Contents

This page would be intentionally left blank if we would not wish to inform about that.

Chapter 1

Introduction

Contents

1.1 What is *competitive programming*?

Competitive programming is basically a type of programming that you need to solve the algorithmic problems in a competitive manner, e.g. fast and efficient, in order to compete with others. Usually, two things needed to consider for programming competitively are (1) **time constraint** and (2) **memory constraint**, and most of the time the deciding factor is (1) time constraint mainly because, for instance, if you have to make an algorithm that takes an input of order $O(n^2)$ but you need to compute an output in $O(n \log n)$, this would be impossible as the bottleneck of your program is $O(n^2)$ from taking in an input.

For the sake of people who still have no idea what *competitive programming* is, I would like to give an example of problems, which are so-called competitive programming tasks:

Problem: Let X be the array of size N of integers with elements x_i. Suppose you want to find the minimum number of swaps with consecutive elements in the list do you need to do in order to make the array sorted, either in ascending order or descending order.
Input constraint: $1 \leq N \leq 10^5$, $1 \leq x_i \leq 10^{18}$.
Time limit: 1 second

In this problem, you have to find how to swap elements in order to make the array sorted and why the method you found is optimal. If you have some backgrounds on this kind of problems, first you would realize that you have to solve this problem twice, one for the ascending order case and another one for the descending order case. Then, you would realize that you just need to find that if you can swap any consecutive elements in order to make the list sorted, then you will swap these two elements and count how many times you do swap. However, the method we just said is in order of $O(N^2)$ and would be impossible to run in the time limit. One approach you will learn in the future is to use either *Merge sort* or *Fenwick Tree/Segment Tree* to count the number of times you do swap, which is in the order of $O(n \log n)$ and faster enough to finish in the time contraint.

1.2 What is the prerequisite for *competitive programming*?

These are very formulaic. I will list everything I know that is important to *competitive programming*:

1. Mathematics
   - Combinatorics (Counting /
   - Number Theory (Inverse modular, Euclid)
   - Geometry (Convex Hull, Trivial Geometry)
   - Graph Theory (BFS/DFS/Tree/Shortest Path/Euclidean/Hamiltonian/Network Flow/Matching)
2. Algorithms
   - Numerical Algorithms
   - Solving paradigm Algorithms (DP/Greedy/Brute force/D&Q)
   - Graph Algorithms
   - Computational Geometry Algorithms
3. Data Structures
   - Basic (Queue/Stack/Linked list)
   - Union-Disjoint Set
   - Fenwick Tree
   - Segment Tree
4. STL

1.3 Where can I practice it?

Again, there are so many websites you can practice at:

1. Topcoder
2. Codeforces
3. SPOJ
4. Google Code Jam
5. Facebook Hacker Cup
6. UVA Online Judge

That's more than enough that you can do.

1.4 How I design this book?

First, I will delve deeper into mathematics you needed to understand and will go to algorithms & data structures.

1.5 Who am I?

Mathematics

This page would be intentionally left blank if we would not wish to inform about that.

Chapter 2

Counting

Contents

This chapter is based on the materials in *Concrete Mathematics* written by Graham, Knuth, and Patashnik.

2.1 Recurrence relations

2.1.1 The Tower of Hanoi

This is a very basic problem in Mathematics. Suppose you are given a tower of n disks, stacked in decreasing size on one of three pegs. You want to move all of the disks to another pegs with the minimum number of moves. It's not that hard to see that the best strategy for moving disks is

- Move the first $n-1$ disks from the initial peg to a temporary peg.
- Move the last disk from the initial peg to a target peg.
- Move $n-1$ disks from the temporary peg to the target peg.

Let T_n denote the smallest number of moves you need to move n disks from any peg to any other peg. It would not hard to see that the answer to this according to our paradigm is

$$\begin{cases} T_0 = 0; \\ T_n = 2T_{n-1} + 1, \quad \text{for } n > 0. \end{cases}$$

2.1.2 Lines in the Plane

Let's move to another example. Suppose you have n lines, with infinite length, on the $\mathbb{R}^2$. You want to find the maximum number of distinct regions created by the set of n lines you have. Let L_n denote the maximum number of regions made by n lines. It's not hard to see that $L_0 = 1$, $L_1 = 2$, and $L_2 = 4$. But, how do we generalize this?

2.1.3 The Josephus Problem

2.1.4 Exercises

2.2 Matrix exponentiation

2.3 Binomial coefficient

2.4 Pigeonhole principle

2.5 Inclusion-exclusion principle

2.6 Subsets and Permutations

2.7 Indistinguishable and distinguishable objects

2.8 Lattice problems

2.9 Counting on graphs

2.10 Polya enumeration theorem

Chapter 3

Moduli Arithmetic

This page would be intentionally left blank if we would not wish to inform about that.

Chapter 4

Probability and Expectation value

This page would be intentionally left blank if we would not wish to inform about that.

Chapter 5

Number Theory

This page would be intentionally left blank if we would not wish to inform about that.

Algorithms