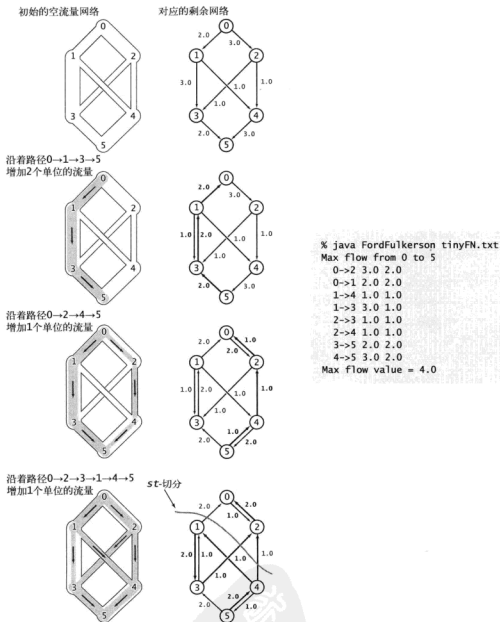




899 图 6.0.23 最短增广路径的 FordFulkerson 算法的轨迹

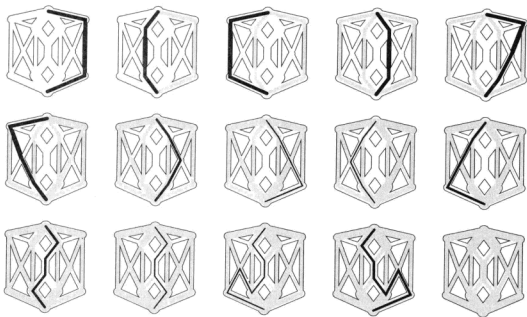


图 6.0.24 一个较大的流量网络中的最短增广路径

6.0.4.8 性能

图 6.0.24 所示的是一个更大的例子。从图中我们可以清晰地看到，增广路径的长度在慢慢变长。这是分析算法性能的第一个要点。

命题 G. 最短增广路径的 Ford-Fulkerson 最大流量算法在处理含有 V 个顶点和 E 条边的流量网络时找到的增广路径最多为 $EV/2$ 条。

简略证明。 每条增广路径中都含有一条关键边——这条边在剩余网络中会被删掉，因为它对应的可能是一条将会被充满的正向边或是将会被抽干的逆向边。每当一条边成为关键边时，通过它的增广路径的长度就会加 2（请见练习 6.39）。因为增广路径的最大长度为 E 且每条边最多可能出现在 $V/2$ 条增广路径上，因此增广路径的总数最多为 $EV/2$ 。

900

推论。 Ford-Fulkerson 算法的最短增广路径实现所需的时间在最坏情况下为 $VE^2/2$ 。

证明。 广度优先搜索最多会检查 E 条边。

命题 G 所述的上界是非常保守的。例如，图 6.0.24 中含有 11 个顶点和 20 条边，该上界说明算法使用的增广路径最多为 110 条，但实际上它只用了 14 条。

6.0.4.9 其他实现

Edmonds 和 Karp 发明的另一种 Ford-Fulkerson 算法的实现是优先处理能够将流量增大最多的增广路径。简单起见，我们将这种方法称为最大容量增广路径的最大流量算法。对于这种（以及其他一些）方法，可以通过稍加修改 Dijkstra 的最短路径算法、由优先队列得到剩余容量最大的正向

边或是流量最大的逆向边来实现。或者也可以寻找最长增广路径，或是随机选择增广路径。要完整分析哪种才是最佳的方法是一个复杂的任务，因为它们的运行时间取决于：

- 找到最大流量所需检查的增广路径数量；
- 寻找每条增广路径所需的时间。

这些量的变化可能很大，和流量网络本身以及图的搜索策略有关。人们还发明了解决最大流量问题的其他几种算法，其中一些在实践中和 Ford-Fulkerson 算法不分高下。但是，为最大流量算法进行数学建模来验证这些猜想是一个非常困难的问题。各种最大流量算法的分析仍然是一个有趣而活跃的研究领域。从理论角度来说，我们已经得到了各种最大流量算法在最坏情况下的上界，但这些上界大多远远高于实际应用中观察到的真实成本，而且也比较小的下界（线性级别）高出许多。最大流量问题的已知成本和潜在成本之间的差距比（目前）本书中讨论过的任何问题都要大。

901

最大流量算法的实际应用仍然既是一门艺术也是一门科学。它的艺术之处在于为特定的应用场景选择最有效的策略；它的科学之处在于对问题本质的理解。是否存在能够在线性时间内解决最大流量问题的新数据结构和算法呢？或者我们能否证明它们不存在呢？请见表 6.0.6。

表 6.0.6 各种最大流量算法的性能特点

算 法	在含有 V 个顶点和 E 条边的流量网络中（各边容量最大为 C ）， 算法的运行时间在最坏情况下的增长数量级
最短增广路径的 Ford-Fulkerson 算法	VE^2
最大容量的 Ford-Fulkerson 算法	$E^2 \log C$
预流推进算法（preflow-push）	$EV \log(E/V^2)$
未知算法？	$V \times E?$

902

6.0.5 问题归约

本书中，我们一直注重说明某个特定的问题，然后给出解决问题的算法和数据结构。在许多情况下（以下列出了很多），我们发现如果能够将某个问题转化为已经解决的问题的某个形式，那么解决它将会更容易。在研究已经学习过的各种算法与形形色色的各种问题之间的关系之前，我们应该正式定义这个解决问题的过程。

定义。如果能够用解决问题 B 的算法得到一个解决问题 A 的算法，则说问题 A 能够被归约为问题 B。

这个概念在软件开发中显然并不陌生：当你使用一个库方法解决某个问题时，正是在将所需要解决的问题归约为该库方法所解决的问题。本书中，我们一直非正式地将能够归约为给定问题的其他问题称为应用。

6.0.5.1 排序问题

我们在第 2 章第一次遇到了问题的归约，当时我们想说明的是高效的排序算法可以用于解决许多看起来与排序无关的其他问题。例如，在许多有趣的问题中，我们研究了以下几个问题。

- 寻找中位数。给定一组数字的集合，找出中位数。

- 不重复的值。在给定的集合中找出所有不同的值。
- 最小平均完成时间的调度问题。给定一组任务的集合和它们的时耗，在一个处理器上应该如何安排调度使得它们的平均完成时间最小呢？

命题 H。以下问题可以被归约为排序问题：

- 寻找中位数；
- 统计不同的值；
- 最小平均完成时间的调度问题。

证明。请见 2.5.3.4 节和练习 2.5.12。

我们还需要注意归约的成本。例如，我们可以在线性时间内找到一组数的中位数，但是如果归约为排序问题，那就需要线性对数级别的时间。即使是这样，额外的成本或许还是可以接受的，因为我们可以使用已有的排序实现。排序的价值在于以下 3 个方面：

- 它有其自身的实用性；
- 我们的算法能够有效解决排序问题；
- 许多问题都能够归约为排序问题。

一般来说，我们将具有这些性质的问题称为问题解决模型。和成熟的库一样，设计良好的问题解决模型能够大大扩展我们能够处理的问题域。但是，在过度关注于问题解决模型时容易犯下的一个错误被称为 Maslow 的锤子，这是由 A.Maslow 在 20 世纪 60 年代提出并广为人知的一句话：如果你有一把锤子，那么什么东西看起来都像颗钉子。如果沉迷于若干问题解决模型，我们就可能将它们当作 Maslow 的锤子一样来解决遇到的所有问题，从而妨碍了发现解决问题的更好方法，甚至是新的问题解决模型。尽管本书所讨论的模型都非常重要、实用且应用广泛，但是考虑各种其他可能性仍然是明智的选择。

6.0.5.2 最短路径问题

在 4.4 节学习最短路径算法时也遇到了问题归约的概念。在许多有趣的问题中，我们研究了以下几个。

- 无向图中的单点最短路径问题。给定一幅加权无向图和起点 s ，其中所有权重非负，回答“是否存在从 s 到给定目的顶点 v 的路径？如果有，找出这样一条最短路径（总权重最小）。”等类似问题。
- 优先级限制下的并行任务调度问题。给定一组需要完成的任务，以及一组关于任务完成的先后次序的优先级限制。在满足限制条件的前提下应该如何在若干相同的处理器上（数量不限）安排任务并在最短的时间内完成所有任务？
- 套汇。在给定的汇率表中找出一个套汇的机会。

和刚才一样，后两个问题看起来和最短路径问题并没有直接的关系，但最短路径算法能够有效地解决它们。这些示例问题虽然都很重要，但并没有什么代表性。许多非常重要的问题（太多了，无法一一讨论）都能够归约为最短路径问题——这是一个非常有效而重要的问题解决模型。

903

904

命题 I。 以下问题能够归约为加权图中的最短路径问题：

- ☐ 非负权重的无向图中的单点最短路径问题；
- ☐ 优先级限制下的并行调度问题；
- ☐ 套汇问题；
- ☐ 其他许多问题。

例证。 请见 4.4.4.2 节命题 R、4.4.5.2 节框注“优先级调度示例问题的关键路径方法”和 4.4.6.9 节框注“货币兑换中的套汇”。

6.0.5.3 最大流量问题

最大流量问题在许多情况下同样非常重要。我们可以去掉流量网络中的各种限制并解决相关的流量问题，也可以用它解决其他网络或者图的处理问题，甚至是非网络问题。例如以下问题。

- ☐ **就业安置。** 大学里的就业指导中心会为学生安排公司面试。这些面试的结果是一系列工作机会。假设一次成功的面试表示了学生和公司之间的相互认可且学生将会接受这份职位，那么这样的就业安置数量当然是越多越好。有可能为每一位学生安排一份工作吗？最多可能安排多少份工作？
- ☐ **产品配送。** 假设有一家只生产一种产品的公司，它拥有能够生产产品的工厂，能够暂时储存产品的物流分配中心以及销售商品的零售直营店。公司需要定期将产品通过物流分配中心分发到各地的直营店，而各地的分配通道的配送能力各有不同。有可能使各地仓库的供应量与直营店的销售量相匹配吗？
- ☐ **网络可靠性。** 一种简化的模型可以将一个计算机网络看成是通过交换机连接所有电脑的一组主干网，任意两台电脑都能够通过交换机和主干线相互连接。切断某一对计算机之间的连接最少需要切断多少条主干线？

同样，这些问题各不相同，也看起来不属于流量网络的问题范畴，但它们都可以被归约为最大

905 流量问题。

命题 J。 以下问题可以归约为最大流量问题：

- ☐ 就业安置；
- ☐ 产品配送；
- ☐ 网络可靠性；
- ☐ 其他许多问题。

例证。 这里只证明第一个问题（又叫做最大二分图匹配问题），其他的将留作练习。我们可以为给定的就业安置问题构造一个对应的最大流量问题。图中的所有边均由学生指向公司，然后添加一个起点且对于每个学生都有一条从起点指向他的边，添加一个终点且对于每个公司都有一条由公司指向终点的边。图中的每条边的容量都是 1，请见图 6.0.25。现在，这个网络中的最大流量问题的每个解都是对应的二分图匹配问题的解（请见命题 F 的推论）。匹配中的所有

边的两个顶点都正好分别属于学生和公司两个集合且它们在最大流量配置中都会是饱和的。首先, 网络流总是会给出一个合法的匹配: 因为每个顶点都既有一条流入边 (来自于起点) 和一条流出边 (指向终点) 且经过的流量最多为 1, 所以每个顶点最多只能出现在一个匹配中。其次, 匹配不可能含有更多的边, 因为任意类似的匹配都意味着一个比最大流量算法的结果更好的流量配置。

二分图匹配问题

1 Alice	7 Adobe
2 Bob	8 Amazon
3 Carol	9 Facebook
4 Dave	10 Google
5 Eliza	11 IBM
6 Frank	12 Yahoo

流量网络的构造

最大流量配置

匹配 (解)

Alice	—	Amazon
Bob	—	Yahoo
Carol	—	Facebook
Dave	—	Adobe
Eliza	—	Google
Frank	—	IBM

图 6.0.25 将二分图匹配问题归约为网络流问题示例

906

例如, 如图 6.0.26 所示, 一个增广路径最大流量算法可能会使用路径 $s \rightarrow 1 \rightarrow 7 \rightarrow t$ 、 $s \rightarrow 2 \rightarrow 8 \rightarrow t$ 、 $s \rightarrow 3 \rightarrow 9 \rightarrow t$ 、 $s \rightarrow 5 \rightarrow 10 \rightarrow t$ 、 $s \rightarrow 6 \rightarrow 11 \rightarrow t$ 和 $s \rightarrow 4 \rightarrow 7 \rightarrow 1 \rightarrow 8 \rightarrow 2 \rightarrow 12 \rightarrow t$ 计算得到匹配 1-8、2-12、3-9、4-7 和 6-11。因此, 在示例中可以找到一种将所有学生和工作相匹配的方法。每条增广路径都会使一条由起点指出的边和一条指向终点的边充满。我们可以注意到, 这些边都不是逆向边, 因此最多只存在 V 条增广路径, 总运行时间与 VE 成正比。

最短路径和最大流量算法都是重要的问题解决模型, 因为它们和排序算法有着相同的性质:

- 它们有其自身的实用性;
- 我们的算法能够有效解决它们;
- 许多问题都能够归约为这些模型。

这段简短的讨论只是为了介绍这个概念。如果你能学习一门有关运筹学的课程, 就将会学到许多能够归约为这些模型的其他问题以及更多的问题解决模型。

6.0.5.4 线性规划

运筹学的基础之一是线性规划 (Linear Programming, LP), 请见图 6.0.27。它的主要思想是将给定的问题归约为以下数学形式。

线性规划。给定一个由 M 个线性不等式组成的集合和含有 N 个决策变量的线性等式, 以及一个由该 N 个决策变量组成的线性目标函数, 找出能够使目标函数的值最大化的一组变量值, 或者证明不存在这样的赋值方案。



