4190.101
Discrete Mathematics

Chapter 8 Advanced Counting Techniques

Gunhee Kim

[Based on S. Srinivasa Rao's Lecture Notes]

Divide-and-Conquer Algorithms and Recurrence Relations

Section 8.3

Section Summary

- Divide-and-Conquer Algorithms and Recurrence Relations
- Examples
  - Binary Search
  - Merge Sort
  - Fast Multiplication of Integers
- Master Theorem
- Closest Pair of Points (*not covered yet in these slides*)

Divide-and-Conquer Algorithmic Paradigm

- **Definition**: A *divide-and-conquer algorithm* works by first *dividing* a problem into one or more instances of the same problem of smaller size and then *conquering* the problem using the solutions of the smaller problems to find a solution of the original problem.
- **Examples**:
  - Binary search, covered in Chapters 3 and 5: It works by comparing the element to be located to the middle element. The original list is then split into two lists and the search continues recursively in the appropriate sublist.
  - Merge sort, covered in Chapter 5: A list is split into two approximately equal sized sublists, each recursively sorted by merge sort. Sorting is done by successively merging pairs of lists.

Divide-and-Conquer Recurrence Relations

- Suppose that a recursive algorithm divides a problem of size n into a subproblems.
- Assume each subproblem is of size n/b.
- Suppose $g(n)$ extra operations are needed in the conquer step.
- Then $f(n)$, the number of operations to solve a problem of size n, satisfies the following recurrence relation:

$$f(n) = af(n/b) + g(n)$$

- This is called a *divide-and-conquer recurrence relation.*

Example: Binary Search

- Binary search reduces the search for an element in a sequence of size n to the search in a sequence of size $n/2$. Two comparisons are needed to implement this reduction;
  - One to decide whether to search the upper or lower half of the sequence and
  - The other to determine if the sequence has elements.
- Hence, if $f(n)$ is the number of comparisons required to search for an element in a sequence of size n, then

$$f(n) = f(n/2) + 2$$

when n is even.

Example: Merge Sort

- The merge sort algorithm splits a list of n (assuming n is even) items to be sorted into two lists with $n/2$ items. It uses fewer than n comparisons to merge the two sorted lists.
- Hence, the number of comparisons required to sort a sequence of size n, is no more than than $M(n)$ where

$$M(n) = 2M(n/2) + n.$$

Example: Fast Multiplication of Integers

- An algorithm for the fast multiplication of two $2n$-bit integers (assuming n is even) first splits each of the $2n$-bit integers into two blocks, each of n bits.
- Suppose that a and b are integers with binary expansions of length $2n$. Let

  $a = (a_{2n-1}a_{2n-2} \ldots a_1a_0)_2$ and $b = (b_{2n-1}b_{2n-2} \ldots b_1b_0)_2$.
- Let $a = 2^nA_1 + A_0$, $b = 2^nB_1 + B_0$, where

  $A_1 = (a_{2n-1} \ldots a_{n+1}a_n)_2$, $A_0 = (a_{n-1} \ldots a_1a_0)_2$,

  $B_1 = (b_{2n-1} \ldots b_{n+1}b_n)_2$, $B_0 = (b_{n-1} \ldots b_1b_0)_2$.
- The algorithm is based on the fact that ab can be rewritten as:

  $ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0$.
- This identity shows that the multiplication of two $2n$-bit integers can be carried out using three multiplications of n-bit integers, together with additions, subtractions, and shifts.
- Hence, if $f(n)$ is the total number of operations needed to multiply two n-bit integers, then

  $f(2n) = 3f(n) + Cn$

  where Cn represents the total number of bit operations; the additions, subtractions and shifts that are a constant multiple of n-bit operations.

Estimating the Size of Divide-and-Conquer Functions

- **Theorem 1**: Let *f* be an increasing function that satisfies the recurrence relation

$$f(n) = af\left(\frac{n}{b}\right) + c$$

whenever n is divisible by b, where $a \geq 1$, b is an integer greater than 1, and c is a positive real number.

- Then

$$f(n) = \begin{cases} O\left(n^{\log_b a}\right) \text{ if } \ a > 1 \\ O(\log n) \quad \text{if } \ a = 1 \end{cases}$$

- Furthermore, when $n = b^k$ and $a \neq 1$, 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 $2 = -c/(a-1)$.

Complexity of Binary Search

- **Binary Search Example**: Give a big-*O* estimate for the number of comparisons used by a binary search.
- **Solution**: Since the number of comparisons used by binary search is $f(n) = f(n/2) + 2$ where n is even, by Theorem 1, it follows that $f(n)$ is $O(\log n)$.

