附录

参考文献

[1] D. E. Knuth. The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd edn.). Addison-Wesley (1997), ISBN:0-201-89683-1

[2] D. E. Knuth. The Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd edn.). Addison-Wesley (1997), ISBN:0-201-89684-8

[3] D. E. Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching (2nd edn.). Addison-Wesley (1998), ISBN:0-201-89685-0

[4] A. V. Aho, J. E.Hopcroft, J. D. Ullman. The Design and Analysis of Computer Algorithms (1st edn.). Addison-Wesley (1974), ISBN:0-201-00029-0

[5] J. Bentley. Writing Efficient Programs. Prentice-Hall (1982), ISBN:0-139-70251-2

[6] J. Bentley. More Programming Pearls: Confessions of a Coder. Addison Wesley (1988), ISBN:0-201-11889-0

[7] R. L. Graham, D. E. Knuth, O. Patashnik. Concrete Mathematics: A Foundation for Computer Science (2nd edn.). Addison-Wesley (1994), ISBN:0-201-55802-5

[8] 严蔚敏 等. 数据结构（C语言版）. 北京：清华大学出版社, 1997年4月第1版, ISBN:7-302-02368-9

[9] J. Bentley. Programming Pearls (2nd edn.). Addison-Wesley (2000), ISBN:0-201-65788-0

[10] T. Budd. Classic Data Structures in Java. Addison-Wesley (2000), ISBN:0-201-70002-6

[11] J. Hromkovic. Design And Analysis Of Randomized Algorithms: Introduction to Design Paradigms. Springer-Verlag (2005), ISBN:3-540-23949-9

[12] H. Samet. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann (2006), ISBN:0-123-69446-9

[13] M. A. Weiss. Data Structures and Algorithm Analysis in C++ (3rd edn.). Addison Wesley (2006), ISBN:0-321-44146-1

[14] E. Horowitz, S. Sahni, D. Mehta. Fundamentals of Data Structures in C++ (2nd edn.). Silicon Press (2006), ISBN:0-929-30637-6

[15] A. Drozdek. Data Structures and Algorithms in C++ (2nd edn.). Thomson Press (2006), ISBN:8-131-50115-9

[16] 殷人昆 等. 数据结构（C++语言描述）. 北京：清华大学出版社, 2007年6月第2版, ISBN:7-302-14811-1

[17] P. Brass. Advanced Data Structures. Cambridge University Press, ISBN:0-521-88037-8

[18] J. Edmonds. How to Think about Algorithms. Cambridge University Press (2008), ISBN:0-521-61410-8

[19] K. Mehlhorn & P. Sanders. Algorithms and Data Structures: The Basic Tools. Springer (2008), ISBN:3-540-77977-9

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. Introduction to Algorithms (3rd edn.). MIT Press (2009), ISBN:0-262-03384-4

[21] R. Bird. Pearls of Functional Algorithm Design. Cambridge University Press (2010), ISBN:0-521-51338-8

[22] M. L. Hetland. Python Algorithms: Mastering Basic Algorithms in the Python Language. Apress (2010), ISBN:1-430-23237-4

[23] M. T. Goodrich, R. Tamassia, D. M. Mount. Data Structures and Algorithms in C++ (2nd edn.). John Wiley & Sons (2011), ISBN:0-470-38327-5

[24] R. Sedgewick & K. Wayne. Algorithms (4th edn.). Addison-Wesley (2011), ISBN:0-321-57351-X

[25] Y. Perl, A. Itai and H. Avni, Interpolation Search: A log(log(n)) Search, Commun. ACM, 21 (1978), pp. 550-553

[26] A. C. Yao & F. F. Yao. The Complexity of Searching an Ordered Random Table. 17th Annual Symposium on Foundations of Computer Science (1976), 173-177

[27] A. C. Yao & J. M. Steele. Lower Bounds to Algebraic Decision Trees. Journal of Algorithms (1982), 3:1-8

[28] A. C. Yao. Lower Bounds for Algebraic Computation Trees with Integer Inputs. SIAM J. On Computing (1991), 20:655-668

[29] L. Devroye. A Note on the Height of Binary Search Trees. J. of ACM (1986), 33(3):489-498

[30] P. Flajolet & A. Odlyzko. The Average Height of Binary Trees and Other Simple Trees. Journal of Computer and System Sciences (1982), 25(2):171-213

[31] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proc. of the American Mathematical Society, 7(1):48-50

[32] B. W. Arden, B. A. Galler, R. M. Graham. An Algorithm for Equivalence Declarations. Communications ACM (1961), 4:310-314

[33] B. A. Galler, M. J. Fisher. An Improved Equivalence Algorithm. Communications ACM (1964), 7:301-303

[34] R. E. Tarjan. Efficiency of a Good but not Linear Set Union Algorithm. Journal of the ACM (1975), 22:215-225

[35] R. Seidel & M. Sharir. Top-Down Analysis of Path Compression. SIAM Journal Computing (2005), 34:515-525

[36] G. Adelson-Velskii & E. M. Landis. An Algorithm for the Organization of Information. Proc. of the USSR Academy of Sciences (1962), 146:263-266

[37] D. S. Hirschberg. An Insertion Technique for One-Sided Heightbalanced Trees. Comm. ACM (1976), 19(8):471-473

[38] S. H. Zweben & M. A. McDonald. An Optimal Method for Deletion in One-Sided Height-Balanced Trees. Commun. ACM (1978), 21(6):441-445

[39] K. Culik, T. Ottman, D. Wood. Dense Multiway Trees. ACM Transactions on Database Systems (1981), 6:486-512

[40] E. Gudes & S. Tsur. Experiments with B-tree Reorganization. SIGMOD (1980), 200-206

[41] D. D. Sleator & R. E. Tarjan. Self-Adjusting Binary Trees. JACM (1985), 32:652-686

[42] R. E. Tarjan. Amortized Computational Complexity. SIAM. J. on Algebraic and Discrete Methods 6(2):306-318

[43] R. Bayer & E. McCreight. Organization and Maintenance of Large Ordered Indexes. Acta Informatica (1972), 1(3):173-189

[44] R. Bayer. Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms. Acta Informatica (1972), 1(4):290-306

[45] L. J. Guibas & R. Sedgewick. A Dichromatic Framework for Balanced Trees. Proc. of the 19th Annual Symposium on Foundations of Computer Science (1978), 8-21

[46] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM (1975), 18(9):509-517

[47] H. J. Olivie. A New Class of Balanced Search Trees: Half Balanced Binary Search Trees. ITA (1982), 16(1):51–71

[48] J. L. Bentley. Decomposable Searching Problems. Information Processing Letters (1979), 8:244-251

[49] J. H. Hart. Optimal Two-Dimensional Range Queries Using Binary Range Lists. Technical Report 76-81, Department of Computer Science, University of Kentucky (1981)

[50] D. E. Willard. New Data Structures for Orthogonal Range Queries. SIAM Journal on Computing (1985), 14:232–253

[51] H. Samet, An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures, in R. Earnshaw, ed., Theoretical Foundations of Computer Graphics and Cad, Springer Berlin Heidelberg, 1988, pp. 51-68

[52] W. Pugh. Skip Lists: a Probabilistic Alternative to Balanced Trees. Lecture Notes in Computer Science (1989), 382:437-449

[53] R. de la Briandais. File Searching Using Variable Length Keys. Proc. of the Western Joint Computer Conference 1959, 295-298