图 6.0.26 二分图匹配中的增广路径

线性规划是一种极为重要的问题解决模型，因为：

- 非常多的重要问题都能够归约为线性规划问题；
- 我们的算法能够有效解决线性规划问题。

在讨论其他问题解决模型时的“该问题有其自身的实用性”就不必提了，因为能够归约为线性规划问题的实际问题实在是太多了。

根据约束条件使得 $f+h$ 最大化

$$\begin{aligned} 0 &\leq a \leq 2 \\ 0 &\leq b \leq 3 \\ 0 &\leq c \leq 3 \\ 0 &\leq d \leq 1 \\ 0 &\leq e \leq 1 \\ 0 &\leq f \leq 1 \\ 0 &\leq g \leq 2 \\ 0 &\leq h \leq 3 \\ a &= c + d \\ b &= e + f \\ c + e &= g \\ d + f &= h \end{aligned}$$

图 6.0.27 线性规划问题示例

命题 K。以下问题均可归约为线性规划问题：

- 最大流量问题；
- 最短路径问题；
- 许许多多其他问题。

例证。我们只证明第一个问题并将第二个留作练习 6.49。考虑一个由不等式和等式所组成的系统，其中每一个约束变量都对应着一条边，两个不等式也对应着一条边，每一个等式对应着一个顶点（起点和终点除外）。约束变量的值就是边中的流量，不等式指明了边中的流量必须在 0 和边的容量之间，而等式说明指向每个顶点的所有边中的流量之和必须和从该顶点指出的所有边中的流量之和相等。任意最大流量问题都可以用这种方式归约为一个线性规划问题，而它的解又可以很容易地归约为最大流量问题的解。图 6.0.28 给出了一个具体的示例。

命题 K 中所说的“许许多多其他问题”有三个含义。第一，添加约束条件和扩展线性规划模型非常简单。第二，问题的归约是有传递性的，因此能够归约为最短路径和最大流量问题的所有问题也能够归约为线性规划问题。第三，也是更普遍的一种情况，即各种最优化问题都能够直接构造为线性规划问题。事实上，线性规划这个词的意思就是“将一个最优化问题构造为一个线性规划问题”。这种用法出现在“programming”这个词被用作计算机领域的“编程”之意之前。和非常多的问题都可以归约为线性规划问题同样重要的是，解决线性规划问题的高效算法已经发明了数十年了。其中最著名的是 G.Dantzig 在 20 世纪 40 年代发明的单纯形法（simplex algorithm）。理解单纯形法并不困难（请见本书网站上对它的简单实现）。更近一些的时候，L.G.Khachian 在 1979 年演示了

椭圆法 (ellipsoid algorithm) 并推动了 20 世纪 80 年代内点法 (interior point methods) 的发展。对于人们在现代应用中遇到的各种大型线性规划问题, 内点法是对单纯形法的有效补充。现在, 解决线性规划问题的程序都已经十分健壮、久经考验、高效并且对于现代公司机构的基本运作起到了关键的作用。它在科学领域甚至应用程序中的运用也在不断扩展。如果线性规划模型能够表示你的问题, 那么离问题的解决也就不远了。

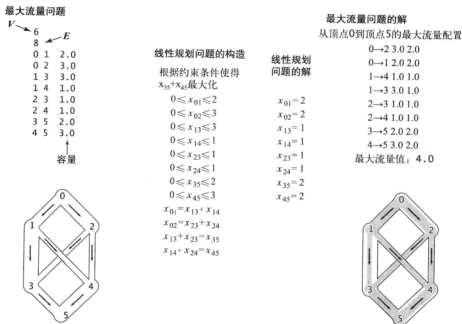


图 6.0.28 将网络流问题归约为线性规划问题

非常现实地说, 线性规划是各种问题解决模型的鼻祖, 因为非常多的问题都能向它归约。很自然, 这一点也使我们不禁思考是否存在比线性规划问题更强大的问题解决模型。还有哪些问题无法归约为线性规划问题? 下面就是一个例子。

负载均衡。给定一组任务和完成它们的时间, 应该如何两个相同的处理器上分配任务使得所有任务的总完成时间最短?

我们能够找到一个更加一般的问题解决模型并高效解决它的实例吗? 这样的思考得到的结果是不可解性, 它也将是本书的最后一个话题。

907
909

6.0.6 不可解性

本书中讨论的算法一般都是用来解决实际问题的, 因此它们消耗的资源都是有限的。大多数算法的实用性是显而易见的, 而且对于许多问题, 我们还很幸运地能够在几种不同的算法之间进行选择。但不幸的是, 现实生活中还有许多其他问题并没有如此有效的解决方法。更糟糕的是, 对于许多类问题, 人们甚至不知道是否存在有效解决它们的方法。这种情况让程序员和算法的设计者都极度沮丧, 因为他们无法为许多实际问题找到有效的算法。对于理论学者而言, 沮丧来自于他们无法证明这些问题到底有多难。在这个领域, 人们已经进行了大量的研究, 并发展出了一种方法来判断

一个新问题从技术的角度来说是否能够归于“难以解决”这个类别。尽管这方面的研究大多数都超出了本书的范畴，但是理解它们的核心思想并不困难。我们将在这里介绍它们，因为当面对一个新问题时，每个程序员都应该了解不存在解决它的高效算法的可能性。

6.0.6.1 准备工作

20 世纪最漂亮和有趣的智力发明之一，就是阿兰·图灵在 20 世纪 30 年代发明的“图灵机”。它是一个简单而又非常通用的计算模型，足以描述任意计算机程序和设备。一台图灵机就是一台能够读取输入、变换状态和打印输出的有限状态机。图灵机是理论计算机科学的基础。它来自于下面两个重要的思想。

□ 普遍性。图灵机可以模拟所有物理可实现的计算设备。这被称为丘奇-图灵论题。这是一个关于自然世界的论断且无法被证明（但可以被证伪）。该论题成立的证据就是数学家和计算机科学家已经发明的无数种计算模型，而它们都已证明和图灵机等价。

□ 可计算性。图灵机（或是任意其他计算设备，根据普遍性可以得到）无法解决的问题是存在的。这在数学上是正确的。停机问题（halting problem）（任意程序都无法保证能够判定给定程序是否会结束）就是这类问题中的一个著名的例子。

在这里，我们感兴趣的是第三个思想，它是关于计算设备效率的。

□ 扩展的丘奇-图灵论题。在任意计算设备上解决某个问题的某个程序所需的运行时间的增长数量级都是在图灵机上（或是任意其他计算设备上）解决该问题的某个程序的多项式倍数。

同样，这也是一个关于自然世界的论断，因为所有已知的计算设备都能够通过图灵机模拟，只是成本最多需要增加一个多项式的倍数。在最近几年，量子计算的概念使得一些研究者开始怀疑扩展的丘奇-图灵论题的正确性。大多数人都认为，从实践的角度来说，这个论题还能支撑一段时间，但许多学者已经在努力证明它是错误的。

6.0.6.2 指数级别的运行时间

不可解性理论的目的在于将能够区别多项式时间内解决的问题和在最坏情况下（可能）需要指数级别时间才能解决的问题。我们可以认为指数级别运行时间的算法在输入规模为 N 时所需的时间（至少）和 2^N 成正比，将底数 2 替换为任意的 $\alpha > 1$ 均可。我们一般认为指数时间的算法无法保证在合理的时间内解决规模超过（例如）100 的问题，因为无论计算机有多快都没人能够等待一个需要 2^{100} 步的算法。指数增长级别使得科技进步忽略不计：一台超级计算机可能比一张算盘快一万亿倍，但两者都不可能解决需要 2^{100} 步才能完成的问题。有时，“简单”问题和“困难”问题之间只有一线之差。例如，4.1 节中学习的那个能够解决以下问题的算法。

最短路径长度。在一幅图中从一个给定的顶点 s 到另一个给定的顶点 t 之间的最短路径的长度是多少？

但并没有学习解决下面这个问题的算法，但两者看起来本质上似乎是一样的。

最长路径长度。在一幅图中从一个给定的顶点 s 到另一个给定的顶点 t 之间的最长路径的长度是多少？

问题的核心在于，据我们目前所知，从难度上来说这些几乎都是最困难的问题。广度优先搜索能够在线性时间内解决第一个问题，但对于第二个问题所有已知算法在最坏情况下均需要指数级别的时间。前面框注的代码用一个深度优先搜索的变种解决了这个问题。它和深度优先搜索非常类似，但它检查了所有从 s 到 t 的简单路径才找到了最长的那一条。

```

public class LongestPath
{
    private boolean[] marked;
    private int max;

    public LongestPath(Graph G, int s, int t)
    {
        marked = new boolean[G.V()];
        dfs(G, s, t, 0);
    }

    private void dfs(Graph G, int v, int t, int i)
    {
        if (v == t && i > max) max = i;
        if (v == t) return;
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) dfs(G, w, t, i+1);
        marked[v] = false;
    }

    public int maxLength()
    { return max; }
}

```

找出图中的两个顶点之间的最长路径的长度

6.0.6.3 搜索问题

本书中已经介绍过的“高效”算法能够解决的问题与还需要如大海捞针一般在各种可能性中寻找解法的问题之间存在巨大差异，这就需要能够用一种简单的形式模型来研究这两类问题之间的关系。第一步就是要说明我们所研究的这类问题。

定义。如果一个问题有解且验证它的解的正确性所需的时间不会超过输入规模的多项式，则称这种问题为搜索问题。当一个算法给出了一个解或是已证明解不存在时，就称它解决了一个搜索问题。

我们将在后面讨论不可解性问题中4个比较有趣的问题。这些问题被为“可满足性”问题。现在，要证明某个问题是一个搜索问题，只需说明你能够快速验证某个完整的解的正确性即可。解决一个搜索问题就好像“在稻草堆里寻找一根针”一样，你唯一的优势只是在看见它的时候能够认得出来。例如，对于后面列出的每个可满足性问题都给定了一组变量赋值，你都能很容易地验证每个等式或不等式都是满足的，但是寻找这样一组变量赋值就完全不同了。我们常用 NP 描述所有搜索问题——我们会在 6.0.6.5 节说明这个名字的由来。

定义。NP 是所有搜索问题的集合。

NP 准确描述了所有科学家、工程师以及应用程序员渴望的能够保证在合理时间范围内解决的所有问题的集合。

部分搜索问题。

- 线性等式可满足性。给定一组由 N 个变量表示的 M 个线性等式，找出一组满足所有等式的变量赋值，或者证明这样的赋值不存在。
- 线性不等式可满足性（线性规划问题的搜索形式）。给定一组由 N 个变量表示的 M 个线性不等式，找出一组满足所有不等式的变量赋值，或者证明这样的赋值不存在。
- $0 \sim 1$ 整数线性不等式可满足性（ $0 \sim 1$ 整数线性规划问题的搜索形式）。给定一组由 N 个整数变量表示的 M 个线性不等式，找出一组满足所有不等式的变量 0 或 1 赋值，或者证明这样的赋值不存在。
- 布尔可满足性。给定一组由 N 个布尔变量以及和 / 或运算符表示的 M 个等式，找出一组满足所有等式的变量赋值，或者证明这样的赋值不存在。

6.0.6.4 其他类型的问题

对于构成了不可解性研究的基础的问题集合，搜索问题的概念是多种描述它的方法之一。其他方法包括决定性问题（解是否存在？）以及最优化问题（最优解是什么？）。例如，6.0.6.2 节中的最长路径长度问题就是一个最优化问题而非一个搜索问题。（给定一个解，无法验证它就是最长路径的长度。）这个问题的搜索版本是找到一条能够连接所有顶点的简单路径。（该问题也叫做汉密尔顿路径问题）。这个问题的决定性版本是询问是否存在一条能够连接所有顶点的简单路径。套汇问题、布尔可满足性问题和汉密尔顿路径问题都是搜索问题；询问这些问题是否有解是决定性问题；而最短或最长路径问题、最大流量问题和线性规划问题都是最优化问题。虽然它们在技术上并不等价，但搜索问题、决定性问题 and 最优化问题一般都能够相互归约（请见练习 6.58 和练习 6.59）且我们的主要结论同时适用于这三种类型的问题。

6.0.6.5 简单的搜索问题

NP 的定义并没有提到寻找解的难度，而只是和解的验证有关。构成不可解性研究的基础的第二类问题的集合被称为 P，它和寻找解的难度有关。在这个模型下，算法的效率是将输入编码所需的比特数量的函数。

定义。 P 是能够在多项式时间内解决的所有搜索问题的集合。

这个定义暗示着多项式时间是一个最坏情况下的时间界限。对于在集合 P 中的一个问题，必然存在一个算法能够保证在多项式时间内解决它。注意，我们完全没有指定这是一个怎样的多项式。线性、线性对数、平方、立方级别都是多项式时间，因此这个定义显然囊括了目前已经学习的所有标准算法。运行一个算法所需的时间取决于所使用的计算机，但扩展的丘奇-图灵论题让这一点变得无关紧要——它说明任意计算设备上的多项式时间的解都意味着任意其他计算设备上也存在多项式时间的解。排序问题属于 P 是因为（例如）插入排序所需的时间与 N^3 成正比（在这里，线性对数时间的排序算法并无意义），最短路径问题、线性等式可满足性问题以及其他许多问题也是这样。一个能够有效解决某个问题的算法足以证明该问题属于集合 P。换句话说，P 准确描述了所有科学家、工程师以及应用程序员能够保证在合理的时间范围内解决的所有问题的集合，请见表 6.0.7 和表 6.0.8。

6.0.6.6 非确定性

NP 中的 N 表示的是非确定性（nondeterminism）。它的意思是，扩展计算机能力的一种（理论上的）方法是赋予它不确定性：即断言当一个算法面对若干个选项时，它有能力“猜出”正确的选择。

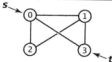
在我们的讨论中,你可以将非确定性的计算机上的一个算法看作是在“猜测”问题的解,然后验证这个解是否成立。在图灵机中,非确定性只是定义为一个给定状态和一个给定输入时的两个不同的后继状态,解则是能够得到期望结果的所有路径。非确定性也许只是一个数学上的幻想,但它也可以是一种很有用的思想。例如,在5.4节中,我们将非确定性用作了一种设计算法的工具——正则表达式模式匹配算法的基础就是有效模拟一个非确定性自动状态机。

914

表 6.0.7 集合 NP 中的问题举例

问 题	输 入	描 述	存在多项式时间算法	实 例	解
哈密顿路径	图 G	找到一条能够访问所有顶点的简单路径	?		0-2-1-3
分解质因数	整数 x	找到 x 的最大因子	?	97605257271	8784561
0-1 线性不等式可满足性	由 N 个 0-1 变量组成的 M 个不等式	找出满足所有不等式的变量赋值	?	$x-y \leq 1$ $2x-z \leq 2$ $x+y \geq 2$ $z \geq 0$	$x=1$ $y=1$ $z=0$
集合 P 中的所有问题	请见表 6.0.7				

表 6.0.8 集合 P 中的问题举例

问 题	输 入	描 述	存在多项式时间算法	实 例	解
最短 s - t 路径	图 G 顶点 s, t	找出从 s 到 t 的最短路径	广度优先搜索 (BFS)		0-3
排序	数组 a	将 a 按升序排列	归并排序	2.8 8.5 4.1 1.3	3 0 2 1
线性等式可满足性	N 个变量 M 个等式	找出满足所有等式的变量赋值	高斯消元法	$x+y=1.5$ $2x-y=0$	$x=0.5$ $y=1$
线性不等式可满足性	N 个变量 M 个不等式	找出满足所有不等式的变量赋值	椭圆法	$x-y \leq 1.5$ $2x-z \leq 0$ $x+y \geq 3.5$ $z \geq 4.0$	$x=2.0$ $y=1.5$ $z=4.0$

915

6.0.6.7 主要问题

非确定性十分强大,严肃认真地考虑它似乎有点荒唐。为什么要花心思用一种想象中的工具将困难的问题变得看起来简单呢?答案是,虽然非确定性看起来十分强大,但没人能够证明它能够帮助我们解决任何问题!换句话说,还没有人能够找到任何一个问题并证明它属于 NP 而不属于 P (甚至证明存在这样一个问题)。这就留下了一个有待解决的问题:

$P=NP$ 成立吗?