Estimating the Size of Divide-and-conquer Functions

- **Theorem 2. Master Theorem**: Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af\left(\frac{n}{b}\right) + cn^d$$

whenever $n = b^k$, where k is a positive integer greater than 1, and c and d are real numbers with c positive and d nonnegative. Then

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

Complexity of Merge Sort

- **Merge Sort example**: Give a big-*O* estimate for the number of comparisons used by merge sort.
- **Solution**: Since the number of comparisons used by merge sort to sort a list of n elements is less than $M(n)$ where $M(n) = 2M(n/2) + n$, by the master theorem $M(n)$ is $O(n \log n)$.

Complexity of Fast Integer Multiplication Algorithm

- **Integer Multiplication Example**: Give a big-*O* estimate for the number of bit operations used needed to multiply two *n*-bit integers using the fast multiplication algorithm.
- **Solution**: We have shown that $f(n) = 3f(n/2) + Cn$, when n is even, where $f(n)$ is the number of bit operations needed to multiply two n-bit integers. Hence by the master theorem with $a = 3$, $b = 2$, $c = C$, and $d = 1$ (so that we have the case where $a > b^d$), it follows that $f(n)$ is $O(n^{\log 3})$.
- Note that $\log 3 \approx 1.6$. Therefore the fast multiplication algorithm is a substantial improvement over the conventional algorithm that uses $O(n^2)$ bit operations.

Inclusion-Exclusion

Section 8.5

Section Summary

- The Principle of Inclusion-Exclusion
- Examples

Principle of Inclusion-Exclusion

- In Section 2.2, we developed the following formula for the number of elements in the union of two finite sets:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

- We will generalize this formula to finite sets of any size.

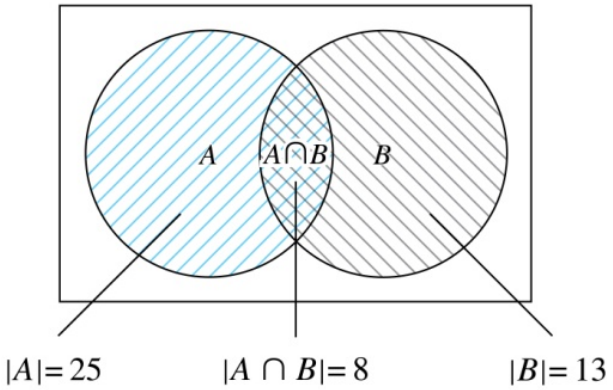
Two Finite Sets

- **Example**: In a discrete math class every student is a major in computer science (CS) or math or both. The number of students having CS as a major (possibly along with math) is 25; the number of students having math as a major (possibly along with CS) is 13; and the number of students majoring in both CS and math is 8. How many students are in the class?

- **Solution**: $|A \cup B| = |A| + |B| - |A \cap B|$

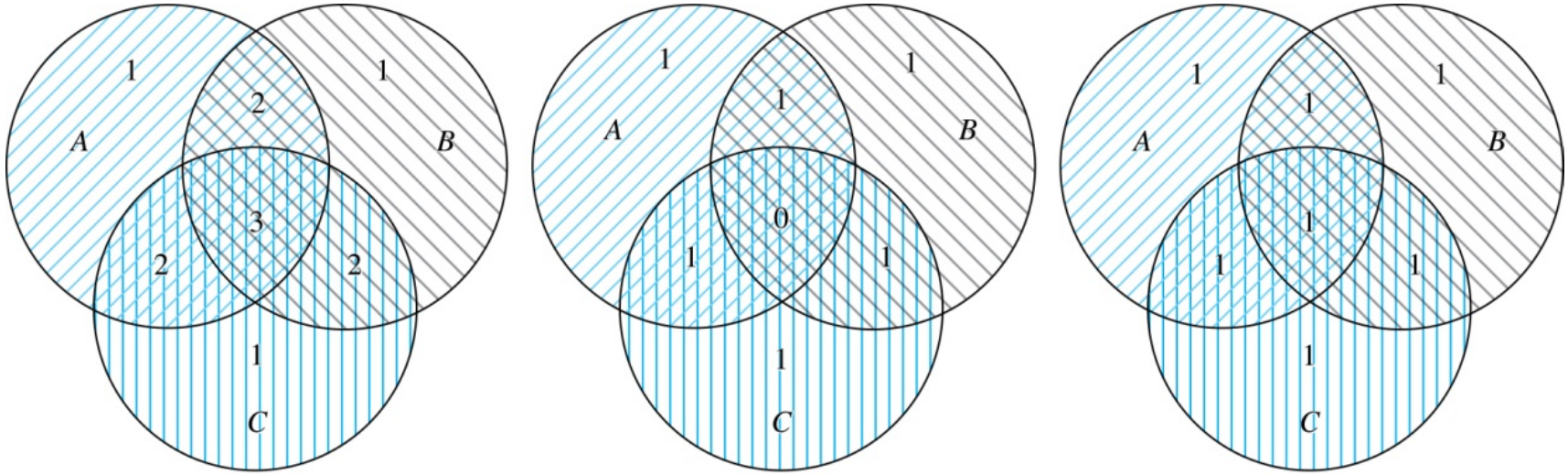
$$= 25 + 13 - 8 = 30$$

$|A \cup B| = |A| + |B| - |A \cap B| = 25 + 13 - 8 = 30$



Three Finite Sets

$$|A \cup B \cup C| =$$
$$|A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$



