CONTENTS IN DETAIL

FOREWORD **xix**

ACKNOWLEDGMENTS **xxi**

INTRODUCTION **xxiii**

What We'll Do xxiii
New to the Second Edition xxiv
Who This Book Is For xxv
Our Programming Language xxvi
    Why Use C? xxvi
    Static Keyword xxvi
    Include Files xxvii
    Freeing Memory xxvii
Topic Selection xxvii
Programming Judges xxviii
Anatomy of a Problem Description xxx
Starter Problem: Food Lines xxxi
    The Problem xxxi
    Solving the Problem xxxii
Online Resources xxxiv
Notes xxxiv

1
HASH TABLES **1**

Problem 1: Unique Snowflakes 1
    The Problem 2
    Simplifying the Problem 4
    Solving the Core Problem 5
    Solution 1: Pairwise Comparisons 8
    Solution 2: Doing Less Work 12
Hash Tables 17
    Hash Table Design 17
    Why Use Hash Tables? 20
Problem 2: Login Mayhem 20
    The Problem 20
    Solution 1: Looking at All Passwords 21
    Solution 2: Using a Hash Table 23

Problem 3: Spelling Check 29
The Problem 29
Thinking About Hash Tables 30
An Ad Hoc Solution 32
Summary 35
Notes 35

2
TREES AND RECURSION **37**

Problem 1: Halloween Haul 37
The Problem 38
Binary Trees 39
Solving the Sample Instance 41
Representing Binary Trees 41
Collecting All the Candy 46
A Completely Different Solution 51
Walking the Minimum Number of Streets 56
Reading the Input 59
Why Use Recursion? 65
Problem 2: Descendant Distance 66
The Problem 66
Reading the Input 68
Number of Descendants from One Node 72
Number of Descendants from All Nodes 73
Sorting Nodes 74
Outputting the Information 75
The main Function 75
Summary 76
Notes 76

3
MEMOIZATION AND DYNAMIC PROGRAMMING **77**

Problem 1: Burger Fervor 78
The Problem 78
Forming a Plan 78
Characterizing Optimal Solutions 80
Solution 1: Recursion 81
Solution 2: Memoization 86
Solution 3: Dynamic Programming 91
Memoization and Dynamic Programming 94
Step 1: Structure of Optimal Solutions 95
Step 2: Recursive Solution 96
Step 3: Memoization 96
Step 4: Dynamic Programming 97

Problem 2: Moneygrubbers . . . 98
The Problem . . . 98
Characterizing Optimal Solutions . . . 99
Solution 1: Recursion . . . 101
The main Function . . . 105
Solution 2: Memoization . . . 106
Problem 3: Hockey Rivalry . . . 108
The Problem . . . 109
About Rivalries . . . 110
Characterizing Optimal Solutions . . . 111
Solution 1: Recursion . . . 114
Solution 2: Memoization . . . 117
Solution 3: Dynamic Programming . . . 119
A Space Optimization . . . 122
Summary . . . 123
Notes . . . 123

4
ADVANCED MEMOIZATION AND DYNAMIC PROGRAMMING 125

Problem 1: The Jumper . . . 125
The Problem . . . 126
Working Through an Example . . . 127
Solution 1: Backward Formulation . . . 128
Solution 2: Forward Formulation . . . 133
Problem 2: Ways to Build . . . 137
The Problem . . . 138
Working Through an Example . . . 139
Solution 1: Using "Exactly" Subproblems . . . 140
Solution 2: Adding More Subproblems . . . 144
Summary . . . 149
Notes . . . 149

5
GRAPHS AND BREADTH-FIRST SEARCH 151

Problem 1: Knight Chase . . . 151
The Problem . . . 152
Moving Optimally . . . 153
Best Knight Outcome . . . 163
The Knight Flip-Flop . . . 165
A Time Optimization . . . 168
Graphs and BFS . . . 169
What Are Graphs? . . . 169
Graphs vs. Trees . . . 170
BFS on Graphs . . . 172
Graphs vs. Dynamic Programming . . . 173

Problem 2: Rope Climb 173
The Problem 173
Solution 1: Finding the Moves 174
Solution 2: A Remodel 179
Problem 3: Book Translation 187
The Problem 187
Reading the Language Names 188
Building the Graph 189
The BFS 192
The Total Cost 194
Summary 195
Notes 195