这个问题是由 K.Gödel 在 1950 年写给 J. von Neumann 的一封信中第一次提出的,并且完全推倒了所有数学家和计算机科学家。陈述这个问题的其他方式说明了一些它的基本性质。

□ 是否存在任何难以解决的搜索问题?

□ 如果能构造一种非确定性的计算设备,能够更快地解决某些搜索问题吗?

无法解答这些问题令人们极度懊恼,因为许多重要的实际问题都属于 NP 但却不一定属于 P。(已

知的最快确定性算法需要指数级别的时间。)如果能够证明它不属于 P, 就可以放弃寻找高效率的算法。既然无法证明, 那么就存在发现某种高效算法的可能性。事实上, 就我们目前的知识水平而言, NP 中的每个问题都可能存在某种高效的算法, 这意味着可能还有许多高效的算法没有被人们发现。但实际上没人相信 $P=NP$, 而且很大一部分人都在努力证明该等式不成立。它仍然是计算机科学领域有待证明的最重要的研究课题。

6.0.6.8 多项式时间问题的相互归约

6.0.5 节通过说明用以下三个步骤可以解决问题 A 的任意实例, 证明了问题 A 是可以归约为问题 B 的:

- 将 A 的实例归约为 B 的实例;
- 解决 B 的实例;
- 将 B 的实例的解归约为 A 的实例的解。

布尔可满足性问题

$$\begin{aligned} &(x_1 \text{ or } x_2 \text{ or } x_3) \text{ and} \\ &(x_1 \text{ or } x_2' \text{ or } x_3) \text{ and} \\ &(x_1' \text{ or } x_2' \text{ or } x_3') \text{ and} \\ &(x_1' \text{ or } x_2' \text{ or } x_3) \end{aligned}$$

0-1 整数线性不等式可满足性问题的构造

当且仅当第一个
子句是可满足时
 c_1 的值为 1

$$\begin{aligned} c_1 &\geq 1 - x_1 \\ c_1 &\geq x_2 \\ c_1 &\geq x_3 \\ c_1 &\leq (1 - x_1) + x_2 + x_3 \end{aligned}$$

$$\begin{aligned} c_2 &\geq x_1 \\ c_2 &\geq 1 - x_2 \\ c_2 &\geq x_3 \\ c_2 &\leq x_1 + (1 - x_2) + x_3 \end{aligned}$$

$$\begin{aligned} c_3 &\geq 1 - x_1 \\ c_3 &\geq 1 - x_2 \\ c_3 &\geq 1 - x_3 \\ c_3 &\leq (1 - x_1) + (1 - x_2) + (1 - x_3) \end{aligned}$$

$$\begin{aligned} c_4 &\geq 1 - x_1 \\ c_4 &\geq 1 - x_2 \\ c_4 &\geq x_3 \\ c_4 &\leq (1 - x_1) + (1 - x_2) + x_3 \end{aligned}$$

$$\begin{aligned} s &\leq c_1 \\ s &\leq c_2 \\ s &\leq c_3 \\ s &\leq c_4 \\ s &\geq c_1 + c_2 + c_3 + c_4 - 3 \end{aligned}$$

当且仅当所有 C
变量的值均为 1
时 s 的值为 1

图 6.0.29 将布尔可满足性问题归约为 0-1 整数线性不等式可满足性问题的示例

只要能够有效完成归约(并解决问题 B), 我们就能有效的解决问题 A。在这里, 为了效率我们采用了能够想象的最弱的定义: 为了解决问题 A 最多需要解决多项式个问题 B 的实例, 且问题归约最多只需多项式时间。在这种情况下, 我们称 A 能够在多项式时间内归约为 B。在前文中, 我们使用问题的归约介绍了各种问题解决模型, 使得高效算法所能解决的问题范围大大拓展了。现在, 我们要从另一个角度使用问题的归约, 即用它来证明一个问题是难以解决的。如果一个问题 A 已知是难以解决的, 且 A 在多项式时间内能够归约为问题 B, 那么问题 B 必然也是难以解决的。否则, 问题 B 的一个多项式时间的解必然也能归约为问题 A 的一个多项式时间内的解。

命题 L. 布尔可满足性问题能够在多项式时间内归约为 0-1 整数线性不等式可满足性问题。

证明. 对于给定的一个布尔可满足性问题的实例, 定义一组不等式, 其中每个布尔变量都对应着一个 0-1 变量, 每个布尔子句也对应着一个 0-1 变量, 如图 6.0.29 所示。若布尔变量的值为真(true)则对应的整数变量的值为 1, 值为假(false)时对应的整数变量的值为 0。这样, 我们就能够将 0-1 整数线性不等式可满足性问题的解归约为布尔可满足性问题的解。

推论. 如果可满足性问题是难以解决的, 那么整数线性规划问题也是难以解决的。

即使我们并没有精确定义难以解决，关于解决这两种问题的难度关系的陈述仍然是有意义的。在这里，“难以解决”的意思是“不包含在集合 P 中”。一般来说，我们用不可解来表示不包含在集合 P 中的问题。以 R.Karp 在 1972 年作出的开创性的工作为起点，一些研究者已经通过这种归约的方式证明了成百上千种各个应用领域的问题都是相关的。此外，这种关系的内涵远比两个单独的问题之间的联系更丰富，下面我们将说明这个概念。

6.0.6.9 NP-完全性

许多问题都属于 NP 但可能并不属于 P 。也就是说，我们可以轻易地验证任意给定的解是否有效，但即使投入了许多努力，也未能开发出一个有效的算法来寻找问题的解。令人惊讶的是，所有这些问题都有一个额外的性质，令人信服地说明了 $P \neq NP$ ：

916
?
917

定义。若 NP 中的所有问题都能在多项式时间内归约为搜索问题 A，那么则称问题 A 是 NP-完全的。

这个定义使得我们可以将“难以解决”的定义升级为“除非 $P=NP$ 否则无解”。如果任意 NP-完全问题能够通过一台有限自动机在多项式时间内解决，那么 NP 中的所有问题都将得到解决（即 $P=NP$ ）。也就是说，所有研究者对于寻找这些问题的高效算法的失败从整体上来看是证明 $P=NP$ 的失败。NP-完全问题的意思是，我们不期望能够找到多项式时间的算法。大多数实际的搜索问题都已知是 P 或 NP-完全问题。

6.0.6.10 Cook-Levin 定理

通过归约，一个问题的 NP-完全性也意味着另一个问题的 NP-完全性。但归约在一种情况下是不可用的：如何证明第一个问题是 NP-完全的？S.Cook 和 L.Levin 在 20 世纪 70 年代早期分别独立地完成了这项工作。

命题 M (Cook-Levin 定理)。布尔可满足性问题是 NP-完全的。

极大简化证明。目标是证明如果布尔可满足性问题存在多项式时间的算法，那么 NP 集合中的所有问题都能在多项式时间内解决。非确定型图灵机是可以解决 NP 中的任意问题的，因此证明的第一步是用与布尔可满足性问题中一样的逻辑表达式描述非确定型图灵机的所有特性。这可以将 NP 中的每个问题（它们都可以表示为非确定型图灵机上的一个程序）和可满足性问题的某个实例（该程序的逻辑表达式形式）联系起来。这样，可满足性问题的解本质上等价于模拟图灵机在给定的输入下运行给定的程序，因此它将产生给定问题的某个实例的解。这份证明的其他细节已经远远超出了本书的范畴。幸运的是，我们只需要证明这一个命题即可：使用归约来证明 NP-完全性要简单的多。

Cook-Levin 定理，再加围绕各种 NP-完全问题所进行的成千上万次多项式时间内的归约，使我们得到了两种可能性：或者 $P=NP$ ，即不存在任何不可解的搜索问题（所有搜索问题都能够都在多项式时间内得到解决）；或者 $P \neq NP$ ，即存在不可解的搜索问题（某些搜索问题无法在多项式时间内得到解决），请见图 6.0.30。NP-完全问题在实际应用中经常出现，因此人们找出解决它们的优秀算法的意愿非常强烈。所有这些问题目前都还未找到有效的算法显然强烈说明了 $P \neq NP$ ，大多数研究者也相信这一点。但从另一方面来说，也没人能够证明这些问题中的任意一个不属于 P ，这也同样是反方向的一个有力证据。无论 $P=NP$ 是否成立，目前的实际状态是所有 NP-完全问题的已知最佳算法在最坏情况下都需要指数级别的时间。

PDF

6.0.6.11 问题的分类

要证明一个搜索问题存在于集合 P 中，我们需要展示一个解决它的多项式时间算法，这或许可以通过将它归约为一个已知 P 类问题。要证明 NP 中的一个问题是 NP -完全的，我们需要证明某个已知的 NP -完全问题能够在多项式时间内归约为它；也就是说，如果一个新的问题的多项式时间的算法能够用于解决 NP -完全问题，那么它也就能够解决 NP 中的所有问题。我们已经用这种方法证明了成千上万的问题都是 NP -完全问题，就像在命题 L 中对整数线性规划问题进行的转换那样。后面列出了一些有代表性的问题，它包含了 Karp 提出的若干问题，但这只是已知的 NP -完全问题中极小的一部分。将新问题归入容易解决（属于集合 P ）或者难以解决（ NP -完全）的类别可能会出现以下几种情况。

- ❑ 显而易见。例如，著名的高斯消元法就能够证明线性等式可满足性问题属于集合 P 。
- ❑ 需要一些技巧但并不困难。例如，给出一份类似于命题 L 的证明需要一些经验和实践，但理解并不困难。
- ❑ 非常有挑战性。例如，线性规划问题曾经长期分类不明，但 Khachian 的椭球法证明了线性规划问题属于集合 P 。
- ❑ 有待解决。例如，图的同构问题（给定两幅图，给出一种能够使得两幅图相同的顶点重命名方案）和分解质因数问题（给定一个整数，找出它的最大因数）仍然是无解的。

目前这仍然是一块内容丰富、研究活跃的领域，每年都会产生数千篇论文。从后面项目列出的最后几个条目可以看出，它涉及了科学界的各个领域。我们在 NP 的定义中包含了科学家、工程师和应用程序员所渴望解决的所有问题——这些问题显然需要分类！

一些著名的 NP -完全问题。

- ❑ 布尔可满足性。给定一组由 N 个布尔变量表示的 M 个等式，找出一组满足所有等式的变量赋值，或者证明这样的赋值不存在。
- ❑ 整数线性规划。给定一组由 N 个整数变量表示的 M 个线性不等式，找出一组满足所有不等式的变量赋值，或者证明这样的赋值不存在。
- ❑ 负载均衡。给定一组任务和完成它们的时间以及一个时间上限 T ，应该如何在两个相同的处理器上分配任务以在时间 T 之内完成所有任务？
- ❑ 顶点覆盖。给定一幅图和一个整数 C ，找出一个含有 C 个顶点的集合，保证图中的每条边都至少依附于集合中的一个顶点。
- ❑ 哈密尔顿路径。给定一幅图，找出一条正好只经过每个顶点一次的简单路径，或者证明这种路径不存在。
- ❑ 蛋白质折叠。给定能量级别 M ，找出一种蛋白质的某种三维折叠结构，其含有的潜在能量小于 M 。
- ❑ 伊辛模型。给定一个三维晶格伊辛模型和一个能量阈值 E ，是否存在一个自由能小于 E 的子图？
- ❑ 给定收益的风险投资组合。给定一组风险投资渠道与一个总成本以及一个给定收益。每项投资都有一定的风险值，风险的总阈值为 M 。找到一种分配投资的方法使得总风险小于 M 。