[54] E. H. Sussenguth. Use of Tree Structures for Processing Files. Communications of the ACM (1963), 6:272-279

[55] D. R. Morrison. PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. Journal of the ACM (1968), 15:514-534

[56] J. L. Bentley & R. Sedgewick. Fast Algorithms for Sorting and Searching Strings. Proc. of 8th ACM-SIAM Symposium on Discrete Algorithms (1997), 360-369

[57] R. W. Floyd. Algorithm 113: Treesort. Communications of the ACM (1962), 5:434

[58] C. A. Crane. Linear Lists and Priority Queues as Balanced Binary Trees. PhD thesis, Stanford University (1972)

[59] E. M. McCreight. Priority Search Trees. SIAM J. Comput. (1985), 14(2):257-276

[60] D. E. Knuth, J. H. Morris, V. R. Pratt. Fast Pattern Matching in Strings. SIAM Journal of Computing (1977), 6(2):323-350

[61] R. S. Boyer & J. S. Moore. A Fast String Searching Algorithm. Communications of the ACM (1977), 20:762-772

[62] L. J. Guibas & A. M. Odlyzko. A New Proof of the Linearity of the Boyer-Moore String Search Algorithm. SIAM Journal on Computing (1980), 9(4):672-682

[63] R. Cole. Tight Bounds on the Complexity of the Boyer-Moore Pattern Matching Algorithm. SIAM Journal on Computing 23(5):1075-1091

[64] C. A. R. Hoare. Quicksort. Computer Journal (1962), 5(1):10-15

[65] D. L. Shell. A High-Speed Sorting Procedure. Communications of the ACM (1959), 2(7):30-32

[66] R. Sedgewick, A New Upper Bound for Shellsort, J. Algorithms, 7 (1986), pp. 159-173

插图索引

图1.1 古埃及人使用的绳索计算机及其算法 2
图1.2 古希腊人的尺规计算机 3
图1.3 通过6趟扫描交换对七个整数排序（其中已就位的元素以深色示意） 4
图1.4 大$\mathcal{O}$记号、大Ω记号和大Θ记号 11
图1.5 复杂度的典型层次：(1)~(7)依次为$\mathcal{O}(\log n)$、$\mathcal{O}(\sqrt{n})$、$\mathcal{O}(n)$、$\mathcal{O}(n\log n)$、$\mathcal{O}(n^2)$、$\mathcal{O}(n^3)$和$\mathcal{O}(2^n)$ 15
图1.6 对sum(A, 5)的递归跟踪分析 18
图1.7 对sum(A, 0, 7)的递归跟踪分析 23
图2.1 可扩充向量的溢出处理 34
图2.2 向量整体置乱算法permute()的迭代过程 37
图2.3 无序向量的顺序查找 39
图2.4 向量元素插入操作insert(r, e)的过程 40
图2.5 向量区间删除操作remove(lo, hi)的过程 41
图2.6 无序向量deduplicate()算法原理 42
图2.7 低效版uniquify()算法的最坏情况 45
图2.8 有序向量中的重复元素可批量删除 46
图2.9 在有序向量中查找互异的相邻元素 46
图2.10 基于减治策略的有序向量二分查找算法（版本A） 48
图2.11 二分查找算法（版本A）实例：search(8, 0, 7)成功，search(3, 0, 7)失败 49
图2.12 二分查找算法（版本A）的查找长度（成功、失败查找分别以实线、虚线白色方框示意） 50
图2.13 Fibonacci查找算法原理 52
图2.14 Fibonacci查找算法的查找长度（成功、失败查找分别以实线、虚线白色方框示意） 53
图2.15 基于减治策略的有序向量二分查找算法（版本B） 54
图2.16 基于减治策略的有序向量二分查找算法（版本C） 56
图2.17 从三只苹果中挑出重量不同者 58
图2.18 有序向量的二路归并实例 （来自两个向量的元素分别以黑、白方框区分，其各自的当前首元素则以灰色长方形示意） 62
图2.19 归并排序实例：S = { 6, 3, 2, 7, 1, 5, 8, 4 } 63
图3.1 首（末）节点是头（尾）节点的直接后继（前驱） 71
图3.2 刚创建的List对象 71
图3.3 ListNode::insertAsPred()算法 73
图3.4 List::remove()算法 75
图3.5 序列的插入排序 79
图3.6 序列的选择排序 80
图4.1 一摞椅子即是一个栈 87
图4.2 栈操作 87

图4.3 函数调用栈实例：主函数main()调用funcA()，funcA()调用funcB()，funcB()再自我调用 88
图4.4 进制转换算法流程 90
图4.5 栈混洗实例：从< 1, 2, 3, 4]到[3, 2, 4, 1 >（上方左侧为栈A，右侧为栈B；下方为栈S） 91
图4.6 迭代式括号匹配算法实例 （上方为输入表达式；下方为辅助栈的演变过程；虚框表示在（右）括号与栈顶（左）括号匹配时对应的出栈操作） 93
图4.7 通过剪枝排除候选解子集 99
图4.8 (a)皇后的控制范围；(b)8皇后问题的一个解 100
图4.9 四皇后问题求解过程（棋盘右侧为记录解的栈solu） 101
图4.10 迷宫寻径算法实例 104
图4.11 在球桶中顺序排列的一组羽毛球可视作一个队列 105
图4.12 队列操作 105
图5.1 有根树的逻辑结构 111
图5.2 二叉树：(a)逻辑结构；(b)实例 111
图5.3 多叉树的“父节点”表示法 112
图5.4 多叉树的“孩子节点”表示法 112
图5.5 多叉树的“父节点 + 孩子节点”表示法 112
图5.6 多叉树的“长子 + 兄弟”表示法（(b)中长子和兄弟指针，分别以垂直实线和水平虚线示意） 113
图5.7 完整的通讯过程由预处理、编码和解码阶段组成 114
图5.8 二叉树中每个节点都由根通路串唯一确定 116
图5.9 Σ = { 'A','E','G','M','S' }两种编码方案对应的二叉编码树 116
图5.10 BinNode模板类的逻辑结构 118
图5.11 二叉树节点左孩子插入过程 119
图5.12 右节点插入过程：(a)插入前；(b)插入后 122
图5.13 右子树接入过程：(a)接入前；(b)接入后 122
图5.14 二叉树遍历的全局次序由局部次序规则确定 124
图5.15 二叉树（上）的先序遍历序列（下） 124
图5.16 二叉树的后序遍历序列 125
图5.17 二叉树的中序遍历序列 126
图5.18 先序遍历过程：先沿最左侧通路自顶而下访问沿途节点，再自底而上依次遍历这些节点的右子树 127
图5.19 中序遍历过程：顺着最左侧通路，自底而上依次访问沿途各节点及其右子树 128
图5.20 迭代式中序遍历实例（出栈节点以深色示意） 129
图5.21 中序遍历过程中，在无右孩子的节点处需做回溯 131
图5.22 后序遍历过程也可划分为模式雷同的若干段 132
图5.23 迭代式后序遍历实例（出栈节点以深色示意，发生gotoHLVFL()调用的节点以大写字母示意） 133
图5.24 二叉树的层次遍历序列 134
图5.25 层次遍历实例（出队节点以深色示意） 134
图5.26 完全二叉树实例及其宏观结构 135
图5.27 满二叉树实例及其宏观结构 135