6
SHORTEST PATHS IN WEIGHTED GRAPHS 197

Problem 1: Mice Maze 198
The Problem 198
Moving On from BFS 199
Finding Shortest Paths in Weighted Graphs 200
Building the Graph 203
Implementing Dijkstra's Algorithm 205
Two Optimizations 207
Dijkstra's Algorithm 210
Runtime of Dijkstra's Algorithm 210
Negative-Weight Edges 211
Problem 2: Grandma Planner 213
The Problem 213
Adjacency Matrix 214
Building the Graph 215
Working Through a Weird Test Case 217
Task 1: Shortest Paths 219
Task 2: Number of Shortest Paths 222
Summary 228
Notes 229

7
BINARY SEARCH 231

Problem 1: Feeding Ants 231
The Problem 232
A New Flavor of Tree Problem 233
Reading the Input 235
Testing Feasibility 237
Searching for a Solution 239

Binary Search 240
Runtime of Binary Search 241
Determining Feasibility 242
Searching a Sorted Array 242
Problem 2: River Jump 243
The Problem 243
A Greedy Idea 244
Testing Feasibility 246
Searching for a Solution 250
Reading the Input 253
Problem 3: Living Quality 254
The Problem 254
Sorting Every Rectangle 256
Using Binary Search 259
Testing Feasibility 260
A Quicker Way to Test Feasibility 261
Problem 4: Cave Doors 267
The Problem 267
Solving a Subtask 268
Using Linear Search 270
Using Binary Search 272
Summary 274
Notes 275

8
HEAPS AND SEGMENT TREES 277

Problem 1: Supermarket Promotion 277
The Problem 277
Solution 1: Maximum and Minimum in an Array 278
Max-Heaps 282
Min-Heaps 293
Solution 2: Heaps 295
Heaps 298
Two More Applications 298
Choosing a Data Structure 300
Problem 2: Building Treaps 300
The Problem 300
Recursively Outputting Treaps 302
Sorting by Label 303
Solution 1: Recursion 303
Range Maximum Queries 307
Segment Trees 308
Solution 2: Segment Trees 316
Segment Trees 317

Problem 3: Two Sum 318
The Problem 318
Filling the Segment Tree 319
Querying the Segment Tree 323
Updating the Segment Tree 324
The main Function 328
Summary 329
Notes 329

9
UNION-FIND 331

Problem 1: Social Network 332
The Problem 332
Modeling as a Graph 333
Solution 1: BFS 336
Union-Find 340
Solution 2: Union-Find 343
Optimization 1: Union by Size 346
Optimization 2: Path Compression 350
Union-Find 352
Relationships: Three Requirements 353
Choosing Union-Find 353
Optimizations 353
Problem 2: Friends and Enemies 354
The Problem 354
Augmenting Union-Find 355
The main Function 359
Find and Union 360
SetFriends and SetEnemies 361
AreFriends and AreEnemies 363
Problem 3: Drawer Chore 364
The Problem 364
Equivalent Drawers 365
The main Function 370
Find and Union 372
Summary 373
Notes 373

10
RANDOMIZATION 375

Problem 1: Yōkan 376
The Problem 376
Randomly Choosing a Piece 376

Generating Random Numbers . . . 378
Determining Number of Pieces . . . 379
Guessing Flavors . . . 381
How Many Attempts Do We Need? . . . 384
Filling the Flavor Arrays . . . 385
The main Function . . . 386
Randomization . . . 387
Monte Carlo Algorithms . . . 387
Las Vegas Algorithms . . . 388
Deterministic vs. Randomized Algorithms . . . 389
Problem 2: Caps and Bottles . . . 390
The Problem . . . 390
Solving a Subtask . . . 391
Solution 1: Recursion . . . 393
Solution 2: Adding Randomization . . . 396
Quicksort . . . 398
Implementing Quicksort . . . 398
Worst-Case and Expected Runtime . . . 400
Summary . . . 402
Notes . . . 402

AFTERWORD 403

A
ALGORITHM RUNTIME 405

The Case for Timing . . . and Something Else . . . 405
Big O Notation . . . 407
Linear Time . . . 407
Constant Time . . . 408
Another Example . . . 409
Quadratic Time . . . 409
Big O in This Book . . . 410

B
BECAUSE I CAN'T RESIST 411

Unique Snowflakes: Implicit Linked Lists . . . 411
Burger Fervor: Reconstructing a Solution . . . 414
Knight Chase: Encoding Moves . . . 416
Dijkstra's Algorithm: Using a Heap . . . 418
Mice Maze: Tracing with Heaps . . . 418
Mice Maze: Implementation with Heaps . . . 421

Compressing Path Compression 422
Step 1: No More Ternary If 423
Step 2: Cleaner Assignment Operator 423
Step 3: Understand the Recursion 424
Caps and Bottles: In-Place Sorting 424

C
PROBLEM CREDITS **427**

INDEX **431**