Discrete Mathematics: Lecture 4

- Last time:
  - Chap 8.1: Applications of recurrence relations
  - Chap 8.2: Solving linear recurrence relations
- Today:
  - Chap 8.3: Divide-and-conquer algorithms and recurrence relations
- Assignment 2 due in two weeks

Review of last time

- Modeling with recurrence relation
- Solving linear homogeneous recurrence relations(线性齐次递推关系) with constant coefficients
- Solving linear nonhomogeneous recurrence relations with constant coefficients
  - the general form of solutions
  - the form of a particular solution when $F(n) = p(n)s^n$
  - example: $a_n = a_{n-1} + n$, find all solutions, find the solution s.t. $a_1 = 1$

Divide-and-conquer algorithms (分治算法)

- Many recursive algorithms take a problem with a given input and divide it into one or more smaller problems.
- Examples: binary search, merge sort
- Divide-and-conquer algorithms
  - divide a problem into one or more instances of the same problem of smaller size and
  - conquer the problem by using the solutions of the smaller problems to find a solution of the original problem, perhaps with some additional work
- We will use recurrence relations to analyze the computational complexity of divide-and-conquer algorithms.

Divide-and-conquer recurrence relations

- Suppose that a recursive algorithm divides a problem of size n into a subproblems, each of size n/b, and
- $g(n)$ extra operations are required in the conquer step
- Let $f(n)$ represent the number of operations required to solve a problem of size n
- Then $f(n) = af(n/b) + g(n)$, called a divide-and-conquer recurrence relation
- Examples: binary search, merge sort, finding the maximum and minimum of a sequence

Example: fast multiplication of integers

- $a = (a_{2n-1}a_{2n-2}\ldots a_1a_0)_2,\ b = (b_{2n-1}b_{2n-2}\ldots b_1b_0)_2$
- $a = 2^nA_1 + A_0,\ b = 2^nB_1 + B_0,$ where
- $A_1 = (a_{2n-1}\ldots a_{n+1}a_n)_2,\ A_0 = (a_{n-1}\ldots a_1a_0)_2,$
- $B_1 = (b_{2n-1}\ldots b_{n+1}b_n)_2,\ B_0 = (b_{n-1}\ldots b_1b_0)_2,$
- $ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0$
- $f(2n) = 3f(n) + Cn$

Example: fast matrix multiplication

- Multiplying two $n \times n$ matrices use $O(n^3)$ operations
- There are more efficient divide-and-conquer algorithms
- Reduce the multiplication of two $n \times n$ matrices to 7 multiplications of two $(n/2) \times (n/2)$ matrices and 15 additions of $(n/2) \times (n/2)$ matrices
- $f(n) = 7f(n/2) + 15n^2/4$

Theorem 1

Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + c,$$

whenever b divides n, $a \geq 1$, $b > 1$ is an integer, and c is a positive real number. Then

$$f(n) \text{ is } O(n^{\log_b a}) \text{ if } a > 1, \text{ and } O(\log n) \text{ if } a = 1.$$

Furthermore, when $n = b^k$, where k is a positive integer,

$$f(n) = C_1 n^{\log_b a} + C_2,$$

where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$.

Examples of applying Theorem 1

- Let $f(n) = 5f(n/2) + 3$ and $f(1) = 7$. Find $f(2^k)$. Also, estimate $f(n)$ if f is an increasing function.
- Binary search
- Finding the maximum and minimum of a sequence

Master Theorem

Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + cn^d,$$

whenever $n = b^k$, where k is a positive integer, $a \geq 1$, b is an integer greater than 1, c is a positive real number, and d is a nonnegative real number. Then

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

Proof: By iterative approach, we can show that if $n = b^k$, then

1. if $a = b^d$, then $f(n) = f(1)n^d + cn^d \log_b n$
2. if $a \neq b^d$, then $f(n) = C_1 n^d + C_2 n^{\log_b a}$, where C_1 and C_2 are certain constants

Examples of applying Master Theorem

- Merge sort
- Fast integer multiplication
- Fast matrix multiplication