Concrete and Abstract Data Types with Algorithms

version 1.0

Robert Strandh

2012

Contents

1 Introduction **1**
- 1.1 Audience . . . 1
- 1.2 Objectives . . . 2
- 1.3 Notation and conventions . . . 6
  - 1.3.1 Mathematical notation . . . 6
  - 1.3.2 Sequence operations and positions . . . 6
  - 1.3.3 Intervals . . . 8
  - 1.3.4 Keys . . . 8
  - 1.3.5 Conventions in figures . . . 11
- 1.4 Modern computer . . . 15
  - 1.4.1 Address space . . . 15
  - 1.4.2 Size of main memory . . . 15
  - 1.4.3 Processor speed . . . 16
  - 1.4.4 Cache memory . . . 16
  - 1.4.5 Branch prediction . . . 17
  - 1.4.6 Cost of certain operations . . . 18
- 1.5 Organization of the book . . . 18
- 1.6 Why this book is self published . . . 18

I Basic Ideas **21**

2 Algorithmic Performance **23**
- 2.1 Asymptotic complexity . . . 24
  - 2.1.1 History . . . 24
  - 2.1.2 RAM model . . . 24
  - 2.1.3 Asymptotic notation . . . 25

2.2 Worst-case complexity 26
2.3 Amortized complexity 27
2.4 Average complexity 28
2.5 Other performance factors 29

3 Data Types **31**

4 Algorithmic Language **35**
4.1 Introduction 35
4.2 Basic syntax 36
4.2.1 Identifiers 36
4.2.2 Literals 36
4.2.3 Separators 38
4.3 Data types 40
4.3.1 Numbers 40
4.3.2 Characters 40
4.3.3 Character strings 40
4.3.4 Arrays 41
4.3.5 Structures 42
4.4 Expressions 43
4.4.1 Constants as expressions 43
4.4.2 Places as expressions 44
4.4.3 Function calls as expressions 44
4.4.4 Operators in expressions 45
4.4.5 Expressions with multiple values 46
4.5 Statements 46
4.5.1 Assignments 47
4.5.2 Conditionals 48
4.5.3 Local blocks 49
4.5.4 Loops 50
4.5.5 Function definitions 52
4.5.6 Return 56
4.6 Uniform reference semantics 56
4.7 Parameter passing 57
4.8 Namespaces 58

II Concrete Types **61**

5 Vectors **63**
5.1 Description . . . 63
5.2 Allocation . . . 65
5.3 Operations on intervals . . . 66
5.3.1 Copying an interval . . . 66
5.3.2 Filling an interval . . . 69
5.3.3 Clearing an interval . . . 71
5.3.4 Reversing an interval . . . 72
5.4 Search algorithms . . . 72
5.4.1 Storing objects associated with keys . . . 72
5.4.2 Sequential search . . . 74
5.4.3 Binary search . . . 75
5.5 Sorting algorithms . . . 81
5.5.1 Insertion sort . . . 82
5.5.2 Quicksort . . . 94
5.6 Exercises . . . 103

6 Simply linked list **105**
6.1 Description . . . 105
6.2 Allocation . . . 106
6.3 Operations by element position . . . 106
6.3.1 Retrieving an object . . . 108
6.3.2 Injecting an object . . . 111
6.3.3 Expunging an object . . . 115
6.3.4 Computing the n^{th} tail of a list . . . 119
6.3.5 Splitting a list at some position . . . 120
6.3.6 Appending two lists . . . 124
6.3.7 Reversing a list . . . 125
6.3.8 Computing the length of a list . . . 125
6.4 Operations using keys . . . 125
6.4.1 Searching for an object . . . 127
6.4.2 Inserting and object . . . 131
6.4.3 Deleting an object . . . 133
6.4.4 Working from the end of the list . . . 135
6.5 Using sentinels . . . 146
6.6 Sorting algorithms . . . 148

6.6.1 Selection sort . . . 149
6.6.2 Merge sort . . . 158
6.7 Exercises . . . 167

7 Search trees **169**
7.1 Operations on search trees . . . 170
7.1.1 Search operation . . . 171
7.1.2 Insert operation . . . 172
7.1.3 Delete operation . . . 174
7.2 Binary Search Tree . . . 174
7.2.1 Finding the object with the smallest key . . . 177
7.2.2 Finding the object with the largest key . . . 178
7.2.3 Searching for an object with a given key . . . 178
7.2.4 Inserting an object . . . 181
7.2.5 Deleting an object . . . 186
7.2.6 Rotation . . . 203
7.3 Balanced Trees . . . 206
7.4 AVL Tree . . . 210
7.4.1 Finding the object with the smallest or largest key . . . 213
7.4.2 Searching for an object with a given key . . . 213
7.4.3 Rebalancing . . . 214
7.4.4 Inserting an object . . . 227
7.4.5 Deleting an object . . . 231
7.5 Splay tree . . . 234
7.5.1 Creating a node . . . 236
7.5.2 Splaying a node . . . 236
7.5.3 Splaying the node with the smallest key . . . 246
7.5.4 Splaying the node with the largest key . . . 249
7.5.5 Splaying a node with a particular key . . . 251
7.5.6 Splitting a tree . . . 266
7.5.7 Joining two trees . . . 268
7.5.8 Inserting an object . . . 270
7.5.9 Deleting an object . . . 275
7.6 2-3 Tree . . . 277
7.6.1 Searching for an object with a given key . . . 279
7.6.2 Inserting an object with a given key . . . 280
7.6.3 Deleting an object with a given key . . . 308
7.7 Exercises . . . 352

