- Quick sort 's average running time is much closer to the best case than to the worst case.
  - Imagine that PARTITION always produces a 9-to-1 split.

$$T(n) \le T(9n/10) + T(n/10) + \Theta(n)$$
$$= \Theta(n \log n)$$



- Consider the modified version of binary search.
- Let us assume that the array is divide into 3 equal parts (ternary search) instead of two equal parts.
- Write the recurrence for this ternary search and find its complexity.

Binary search : Time Complexity Analysis

- Binary search has the recurrence relation:

$$T(n) = T(\frac{n}{2}) + O(1)$$

- Instead of "2" in the recurrence relation we need use "3". That indicates that we are dividing the array into 3 sub-arrays with equal and considering only one of them.
- So, the recurrence for the ternary search can be given as

$$T(n) = T(\frac{n}{3}) + O(1)$$

- Using Master theorem, we get the complexity as $O(\log_3 n) = O(\log n)$

Binary search : Time Complexity Analysis

- For previous problem, what if we divide the array into two sets of sizes approximately one-third and two-thirds.
- We now consider a slightly modified version of ternary search which only one comparison is made which creates two partitions, one of roughly n/3 elements and the other of 2n/3.
- Here the worst case comes when the recursive call is on the larger 2n/3 element part. So the recurrence corresponding to the worst case is

Binary search : Time Complexity Analysis

- Using master method, we get the complexity as O(log n)
- It is interesting to note that we will get the same results for general k-ary search (as long as k is a fixed constant which does not depend on n) as n approaches infinity.