图5.28 为实现PFC编码和解码过程所需的数据结构和算法 136
图5.29 子集的PFC编码树合并后，即是全集的一棵PFC编码树 136
图5.30 最优编码树的双子性 140
图5.31 最优编码树的层次性 140
图5.32 通过节点交换提高编码效率完全二叉树与满二叉树 140
图5.33 考虑字符出现频率，以平均带权深度衡量编码效率 141
图5.34 若考虑出现频率，完全二叉树或满树未必最优 142
图5.35 若考虑出现频率，最优编码树往往不是完全二叉树 142
图5.36 最优编码树的层次性 142
图5.37 最优编码树中底层兄弟节点合并后，依然是最优编码树 143
图5.38 Huffman树构造算法实例 144
图6.1 (a)无向图、(b)混合图和(c)有向图 151
图6.2 通路与简单通路 152
图6.3 环路与简单环路 152
图6.4 欧拉环路与哈密尔顿环路 152
图6.5 邻接矩阵（空白单元对应的边不存在，其统一取值标注于矩阵最左上角） 155
图6.6 以邻接表方式描述和实现图 158
图6.7 广度优先搜索示例 161
图6.8 深度优先搜索实例（粗边框白色，为当前顶点；细边框白色、双边框白色和黑色，分别为处于UNDISCOVERED、DISCOVERED和VISITED状态的顶点；dTime和fTime标签，分别标注于各顶点的左右） 163
图6.9 活跃期与“祖先-后代”关系之间的对应关系 164
图6.10 拓扑排序 165
图6.11 利用“DAG必有零入度顶点”的特性，实现拓扑排序 166
图6.12 基于DFS搜索的拓扑排序实例 168
图6.13 关节点 168
图6.14 双连通域 168
图6.15 DFS树根节点是关节点，当且仅当它拥有多个分支 169
图6.16 内部节点C是关节点，当且仅当C的某棵极大真子树不（经后向边）联接到C的真祖先 169
图6.17 基于DFS搜索的双连通域分解实例 171
图6.18 支撑树 174
图6.19 极小支撑树与最小支撑树 175
图6.20 最小支撑树总是会采用联接每一割的最短跨越边 176
图6.21 Prim算法示例（阴影区域示意不断扩展的子树T_k，粗线示意树边） 177
图6.22 最短路径的任一前缀也是最短路径 178
图6.23 有向带权图(a)，及其最短路径树(b)和(c) 178
图6.24 最短路径子树序列 179
图6.25 Dijkstra算法示例（阴影区域示意不断扩展的子树T_k，粗线示意树边） 180
图7.1 第7章和第8章内容纵览 182

图7.2 二叉搜索树即处处满足顺序性的二叉树 184
图7.3 二叉搜索树的三个实例（左），以及三个反例（右） 184
图7.4 二叉搜索树（上）的中序遍历序列（下），必然单调非降 184
图7.5 二叉搜索树的查找过程（查找所经过的通路，以粗线条示意） 186
图7.6 searchIn()算法对返回值和_hot的语义定义：(a) 查找成功；(b) 查找失败 187
图7.7 二叉搜索树节点插入算法实例 188
图7.8 二叉搜索树节点删除算法实例 189
图7.9 由三个关键码{ 1, 2, 3 }的6种全排列生成的二叉搜索树 191
图7.10 由同一组共11个节点组成，相互等价的两棵二叉搜索树（二者在拓扑上的差异，以阴影圈出） 192
图7.11 zig(v)：顺时针旋转 193
图7.12 zag(v)：逆时针旋转 193
图7.13 在高度固定为h的前提下，节点最少的AVL树 195
图7.14 经节点删除和插入操作后，AVL树可能失衡（加减号示意平衡因子，双圈表示失衡节点） 195
图7.15 节点插入后，通过单旋操作使AVL树重新平衡 196
图7.16 节点插入后通过连续的两次旋转操作使AVL树重新平衡 197
图7.17 节点删除后经一次旋转恢复局部平衡 198
图7.18 节点删除后通过两次旋转恢复局部平衡 199
图7.19 节点插入后的统一重新平衡 200
图8.1 通过自下而上的一系列等价变换，可使任一节点上升至树根 205
图8.2 简易伸展树的最坏情况 205
图8.3 通过zig-zig操作，将节点v上推两层 206
图8.4 通过zig-zag操作，将节点v上推两层 206
图8.5 通过zig操作，将节点v上推一层，成为树根 207
图8.6 双层调整策略的高度折半效果 207
图8.7 伸展树中较深的节点一旦被访问到，对应分支的长度将随即减半 208
图8.8 伸展树的节点插入 210
图8.9 伸展树的节点删除 211
图8.10 二叉搜索树与四路搜索树 213
图8.11 B-树的宏观结构（外部节点以深色示意，深度完全一致，且都同处于最底层） 214
图8.12 (a) 4阶B-树；(b) B-树的紧凑表示；(c) B-树的最紧凑表示 215
图8.13 B-树的查找过程 217
图8.14 通过分裂修复上溢节点 220
图8.15 3阶B-树插入操作实例（I） 221
图8.16 3阶B-树插入操作实例（II） 222
图8.17 下溢节点向父亲“借”一个关键码，父亲再向左兄弟“借”一个关键码 223
图8.18 下溢节点向父亲“借”一个关键码，父亲再向右兄弟“借”一个关键码 223
图8.19 下溢节点向父亲“借”一个关键码，然后与左兄弟“粘接”成一个节点 224
图8.20 3阶B-树删除操作实例（I） 226

图8.21 3阶B-树删除操作实例（II） 227
图8.22 通过假想式地引入外部节点（黑色正方形），将二叉树扩展为真二叉树 228
图8.23 红黑树到4阶B-树的等价转换（在完全彩色版尚未出版之前本书约定，分别以圆形、正方形和八角形表示红黑树的红节点、黑节点和颜色未定节点，以长方形表示B-树节点） 229
图8.24 红黑树的黑高度不低于高度的一半；反之，高度不超过黑高度的两倍 229
图8.25 双红修正第一种情况（RR-1）及其调整方法（上方、下方分别为红黑树及其对应B-树的局部） 231
图8.26 双红修正第二种情况（RR-2）及其调整方法（带问号的关键码可能存在） 232
图8.27 双红修正流程图 232
图8.28 删除节点x之后，红黑树条件(4)：(a)或依然满足，(b)或经重染色后重新满足，(c)或不再满足 234
图8.29 双黑修正（情况BB-1）（带问号的关键码可能存在，且颜色不定） 235
图8.30 双黑修正（情况BB-2-R）（带问号的黑关键码可能但不会同时存在） 235
图8.31 双黑修正（情况BB-2-B） 236
图8.32 双黑修正（情况BB-3） 236
图8.33 双黑修正流程图 237
图8.34 一维范围查询 239
图8.35 通过预先排序，高效地解决一维范围查询问题（p_{-1}为假想着引入的哨兵，数值等于$-\infty$） 239
图8.36 平面范围查询（planar range query） 240
图8.37 平衡二叉搜索树：叶节点存放输入点，内部节点等于左子树中的最大者 240
图8.38 借助平衡二叉搜索树解决一维范围查询问题（针对区间端点的两条查找路径加粗示意） 241
图8.39 2d-树中各节点对应的区域，逐层递归地按所包含的输入点数均衡切分 242
图8.40 2d-树的构造过程，就是对平面递归划分的过程 243
图8.41 基于2d-树的平面范围查询 （A～J共计10个输入点；命中子树的根节点，以双线圆圈示意） 244
图9.1 三国人物的词典结构 247
图9.2 跳转表的总体逻辑结构 250
图9.3 跳转表节点插入过程(a~d)，也是节点删除的逆过程(d~a) 256
图9.4 四联表节点插入过程 257
图9.5 直接使用线性数组实现电话簿词典 260
图9.6 散列函数 261
图9.7 除余法 262
图9.8 素数表长可降低冲突的概率并提高空间的利用率 262
图9.9 MAD法可消除散列过程的连续性 263
图9.10 通过槽位细分排解散列冲突 267
图9.11 利用建立独立链排解散列冲突 267
图9.12 利用公共溢出区解决散列冲突 268
图9.13 线性试探法 268
图9.14 线性试探法对应的查找链 269
图9.15 通过设置懒惰删除标记，无需大量词条的重排即可保证查找链的完整 270
图9.16 线性试探法会加剧聚集现象，而平方试探法则会快速跳离聚集区段 273