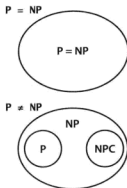


图 6.0.30 问题集的可能情况

6.0.6.12 处理 NP- 完全性

在实践中，我们必须为这些各种各样的问题找到某种解决办法，因此人们对解决这些问题非常感兴趣。我们不可能在这一小段文字中说明这个庞大的研究领域，但我们可以简要描述一下人们已经尝试过的各种手段。一种方法是，修改问题并寻找一种“近似”算法来给出接近但并非最佳的解。例如，欧几里德旅行销售员问题（traveling salesman problem），我们很容易找到一个长度小于最优路线的两倍的解。但不幸的是，在寻找更好的近似时，这种方法并不足以绕开 NP- 完全性。第二种方法是，给出一种能够有效解决实际应用中所出现的问题的实例算法，但对于最坏情况下的输入，这种算法仍然是无法找到问题的解。这种方法最著名的例子是解决整数线性规划问题的程序，它们是数十年来解决无数工业应用中的大量最优化问题的主力军。尽管它们有可能需要指数级别的时间，但实际应用中的输入数据也显然不是最坏情况下的输入。第三种方法是，使用一种叫做“回溯法”的技术来避免检查所有可能的解，以期找到尽可能“高效”的指数级别算法。最后，计算机科学的理论并没有提到多项式时间和指数时间之间的一个相当大的空档。存在运行时间与 $N^{\log N}$ 以及 $2^{\sqrt{N}}$ 成正比的算法吗？

NP- 完全性触及了本书中我们所研究过的所有应用领域：NP- 完全问题会出现在初级的编程问题、排序和查找、图处理、字符串处理、科学计算、系统编程、运筹学以及所有能够想到的需要计算的地方。NP- 完全性理论对实际生产最重要的贡献在于它给出了一种方法来鉴别来自于这些广泛领域的一个新问题是“容易”还是“困难”呢。如果有人找到了一种解决新问题的有效方法，那么它显然就没什么难度了。如果找不到，那么要是能够证明该问题是 NP- 完全的，这就说明找到一个高效算法基本上是不可能的。（因此或许应该尝试另一种思路。）本书中已经研究过的所有高效算法说明我们已经学习了自欧拉以来的多种高效的计算方法，但 NP- 完全性理论也说明事实上人们还有很长的路要走。

921

练习：碰撞模拟

- 6.1 根据正文完成 `predictCollisions()` 和 `Particle` 的实现。决定一对刚性球体进行弹性碰撞后的运动状态需要 3 个公式：(a) 动量守恒，(b) 动能守恒，(c) 碰撞时，相互作用力和碰撞点的切面垂直（假设没有摩擦力和自旋）。更多细节请见本书的网站。
- 6.2 开发一个版本的 `CollisionSystem`、`Particle` 和 `Event` 类，使之能够处理多个粒子的相互碰撞。在模拟台球比赛的开球时这是非常重要的。（这道习题很难！）
- 6.3 开发一个三维版本的 `CollisionSystem`、`Particle` 和 `Event` 类。
- 6.4 尝试将大片区域分割为长方形的小格，并在一种新的事件类型中仅预测某个粒子在某一时刻和相邻的 9 个方格中的所有粒子的碰撞。用这种方法改进 `CollisionSystem` 的 `simulate()` 方法的性能。这种方法减少了需要计算的预测碰撞数量，代价是需要监视所有粒子在方格之间的运动。
- 6.5 在 `CollisionSystem` 中引入熵的概念并用它验证（信息论中的）经典结论。
- 6.6 布朗运动。1827 年，植物学家罗伯特·布朗在用显微镜观察到浸入水中的野生花粉颗粒时，发现它们在进行无规则的运动。这种运动后来被称为布朗运动。人们讨论了这种现象，但没人能够给出令人信服的解释，直到爱因斯坦在 1905 年在数学上说明了这个问题。爱因斯坦的解释是：花粉颗粒的运动是由无数微小的分子和花粉粒子相撞造成的。请用模拟来说明这个现象。
- 6.7 温度。为 `Particle` 类添加一个 `temperature()` 方法，返回粒子的质量和速度的平方除以 dk_B 的商之积，其中空间维数 $d=2$ ，Boltzmann 常数 $k_B=1.3806503 \times 10^{-23}$ 。系统的温度是所有粒子的这些量



的平均值。为 `CollisionSystem` 添加一个 `temperature()` 方法，周期性采集温度数据并绘成图表，检查温度是否恒定。

922

6.8 Maxwell-Boltzmann. 刚性球体模型中的所有粒子的速度分布遵循 Maxwell-Boltzmann 分布（假设系统已经被加热且粒子的质量足以忽略量子力学效应），在二维系统中又被称为 Rayleigh 分布。分布的形状取决于温度。编写一个方法计算粒子速度的直方图并在各种温度下测试它。

6.9 任意形状。分子的移动速度非常快（超过喷气式飞机）但扩散却很慢，因为它们会互相碰撞并因此改变方向。扩展模型，将两个容器用一根管道相连，容器中分别含有两种不同类型的粒子。模拟粒子的运动并以时间的函数测量每个容器中每种类型的粒子的比例。

6.10 回退。在某次模拟结束后，将所有速度变为相反的方向并继续模拟系统中的运动，它应该能够回到最初的状态！测量系统的最终状态和初始状态的差异来估计四舍五入造成的误差。

6.11 压强。为 `Particle` 类添加一个 `pressure()` 方法来测量大量粒子和墙体碰撞造成的压强。系统的压强为所有粒子的冲击力之和。为 `CollisionSystem` 类添加一个 `pressure()` 方法并编写一个用例验证等式 $pv=nRT$ 。

6.12 基于索引优先队列的实现。开发一个版本的 `CollisionSystem`，使用索引优先队列来保证优先队列的长度最多与粒子数量呈线性关系（而非平方级别或者更糟）。

6.13 优先队列的性能。使用优先队列，在多种温度下测试 `Pressure` 类来定位计算的瓶颈。如果可以，尝试切换到另一种不同的优先队列实现，在高温下获取更好的性能。

923

练习：B-树

6.14 假设在一棵三层树中，总共可以在内存中保存 a 条链接。每个页中可以保存 $b \sim 2b$ 条指向内部结点的链接和 $c \sim 2c$ 条指向外部结点中的链接。在这样一棵树中最多可以含有多少个项（作为 a 、 b 、 c 的函数）？

6.15 开发一个 `Page` 的实现，将 B-树的结点表示为一个 `BinarySearchST` 类的对象。

6.16 扩展 `BTreeSET` 来实现能够关联键和值的 `BTreeST` 类，并完整支持有序符号表 API，包括 `min()`、`max()`、`floor()`、`ceiling()`、`deleteMin()`、`deleteMax()`、`select()`、`rank()` 方法以及接受两个参数的 `size()` 和 `get()` 方法。

6.17 编写一个程序，使用 `StdDraw` 将 B-树的生长过程可视化，如同正文描述的方式一样。

6.18 在一个有缓存的典型系统中，估计对 B-树的 S 次随机查找中，每次查找的平均探查次数。缓存可以将 T 个最近访问的页保存在内存中（因此无需探查）。假设 S 远大于 T 。

6.19 网络搜索。开发一个 `Page` 类的实现，为了索引网页，用 B-树的结点表示网页中的文本。用一个文件表示搜索的关键词。从标准输入接受被索引的网页。为了控制规模，接受命令行参数 m 并将内部结点的数量限制在 10^m 内。（在使用较大的 m 前请联系系统管理员。）使用一个 m 位的数字来表示内部结点。例如，当 m 为 4 时，结点名可以是 `BTreeNode0000`、`BTreeNode0001`、`BTreeNode0002` 等。在页中保存成对的字符串。向 API 中添加一个 `close()` 操作来排序并写入数据。为了测试实现，尝试在你的学校的网站上搜索你和朋友的名字。

6.20 B*-树。在 B-树中启发式地分裂兄弟结点：当某个结点含有 M 个条目并需要分裂时，将它和它的一个兄弟结点合并。如果该兄弟结点只含有 k 个条目且 $k < M-1$ ，可以重新分配并使得两者都只含有 $(M+k)/2$ 个条目。否则，我们创建一个新结点并使 3 个结点中都只含有 $2M/3$ 个条目。同时，我们允许根结点保存 $4M/3$ 个条目，并在它饱和时将它分裂并创建一个只含有两个条目的新根结

点。找出在含有 N 个元素的 M 阶 B*- 树中每次查找或插入所需的探查数的上下界限。将你的结果和 B- 树的相应上下界（请见命题 B）进行比较。实现 B* 树中的插入操作。

924

- 6.21 编写一段程序，计算在 N 次随机插入所构造的一棵 M 阶 B- 树中外部的平均数量。用合理的 M 和 N 值运行你的程序。
- 6.22 如果你的系统支持虚拟内存，设计并用实验比较 B- 树和二分查找在一张庞大的符号表中的随机查找性能。
- 6.23 对于你为练习 6.15 给出的保存在内存中的 Page 的实现，用实验确定能够使 B- 树在一张庞大的符号表中的使随机查找操作速度最快的 M 值。特别注意 M 为 100 的倍数的情况。
- 6.24 运行实验比较保存在内存中的 B- 树（使用练习 6.23 中确定的 M 值）、线性探测散列法和红黑树在一张庞大的符号表中的随机查找用时。

925

练习：后缀数组

- 6.25 按照图 6.0.15 的样式给出由以下字符串的后缀、后缀的排序、`index()` 和 `lcp()` 方法的返回值组成的表格。

a. abacadaba
b. mississippi
c. abcdefghij
d. aaaaaaaaaa

- 6.26 下面这段代码用于计算字符串的所有后缀，找出其中的问题。

```
suffix = "";
for (int i = s.length() - 1; i >= 0; i--)
{
    suffix = s.charAt(i) + suffix;
    suffixes[i] = suffix;
}
```

答：它需要平方级别的时间和空间。

- 6.27 有些应用需要对文本进行回环移位，这个操作会涉及文本中的所有字符。对于 0 到 $N-1$ 之间的 i ，长度为 N 的文本的第 i 次回环移位得到的是它的后 $N-i$ 个字符和前 i 个字符相连所得的字符串。下面这段代码用于计算文本的所有回环移位，找出其中的问题。

```
int N = s.length();
for (int i = 0; i < N; i++)
    rotation[i] = s.substring(i, N) + s.substring(0, i);
```

答：它需要平方级别的时间和空间。

- 6.28 设计一个线性时间的算法来计算给定文本字符串的所有回环移位。

答：

```
String t = s + s;
int N = s.length();
for (int i = 0; i < N; i++)
    rotation[i] = r.substring(i, i + N);
```

926

- 6.29 按照 1.4 节中的假设，给出一个长度为 N 的字符串 SuffixArray 对象对内存的使用情况。

- 6.30 最长公共子字符串。编写一个 SuffixArray 的用例 LCS，接受两个文件名作为命令行参数，读取这两个文本文件并在线性时间内找出同时出现在两个文件中的最长子字符串。（在 1970 年，D.Knuth

猜测这是不可能的。)提示:为字符串 $s\#t$ 创建后缀数组,其中 s 和 t 是文本字符串,而 $\#$ 是一个两者都不包含的字符)。

- 6.31 Burrow-Wheeler 变换。Burrow-Wheeler 变换 (BWT) 是一种用于数据压缩算法中的变换,包括 bzip2 和高吞吐量的基因组测序等。编写一个 SuffixArray 的用例用以下方法在线性时间内计算 BWT。给定一个长度为 N 的字符串 (以一个文件结束符 $\$$ 结尾,它小于其他任意字符)。使用一个 $N \times N$ 的矩阵,其中每一行均为原文的一个不同的循环变位。按照字典顺序将所有行排序。Burrow-Wheeler 变换就是排序后的矩阵中最右侧的列。例如,mississippi 的 BWT 是 ipssm\$piisii。Burrow-Wheeler 逆变换是 BWT 的逆序。例如,ipssm\$piisii 的 BWI 是 mississippi\$。编写一个用例,在线性时间内,为某个字符串的 BWT 计算它的 BWI。
- 6.32 环形字符串的线性化。编写一个 SuffixArray 的用例,对于给定的字符串,在线性时间内找出它的字典序列最小的循环变位。这个问题来源于化学数据库中的各种环形分子,每一种分子都表示为一个环形的字符串。人们需要一种标准的表示方法 (最小的循环变位) 使得用字符串的任意循环变位作为键都能找到该分子。(请见练习 6.27 和练习 6.28。)
- 6.33 重复 k 次的最长子字符串。编写一个 SuffixArray 的用例,对于给定的字符串和一个整数 k ,找出其中被至少重复了 k 次的最长子字符串。
- 6.34 较长的重复字符串。编写一个 SuffixArray 的用例,对于给定的字符串和一个整数 L ,找出长度至少为 L 的重复子字符串。
- 6.35 k -gram 频率统计。开发并实现一个抽象数据类型,对字符串进行预处理以支持高效回答如下形式的问题:“给定的 k -gram 出现了多少次?”每次查询在最坏情况下所需的时间应该与 $k \log N$ 成正比,其中 N 为字符串的长度。

927

练习:最大流问题

- 6.36 在含有 V 个顶点和 E 条边的任意 st -流量网络中,如果所有边的容量都是小于 M 的正整数,可能的最大流量值是多少?为存在和不存在平行边的情况分别给出答案。
- 6.37 如果原流量网络在删去终点时将变成一棵树,给出一个算法解决这种流量网络中的最大流量问题。
- 6.38 真假判断。如果为真,给出简短的证明;如果为假,给出一个反例。
- 在任意最大流配置中均不存在所有边的正流均为正的有向环。
 - 存在一种不包含所有边的流量均为正的有向环的最大流配置。
 - 如果所有边的容量均不同,那么最大流量配置是唯一的。
 - 如果所有边的容量是一个等差数列,那么最小切分是唯一的 (remains unchanged)。
 - 如果所有边的容量是一个等比数列,那么最小切分是唯一的。
- 6.39 完成命题 G 的证明。说明为何每当一条边成为关键边时,经过它的增广路径的长度必然会加 2。
- 6.40 在互联网上找出一个大型网络,使用真实数据测试最大流算法。你可以选择交通运输网络 (公路、铁路或者航空)、通信网络 (电话或者计算机网络) 或者物流配送网络。如果边的容量不明,根据一个合理的模型自己添加这些数据。编写一个程序使用我们学过的接口根据你的数据实现流量网络的配置。如有需要,编写一个私有方法清理数据。
- 6.41 编写一个随机网络生成器来生成稀疏网络,其中边的容量为 0 到 2^{30} 之间的整数。用一个单独的类表示容量并开发两种实现:一种生成均匀分布的容量值,一种根据高斯分布生成容量值。实现一个用例,对于一组精心选择的 V 和 E 值用两种分布方法生成随机网络,这样你就可以使用它们进行各种测试了。