(a) Count of elements by $|A|+|B|+|C|$

(b) Count of elements by $|A|+|B|+|C|-|A\cap B|-|A\cap C|-|B\cap C|$

(c) Count of elements by $|A|+|B|+|C|-|A\cap B|-|A\cap C|-|B\cap C|+|A\cap B\cap C|$

Three Finite Sets

- **Example**: 1232 students have taken a course in Spanish, 879 have taken a course in French, and 114 have taken a course in Russian. Further, 103 have taken courses in both Spanish and French, 23 have taken courses in both Spanish and Russian, and 14 have taken courses in both French and Russian. If 2092 students have taken a course in at least one of Spanish, French and Russian, how many students have taken a course in all 3 languages.
- **Solution**: Let S be the set of students who have taken a course in Spanish, F the set of students who have taken a course in French, and R the set of students who have taken a course in Russian. Then, $|S| = 1232$, $|F| = 879$, $|R| = 114$, $|S \cap F| = 103$, $|S \cap R| = 23$, $|F \cap R| = 14$, and $|S \cup F \cup R| = 23$.

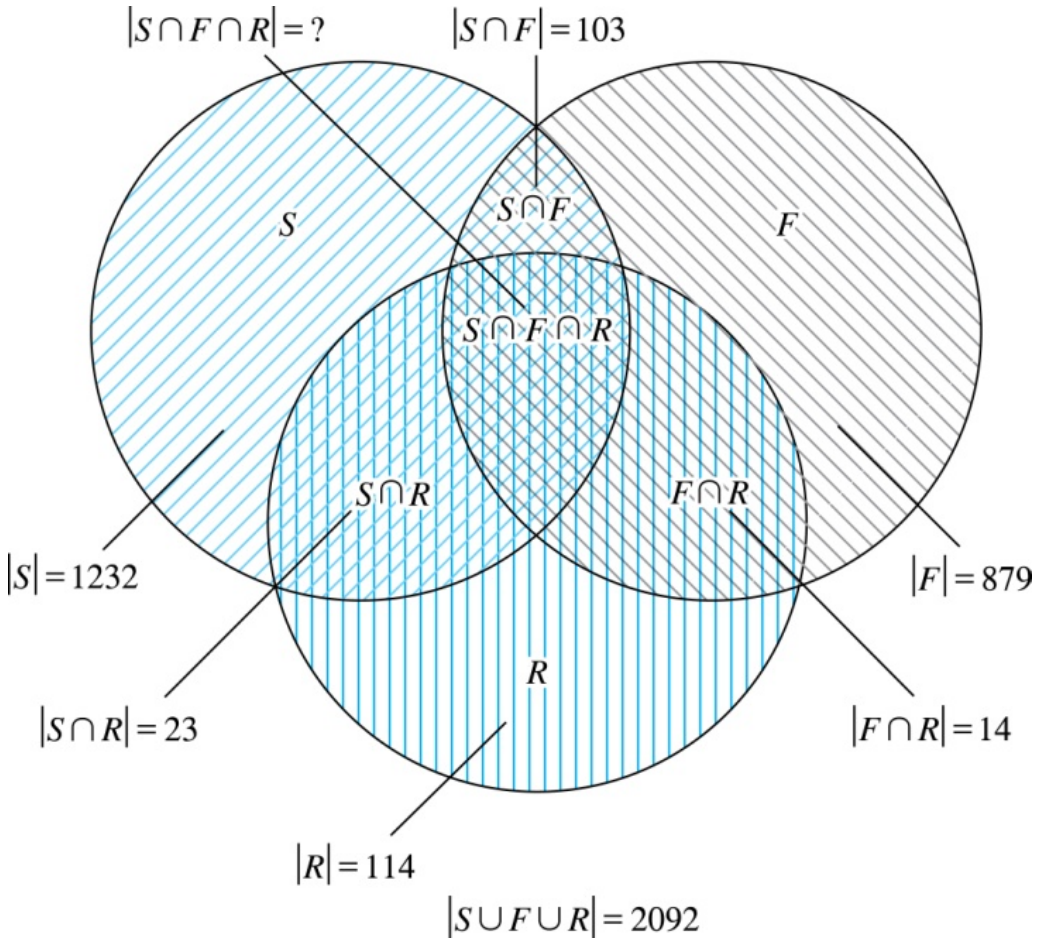
  Using the equation

  $$|S \cup F \cup R| = |S| + |F| + |R| - |S \cap F| - |S \cap R| - |F \cap R| + |S \cap F \cap R|,$$

  we obtain $2092 = 1232 + 879 + 114 - 103 - 23 - 14 + |S \cap F \cap R|$.

