Graduate Algorithms

1/11/10

15-750

Gary Miller GHC 8109

Office hours T 11-12 Th 2-3

Web Page ~glmiller

TAs Richard Peng
Aaron Roth

Class MWF 1:30-3 GHC 4215

HW	Every 2 weeks	25%
	2 Midterms	40%
	1 Final	35%

Course Goals

1) Understand many known:
   a) Algorithms
   b) Design Techniques
2) Analyze algorithm efficiency
3) Algorithm correctness
4) Communicate about code
5) Know key words
6) Design your own alg.

Machine Models

RAM = Random Access Machine (C)

not considered

- Caching models
- memory hieracchy
- pipelining

Parallel Models

PRAM

Circuits

Asymptotic Complexity

Def:

$f(n) \in O(g(n))$ if

$$\exists c, n_0 \geq 0 \; \forall n \geq n_0 \quad f(n) \leq c\, g(n)$$

$$O(g(n)) = \{ f(n) \mid f(n) \in O(g) \}$$

$f \in o(g(n))$ if

$$\forall c > 0 \; \exists n_0 \geq 0 \; \forall n \geq n_0 \quad f(n) \leq c\, g(n)$$

---

1) $f \in \Omega(g)$ if $g \in O(f)$

2) $f \in \Omega(g)$ if

$$\exists c \geq 0 \; \forall n_0 \geq 0 \; \exists n_1 \geq n_0 \quad f(n_1) \geq c\, g(n_1)$$

(infinitely often)

Matrix Multiplication

Naive A, B are $n \times n$ matrices

Def $AB = C$ if $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

n^3 multiplications $(n-1)n^2$ additions

$O(n^3)$ operations

Recursive Alg $M(A,B)$ $n = 2^k$

1) if A is 1×1 then return $a_{11} \cdot b_{11}$

2) write $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$

A_{ij} are $n/2 \times n/2$ B_{ij} are $n/2 \times n/2$

3) $C_{ij} = M(A_{i1}, B_{1j}) + M(A_{i2}, B_{2j})$

4) return $\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

Correctness induction on n.

$n=1$ done

assume $M(A,B) = A \cdot B \quad n < n_0$

we know $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$

$\therefore \quad C_{ij} = M(A_{i1}, B_{1j}) + M(A_{i2}, B_{2j})$

Timing Let $T(n)$ = number of ops for $n \times n$

$T(n) \leq 8T(n/2) + cn^2$ & $T(1) = 1$

$\Rightarrow T(n) = O(n^3)$

Consider recurrence

$T(n) = 7T(n/2) + cn^2$ & $T(1) = 1$

$\Rightarrow T(n) = O(n^{\log_2 7})$

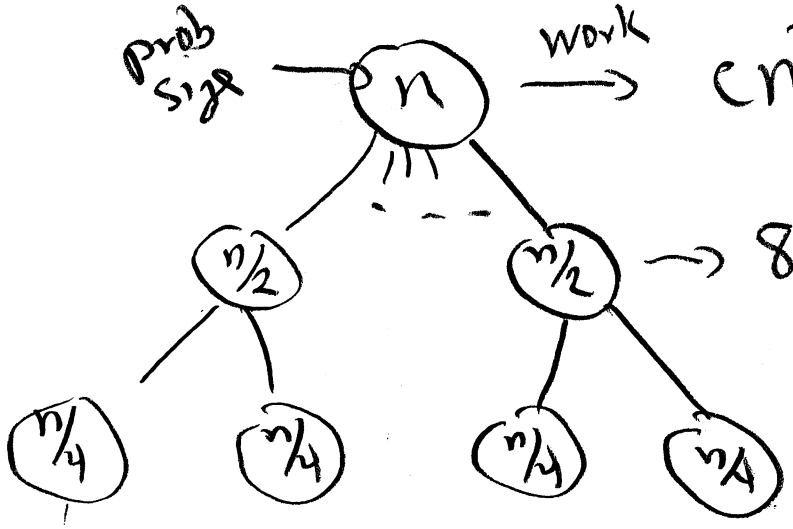
Solving Recurrences

Methods 1) Use formula

2) Induction on n

3) consider tree of recursive calls

3)