图9.17 平方试探法 274
图9.18 即便散列表长取为素数（M = 11），在装填因子λ > 50%时仍可能找不到实际存在的空桶 274
图9.19 分两步将任意类型的关键码，映射为桶地址 275
图9.20 利用散列表对一组互异整数排序 277
图9.21 利用散列表对一组可能重复的整数排序 277
图9.22 利用散列法，在线性时间内确定n个共线点之间的最大间隙 278
图10.1 以获奖先后为优先级，由前12届图灵奖得主构成的完全二叉堆 286
图10.2 按照层次遍历序列，对完全二叉树节点做编号（其中圆形表示内部节点，方形表示外部节点） 287
图10.3 完全二叉堆词条插入过程 289
图10.4 完全二叉堆词条插入操作实例 290
图10.5 完全二叉堆词条删除过程 291
图10.6 完全二叉堆词条删除操作实例 292
图10.7 堆合并算法原理 293
图10.8 Floyd算法实例（虚线示意下滤过程中的交换操作） 294
图10.9 就地堆排序 295
图10.10 就地堆排序实例：建堆 296
图10.11 就地堆排序实例：迭代 296
图10.12 堆合并 297
图10.13 整体结构向左倾斜，右侧通路上的节点不超过𝒪(logn)个 298
图10.14 空节点路径长度（其中有个节点违反左倾性，以双圈标出） 299
图10.15 左式堆：左孩子的npl值不小于右孩子，而前者的高度却可能小于后者 299
图10.16 左式堆的最右侧通路 300
图10.17 左式堆合并算法原理 300
图10.18 左式堆合并算法实例 301
图10.19 基于堆合并操作实现删除接口 303
图10.20 基于堆合并操作实现词条插入算法 303
图11.1 串模式匹配的蛮力算法 309
图11.2 蛮力算法的最坏情况 （也是基于坏字符策略BM算法的最好情况） 310
图11.3 蛮力算法的最好情况 （也是基于坏字符策略BM算法的最坏情况） 310
图11.4 利用以往的成功比对所提供的信息，可以避免文本串字符指针的回退 311
图11.5 利用以往的成功比对所提供的信息，有可能使模式串大跨度地右移 311
图11.6 利用此前成功比对所提供的信息，在安全的前提下尽可能大跨度地右移模式串 312
图11.7 P[j] = P[next[j]]时，必有next[j + 1] = next[j] + 1 313
图11.8 P[j] ≠ P[next[j]]时，必有next[j + 1] = next[... next[j] ...] + 1 314
图11.9 按照此前定义的next表，仍有可能进行多次本不必要的字符比对操作 315
图11.10 坏字符策略：通过右移模式串P，使T[i + j]重新得到匹配 318
图11.11 借助bc[]表的串匹配 320
图11.12 好后缀策略：通过右移模式串P，使与P后缀U匹配的W重新得到匹配 321

图11.13 借助gs[]表的串匹配：（a）模式串P及其gs[]表；(b) 文本串T 322
图11.14 MS[j]和ss[j]表的定义与含义 323
图11.15 由ss[]表构造gs[]表 324
图11.16 构造ss[]表 325
图11.17 典型串匹配算法的复杂度概览 326
图11.18 随着单次比对成功概率（横轴）的提高，串匹配算法的运行时间（纵轴）通常亦将增加 327
图11.19 Karp-Rabin串匹配算法实例： 模式串指纹hash("82818") = 82,818 % 97 = 77 329
图11.20 Karp-Rabin串匹配算法实例：模式串指纹hash("18284") = 18,284 % 97 = 48..... 330
图11.21 相邻子串内容及指纹的相关性 330
图12.1 序列的轴点（这里用高度来表示各元素的大小） 334
图12.2 有序向量经循环左移一个单元后，将不含任何轴点 335
图12.3 轴点构造算法的构思 336
图12.4 轴点构造过程 337
图12.5 轴点构造算法实例 337
图12.6 partition()算法的退化情况，也是最坏情况 339
图12.7 选取与中位数 341
图12.8 通过减治策略计算众数 342
图12.9 采用减治策略，计算等长有序向量归并后的中位数 344
图12.10 基于堆结构的选取算法 346
图12.11 基于快速划分算法逐步逼近选取目标元素..... 347
图12.12 k-选取目标元素所处位置的三种可能情况 349
图12.13 各子序列的中位数以及全局中位数 349
图12.14 将待排序向量视作二维矩阵..... 350
图12.15 递减增量、逐渐逼近策略 351
图12.16 希尔排序实例：采用增量序列{ 1, 2, 3, 5, 8, 13, 21, ... }..... 352
图12.17 (g, h)-有序向量必然(mg + nh)-有序..... 354
图12.18 经多步迭代，逆序元素可能的范围必然不断缩小 355

表格索引

表1.1 countOnes(441)的执行过程 12
表2.1 向量ADT支持的操作接口 29
表2.2 向量操作实例 30
表3.1 列表节点ADT支持的操作接口 67
表3.2 列表ADT支持的操作接口 68
表3.3 插入排序算法实例 79
表3.4 选择排序算法实例 81
表4.1 栈ADT支持的操作接口 87
表4.2 栈操作实例 87
表4.3 RPN表达式求值算法实例（当前字符以方框注明，操作数栈的底部靠左） 97
表4.4 队列ADT支持的操作接口 105
表4.5 队列操作实例（元素均为整型） 105
表5.1 Σ = { 'A', 'E', 'G', 'M', 'S' }的一份二进制编码表 114
表5.2 二进制解码过程 114
表5.3 Σ = { 'A', 'E', 'G', 'M', 'S' }的另一份编码表 115
表5.4 按照表5.3“确定”的编码协议，可能有多种解码结果 115
表5.5 在一篇典型的英文文章中，各字母出现的次数 141
表5.6 由6个字符构成的字符集Σ，以及各字符的出现频率 143
表6.1 图ADT支持的边操作接口 153
表6.2 图ADT支持的顶点操作接口 153
表8.1 双红修正算法所涉及局部操作的统计 232
表8.2 双黑修正算法所涉及局部操作的统计 237
表9.1 词典ADT支持的标准操作接口 247
表9.2 词典结构操作实例 247
表9.3 基数排序实例 280
表10.1 优先级队列ADT支持的操作接口 283
表10.2 优先级队列操作实例：选择排序（当前的最大元素以方框示意） 284
表11.1 串ADT支持的操作看接口 307
表11.2 串操作实例 307
表11.3 next表实例：假想地附加一个通配符P[-1] 313
表11.4 next表仍有待优化的实例 315
表11.5 改进后的next表实例 316
表11.6 模式串P = "DATA STRUCTURES"及其对应的BC表 319
表11.7 模式串P = "ICED RICE PRICE"对应的GS表 322
表11.8 模式串P = "ICED RICE PRICE"对应的SS表 323