928

- 6.42 编写一个程序，在平面上随机生成 V 个点。构造流量网络时，对于每个点都将它和距离 d 以内的所有点相互连接，用练习 6.42 中的随机模型设置每条边的容量。
- 6.43 简单的归约。编写 FordFulkerson 的用例，在以下类型的流量网络中寻找最大流配置。
- 管道没有方向。
 - 起点和终点的数量不限，也不限制指向起点或是由终点指出的边的数量。
 - 容量有下限。
 - 顶点有流量限制。
- 6.44 产品分发。假设流量表示城市之间用卡车运送的产品，边 $u-v$ 上的流量表示某一天从 u 市运送到 v 市的产品数量。编写一个用例，为卡车司机打印出每天的订单，告诉他们应该去哪个城市上多少货，然后去哪个城市卸多少货。假设卡车司机的数量无限多且对于任意一个分发点，所有货物全部收到了之后才会开始发货。
- 6.45 就业安置。开发一个 FordFulkerson 的用例，根据命题 J 中的归约解决就业安置问题。使用一张符号表将名字变为数字并用于流量网络中。
- 6.46 构造一系列的二分图匹配问题，其中任意增广路径算法解决对应的最大流问题所使用的所有增广路径的平均长度与 E 成正比。
- 6.47 st -连通性。开发一个 FordFulkerson 的用例，对于给定的无向图 G 和顶点 s 和 t ，找出在 G 中使 t 和 s 不连通所需切断的最小边数。
- 6.48 不同的路径。开发一个 FordFulkerson 的用例，对于给定的无向图 G 和顶点 s 和 t ，找出从 s 到 t 最多有多少条任意边均不相同的路径。

929

练习：问题的归约与不可解性

- 6.49 找到 37 703 491 的最大因数。
- 6.50 证明最短路径问题可以归约为线性规划问题。
- 6.51 如果 $P \neq NP$ ，是否存在能够在 N^{poly} 时间内解决某个 NP-完全问题的算法？解释你的回答。
- 6.52 假设某人发明了一种保证能够在与 1.1^N 成正比的时间内解决布尔可满足性问题的算法。这说明我们能够在与 1.1^N 成正比的时间内解决其他 NP-完全问题吗？
- 6.53 一个能够在与 1.1^N 成正比的时间内解决整数线性规划问题的程序的意义是什么？
- 6.54 给出一个从顶点覆盖问题向 0-1 整数线性不等式可满足性问题的多项式时间的归约。
- 6.55 使用无向图中的哈密尔顿路径问题的 NP-完全性证明在有向图中寻找哈密尔顿路径的问题也是 NP-完全的。
- 6.56 假设两个问题都已知是 NP-完全的，这说明能够在多项式时间内将两者相互归约吗？
- 6.57 假设问题 X 是 NP-完全的， X 能够在多项式时间内归约为问题 Y ，而且 Y 也能在多项式时间内归约为 X ，那么 Y 一定是 NP-完全的吗？
- 答：不，因为 Y 不一定属于 NP。
- 6.58 假设我们有一个能够解决布尔可满足性问题的确定性版本的算法，这说明存在某种变量赋值能够满足所有的布尔表达式。说明如何找到这种赋值方案。
- 6.59 假设我们有一个能够解决顶点覆盖问题的确定性版本的算法，这说明对于某个给定的大小存在顶点覆盖的方案。说明如何解决最小顶点覆盖问题的最优化版本。

930

6.60 解释为何顶点覆盖问题的最优化版本不一定是一个搜索问题。

答：因为并没有很好的方法来验证给定的解是否是最优的。（尽管我们可以用二分查找在这个问题的搜索版本上找到最优解。）

6.61 假设问题 X 和问题 Y 均为搜索问题，且 X 能够在多项式时间内归约为 Y 。我们可以得到以下哪些结论。

- 如果 Y 是 NP-完全的，那么 X 也是。
- 如果 X 是 NP-完全的，那么 Y 也是。
- 如果 X 属于 P，那么 Y 也属于 P。
- 如果 Y 属于 P，那么 X 也属于 P。

6.62 假设 $P \neq NP$ ，我们可以得到以下哪些结论。

- 如果问题 X 是 NP-完全的，那么 X 无法在多项式时间内得到解决。
- 如果问题 X 属于 NP，那么 X 无法在多项式时间内得到解决。
- 如果问题 X 属于 NP 但并不是 NP-完全的，那么 X 可以在多项式时间内得到解决。
- 如果问题 X 属于 P，那么 X 就不是 NP-完全的。

931



索引

(索引中的页码为英文原书的页码, 与书中边栏的页码一致。)

- 2-3-4 search tree (2-3-4 查找树), 441, 451
 - 2-3 search tree (2-3 查找树), 424-431
 - 2-nodes and 3-nodes (2-节点和 3-节点), 424
 - analysis of (2-3 查找树分析), 429
 - height (高度), 429
 - insertion (插入), 425-427
 - order (有序), 424
 - perfect balance (完美平衡), 424
 - and red-black BST (红黑二叉查找树), 432
 - search (查找), 425
 - 2-3 tree. See 2-3 search tree (2-3 树, 见 2-3 查找树)
 - 2-colorability problem (双色问题), 546
 - 2-dimensional array (二维数组), 19
 - 2-satisfiability problem (2-可满足性), 599
 - 2-sum problem (2-sum 问题), 189
 - 3-collinear problem (三点共线问题), 211
 - 3-sum problem (3-sum 问题), 173, 190
 - 3-way partitioning (三向切分), 298
 - 3-way quicksort (三向快速排序), 298-301
 - 3-way string quicksort (三向字符串快速排序), 719-723
 - 8-puzzle problem (8 字谜题), 358
 - 32-bit architecture (32 位架构), 13, 201, 212
 - 64-bit architecture (64 位架构), 13, 201
- A**
- A* algorithm (A* 算法), 350
 - Abstract data type (抽象数据类型), 64
 - API, 65
 - client (用例), 88-89
 - design (数据类型的设计), 96-97
 - implementing an (实现一种数据类型), 84-87
 - multiple implementations (实现多种数据类型), 90
 - Abstract in-place merge (抽象原地递归), 270
 - Accumulator data type (累加器数据类型), 92-93
 - Actual type (实际类型), 134, 328
 - Acyclic digraph. See Directed acyclic graph (无环有向图, 见 Directed acyclic graph)
 - Acyclic edge-weighted digraph. (无环加权有向图) See Edge-weighted DAG (见 Edge-weighted DAG)
 - Acyclic graph (无环图), 520, 547, 576
 - Adjacency list (邻接表)
 - directed graph (有向图), 568-569
 - edge-weighted digraph (加权有向图), 644
 - edge-weighted graph (加权无向图), 609
 - undirected graph (无向图), 524-525
 - Adjacency matrix (邻接矩阵), 524, 527
 - Adjacency set (邻接集), 527
 - Adjacent vertex (邻接顶点), 519
 - ADT. See Abstract data type (ADT, 见 Abstract data type)
 - Algorithm (算法)
 - century of (算法的世纪), 853
 - defined (定义算法), 4
 - deterministic (可判定的), 4
 - nondeterministic (非确定性的), 914
 - randomized (随机的), 198
 - Aliasing
 - of arrays (别名), 19
 - of objects (对象), 69
 - of substrings (子数组), 202
 - All-pairs reachability (顶点对的可达性), 590
 - All-pairs shortest paths (任意顶点对之间的最短路径), 656
 - Alphabet data type (字母表类型), 698-700
 - Amortized analysis (均摊分析)
 - binary heap (二叉堆), 320
 - defined (均摊分析的定义), 198-199
 - hash table (散列表), 475
 - resizing array (可调整大小的数组), 199
 - union-find (union-find 算法), 231, 237
 - weighted quick-union with path compression (路径压缩的加权 quick-union 算法), 231
 - Analysis of algorithms (算法分析), 172-215
 - See also Propositions (另见命题);
 - See also Properties (另见性质);
 - amortized analysis (均摊分析), 198-199
 - big-Oh notation (大 O 记法), 206-207
 - cost model (成本模型), 182
 - divide-and-conquer (分治思想), 272
 - doubling ratio (倍率), 192-193
 - doubling test (双倍测试), 176-177
 - input models (输入模型), 197
 - log-log plot (对数图像), 176
 - mathematical models (数学模型), 178
 - memory usage (内存使用), 200-204

multiple parameters (多个参量), 196
 observations (观察), 173–175
 order-of-growth (增长的数量级), 179
 order-of-growth classifications (增长数量级的分类), 186–188
 order-of-growth hypothesis (增长数量级的猜想), 180
 problem size (问题规模), 173
 randomized algorithm (随机化算法), 198
 scientific method (科学方法), 172
 tilde approximation (近似), 178
 worst-case guarantee (对最坏情况下的性能保证), 197

Antisymmetric relation (反对称性), 247

APIs

Accumulator, 93
 Alphabet (字母表), 698
 Bag, 121
 BinaryStdIn, 812
 BinaryStdOut, 812
 Buffer, 170
 CC, 543
 Counter, 65
 Date, 79
 Degrees, 596
 Deque, 167
 Digraph, 568
 DirectedCycle, 576
 DirectedDFS, 570
 DirectedEdge, 641
 Draw, 83
 Edge, 608
 EdgeWeightedDigraph, 641
 EdgeWeightedGraph, 608
 FixedCapacityStack, 135
 FixedCapacityStackOfStrings, 133
 FlowEdge, 890
 FlowNetwork, 890
 GeneralizedQueue, 169
 Graph, 522
 GraphProperties, 559
 In, 41, 83
 IndexMaxPQ, 320
 IndexMinPQ, 320
 Interval1D, 77
 Interval2D, 77
 java.lang.Double, 34
 java.lang.Integer, 34
 java.lang.Math, 28
 java.lang.String, 80
 java.util.Arrays, 29

KMP, 769
 List, 511
 MathSET, 509
 Matrix, 60
 MaxPQ, 309
 MinPQ, 309
 MST, 613
 Out, 41, 83
 Page, 870
 Particle, 860
 Paths, 535
 Point2D, 77
 Queue, 121
 RandomBag, 167
 RandomQueue, 168
 Rational, 117
 SCC, 586
 Search, 528
 SET, 489
 SP, 644, 677
 ST, 363, 366, 860, 870, 879
 Stack, 121
 StaticSETofInts, 99
 StdDraw, 43
 StdIn, 39
 StdOut, 37
 StdRandom, 30
 StdStats, 30
 Stopwatch, 175
 StringSET, 754
 StringST, 730
 SuffixArray, 879
 SymbolDigraph, 581
 SymbolGraph, 548
 Topological, 578
 Transaction, 79
 TransitiveClosure, 592
 UF, 219
 VisualAccumulator, 95

Application programming interface. See also APIs (应用程序接口, 见 API)
 client (用例), 28
 contract (契约), 33
 data type definition (数据类型定义), 65
 implementation (实现), 28
 library of static methods (静态方法库), 28
 Arbitrage detection (套汇检测), 679–681
 Arithmetic expression evaluation (算术表达式求值), 128–131

Array (数组), 18–21
 2-dimensional (二维数组), 19
 aliasing (别名), 19
 as object (数组对象), 72
 bounds checking (边界检查), 19
 memory usage of (内存使用), 202
 of objects (数组对象), 72
 ragged (参差不齐), 19
 Array resizing. See Resizing array (调整数组大小, 见调整大小的数组)
 Arrays.sort(), 29, 306
 Articulation point (关节点), 562
 ASCII encoding (ASCII 编码), 696, 815
 Assertion (断言), 107
 assert statement (断言语句), 107
 Assignment statement (赋值语句), 14
 Associative array (关联数组), 363
 Augmenting path (增广路径), 891
 Autoboxing (自动装箱), 122, 214
 AVL tree (AVL 树), 452

B

Backtracking (回溯法), 921
 Bag data type (背包数据类型), 124, 154–156
 Balanced search tree (平衡查找树), 424–457
 2-3 search tree (2-3 查找树), 424–431
 AVL tree (AVL 树), 452
 B-tree (B-树), 866–874
 red-black BST (红黑二叉查找树), 432–447
 Base case (最简单的情况), 25
 Bellman-Ford (Bellman-Ford 算法), 671–678
 Bellman, R., 683
 Bentley, J., 298, 306
 BFS. See Breadth-first search (BFS, 见 Breadth-first search)
 Biconnectivity (双向连通性), 562
 Big-Oh notation (大 O 记法), 206–207
 Big-Omega notation (大 Omega 记法), 207
 Big-Theta notation (大 Theta 记法), 207
 Binary data (二进制数据), 811–815
 Binary dump (二进制转储), 813–814
 Binary heap (二叉堆), 313–322
 amortized analysis of (二叉堆的均摊分析), 320
 analysis of (分析二叉堆), 319
 change priority (改变优先级), 321
 definition (二叉堆的定义), 314
 deletion (删除), 321
 heapsort (堆排序), 323–327

insertion (插入元素), 317
 remove the maximum (删除最大元素), 317
 remove the minimum (删除最小元素), 321
 representation (二叉堆表示法), 313
 sink and swim (下沉和上浮), 315–316
 Binary logarithm function (以 2 为底的对数函数), 185
 Binary search (二分查找), 8
 analysis of (二分查找的分析), 383, 391
 bitonic search (双调查找), 210
 for a fraction (分数的二分查找), 211
 in a sorted array (在有序数组中进行二分查找), 46–47, 98–99
 local minimum (数组的局部最小元素), 210
 symbol table (数组符号表), 378–384
 Binary search tree (二叉查找树), 396–423
 analysis of (二叉查找树的分析), 403
 anatomy of (详解二叉查找树), 396
 AVL tree (AVL 树), 452
 certification (认证), 419
 definition (二叉查找树的定义), 396
 delete the min/max (删除最大键和删除最小键), 408
 floor and ceiling (向上取整和向下取整函数), 406
 height (树的高度), 412
 Hibbard deletion (Hibbard 删除方法), 410, 422
 insertion (插入), 400–401
 minimum and maximum (最大键和最小键), 406
 nonrecursive (非递归), 417
 perfectly balanced (完美平衡的), 403
 range query (范围查找), 412
 rank and select (排名和选择), 415
 recursion (递归), 415
 representation (数据表示), 397
 rotation (旋转), 433–434
 search (查找), 397–401
 selection and rank (选择和排名), 406, 408
 symmetric order (二叉树的对称性), 410
 threading (线性符号表), 420
 BinaryStdIn library (BinaryStdIn 库), 811–815
 BinaryStdOut library (BinaryStdOut 库), 811–815
 Binary tree (二叉树)
 anatomy of (详解二叉树), 396
 binary heap (二叉堆), 313
 complete (实现), 313, 314
 external path length (外部路径总长度), 418
 heap-ordered (堆有序), 313
 height (完全二叉树的高度), 314
 inorder traversal (中序遍历), 412
 internal path length (内部路径长度), 412
 level-order traversal (按层遍历), 420

