Introduction

Yangtao Ge

June 18, 2019

1 Preface

1.1 Purpose

How does the books go:
Specific problems = Coding + Math Analysing

Knowlegde preferred:

- intermediate programming(OOP & recursion)
- discrete Math – *Ref: COMP0147 & "Discrete Mathematics and Its Application"*

1.2 Overview

$\boxed{Part1 : Basic\ Knowlegde}$

- Chapter 1: Reviewing material on discrete math & recursion + Java related(out of date, not focus on)
- Chapter 2: Algorithm analysis (important and doing exercise)
- Chapter 3: List, Stack and Queues
- Chapter 4: Tress (Basic, AVL & game trees refer to advanced part)

- Chapter 5: Hash tables
- Chapter 6: Priority Queues
- Chapter 7: Sorting
- Chapter 8: Disjoint set
- Chapter 9: Graph Algorithm

$\boxed{Part2 : Advanced\ Knowlegde}$

- Chapter 10: Algorithm on problem-solving techniques (Lots of Examples)
- Chapter 11: amortized analysis(Three data structure from C4 & C6 + Fibonacci heap)
- Chapter 12: Search tree Algorithms(advanced trees)

1.3 Exercise

From easy to hard(marked with *), Last question demo the whole Chapter
Ref: `www.pearsonhighered.com/cssupport`

2 Chapter 1: Introduction

2.1 What is the Book About?

Running code fast and analysis them

N.B. detail contents for every chapter are in the previous section

2.2 Mathematics Review

Ref: pp.3-8

2.2.1 Exponents

$$X^A X^B = X^{A+B}$$
$$\frac{X^A}{X^B} = X^{A-B}$$
$$(X^A)^B = X^{AB}$$
$$X^N + X^N = 2X^N \neq X^{2N}$$
$$2^N + 2^N = 2^{N+1}$$

2.2.2 Logarithms

In computer science, all Logarithms are to the '*base 2*' unless specified otherwise.

Definition 2.1. $X^A = B$ iff $\log_X B = A$

Theorem 2.1.

$$\log_A B = \frac{\log_C B}{\log_C A};\ \ A, B, C > 0, A \neq 1$$

Theorem 2.2.

$$\log AB = \log A + \log B;\ \ A, B > 0$$

2.2.3 Series

Theorem 2.3 (Sum of Geometric Progression)**.**

$$\sum_{i=0}^{N} A^i = \frac{A^{N+1} - 1}{A - 1}$$

Theorem 2.4 (Infinite Geometric Series)**.**

$$\sum_{i=0}^{N} A^i \leq \frac{1}{1 - A}$$

Hence, when $N \to \infty$, we have:

$$\sum_{i=0}^{\infty} ar^i = \frac{a}{1 - i}$$

Theorem 2.5 (Sum of Arithmetic Series)**.**

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2}$$

Theorem 2.6 (Inference of Arithmetic Progression)**.**

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{2} \approx \frac{N^3}{3}$$

$$\sum_{i=1}^{N} i^k \approx \frac{N^{k+1}}{|k+1|};\ k \neq -1$$

Theorem 2.7 (Harmonic Numbers)**.**

$$H_N = \sum_{i=1}^{N} \frac{1}{i} \approx \log_e N$$

The error is called **Euler's constant**, which value is $\gamma \approx 0.57721566$

2.2.4 Modular Arithmetic

Definition 2.2. We say A os *congruent* to B modulo N, written in $A \equiv B$ (mod N)

Theorem 2.8. If a prime number N divides a product of two numbers, it divides at least one of the two numbers

$$ab \equiv 0\ (modN)\ \ iff\ \ a \equiv 0\ (modN)\ or\ b \equiv 0\ (modN)$$

Theorem 2.9. If N is prime, then the equation $ax \equiv 1$ (mod N) has a *unique* solution (mod N) for all $0 < a < N$

This solution $0 < x < N$ is the *multiplicative inverse*

2.2.5 The P Word

Three common ways of proving statements in data structure analysis:

- Proof by induction
- Proof by contradiction
- Proof by counterexample

Ref: pp.6 - 8 & COMP 0147 & COMP 0003(1)

2.3 A Brief Introduction to Recursion

Ref: pp. 8-12 & COMP 0002 Haskell & COMP0005 Algorithm

Basic Rules of recursion:

- **Base cases**: you must have some base cases, which can be solved *without* recursion
- **Making progress**: For the case that are to besolved recursively, the recursive call must always be to a case that makes progress toward a base case (from base case)
- **Design rule**: Assume that all the recursive calls work
- **Compound interest rule**: Never duplicate work by solving the same instance of a problem in separate recursive calls

2.4 Implementing Generic Components

Review after finish Core Java 'Generic Programming'

2.5 Function Objects

Review after finish Core Java 'Generic Programming'