算法索引

算法1.1 过直线上给定点作直角 3
算法1.2 三等分给定线段 3
算法1.3 取非极端元素 12
算法2.1 从三个苹果中选出重量不同者 57
算法4.1 RPN表达式求值 97
算法4.2 利用队列结构实现的循环分配器 107
算法8.1 构造2d-树 242
算法8.2 基于2d-树的平面范围查询 244
算法12.1 线性时间的k-选取 348
算法12.2 希尔排序 351

代码索引

代码1.1 整数数组的起泡排序 5
代码1.2 整数二进制展开中数位1总数的统计 13
代码1.3 数组元素求和算法sumI() 13
代码1.4 幂函数算法（蛮力迭代版） 14
代码1.5 数组求和算法（线性递归版） 17
代码1.6 数组倒置算法的统一入口 19
代码1.7 数组倒置的递归算法 20
代码1.8 优化的幂函数算法（线性递归版） 21
代码1.9 由递归版改造而得的数组倒置算法（迭代版） 22
代码1.10 进一步调整代码1.9的结构，消除goto语句 22
代码1.11 通过二分递归计算数组元素之和 23
代码1.12 通过二分递归计算Fibonacci数 24
代码1.13 通过线性递归计算Fibonacci数 25
代码1.14 基于动态规划策略计算Fibonacci数 26
代码2.1 向量模板类Vector 31
代码2.2 基于复制的向量构造器 32
代码2.3 重载向量赋值操作符 33
代码2.4 向量内部数组动态扩容算法expand() 34
代码2.5 向量内部功能shrink() 36
代码2.6 重载向量操作符[] 37
代码2.7 向量整体置乱算法permute() 37
代码2.8 向量区间置乱接口unsort() 38
代码2.9 重载比较器以便比较对象指针 38
代码2.10 无序向量元素查找接口find() 39
代码2.11 向量元素插入接口insert() 40
代码2.12 向量区间删除接口remove(lo, hi) 41
代码2.13 向量单元素删除接口remove() 41
代码2.14 无序向量清除重复元素接口deduplicate() 42
代码2.15 向量遍历接口traverse() 43
代码2.16 基于遍历实现increase()功能 44
代码2.17 有序向量甄别算法disordered() 45
代码2.18 有序向量uniquify()接口的平凡实现 45
代码2.19 有序向量uniquify()接口的高效实现 46
代码2.20 有序向量各种查找算法的统一search()接口 47
代码2.21 二分查找算法（版本A） 48

代码2.22 Fibonacci查找算法 53
代码2.23 二分查找算法（版本B） 55
代码2.24 二分查找算法（版本C） 56
代码2.25 向量排序器接口 60
代码2.26 向量的起泡排序 60
代码2.27 单趟扫描交换 60
代码2.28 向量的归并排序 62
代码2.29 有序向量的二路归并 63
代码3.1 列表节点模板类 68
代码3.2 列表模板类 70
代码3.3 列表类内部方法init() 71
代码3.4 重载列表类的下标操作符 72
代码3.5 无序列表元素查找接口find() 72
代码3.6 列表节点插入接口 73
代码3.7 ListNode::insertAsPred()算法 73
代码3.8 ListNode::insertAsSucc()算法 74
代码3.9 列表类内部方法copyNodes() 74
代码3.10 基于复制的列表构造方法 75
代码3.11 列表节点删除接口remove() 75
代码3.12 列表析构方法 76
代码3.13 列表清空方法clear() 76
代码3.14 无序列表剔除重复节点接口deduplicate() 76
代码3.15 列表遍历接口traverse() 77
代码3.16 有序列表剔除重复节点接口uniquify() 77
代码3.17 有序列表查找接口search() 78
代码3.18 有序列表基于排序的构造方法 78
代码3.19 列表的插入排序 80
代码3.20 列表的选择排序 81
代码3.21 列表最大节点的定位 82
代码3.22 有序列表的二路归并 82
代码3.23 列表的归并排序 83
代码4.1 Stack模板类 88
代码4.2 进制转换算法（递归版） 90
代码4.3 进制转换算法（迭代版） 91
代码4.4 括号匹配算法（递归版） 92
代码4.5 括号匹配算法（迭代版） 93
代码4.6 运算符优先级关系的定义 94
代码4.7 表达式的求值及RPN转换 95

代码4.8 皇后类 100
代码4.9 N皇后算法 101
代码4.10 迷宫格点类 102
代码4.11 查询相邻格点 103
代码4.12 转入相邻格点 103
代码4.13 迷宫寻径 103
代码4.14 Queue模板类 106
代码4.15 顾客对象 107
代码4.16 银行服务模拟 108
代码4.17 查找最短队列 108
代码5.1 二叉树节点模板类BinNode 117
代码5.2 以宏的形式对基于BinNode的操作做一归纳整理 119
代码5.3 二叉树节点左、右孩子的插入 119
代码5.4 二叉树中序遍历算法的统一入口 120
代码5.5 二叉树模板类BinTree 121
代码5.6 二叉树节点的高度更新 121
代码5.7 二叉树根、左、右节点的插入 122
代码5.8 二叉树子树的接入 122
代码5.9 二叉树子树的删除 123
代码5.10 二叉树子树的分离 123
代码5.11 二叉树先序遍历算法（递归版） 124
代码5.12 二叉树后序遍历算法（递归版） 125
代码5.13 二叉树中序遍历算法（递归版） 125
代码5.14 二叉树先序遍历算法（迭代版#2） 127
代码5.15 二叉树中序遍历算法（迭代版#1） 129
代码5.16 二叉树节点直接后继的定位 129
代码5.17 二叉树中序遍历算法（迭代版#2） 130
代码5.18 二叉树中序遍历算法（迭代版#3） 131
代码5.19 二叉树后序遍历算法（迭代版） 133
代码5.20 二叉树层次遍历算法 134
代码5.21 基于二叉树的PFC编码 136
代码5.22 实现PFC编码所需的数据结构 137
代码5.23 初始化PFC森林 137
代码5.24 构造PFC编码树 138
代码5.25 生成PFC编码表 138
代码5.26 PFC编码 138
代码5.27 PFC解码 139
代码5.28 基于二叉树的Huffman编码 145