preorder traversal (前序遍历), 834
 weighted external path length (加权外部路径长度), 832
 Binomial coefficient (二项式系数), 185
 Binomial distribution (二项分布), 59, 466
 Binomial tree (二项树), 237
 Bipartite graph (二分图), 521, 546–547
 Birthday problem (生日问题), 215
 Bitmap (位图), 822
 Bitonic array (双调数组), 210
 Bitonic search (双调查找), 210
 Bitonic shortest paths (双调最短路径), 689
 Blacklist filter (黑名单过滤), 491
 Boerner's theorem (Boerner 定理), 357
 boolean primitive data type (布尔型原始数据类型), 12
 Boolean satisfiability (布尔可满足性), 913, 920
 Boruvka, O., 628
 Boruvka's algorithm (Boruvka 算法), 629, 636
 Bottleneck shortest paths (瓶颈最短路径), 690
 Bottom-up 2-3-4 tree (自底向上的 2-3-4 树), 451
 Bottom-up mergesort (自底向上的合并排序), 277
 Boyer-Moore (Boyer-Moore 算法), 770–773
 Boyer, R. S., 759
 Breadth-first search (广度优先搜索)
 in a digraph (在有向图中), 573
 in a graph (在图中), 538–542
 break statement (break 语句), 15
 Bridge in a graph (连通图中的桥), 562
 B-tree (B-树), 448, 866–874
 analysis of (B-树分析), 871
 insertion (插入), 868
 perfect balance (完美平衡), 868
 search (搜索), 868
 Buffer data type (缓冲区数据类型), 170
 Byte (8 bits) (字节(1 字节等于 8 比特)), 200
 byte primitive data type (byte 原书数据类型), 13

C

Cache (缓存), 195, 307, 327, 343, 394, 419, 423
 Call a method (调用方法), 22
 Callback (回调), 339. See also Interface (另见接口)
 Cast (类型转换), 13, 328, 346
 Catenable queue (可连接的队列), 171
 Ceiling function (向上取整函数)
 binary search tree (二叉查找树), 406
 mathematical function (数学函数), 185
 ordered array (有序数组), 380
 symbol table (符号表), 367

Cell-probe model (cell-probe 模型), 234
 Center of a graph (图的中点), 559
 Certification (认证)
 binary heap (二叉堆), 330
 binary search (二分查找), 392
 binary search tree (二叉查找树), 419
 minimum spanning tree (最小生成树), 634
 NP complexity class (NP 复杂性类), 912
 red-black BST (红黑二叉查找树), 452
 search problem (搜索问题), 912
 shortest paths (最短路径), 651
 sorting (排序), 246, 265
 char primitive data type (char 原始数据类型), 12, 696
 Chazelle, B., 629, 853
 Chebyshev's inequality (Chebyshev 不等式), 303
 Church-Turing thesis (丘奇-图灵论题), 910
 Circular linked list (环形链表), 165
 Circular queue (环形队列), 169
 Circular rotation (回环变位), 114
 Classpath (类路径), 66
 Client (用例), 28
 Closest pair (最接近的一对), 210
 Collections (集合), 120
 bag (背包), 124–125
 catenable (可连接的), 171
 deque (出列), 167
 generalized queue (一般队列), 169
 priority queue (优先队列), 308–334
 pushdown stack (下压栈), 127
 queue (队列), 126
 random bag (随机背包), 167
 random queue (随机队列), 168
 ring buffer (环形缓冲区), 169
 stack (栈), 127
 steque, 167
 symbol table (符号表), 360–513
 trie (单词查找树), 730–757
 Collision resolution (处理碰撞冲突), 458
 Combinatorial search (组合搜索), 350
 Command-line argument (命令行参数), 36
 Command-line interface (命令行接口)
 command-line argument (命令行参数), 36
 compile a Java program (编译 Java 程序), 10
 piping (管道), 40
 redirection (重定向), 40
 run a Java program (运行 Java 程序), 10
 standard input (标准输入), 39
 standard output (标准输出), 37–38
 terminal window (终端窗口), 36

- Comma-separated-value (逗号分隔的值), 493
- Comparable interface (Comparable 接口)
 - compareTo() method (compareTo() 方法), 246–247
 - Date, 247
 - natural order (自然排序), 337
 - sorting (排序), 244, 246–247
 - String, 353
 - symbol table (符号表), 368–369
 - Transaction, 266
- Comparator interface (Comparator 接口), 338–340
 - compare() method, 338–339
 - priority queue (优先队列), 340
 - Transaction, 339
- compare() method (compare() 方法)
 - See Comparator interface (见 Comparator 接口)
- compareTo() method (compareTo() 方法)
 - See Comparable interface (见 Comparable 接口)
- Compile a program (编译程序), 10
- Compiler (编译器), 492, 498
- Complete binary tree (完全二叉树), 314
- Complete graph (完全有向图), 681
- Compression. See Data compression (压缩, 见数据压缩)
- Computability (可计算性), 910
- Computational complexity (计算复杂性)
 - Cook-Levin theorem (Cook-Levin 定理), 918
 - Intractability (不可解性), 910–921
 - NP-complete (NP-完全), 917–918
 - NP, 912
 - P, 914
 - P=NP question, 916
 - poly-time reduction (多项式时间问题的相互归约), 916–917
 - sorting (排序), 279–282
- Computational geometry (计算几何学), 76
- Concatenation of strings (字符串连接), 34
- Concordance (对照索引), 510
- Concrete type (具体类型), 122, 134
- Conditional statement (条件语句), 15
- Connected components (连通分量)
 - Computing (计算), 543–546
 - Defined (定义), 519
 - union-find (union-find 算法), 217
- Connected graph (连通图), 519
- Connectivity (连通性)
 - articulation point (关节点), 562
 - biconnectivity (双向连通性), 562
 - bridge (桥), 562
 - components (分量), 543–546
 - dynamic (动态的), 216
 - edge-connected graph (边连通图), 562
 - strong connectivity (强连通性), 584–591
 - undirected graph (无向图), 530
 - union-find (union-find 算法), 216–241
- Constant running time (常数级别的运行时间), 186
- Constructor (构造函数), 65, 84–85
- continue statement (continue 语句), 15
- Contract (契约), 33
- Cook-Levin theorem (Cook-Levin 定理), 918
- Cook, S., 759, 918
- Cost model (成本模型), 182
 - array accesses (数组的访问次数), 182, 220, 369
 - binary search (二分查找), 184
 - B-tree (B-树), 866
 - compares (比较), 369
 - equality tests (等价性测试), 369
 - searching (查找), 369
 - sorting (排序), 246
 - symbol table (符号表), 369
 - 3-sum (3-sum 问题), 182
 - union-find (union-find 算法), 220
- Coupon collector problem (优惠券收集问题), 215
- Covariant arrays (共变数组), 158
- CPM. See Critical-path method (CPM, 见最短路径算法)
- C language (C 语言), 104
- C++ language (C++ 语言), 104
- Critical edge (关键边), 633, 690, 900
- Critical path (关键路径), 663
- Critical-path method (关键路径法), 663, 664
- Crossing edge (横切边), 606
- Cubic running time (立方级别的运行时间), 186
- Cuckoo hashing (Cuckoo 散列函数), 484
- Cut (切分), 606
 - See also Mincut problem (另见 Mincut problem)
 - capacity of (容量), 892
 - optimality conditions (最优条件), 634
 - property for MST (最小生成树定理), 606
 - st-cut (st-切分), 892
- Cycle (环)
 - Eulerian (欧拉), 562, 598
 - Hamiltonian (哈密尔顿), 562
 - in a digraph (在有向图中), 567
 - in a graph (在图中), 519
 - odd length (奇数长度), 562
 - simple (简单), 519, 567
- Cycle detection (检测环), 546–547
- Cyclic rotation of a string (字符串的循环移位), 784

D

DAG. See Directed acyclic graph (DAG, 见 Directed acyclic graph)

Dangling else (无主的 else), 52

Dantzig, G., 909

Data abstraction (数据抽象), 64–119

Data compression (数据压缩), 810–851

fixed-length code (定长编码), 819–821

Huffman (霍夫曼), 826–838

lossless (无损), 811

lossy (有损), 811

LZW algorithm (LZW 压缩算法), 839–845

prefix-free code (前缀码), 826–827

run-length encoding (游程编码), 822–825

2-bit genomics code (双位基因编码), 819–821

undecidability (不可判定性), 817

uniquely decodable code (唯一解码), 826

universal (通用), 816

variable-length code (变长码), 826

Data structure (数据结构)

adjacency lists (邻接表), 525

adjacency matrix (邻接矩阵), 524

binary heap (二叉堆), 313

binary search tree (二叉查找树), 396

binary tree (二叉树), 396

circular linked list (环形链表), 165

doubly-linked list (双向链表), 146

linked list (链表), 142–146

multiway trie (多路单词查找树), 732

ordered array (有序数组), 312

ordered list (有序列表), 312

parallel arrays (平行的数组), 378

parent-link (父链接), 225

resizing array (调整数组的大小), 136

ternary search trie (三向单词查找树), 746

unordered array (无序数组), 310

unordered list (无序列表), 312

Data type (数据类型)

abstract (抽象数据类型), 64

design of (抽象数据类型的设计), 96–97

encapsulation (抽象数据类型的封装), 96

Date data type (日期数据类型), 78–79

compareTo() method (compareTo() 方法), 247

equals() method (equals() 方法), 103

implementation (实现), 91

toString() method (toString() 方法), 103

Decision problem (决定性问题), 913

Declaration statement (声明语句), 14

Dedup (dedup 过滤器), 490

Default initialization (默认初始化), 18, 86

Defensive copy (保护性复制), 112

Degree of a vertex (顶点度数), 519

Degrees of separation (间隔的度数), 553–554

Denial-of-service attacks (拒绝服务攻击), 197

Dense graph (稠密图), 520

Deprecated method (弃用的方法), 113

Depth-first search (深度优先搜索), 530–534

bipartiteness (二分图), 547

connected components (连通分量), 543

cycle detection (检测环), 547

directed cycle (有向环), 574–581

longest path (最长路径), 912

maze exploration (探索迷宫), 530

path finding (寻找路径), 535–537

reachability (可达性), 570–573

strong components (强连通分量), 584–591

topological order (拓扑排序), 574–581

transitive closure (传递闭包), 592

Tremaux exploration (Tremaux 搜索), 530

2-colorability (双色), 547

union-find (union-find 算法), 546

Depth of a node (节点的深度), 226

Deque data type (双向队列数据类型), 167, 212

Design by contract (契约式设计), 107

Deterministic finitestate automaton (有限状态自动机), 764

Devroye, L., 412

DFA. See Deterministic finite state automaton (DFA, 见 Deterministic finite-state automaton)

Diameter of a graph (图的直径), 559, 685

Dictionary (字典), 361. See also Symbol table (另见 Symbol table)

Digraph. See Directed graph (有向图, 见 Directed graph)

Digraph data type (有向图的数据类型), 568–569

Dijkstra, E. W., 128, 298, 628, 682

Dijkstra's 2-stack algorithm (Dijkstra 双栈算法), 128–131

Dijkstra's algorithm (Dijkstra 算法), 652–657

bidirectional search (双向搜索), 690

negative weights (负权重), 668

Directed acyclic graph (有向无环图), 574–583

depth-first orders (深度优先次序), 578

edge-weighted (加权), 658–667

Hamiltonian path (哈密顿路径), 598

lowest common ancestor (最近共同祖先), 598

shortest ancestral path (最短先导路径), 598

- topological order (拓扑排序), 575
- topological sort (拓扑排序), 575
- Directed cycle (有向环), 567
- Directed cycle detection (有向环检测) 576
- Directed edge (有向边), 566
- Directed graph (有向图), 566–603
 - See also Edge-weighted digraph (另见 Edge-weighted digraph)
 - acyclic (无环), 574–583
 - adjacency-lists representation (邻接表表示), 568, 568–569
 - all-pairs reachability (顶点对的可达性), 590
 - anatomy of (详解), 567
 - breadth-first search (广度优先搜索), 573
 - cycle (环), 567
 - cycle detection (检测环), 576
 - defined (定义), 566
 - directed paths (有向路径), 573
 - edge (边), 566
 - Euler cycle (Euler 环), 598
 - indegree and outdegree (入度和出度), 566
 - Kosaraju's algorithm (Kosaraju 算法), 586–590
 - path (路径), 567
 - postorder traversal (后序遍历), 578
 - preorder traversal (前序遍历), 578
 - reachability (可达性), 570–572
 - reachable vertex (可达顶点), 567
 - reverse (取反), 568
 - reverse postorder (逆后序), 578
 - shortest ancestral path (最短先导路径), 598
 - shortest directed paths (最短有向路径), 573
 - simple (简单), 567
 - strong component (强连通分量), 584
 - strong connectivity (强连通性), 584–591
 - strongly-connected (强连通), 584
 - topological order (拓扑排序), 575–583
 - transitive closure (传递闭包), 592
- Directed path (有向路径), 567
- Disjoint set union. See Union find (不相交集合并, 见 Union find)
- Divide-and-conquer paradigm (分治思想的典型应用)
 - mergesort (合并排序), 270
 - quicksort (快速排序), 288, 293
- Division by zero (除零异常), 51
- Documentation (文档), 28
- Double hashing (二次散列), 483
- double primitive data type (double 原始数据类型), 12
- Double probing (二次探测), 483
- Doubling ratio experiment (倍率实验), 192
- Doubling test (双倍测试), 176–177
- Doubly-linked list (双向链表), 146
- Draw data type (Draw 数据类型), 82, 83
- Dump (转储), 813
- Duplicate keys (重复元素)
 - 3-way quicksort (三向切分的快速排序), 301
 - hash table (散列表), 488
 - in a symbol table (符号表中的重复元素), 363
 - MSD string sort (高位优先的字符串排序), 715
 - priority queue (优先队列), 309
 - quicksort (快速排序), 292
 - sorting (排序), 344
 - stability (稳定性), 341
- Dutch National Flag (荷兰国旗问题), 298
- Dynamic connectivity (动态连通性), 216
- Dynamic memory allocation (动态内存分配), 104
- Dynamic resizing array. See Resizing array (动态调整数组大小, 见 Resizing array)