7.8 Bibliographic notes 353

8 Trees as sequences **355**
8.1 The basic idea 355
8.2 Binary sequence tree 364
8.2.1 Rotation 364
8.3 Splay trees as sequences 366
8.3.1 Splaying a node at a particular position 367
8.3.2 Retrieving an object 375
8.3.3 Splitting a tree 376
8.3.4 Joining two trees 377
8.3.5 Injecting an object 377
8.3.6 Expunging an object 379
8.4 2-3 trees as sequences 380
8.4.1 Retrieving an object 380
8.4.2 Injecting an object 382
8.4.3 Expunging an object 389

9 Hash table **403**
9.1 Description 403
9.2 Representation of a hash bucket 405
9.3 Operations 408
9.4 Performance 412
9.5 Alternative representations for hash buckets 415
9.5.1 Vector 415
9.5.2 Balanced tree 416
9.6 Perfect and minimal-perfect hashing 416

10 Heap **419**
10.1 Description 419
10.2 Operations 423
10.2.1 Inserting an object 424
10.2.2 Retrieving the object with the smallest key 430
10.2.3 Deleting the object with the smallest key 430
10.3 Variations 437
10.4 Exercises 437

11 Other concrete data types **439**

11.1 Multi-dimensional arrays 439
11.1.1 Description 439
11.1.2 Performance 440
11.1.3 Allocation 440
11.2 Doubly linked list 441
11.2.1 Description 441
11.2.2 Space requirement 442
11.2.3 Allocation 443

III Abstract Data types 445

12 General considerations 447
12.1 Consistent interface 448
12.2 Preconditions 449
12.3 Use of header objects 450
12.4 Size of contained objects 451
12.5 Throughput versus response time 452

13 Stack 455
13.1 Description 455
13.2 Creation 455
13.3 Operations 456
13.3.1 Checking whether the stack is empty 456
13.3.2 Adding an object to the top of the stack 457
13.3.3 Accessing the top object of the stack 457
13.3.4 Removing the top-object of the stack 457
13.4 Implementations 458
13.4.1 Simply linked list 458
13.4.2 Vector 463
13.4.3 List of vectors 470
13.5 Variations 475

14 Queue 477
14.1 Description 477
14.2 Creation 477
14.3 Operations 478
14.3.1 Checking whether the queue is empty 478

14.3.2 Adding an object to the tail of the queue 479
14.3.3 Accessing the head object of the queue 479
14.3.4 Removing the head object of the queue 479
14.4 Implementations 480
14.4.1 Simply linked list 480
14.4.2 Vector 484
14.4.3 List of vectors 492

15 Priority queue **497**
15.1 Description 497
15.2 Creation 497
15.3 Operations 498
15.3.1 Checking whether the priority queue is *empty* 499
15.3.2 Adding an object to a priority queue 499
15.3.3 Retrieving the object with the largest priority 499
15.3.4 Deleting the object with the largest priority 500
15.4 Implementations 500
15.4.1 Balanced tree 500
15.4.2 Heap 507
15.5 Variations 513
15.6 Exercises 513

16 Dictionary **515**
16.1 Description 515
16.2 Operations 515
16.2.1 Checking whether the dictionary is empty 516
16.2.2 Inserting an object 516
16.2.3 Searching for an object with a particular key 516
16.2.4 Deleting an object with a particular key 517
16.3 Implementations 517
16.3.1 Simply linked list 518
16.3.2 Vector 522
16.3.3 Balanced tree 529
16.3.4 Hash table 534
16.4 Variations 540
16.5 Exercises 540

IV Case Studies **541**

17 Editable sequence **545**
17.1 Interface . . . 545
17.2 Possible strategies for implementation . . . 547
17.2.1 Linked list . . . 548
17.2.2 Vector . . . 548
17.2.3 Tree . . . 549
17.3 Compromise implementation strategy . . . 550
17.4 Algorithms . . . 552
17.4.1 Retrieving an object . . . 553
17.4.2 Injecting an object . . . 553
17.4.3 Expunging an object . . . 567
17.5 Analysis . . . 583
17.6 Exercises . . . 583

18 Text editor buffer **587**
18.1 Requirements . . . 588
18.1.1 Size of the entire buffer . . . 588
18.1.2 Size of a line . . . 589
18.1.3 Number of lines . . . 589
18.1.4 Contents . . . 590
18.1.5 Locality . . . 590
18.1.6 Multiple cursors . . . 590
18.1.7 Positions of editing operations . . . 591
18.2 Existing buffer implementations . . . 591
18.2.1 Gap buffer . . . 591
18.2.2 Sequence of lines . . . 603
18.3 Interface . . . 604
18.3.1 Stratified design and internal interfaces . . . 604
18.3.2 Additional terminology . . . 605
18.3.3 The cursor/line interface . . . 605
18.3.4 The buffer interface . . . 608
18.4 Final choice of representation . . . 610
18.4.1 Representation of a single line . . . 611
18.4.2 Representation of the sequence of lines . . . 632
18.5 Undo facility . . . 649
18.6 Exercises . . . 649

V Appendix **651**

A Automatic memory management **653**
A.1 Reasons for having automatic memory management 653
A.2 How automatic memory management works 656
A.2.1 Reference counting 656
A.2.2 Tracing 657
A.3 Consequences on programming methodology 659
A.4 Performance of automatic memory management 660

Bibliography **661**

Index **663**