代码5.29 HuffChar结构 145
代码5.30 Huffman编码树结构 145
代码5.31 Huffman森林结构 145
代码5.32 Huffman二进制编码串 145
代码5.33 Huffman编码表 146
代码5.34 Huffman算法：字符出现频率的样本统计 146
代码5.35 初始化Huffman森林 146
代码5.36 构造Huffman编码树 147
代码5.37 生成Huffman编码表 147
代码5.38 Huffman编码 148
代码5.39 Huffman解码 148
代码6.1 图ADT操作接口 154
代码6.2 基于邻接矩阵实现的图结构 157
代码6.3 BFS算法 160
代码6.4 DFS算法 162
代码6.5 基于DFS搜索框架实现拓扑排序算法 167
代码6.6 基于DFS搜索框架实现双连通域分解算法 170
代码6.7 优先级搜索算法框架 173
代码6.8 Prim算法的顶点优先级更新器 177
代码6.9 Dijkstra算法的顶点优先级更新器 179
代码7.1 词条模板类Entry 183
代码7.2 由BinTree派生的二叉搜索树模板类BST 185
代码7.3 二叉搜索树searchIn()算法的递归实现 186
代码7.4 二叉搜索树search()接口 186
代码7.5 二叉搜索树insert()接口 188
代码7.6 二叉搜索树remove()接口 190
代码7.7 二叉搜索树removeAt()算法 190
代码7.8 基于BST定义的AVL树接口 194
代码7.9 用于简化AVL树算法描述的宏 194
代码7.10 恢复平衡的调整方案，决定于失衡节点的更高孩子、更高孙子节点的方向 196
代码7.11 AVL树节点的插入 197
代码7.12 AVL树节点的删除 199
代码7.13 “3 + 4”重构 201
代码7.14 AVL树的统一重平衡 201
代码8.1 基于BST定义的伸展树接口 208
代码8.2 伸展树节点的调整 209
代码8.3 伸展树节点的查找 210
代码8.4 伸展树节点的插入 211

代码8.5 伸展树节点的删除 212
代码8.6 B-树节点 215
代码8.7 B-树 216
代码8.8 B-树关键码的查找 217
代码8.9 B-树关键码的插入 219
代码8.10 B-树节点的上溢处理 221
代码8.11 B-树关键码的删除 222
代码8.12 B-树节点的下溢处理 226
代码8.13 基于BST定义的红黑树接口 230
代码8.14 用以简化红黑树算法描述的宏 230
代码8.15 红黑树节点的黑高度更新 230
代码8.16 红黑树insert()接口 231
代码8.17 双红修正solveDoubleRed() 233
代码8.18 红黑树remove()接口 234
代码8.19 双黑修正solveDoubleBlack() 238
代码9.1 词典结构的操作接口规范 248
代码9.2 Skiplist模板类 249
代码9.3 Quadlist模板类 251
代码9.4 QuadlistNode模板类 251
代码9.5 Quadlist对象的创建 252
代码9.6 Skiplist::get()查找 252
代码9.7 Skiplist::skipSearch()查找 253
代码9.8 Skiplist::put()插入 255
代码9.9 Quadlist::insertAfterAbove()插入 257
代码9.10 QuadlistNode::insertAsSuccAbove()插入 257
代码9.11 Skiplist::remove()删除 258
代码9.12 Quadlist::remove()删除 258
代码9.13 基于散列表实现的映射结构 265
代码9.14 散列表构造 265
代码9.15 确定散列表的素数表长 266
代码9.16 散列表析构 266
代码9.17 散列表的查找 271
代码9.18 散列表的查找probe4Hit() 271
代码9.19 散列表元素删除（采用懒惰删除策略） 271
代码9.20 散列表元素插入 272
代码9.21 散列表的查找probe4Free() 272
代码9.22 散列表的重散列 273
代码9.23 散列码转换函数hashCode() 276

代码10.1 优先级队列标准接口 284
代码10.2 利用统一的优先级队列接口，实现通用的Huffman编码 285
代码10.3 为简化完全二叉堆算法的描述及实现而定义的宏 288
代码10.4 完全二叉堆接口 288
代码10.5 完全二叉堆getMax()接口 288
代码10.6 完全二叉堆insert()接口的主体框架 289
代码10.7 完全二叉堆的上滤 290
代码10.8 完全二叉堆delMax()接口的主体框架 291
代码10.9 完全二叉堆的下滤 292
代码10.10 Floyd建堆算法 294
代码10.11 基于向量的就地堆排序 297
代码10.12 左式堆PQ_LeftHeap模板类定义 298
代码10.13 左式堆合并接口merge() 302
代码10.14 左式堆节点删除接口delMax() 303
代码10.15 左式堆节点插入接口insert() 303
代码11.1 蛮力串匹配算法（版本一） 309
代码11.2 蛮力串匹配算法（版本二） 310
代码11.3 KMP主算法（待改进版） 313
代码11.4 next表的构造 314
代码11.5 改进的next表构造算法 316
代码11.6 BM主算法 317
代码11.7 BC表的构造 319
代码11.8 GS表的构造 326
代码11.9 Karp-Rabin算法相关的预定义 328
代码11.10 Karp-Rabin算法主体框架 329
代码11.11 指纹相同时还需逐个字符地比对 330
代码11.12 串指纹的快速更新 331
代码11.13 提前计算M^(m-1) 331
代码12.1 向量的快速排序 335
代码12.2 轴点构造算法（版本A） 336
代码12.3 轴点构造算法（版本B） 340
代码12.4 众数查找算法主体框架 342
代码12.5 候选众数核对算法 342
代码12.6 候选众数选取算法 343
代码12.7 中位数蛮力查找算法 343
代码12.8 等长有序向量归并后中位数算法 344
代码12.9 不等长有序向量归并后中位数算法 346
代码12.10 基于快速划分的k-选取算法 348

关键词索引

（按关键词中各汉字的声母及各英文单词的首字母排序，比如“大O记号”对应于“DOJH”）

A

AVL树 (AVL tree) 194

B

边 (edge) 110, 150
八叉树 (octree) 204
波峰集（frontier） 159
比较树 (comparison tree) 58
遍历 (traversal) 123, 150
遍历树 (traversal tree) 159
编码 (encoding) 114
Boyer-Moore算法（Boyer-Moore Algorithm） 317
B-树 (B-tree) 214
闭散列 (closed hashing) 268
不稳定算法 (unstable algorithm) 61, 337
半线性结构 (semi-linear structure) 110, 150

C

层 (level) 250
出边 (outgoing edge) 151
成本 (cost) 174
层次遍历 (level-order traversal) 133
出队 (dequeue) 105
出度 (out-degree) 151
词典（dictionary） 246, 247, 248
持久性结构 (persistent structure) 204
串模式匹配 (string pattern matching) 308
插入排序 (insertionsort) 79, 353
初始化 (initialization) 32
重散列（rehashing） 273

常数时间复杂度算法 (constant-time algorithm) 12
词条 (entry) 183, 246, 283
错误 (error) 42
槽位 (slot) 267
重写 (override) 121, 185, 279
抽象数据类型 (abstract data type, ADT) 26
除余法（division method） 262
出栈 (pop) 87
重载（overload） 20, 37, 100, 145, 276, 285
查找长度 (search length) 50, 191, 269
查找链 (probing chain) 269

D

堆 (heap) 286
顶层 (top) 250
底层 (bottom) 250
对称二叉B-树（symmetric binary B-tree） 228
多重继承（multiple inheritance） 249, 288, 298
多槽位法（multiple slots） 267
顶点 (vertex) 110, 150
迭代式后序遍历（iterative postorder traversal） 131
迭代式先序遍历（iterative preorder traversal） 126
迭代式中序遍历（iterative inorder traversal） 128
递归调用 (recursive call) 16
递归跟踪 (recursion trace) 17, 63, 83, 89, 168, 302
递归基 (base case of recursion) 17
递归式后序遍历（recursive postorder traversal） 125
递归式先序遍历（recursive preorder traversal） 124
递归式中序遍历（recursive inorder traversal） 125
递减增量 (diminishing increment) 351
队列 (queue) 104
多路递归 (multi-way recursion) 23
独立链（separate chaining） 267
多路搜索树 (multi-way search tree) 213
大Ω记号（big-omega notation） 10
大O记号 (big-O notation) 9
堆排序 (heapsort) 295