(1)

(1) $2^{\log n} cn^2$

$O(n^3)$

For 7 calls

$$cn^2$$

$$7c(n/2)^2 = \frac{7}{4}cn^2$$

$$7^2c(n/4)^2 = \left(\frac{7}{4}\right)^2 cn^2$$

$$\vdots$$

$$\left(\frac{7}{4}\right)^{\log n} cn^2 = \frac{n^{\log 7}}{n^{\log 4}} cn^2$$

$$= cn^{\log 7}$$

Total $O(n^{\log 7})$

improving technology in the design and analysis of algorithms than any other mathematical abstraction. On the other hand, one should be aware of its limitations and realize that an asymptotically optimal solution is not necessarily the best one.

A good rule of thumb in the design and analysis of algorithms, as in life, is to use common sense, exercise good taste, and always listen to your conscience.

1.4 Strassen's Matrix Multiplication Algorithm

Probably the single most important technique in the design of asymptotically fast algorithms is *divide-and-conquer*. Just to refresh our understanding of this technique and the use of recurrences in the analysis of algorithms, let's take a look at Strassen's classical algorithm for matrix multiplication and some of its progeny. Some of these examples will also illustrate the questionable lengths to which asymptotic analysis can sometimes be taken.

The usual method of matrix multiplication takes 8 multiplications and 4 additions to multiply two 2×2 matrices, or in general $O(n^3)$ arithmetic operations to multiply two $n \times n$ matrices. However, the number of multiplications can be reduced. Strassen [97] published one such algorithm for multiplying 2×2 matrices using only 7 multiplications and 18 additions:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} s_1 + s_2 - s_4 + s_6 & s_4 + s_5 \\ s_6 + s_7 & s_2 - s_3 + s_5 - s_7 \end{bmatrix}$$

where

$$\begin{aligned} s_1 &= (b-d)\cdot(g+h) \\ s_2 &= (a+d)\cdot(e+h) \\ s_3 &= (a-c)\cdot(e+f) \\ s_4 &= \text{✗}\cdot(a+b)\cdot h \\ s_5 &= a\cdot(f-h) \\ s_6 &= d\cdot(g-e) \\ s_7 &= \text{✗}\cdot(c+d)\cdot e \end{aligned}$$

Assume for simplicity that n is a power of 2. (This is not the last time you will hear that.) Apply the 2×2 algorithm recursively on a pair of $n \times n$ matrices by breaking each of them up into four square submatrices of size $\frac{n}{2} \times \frac{n}{2}$:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} S_1 + S_2 - S_4 + S_6 & S_4 + S_5 \\ S_6 + S_7 & S_2 - S_3 + S_5 - S_7 \end{bmatrix}$$

where

$$S_1 = (B-D)\cdot(G+H)$$

$$\begin{aligned} S_2 &= (A+D)\cdot(E+H) \\ S_3 &= (A-C)\cdot(E+F) \\ S_4 &= \text{✗}\cdot(A+B)\cdot H \\ S_5 &= A\cdot(F-H) \\ S_6 &= D\cdot(G-E) \\ S_7 &= \text{✗}\cdot(C+D)\cdot E \end{aligned}$$

Everything is the same as in the 2×2 case, except now we are manipulating $\frac{n}{2} \times \frac{n}{2}$ matrices instead of scalars. (We have to be slightly cautious, since matrix multiplication is not commutative.) Ultimately, how many scalar operations $(+, -, \cdot)$ does this recursive algorithm perform in multiplying two $n \times n$ matrices? We get the recurrence

$$T(n) = 7T(\frac{n}{2}) + dn^2$$

with solution

$$\begin{aligned} T(n) &= (1 + \frac{4}{3}d)n^{\log_2 7} + O(n^2) \\ &= O(n^{\log_2 7}) \\ &= O(n^{2.81\ldots}) \end{aligned}$$

which is $o(n^3)$. Here d is a fixed constant, and dn^2 represents the time for the matrix additions and subtractions.