E

- Eccentricity of a vertex (顶点的离心率), 559
- Edge (边)
 - backward (逆向), 891
 - critical (关键), 633, 900
 - crossing (横切), 606
 - data type (数据类型), 608
 - directed (有向), 566, 638
 - eligible (有效的), 646
 - forward (正向), 891
 - incident (依附于), 519
 - ineligible (无效的), 616, 646
 - parallel (平行), 518
 - self-loop (自环), 518
 - undirected (无向), 518
 - weighted (加权), 608, 638
- Edge-connected graph (边连通的图), 562
- Edge relaxation (边放松), 646–647
- Edge-weighted DAG (加权有向图), 658–667
 - critical path method (关键路径法), 663–667
 - longest paths (最长路径), 661
 - shortest paths (最短路径), 658–660
- Edge-weighted digraph (加权有向图)
 - adjacency-lists (邻接表), 644
 - complete (完全), 679
 - data type (数据类型), 641
 - diameter of (直径), 685
 - shortest paths (最短路径), 638–693

Edge-weighted graph (加权无向图)

adjacency-lists (邻接表), 609

data type (数据类型), 608

min spanning forest (最小生成森林), 605

min spanning tree (最小生成树), 604–637

Edmonds, J., 901

Eligible edge (有效边), 616, 646

Ellipsoid algorithm (椭球法), 909

Empty string epsilon (空字符串 ϵ), 789, 805

Encapsulation (封装), 96

Entropy (熵), 300–301

Epsilon-transition (ϵ -转换), 795

Equal keys. See Duplicate keys (等值键, 见 Duplicate keys)

equals() method (equals() 方法), 102–103

symbol table (符号表), 365

Equivalence class (等价类), 216

Equivalence relation (等价性)

connectivity (连通性), 216, 543

equals() method (equals() 方法), 102

strong connectivity (强连通性), 584

Erdős number (Erdős 数), 554

Erdős, P., 554

Erdős-Renyi model (Erdős-Renyi 模型), 239

Error. See also Exception (错误, 另见异常)

OutOfMemoryError, 107

StackOverflowError, 57, 107

Euclid's algorithm (欧几里德算法), 4, 58

Eulerian cycle (欧拉环), 562, 598

Event-driven simulation (事件驱动模拟), 349, 856–865

Exception. See also Error (异常, 另见错误)

Arithmetic, 107

ArrayIndexOutOfBoundsException, 107

ClassCast, 387

NoSuchElement, 139

NullPointerException, 159

Runtime, 107

UnsupportedOperationException, 139

ConcurrentModification, 160

exch() method (exch() 方法), 245, 315

Exhaustive search (穷举搜索), 912

Exponential inequality (指数级别), 185

Exponential running time (指数级别的运行时间), 186, 661, 911

Extended Church-Turing thesis (扩展丘奇-图灵论题), 910

Extensible library (可扩展的库), 101

External path length (外部路径长度), 418, 832

Factor an integer (整数的因数), 919

Factorial function (阶乘函数), 185

Fail-fast iterator (快速出错的迭代器), 160, 171

Farthest pair (最遥远的一对), 210

Fibonacci heap (斐波纳契堆), 628, 682

Fibonacci numbers (斐波纳契数), 57

FIFO. See First-in first-out policy (先进先出, 见先进先出策略)

FIFO queue. See Queue data type (先进先出队列, 见队列数据类型)

File system (文件系统), 493

Filter (过滤器), 60

blacklist (黑名单), 491

dedup (dedup 过滤器), 490

whitelist (白名单), 8, 491

final access modifier (final 访问修饰符), 105–106

Fingerprint search (指纹搜索), 774–778

Finite state automaton. See Deterministic finite state automaton

First-in-first-out policy (先进先出策略), 126

Fixed-capacity stack (定容栈), 132, 134–135

Fixed-length code (定长编码), 826

Float primitive data type (浮点型原始数据类型), 13

Flood fill (填充), 563

Floor function (向上取整函数)

binary search tree (二叉查找树), 406

mathematical function (数学函数), 185

ordered array (有序数组), 380

symbol table (符号表), 367, 383

Flow (流量), 888. See also Maxflow problem (另见 Maxflow problem)

flow network (流量网络), 888

inflow and outflow (流入量和流出量), 888

residual network (剩余网络), 895

st-flow (st-流量), 888

st-flow network (st-流量网络), 888

value (值), 888

Floyd, R. W., 326

Floyd's method (Floyd 方法), 327

for loop (for 循环), 16

Ford-Fulkerson (Ford-Fulkerson 算法), 891–893

analysis of (分析), 900

maximum-capacity path (最大容量增广路径), 901

shortest augmenting path (最短增广路径), 897

Ford, L., 683

Foreach loop (foreach 循环), 138

arrays (数组), 160
 strings (字符串), 160
 Forest (森林)
 graph (图), 520
 spanning (生成), 520
 Forest-of-trees (森林), 225
 Formatted output (格式化输出), 37
 Fortran language (Fortran 语言), 217
 Fragile base class problem (脆弱的基类问题), 112
 Frazer, W., 306
 Fredman, M. L., 628
 Function-call stack (函数调用所需的栈), 246, 415

G

Garbage collection (垃圾收集), 104, 195
 loitering (对象游荡), 137
 mark-and-sweep (标记-清除), 573
 Gaussian elimination (高斯消元法), 919
 Generics (泛型), 122–123, 134–135
 and covariant arrays (泛型与共变数组), 158
 and type erasure (泛型与类型擦除), 158
 array creation (创建数组), 134, 158
 parameterized type (参数化类型), 122
 priority queues (优先队列), 305
 stacks and queues (栈和队列), 134–135
 symbol tables (符号表), 363
 type parameter (类型参数), 122, 134
 Genomics (基因组), 492, 498
 Geometric data types (几何对象数据类型), 76–77
 Geometric sum (等比数列之和), 185
 getClass() method (getClass() 方法), 101, 103
 Girth of a graph (图的周长), 559
 Global variable (全局变量), 113
 Gosper, R. W., 759
 Graph data type (图数据类型), 522–527
 Graph isomorphism (图同构), 561, 919
 Graph processing (图处理), 514–693. See also Directed graph (另见 Directed graph); See also Edge-weighted digraph (另见 Edge-weighted digraph); See also Edge-weighted graph (另见 Edge-weighted graph); See also Undirected graph (另见 Undirected graph); See also Directed acyclic graph (另见 Directed acyclic graph)
 Bellman-Ford (Bellman-Ford 算法), 668–681
 breadth-first search (广度优先搜索), 538–541
 components (分量), 543–546
 critical-path method (关键路径法), 664–666
 depth-first search (深度优先搜索), 530–537

Dijkstra's algorithm (Dijkstra 算法), 652
 Kosaraju's algorithm (Kosaraju 算法), 586–590
 Kruskal's algorithm (Kruskal 算法), 624–627
 longest paths (最长路径), 911–912
 max bipartite matching (最大二分图匹配), 906
 min spanning tree (最小生成树), 604–637
 Prim's algorithm (Prim 算法), 616–623
 reachability (可达性), 570–573
 shortest paths (最短路径), 638–693
 strong components (强连通分量), 584–591
 symbol graphs (符号图), 548
 transitive closure (传递闭包), 592–593
 union-find (union-find 算法), 216–241
 Greatest common divisor (最大公约数), 4
 Greedy algorithm (贪心算法)
 Huffman encoding (霍夫曼编码), 830
 minimum spanning tree (最小生成树), 607
 Grep, 804

H

Halting problem (停机问题), 910
 Hamiltonian cycle (哈密尔顿环), 562, 920
 Hamiltonian path (哈密尔顿路径), 598, 913, 920
 Handle (句柄), 112
 Hard-disc model (刚性球体模型), 856
 Harmonic number (调和数), 23, 185
 Harmonic sum (调和数之和), 185
 hashCode() method (hashCode() 方法), 101, 102, 461–462
 Hash function (散列函数), 458, 459–463
 modular (除留余数), 459
 perfect (完美散列函数), 480
 Rabin-Karp algorithm (Rabin-Karp 算法), 774
 Hashing. See Hash function (散列, 见 Hash function)
 See also Hash table (另见 Hash table)
 hash function (散列函数), 459–463
 time-space tradeoff (时间和空间作出权衡), 458
 Hash table (散列表), 458–485
 array resizing (调整数组大小), 474–475
 clustering (键簇), 472
 collision resolution (处理碰撞冲突), 458
 cuckoo hashing (布谷鸟散列), 484
 deletion (删除), 468
 double hashing (二次散列), 483
 double probing (二次探测), 483
 duplicate keys (重复元素), 488
 hashCode() method (hashCode() 方法), 61–462

- hash function (散列函数), 458
 - Java library (Java 库), 489
 - linear probing (线性探测), 469-474
 - load factor (使用率), 471
 - memory usage of (内存使用), 476
 - primitive types (原始数据类型), 488
 - separate chaining (拉链法), 464-468
 - uniform hashing assumption (均匀散列假设), 463
 - Head vertex (头顶点), 566
 - Heap. See Binary heap (堆, 见 Binary heap)
 - multiway (多叉堆), 319
 - Heap order (堆有序), 313
 - Heapsort (堆排序), 323-327
 - Height (树的高度)
 - 2-3 search tree (2-3 查找树), 429
 - binary search tree (二叉查找树), 412
 - complete binary tree (完全二叉树), 314
 - red-black BST (红黑二叉查找树), 444
 - tree (树), 226
 - Hibbard deletion (Hibbard 删除方法), 422
 - Hibbard, T., 410
 - Hoare, C. A. R., 205
 - Horner's method (Horner 方法), 460
 - h*-sorted array (*h* 有序数组), 258
 - Huffman compression (霍夫曼压缩), 350, 826-838
 - analysis of (分析), 833
 - optimality of (最优性), 833
 - Huffman, D., 827
- I
- if statement (if 语句), 15
 - if-else statement (if-else 语句), 15
 - Immutability (不可变性), 105-106
 - defensive copy (保护性复制), 112
 - of strings (字符串的不可变性), 114, 202, 696
 - priority queue keys (优先队列中的元素), 320
 - symbol table keys (符号表中的键), 365
 - Implementation (实现), 28, 88
 - Implementation inheritance (实现继承), 101
 - import statement (import 语句), 27, 29, 66
 - Incident edge (关联边), 519
 - Increment sequence (递增序列), 258
 - In data type (In 数据类型), 41, 83
 - Indegree of a vertex (顶点的入度), 566
 - Index (索引), 361, 496-501
 - string (字符串索引), 877
 - files (文件索引), 500-501
 - inverted (反向索引), 498-501
 - Index priority queue (索引优先队列), 320-322
 - Dijkstra's algorithm (Dijkstra 算法), 652
 - Prim's algorithm (Prim 算法), 620
 - Indirect sort (间接排序), 286
 - Ineligible edge (无效边)
 - minimum spanning tree (最小生成树), 616
 - shortest paths (最短路径), 646
 - Infix notation (中缀记法), 13, 128, 162
 - Inherited methods (继承的方法), 66, 100-101
 - compare(), 338-339
 - compareTo(), 246-247
 - equals(), 102-103
 - getClass(), 101
 - hashCode(), 101, 461-462
 - hasNext(), 138
 - iterator(), 138
 - next(), 138
 - toString(), 66, 101
 - Inner loop (内循环), 180, 184, 195
 - Inorder tree traversal (中序遍历), 412
 - In-place merge (原地合并), 270
 - Input and output (输入和输出), 82-83
 - binary data (二进制数据), 812-815
 - from a file (从文件重定向标准输入或将标准输出重定向到文件), 41
 - pipng (管道), 40
 - redirection (重定向), 40
 - Input model (输入模型), 197
 - Input size (输入规模), 173
 - Insertion sort (插入排序), 250-252
 - Instance method (实例方法), 65, 84
 - Instance variable (实例变量), 84
 - int primitive data type (int 原始数据类型), 12
 - Integer linear inequality satisfiability problem (整数线性不等式可满足性问题), 913
 - Integer linear programming (整数线性规划), 920
 - Integer overflow (整数溢出), 51
 - Interface (接口), 100
 - Comparable, 246-247
 - Comparator, 338-340
 - Iterable, 138
 - Iterator, 139
 - Interface inheritance (接口继承), 100
 - Interior point method (内点法), 909
 - Internal path length (内部路径长度), 412
 - Internet DNS (互联网 DNS), 493
 - Internet Movie Database (互联网电影数据库), 497
 - Interpreter (解释器), 130

Interval graph (区间图), 564
 Intractability (不可解性), 910–921
 Inversion (反向), 252,, 286
 Inverted index (反向索引), 498–501
 Ising model (伊辛模型), 920
 Isomorphic graph (同构图), 561
 Item (元素)
 contains a key (每个元素有一个主键), 244
 sorting (排序), 244
 symbol table (符号表), 387
 with multiple keys (多键数组), 339
 Item type parameter (Item 数据类型), 134
 Iteration (迭代), 123, 138–141
 fail-fast (快速出错), 171
 foreach loop (foreach 循环), 123