带权图 (weighted graph) 152
度数 (degree) 111, 151
对数多项式时间复杂度算法 (polylogarithmic-time algorithm) 13
Dijkstra算法（Dijkstra Algorithm） 178
对数时间复杂度算法 (logarithmic-time algorithm) 13
队头 (front) 105
递推方程 (recurrence equation) 19, 51, 64
动态规划 (dynamic programming) 25
大Θ记号（big-theta notation） 11
队尾 (rear) 105
对外功能接口 (interface) 26
低位字段优先 (least significant digit first) 279
多项式时间复杂度算法 (polynomial-time algorithm) 14
多项式散列码 (polynomial hash code) 276
调用栈 (call stack) 89
地址空间 (address space) 259

E

二叉编码树（binary encoding tree） 116
二叉树 (binary tree) 111
二叉树节点 (binary tree node) 117
二叉搜索树 (binary search tree) 184
二分查找（binary search） 48, 49, 50, 54, 55, 56, 183, 217, 239, 240
二分递归 (binary recursion) 23

F

封闭定址（closed addressing） 268
Fibonacci查找 (Fibonaccian search) 52
分而治之 (divide-and-conquer) 22, 23
符号表 (symbol table) 246
返回地址 (return address) 89
父节点 (parent) 111, 112
Floyd算法（Floyd Algorithm） 293
分摊分析 (amortized analysis) 35, 53, 204, 206, 227, 273, 315
分摊运行时间 (amortized running time) 35
范围查询 (range query) 239

非线性结构 (non-linear structure) 150
复杂度下界 (lower bound) 58

G

根 (root) 110
割（cut） 175
归并排序 (mergesort) 61, 83
高度 (height) 111
广度优先搜索 (Breadth-First Search, BFS) 159
广度优先搜索树 (BFS tree) 160
广度优先搜索森林 (BFS forest) 161
公共溢出区 (overflow area) 268
关节点 (articulation point) 168
关键码 (key) 61, 137, 146, 213, 246, 248, 283, 327
关联 (incident) 151
关联数组 (associative array) 247
根通路串 (root path string) 116
构造函数 (constructor) 32

H

弧（arc） 150
Huffman编码树（Huffman encoding tree） 143
后代 (descendant) 111
黑高度 (black height) 118, 228, 230
红黑树 (red-black tree) 228
混合图 (mixed graph) 151
好后缀 (good suffix) 321
后继 (successor) 28, 67
画家算法 (painter's algorithm) 320, 324
后进先出（last-in-first-out, LIFO） 87, 90
环路 (cycle) 152
h-排序 (h-sorting) 354
回溯 (backtracking) 99
黑深度 (black depth) 228
后向边 (back edge) 163
活跃函数实例 (active function instance) 89

活跃期 (active duration) 163
h-有序 (h-ordered) 354
后缀 (suffix) 28, 67, 306
后缀表达式 (postfix) 97
坏字符 (bad character) 318
孩子节点 (child) 111

J

节点 (node) 66, 67, 110, 112, 150, 186
建堆 (heapification) 292
节点的分裂 (split) 219
节点的合并 (merge) 61, 224
简单环路 (simple cycle) 152
就地算法 (in-place algorithm) 12
简单图 (simple graph) 151
简单通路 (simple path) 152
减而治之 (decrease-and-conquer) 17, 48, 185, 343, 344
几何分布（geometric distribution） 254
聚集 (clustering) 263
渐进分析 (asymptotic analysis) 9
解码 (decoding) 114
计算机科学 (computer science) 2
计算科学 (computing science) 2
基数排序 (radixsort) 279
具体实现 (implementation) 26
极小支撑树 (minimal spanning tree, MST) 175
记忆（memoization） 25
基于比较式算法 (comparison-based algorithm, CBA) 58, 59, 82, 246, 277, 278
剪枝 (pruning) 99

K

跨边 (cross edge) 160, 163
空串 (null string) 306
k叉树 (k-ary tree) 112
可达分量 (reachable component) 159
kd-树 (kd-tree) 242

开放定址 (open addressing) 268
客户 (client) 107
空节点路径长度 (null path length) 118, 299
空间复杂度 (space complexity) 11
可计算性 (computability) 7
可扩充向量 (extendable vector) 33, 34
KMP算法（KMP Algorithm） 311
Karp-Rabin算法（Karp-Rabin Algorithm） 327
快速划分 (quick partitioning) 336
开散列（open hashing） 268
快速排序 (quicksort) 334
k-选取 (k-selection) 341
跨越边 (crossing edge) 175
可有效求解的（tractable） 14

L

列表 (list) 28, 66
链表 (linked list) 66
鲁棒性 (robustness) 7
连通分量 (connected component) 159
懒惰删除 (lazy removal) 270
邻接 (adjacent) 151
链接 (link) 66
路径 (path) 151
邻接表 (adjacency list) 158, 174
邻接矩阵 (adjacency matrix) 155
离线算法 (offline algorithm) 57
旅行商问题 (traveling salesman problem) 99
轮值 (round robin) 107

M

MAD法（multiply-add-divide method） 263
满二叉树 (full binary tree) 135, 192, 293, 294, 300
末节点 (last node) 71, 131, 251
模式定位 (pattern location) 308
模式检测 (pattern detection) 308

模式计数 (pattern counting) 308
模式枚举 (pattern enumeration) 308

N

内部节点 (internal node) 111
逆波兰表达式 (reverse Polish notation, RPN) 96
难解的（intractable） 15
难解性 (intractability) 8
逆序对（inversion） 355

O

欧拉环路 (Eulerian tour) 152

P

排队论（queuing theory） 107
PFC编码树（PFC encoding tree） 116
平凡后缀 (trivial suffix) 306
平凡前缀 (trivial prefix) 306
平方取中法 (mid-square) 264
平方试探（quadratic probing） 274
平凡子串 (trivial substring) 306
平衡二叉搜索树 (balanced binary search tree, BBST) 192
平衡因子 (balance factor) 194
平均情况 (average case) 10
平均运行时间 (average running time) 35
平面范围查询（planar range query） 240
Prim算法（Prim Algorithm） 175
排序 (sorting) 4
偏序 (partial order) 282

Q

桥 (bridge) 175
起点 (origin) 151
起点 (source) 178

切割节点 (cut vertex) 168
清理 (cleanup) 33
起泡排序 (bubblesort) 4, 5, 9, 10, 14, 60
前驱 (predecessor) 28, 67
期望运行时间 (expected running time) 35
全序 (full order) 282
前向边 (forward edge) 163
权重 (weight) 152
前缀 (prefix) 28, 67
前缀 (prefix) 306
前缀无歧义编码 (prefix-free code) 115

R

入边 (incoming edge) 151
入队 (enqueue) 105
入度 (in-degree) 151
入栈 (push) 87

S

树边 (tree edge) 160, 162
哨兵节点 (sentinel node) 71, 73, 74, 75, 239, 250
输出 (output) 5
输出敏感的 (output sensitive) 240, 241
四叉树 (quadtree) 204
深度 (depth) 111
深度优先搜索（Depth-First Search, DFS） 162
深度优先搜索树 (DFS tree) 163
深度优先搜索森林 (DFS forest) 163
双红 (double red) 231
双黑 (double black) 234
随机存储机（Random Access Machine, RAM） 9
首节点 (first node) 71, 76, 83, 135, 251, 253
时间复杂度 (time complexity) 8
数据局部性（data locality） 204, 269, 274
随机生成 (randomly generated) 191
随机组成 (randomly composed) 191