  Solving for $|S \cap F \cap R|$ yields 7.

Illustration of Three Finite Set Example



The Principle of Inclusion-Exclusion

- **Theorem 1. The Principle of Inclusion-Exclusion**: Let A_1, A_2, …, A_n be finite sets. Then:

$$|A_1 \cup A_2 \cup \cdots \cup A_n| =$$

$$\sum_{1 \le i \le n} |A_i| - \sum_{1 \le i \le j \le n} |A_i \cap A_j| +$$

$$\sum_{1 \le i \le j \le k \le n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n+1} |A_1 \cap A_2 \cap \ldots \cap A_n|$$

The Principle of Inclusion-Exclusion

- **Proof:** An element in the union is counted exactly once in the right-hand side of the equation. Consider an element *a* that is a member of *r* of the sets A_1,…., A_n where $1 \leq r \leq n$.
  - It is counted $C(r,1)$ times by $\sum |A_i|$
  - It is counted $C(r,2)$ times by $\sum |A_i \cap A_j|$
  - In general, it is counted $C(r,m)$ times by the summation of *m* of the sets A_i.

The Principle of Inclusion-Exclusion

– Thus the element is counted exactly

$C(r,1) - C(r,2) + C(r,3) - \cdots + (-1)^{r+1} C(r,r)$

times by the right hand side of the equation.

– By Corollary 2 of Section 6.4, we have

$C(r,0) - C(r,1) + C(r,2) - \cdots + (-1)^{r+1} C(r,r) = 0.$

– Hence,

$1 = C(r,0) = C(r,1) - C(r,2) + \cdots + (-1)^{r+1} C(r,r).$

Applications of Inclusion-Exclusion

Section 8.6

Section Summary

- Counting Onto-Functions
- Derangements

The Number of Onto Functions

- **Example**: How many onto functions are there from a set with six elements to a set with three elements?
- **Solution**: Suppose that the elements in the codomain are b_1, b_2, and b_3. Let P_1, P_2, and P_3 be the properties that b_1, b_2, and b_3 are not in the range of the function, respectively. The function is onto if none of the properties P_1, P_2, and P_3 hold.
- By the inclusion-exclusion principle the number of onto functions from a set with six elements to a set with three elements is
  $N - [N(P_1) + N(P_2) + N(P_3)] + [N(P_1P_2) + N(P_1P_3) + N(P_2P_3)] - N(P_1P_2P_3)$
  - Here the total number of functions from a set with six elements to one with three elements is $N = 3^6$.
  - The number of functions that do not have in the range is $N(P_1) = 2^6$. Similarly, $N(P_2) = N(3_1) = 2^6$.
  - Note that $N(P_1P_2) = N(P_1P_3) = N(P_2P_3) = 1$ and $N(P_1P_2P_3) = 0$.
- Hence, the number of onto functions from a set with six elements to a set with three elements is:
  $$3^6 - 3 \cdot 2^6 + 3 = 729 - 192 + 3 = 540$$

The Number of Onto Functions

- **Theorem 1**: Let m and n be positive integers with $m \geq n$. Then there are

$$n^m - C(n,1)(n-1)^m + C(n,2)(n-2)^m - \cdots + (-1)^{n-1}C(n,n-1)\cdot 1^m$$

onto functions from a set with *m* elements to a set with *n* elements.

- Proof follows from the principle of inclusion-exclusion (*see Exercise* 27).

Derangements

- **Definition**: A *derangement* is a permutation of objects that leaves no object in the original position.

- **Example**: The permutation of 21453 is a derangement of 12345 because no number is left in its original position. But 21543 is not a derangement of 12345, because 4 is in its original position.

Derangements

- **Theorem 2**: The number of derangements of a set with *n* elements is

$$D_n = n!\left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}\right].$$

  - Proof follows from the principle of inclusion-exclusion (see text)

Derangements

- **The Hatcheck Problem**: A new employee checks the hats of *n* people at restaurant, forgetting to put claim check numbers on the hats. When customers return for their hats, the checker gives them back hats chosen at random from the remaining hats. What is the probability that no one receives the correct hat?
- **Solution**: The answer is the number of ways the hats can be arranged so that there is no hat in its original position divided by *n*!, the number of permutations of *n* hats.

Remark: It can be shown that the probability of a derangement approaches $1/e$ as n grows without bound.

$$\frac{D_n}{n!} = \left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}\right]$$

TABLE 1 The Probability of a Derangement.

n	2	3	4	5	6	7
$D_n/n!$	0.50000	0.33333	0.37500	0.36667	0.36806	0.36786