This is already a significant asymptotic improvement over the naive algorithm, but can we do even better? In general, an algorithm that uses c multiplications to multiply two $d \times d$ matrices, used as the basis of such a recursive algorithm, will yield an $O(n^{\log_d c})$ algorithm. To beat Strassen's algorithm, we must have $c < d^{\log_2 7}$. For a 3×3 matrix, we need $c < 3^{\log_2 7} = 21.8\ldots$, but the best known algorithm uses 23 multiplications.

In 1978, Victor Pan [83, 84] showed how to multiply 70×70 matrices using 143640 multiplications. This gives an algorithm of approximately $O(n^{2.795\ldots})$. The asymptotically best algorithm known to date, which is achieved by entirely different methods, is $O(n^{2.376\ldots})$ [25]. Every algorithm must be $\Omega(n^2)$, since it has to look at all the entries of the matrices; no better lower bound is known.

C_i can be found in a constant number of bit vector steps, and line 6 requires $O_{BV}(n)$. Therefore the loop of lines 1–6 is $O_{BV}(n^2/\log n)$ and line 7 is of the same complexity. □

EXERCISES

6.1 Show that the integers modulo n form a ring. That is, $\mathbf{Z}_n$ is the ring $(\{0, 1, \ldots, n-1\}, +, \cdot, 0, 1)$, where $a + b$ and $a \cdot b$ are ordinary addition and multiplication modulo n.

6.2 Show that M_n, the set of $n \times n$ matrices with elements chosen from some ring R, itself forms a ring.

6.3 Give an example to show that the product of matrices is not commutative, even if the elements are chosen from a ring in which multiplication is commutative.

6.4 Use Strassen's algorithm to compute the product

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}.$$

6.5 Another version of Strassen's algorithm uses the following identities to help compute the product of two 2×2 matrices.

$$\begin{aligned}
s_1 &= a_{21} + a_{22} & m_1 &= s_2 s_6 & t_1 &= m_1 + m_2 \\
s_2 &= s_1 - a_{11} & m_2 &= a_{11} b_{11} & t_2 &= t_1 + m_4 \\
s_3 &= a_{11} - a_{21} & m_3 &= a_{12} b_{21} & & \\
s_4 &= a_{12} - s_2 & m_4 &= s_3 s_7 & & \\
s_5 &= b_{12} - b_{11} & m_5 &= s_1 s_5 & & \\
s_6 &= b_{22} - s_5 & m_6 &= s_4 b_{22} & & \\
s_7 &= b_{22} - b_{12} & m_7 &= a_{22} s_8 & & \\
s_8 &= s_6 - b_{21} & & & &
\end{aligned}$$

The elements of the product matrix are:

$$\begin{aligned}
c_{11} &= m_2 + m_3, \\
c_{12} &= t_1 + m_5 + m_6, \\
c_{21} &= t_2 - m_7, \\
c_{22} &= t_2 + m_5.
\end{aligned}$$

Show that these elements compute Eq. (6.1). Note that only 7 multiplications and 15 additions have been used.

† We can get around the detail that NUM($\mathbf{a}_i$) is the integer representing the reverse of $\mathbf{a}_i$ by taking the "jth row" of B_i to be the jth row from the bottom instead of the top as we have previously done.

What is a Space Efficient Strassen?

1) Add in place
2) Malloc $3n^2$ space per call.
3) Do final additions in output space of parent.

$W(n)$ be space used

$$W(n) = 3n^2 + W(n/2)$$

$$= 3n^2 + 3(n/2)^2 + 3(n/4)^2 + \cdots$$

$$= 3n^2(1 + 1/4 + 1/16 + \cdots)$$

---

note $1 + \alpha + \alpha^2 + \cdots = \frac{1}{1-\alpha} \quad \alpha < 1$

$$= 3n^2\left(\frac{1}{1-1/4}\right) = 3n^2(4/3) = 4n^2$$