上滤 (percolate up) 289
散列 (hashing) 259
四联表 (quadlist) 250
散列表 (hashtable) 145
散列表 (hashtable) 259
散列冲突 (collision) 261
散列地址 (hashing address) 259
散列函数 (hash function) 259
散列码 (hash code) 275
双连通域 (bi-connected component) 168
势能分析法（potential analysis） 208
输入 (input) 5
输入敏感的（input sensitive） 39, 353
搜索 (search) 183
试探 (probing) 99
顺序查找 (sequential search) 39
上溢 (overflow) 33, 219
伸展 (splaying) 205
数组 (array) 28
数字分析法 (selecting digits) 264
伸展树 (splay tree) 204

T

图 (graph) 150
塔 (tower) 250
桶 (bucket) 259
头顶点 (head) 151
退化 (degeneracy) 7
头节点 (header) 71
通路 (path) 151
图灵机（Turing Machine, TM） 9
拓扑排序 (topological sorting) 166
桶排序 (bucketsort) 277
图搜索 (graph search) 159
桶数组 (bucket array) 259
跳转表 (skip list) 249

W

外部节点 (external node) 111, 186, 214
尾顶点 (tail) 151
尾递归 (tail recursion) 22
伪对数的 (pseudo-logarithmic) 16
稳定算法 (stable algorithm) 61, 80, 277
稳定性 (stability) 55, 61, 280
尾节点 (trailer) 71, 255
网络 (network) 152
完美散列 (perfect hashing) 260
完全二叉堆 (complete binary heap) 286
完全二叉树 (complete binary tree) 135, 194, 286, 287, 293, 298
伪随机试探法（pseudo-random probing） 275
无向边 (undirected edge) 150
无向图 (undigraph) 151
伪线性的 (pseudo-linear) 16
无序向量 (unsorted vector) 39
位异或法 (xor) 264
位置 (position) 28, 66, 118

X

希尔排序 (Shellsort) 350
析构函数 (destructor) 33
循关键码访问 (call-by-key) 183, 185
先进先出 (first-in-first-out, FIFO) 105, 107, 282
下滤 (percolate down) 291
向量 (vector) 28, 29
序列 (sequence) 28
相邻 (adjacent) 151
循链接访问 (call-by-link) 66
选取 (selection) 341
稀疏图 (sparse graph) 158
循位置访问 (call-by-position) 66
线性递归 (linear recursion) 17
线性结构 (linear structure) 110

线性时间复杂度算法 (linear-time algorithm) 14
线性试探（linear probing） 268
线性数组 (linear array) 28
下溢 (underflow) 36, 223
循优先级访问 (call-by-priority) 282
循秩访问 (call-by-rank) 29, 66, 157, 291, 344
循值访问 (call-by-value) 246
选择排序 (selectionsort) 80, 283

Y

源点 (source) 178
有根树 (rooted tree) 110
叶节点 (leaf) 111
易解的（tractable） 14
叶节点带权平均深度（weighted average leaf depth） 141
叶节点平均深度 (average leaf depth) 139
有穷性 (finiteness) 6
映射 (map) 246
意外 (exception) 42
有向边 (directed edge) 150
有序二叉树 (ordered binary tree) 111
优先级 (priority) 283
优先级队列 (priority queue) 283
优先级数 (priority number) 173
优先级搜索 (Priority-First Search, PFS) 173
有序列表 (sorted list) 77
有序树 (ordered tree) 113
有向图 (digraph) 151
有向无环图 (directed acyclic graph, DAG) 152, 166
有序向量 (sorted vector) 29, 44
野指针 (wild pointer) 34

Z

栈 (stack) 86
秩 (rank) 28, 29
帧 (frame) 89

制表（tabulation） 25
子串 (substring) 306
支撑树 (spanning tree) 174
终点 (destination) 151
栈顶 (stack top) 87
栈底 (stack bottom) 87
轴点 (pivot) 335
折叠法 (folding) 264
最低共同祖先 (lowest common ancestor, LCA) 241
最大间隙 (maximum gap) 278
最短路径树（shortest-path tree） 178
字典序（lexicographical order） 61, 113, 279
真二叉树 (proper binary tree) 111, 228
字符表 (alphabet) 306
字符串 (string) 306
字符集 (alphabet) 114, 116, 136, 139
最高连通祖先 (highest connected ancestor, HCA) 169
最高左侧可见叶节点 (highest leaf visible from left, HLVFL) 132
自环 (self-loop) 151
组合 (combination) 354
真后代 (proper descendant) 111
最好情况 (best case) 10
最坏情况 (worst case) 10
最坏情况下最优的 (worst-case optimal) 58
栈混洗 (stack permutation) 91
真后缀 (proper suffix) 306
直接后继 (immediate successor) 28, 67
直接前驱 (immediate predecessor) 28, 67
最佳优先搜索 (Best-First Search, BFS) 173
增量 (increment) 350
正确性 (correctness) 6
真前缀 (proper prefix) 306
子树 (subtree) 111
众数 (majority) 342
左式堆 (leftist heap) 298
再散列（double hashing） 275
指数时间复杂度算法 (exponential-time algorithm) 15
装填因子 (load factor) 33, 261, 272

自调整列表（self-adjusting list） 205
指纹（fingerprint） 328
中位点 (median point) 242
中位数 (median) 341
祖先 (ancestor) 111
在线算法 (online algorithm) 57, 116
执行栈 (execution stack) 89
最小支撑树 (minimum spanning tree, MST) 175
最优编码树（optimal encoding tree） 139
最右侧通路 (rightmost path) 300
中缀表达式（infix） 97
真子串 (proper substring) 306
最左侧通路 (leftmost path) 126
真祖先 (proper ancestor) 111

内容简介

本书按照面向对象程序设计的思想，根据作者多年的教学积累，系统地介绍各类数据结构的功能、表示和实现，对比各类数据结构适用的应用环境；结合实际问题展示算法设计的一般性模式与方法、算法实现的主流技巧，以及算法效率的评判依据和分析方法；以高度概括的体例为线索贯穿全书，并通过对比和类比揭示数据结构与算法的内在联系，帮助读者形成整体性认识。

配套《习题解析》涵盖验证型、拓展型、反思型、实践型和研究型习题，总计290余道大题、525多道小题，激发读者的求知欲，培养自学能力和独立思考习惯。本书及《习题解析》共计配有340多组、400余幅插图结合简练的叙述，40多张表格列举简明的规范、过程及要点，280余段代码及算法配合详尽而简洁的注释，使深奥抽象的概念和过程得以具体化且便于理解和记忆；推荐20余册经典的专著与教材，提供40余篇重点的学术论文，便于读者进一步钻研和拓展。

结合学生基础、专业方向、教学目标及允许课时总量等各种因素，本书推荐了若干种典型的教学进度及学时分配方案，供授课教师视具体情况参考和选用。

勘误表、插图、代码以及配套讲义等相关教学资料，均以电子版形式向公众开放，读者可从本书主页直接下载：http://dsa.cs.tsinghua.edu.cn/~deng/ds/dsacpp/