J

Jacquet, P., 882
 Jarnik's algorithm (Jarnik 算法), 628
 See also Prim's algorithm (另见 Prim 算法)
 Jarnik, V., 628
 Java programming (Java 编程)
 array (数组), 18–21
 arrays as objects (数组对象), 72
 arrays of objects (对象的数组), 72
 assertion (断言), 107
 assert statement (assert 语句), 107
 assignment statement (赋值语句), 14
 autoboxing (自动装箱), 122
 autounboxing (自动拆箱), 122
 base class (基类), 101
 bitwise operators (位运算符), 52
 block statement (语句块), 15
 boolean expression (布尔表达式), 13
 break statement (break 语句), 15
 bytecode (字节码), 10
 cast (类型转换), 13
 class (类), 10, 64
 classpath (类路径), 66
 comparison operator (比较运算符), 13
 conditional statement (条件语句), 15
 constructor (构造函数), 65, 84
 continue statement (continue 语句), 15
 covariant arrays (共变数组), 158
 create an object (创建对象), 67
 declaration statement (声明语句), 14
 default initialization (默认初始化), 18, 86

deprecated method (弃用的方法), 113
 derived class (派生类), 101
 Error, 107
 Exception, 107
 expression (表达式), 11, 13
 final modifier (final 修饰符), 84, 105–106
 for loop (for 循环), 16
 foreach loop (foreach 循环), 123
 garbage collection (垃圾收集), 104
 generic array creation (创建泛型数组), 158
 generics (泛型), 122–123, 134–135
 identifier (标识符), 11
 if statement (if 语句), 15
 if-else statement (if-else 语句), 15
 implicit assignment (隐式赋值), 16
 import statement (import 语句), 29
 imported system libraries (导入的系统库), 27
 infix expression (中缀表达式), 13
 inheritance (继承), 100–101
 inherited method (继承的方法), 66
 initializing declarations (初始化声明), 16
 inner class (内部类), 159
 instance method (实例方法), 65, 84, 86
 instance method signature (实例方法签名), 86
 instance variable (实例变量), 84
 invoke instance method (调用实例方法), 68
 iterable collections (可迭代的集合), 123
 just-in-time compiler (JIT 编译器), 195
 literal (字面量), 11
 loitering (游离对象), 137
 loop statement (循环语句), 15
 memory management (内存管理), 104
 modular programming (模块化编程), 26
 nested class (嵌套类), 159
 new(), 67
 objects (对象), 67–74
 objects as arguments (对象作为参数), 71
 objects as return values (对象作为返回值), 71
 operator (运算符), 11
 operator precedence (运算符优先级), 13
 orphan (孤儿), 137
 orphaned object (孤儿对象), 104
 overloading (重载), 12, 24
 override a method (重载方法), 101
 parameterized type (参数化类型), 122, 134
 pass by reference (按引用传递), 71
 pass by value (按值传递), 24, 71
 primitive data type (原始数据类型), 11–12
 private class (private 类), 159

- private modifier (private 修饰符), 84
- protected modifier (protected 修饰符), 110
- public modifier (public 修饰符), 84, 110
- ragged array (参差不齐的数组), 19
- recursion (递归), 25
- reference (引用), 67
- reference type (引用类型), 64
- return statement (return 语句), 86
- scope (作用域), 14, 87
- short-circuiting (短路求值法则), 52
- side effects (副作用), 24
- single-statement blocks (单语句代码段), 16
- standard libraries (标准库), 27
- standard system libraries (标准系统库), 27
- statement (语句), 14
- static method (静态方法), 22-25
- static variable (静态变量), 113
- strong typing (强类型), 14
- subclass (子类), 101
- superclass (父类), 101
- this reference (this 引用), 87
- throw an error/exception (抛出错误或异常), 107
- two-dimensional array (二维数组), 19
- type conversion (类型转换), 13, 35
- type erasure (类型擦除), 158
- type parameter (类型参数), 122
- unit testing (单元测试), 26
- using objects (使用对象), 69
- variable (变量), 11
- visibility modifier (可见性修饰符), 84
- while loop (while 循环), 15
- wrapper type (封装类型), 122
- Java system sort (Java 系统排序), 306
- Java virtual machine (Java 虚拟机), 51
- java.awt
 - Color, 75
 - Font, 75
- java.io
 - File, 75
- java.lang
 - ArithmeticException, 107
 - ArrayIndexOutOfBoundsException, 107
 - Boolean, 102
 - Byte, 102
 - Character, 102
 - ClassCastException, 387
 - Comparable, 100
 - Double, 34, 102
 - Float, 102
 - Integer, 102
 - Iterable, 100, 123, 138, 154
 - Long, 102
 - Math, 28
 - NullPointerException, 107, 113, 159
 - Object, 101
 - OutOfMemoryError, 107
 - RuntimeException, 107
 - Short, 102
 - StackOverflowError, 57, 107
 - StringBuilder, 27, 105, 697
 - UnsupportedOperationException, 139
- java.net
 - URL, 75
- java.util
 - ArrayList, 160
 - Arrays, 29
 - Comparator, 100, 339
 - ConcurrentModification, 160
 - Date, 113
 - HashMap, 489
 - Iterator, 100, 138-141, 154
 - LinkedList, 160
 - NoSuchElementException, 139
 - PriorityQueue, 352
 - Stack, 159
 - TreeMap, 489
- Job-scheduling problem. See Scheduling (任务调度问题, 见 Scheduling)
- Josephus problem (Josephus 问题), 168
- Just-in-time compiler (JIT 编辑器), 195

K

- Karp, R., 759, 901
- Kendall tau distance (Kendall's tau 距离), 286, 345, 356
- Kevin Bacon number (Kevin Bacon 数), 553-554
- Key (键), 244
- Key equality (键的等价性)
 - ordered symbol table (有序符号表), 368
 - symbol table (符号表), 365
- Key-indexed counting (键索引计数法), 703-705
- Key type parameter (Key 类型参数)
 - priority queue (优先队列), 309
 - symbol table (符号表), 361
- Keyword in context (上下文中的关键词), 879
- Khachian, L. G., 909
- Kleene's theorem (Kleene 定理), 794

L

- Knuth, D. E., 178, 205, 759
 Knuth-Morris-Pratt, 762–769
 Kosaraju's algorithm (Kosaraju 算法), 586–590
 Kruskal, J., 628
 Kruskal's algorithm (Kruskal 算法), 624–627
 KWIC. See Keyword-in-context (KWIC, 见 Keyword-in-context)
- Last-in-first-out policy (后进先出策略), 127**
 Las Vegas algorithm (拉斯维加斯算法), 778
 Leading-term approximation. See Tilde notation (首项近似, 见 Tilde notation)
 Least-significant digit (最低有效位数)
 See LSD string sort (见 LSD string sort)
 Leipzig Corpora Collection (Leipzig Corpora 数据库), 371
 Lempel, A., 839
 less() method (less() 方法), 245, 315
 Level-order traversal (按层遍历)
 binary heap (二叉堆), 313
 binary search tree (二叉查找树), 420
 Levin, L., 918
 LIFO. See Last-in first-out policy (LIFO, 见 Last-in first-out policy)
 LIFO stack. See Stack data type (LIFO 栈, 见 Stack data type)
 Linear equation satisfiability (线性等式可满足性) 913
 Linear inequality satisfiability (线性不等式可满足性), 913
 Linear probing (线性探测), 469–474
 Linear programming (线性规划), 907–909
 ellipsoid algorithm (椭球法), 909
 interior point method (内点法), 909
 reductions (归约), 907–909
 simplex algorithm (单纯形法), 909
 Linear running time (线性级别的运行时间), 186
 Linearithmic running time (线性对数级别的运行时间), 186
 Linked allocation (链式存储), 156
 Linked list (链表), 142–146
 building (创建链表), 143
 circular (环形链表), 165
 defined (定义), 142
 deletion (删除元素), 145
 deletion from beginning (从表头删除元素), 145
 garbage collection (垃圾收集), 145
 insertion (插入), 145
 insertion at beginning (在表头插入节点), 144
 insertion at end (在表尾插入节点), 145
 iterator (迭代器), 154–155
 memory usage of (内存使用), 201
 Node data type (Node 数据类型), 142
 queue (队列), 150
 reverse a (将链表反转), 165
 sequential search (顺序查找), 374
 shuffle a (打乱链表), 288
 sort a (链表排序), 286
 stack (栈), 147–149
 traversal (遍历), 146
 Literal (字面量)
 null, 112–113
 primitive type (原始数据类型), 11
 string (字符串), 80
 Load-balancing (负载均衡), 349, 909
 Load factor (使用率), 471
 Local minimum (数组的局部最小元素), 210
 Logarithm function (对数函数)
 binary (以 2 为底的对数函数), 185
 integer binary (以 2 为底的整型对数函数), 185
 natural (自然对数函数), 185
 Logarithmic running time (线性对数级别的运行时间), 186
 Log-log plot (对数图像), 176
 Loitering (游离对象), 137
 Longest common prefix (最长公共前缀), 875
 Longest paths (最长路径), 661, 911
 Longest prefix match (匹配的最长前缀), 842
 Longest processing-time first rule (最大优先), 349
 Longest repeated substring (最长重复子字符串), 875
 long primitive data type (long 原始数据类型), 13
 Loop (循环)
 for, 16
 foreach, 138
 inner (内部循环), 180
 while, 15
 Lossless data compression (无损数据压缩), 811
 Lossy data compression (有损数据压缩), 811
 Lower bound (下界)
 priority queue (优先队列), 332
 sorting (排序), 279–282
 3-sum problem (3-sum 问题), 190
 union-find (union-find 算法), 231
 Lowest common ancestor (最近公共祖先), 598
 Lloyd, S., 358
 LSD string sort (低位优先的字符串排序), 706–709

LZW algorithm (LZW 压缩算法), 839-845
 compression (压缩), 840
 expansion (压缩的展开), 841
 trie representation (单词查找树表示), 840

M

Manber, U., 884
 Mark-and-sweep garbage collection (标记-清除的垃圾收集), 573
 Maslow, A., 904
 Maslow's hammer (Maslow 的锤子), 904
 Matrix data type (矩阵数据类型), 60
 Maxflow-mincut theorem (最大流-最小切分定理), 894
 Maxflow problem (最大流问题), 886-902
 See also Mincut problem (另见 Mincut problem)
 Ford-Fulkerson (Ford-Fulkerson 算法), 891-893
 integrality property (完整性), 894
 maxflow-mincut theorem (最大流-最小切分定理), 892-894
 max bipartite matching (最大二分图匹配问题) 906
 preflow-push algorithm (preflow-push 算法), 902
 reductions (归约), 905-907
 residual network (剩余网络), 895-897
 Maximum (最大元素)
 in array (数组中的最大元素), 30
 in binary heap (二叉堆中的最大元素), 313
 in binary search tree (二叉查找树中的最大元素), 406
 in ordered symbol table (有序符号表中的最大元素), 367
 Maximum *st*-flow problem. See Maxflow problem (最大 *st*-流量问题, 见 Maxflow problem)
 Max bipartite matching (最大二分图匹配问题), 906
 Maze (迷宫), 530
 McIlroy, D., 298, 306
 McKellar, A., 306
 Median (中位数), 332, 345-347
 Median-of-3 partitioning (三取样切分), 305
 Memory management (内存管理), 104
 linked allocation (链式存储), 156
 loitering (游离对象), 137
 orphan (孤儿), 137
 Sequential allocation (顺序存储), 156
 Memory usage (内存使用), 200-204
 array (数组), 202
 hash table (散列表), 476
 linked list (链表), 201
 nested class (嵌套类), 201

object (对象), 67, 201
 primitive types (原始数据类型), 200
 R-way trie (R 向单词查找树), 746
 stack (栈), 213
 string (字符串), 202
 substring (子字符串), 202-204
 Mergesort (合并排序), 270-288
 abstract in-place merge (抽象原地合并算法), 270
 analysis of (合并排序分析), 272
 bottom-up (自底向上的合并排序), 277
 linked list (链表), 279, 286
 multiway (多向合并排序), 287
 natural (自然合并排序), 285
 optimality (最优算法), 282
 stability (稳定性), 341
 top-down (自顶向下的合并排序), 272
 Merging (合并), 270-271
 Method (方法)
 inherited (继承的方法), 100-101
 instance (实例方法), 68-69, 86-87
 static (静态), 22-25
 Mincut problem (最小切分问题), 893
 See also Maxflow problem (另见 Maxflow problem)
 Minimum (最小元素)
 in array (数组中的最小元素), 30
 in binary search tree (二叉查找树中的最小元素), 406
 in ordered symbol table (有序符号表中的最小元素), 367
 Min spanning forest (最小生成森林), 605
 Min spanning tree (最小生成树), 604-637
 Boruvka's algorithm (Boruvka 算法), 636
 bottleneck shortest paths (瓶颈最短路径), 690
 critical edge (关键边), 633
 crossing edge (横切边), 606
 cut (切分), 606
 cut optimality conditions (最优切分条件), 634
 cut property (切分定理), 606
 defined (定义), 604
 greedy algorithm (贪心算法), 607
 Kruskal's algorithm (Kruskal 算法), 624-627
 Prim's algorithm (Prim 算法), 616-623
 reverse-delete algorithm (逆向删除算法), 633
 Vyssotsky's algorithm (Vyssotsky 算法), 633
 Minimum *st*-cut problem. See Mincut problem (最小 *st*-切分问题, 见 Mincut problem)
 Minotaur (米诺陶), 530
 Mismatched character rule (启发式处理不匹配的字符法则), 770
 M. L. Fredman, 628

Modular hash function (除留余数法散列函数), 459, 774
 Modular programming (模块化编程), 26
 Monte Carlo algorithm (蒙特卡洛法), 776
 Moore, J. S., 759
 Moore's law (摩尔定律), 194-195
 Morris, J. H., 759
 Most-significant-digit sort. See MSD string sort (高位优先的字符串排序, 见 MSD string sort)
 Move-to-front (前移编码), 169
 MSD string sort (高位优先的字符串排序), 710-718
 Multidimensional sort (多维排序), 356
 Multigraph (多重图), 518
 Multiple-source reachability problem (多点可达性问题), 570, 797
 Multiset, 509
 Multway mergesort (多向合并排序), 287
 Multway trie. See R-way trie (多向单词查找树, 见 R-way trie)
 Myers, E., 884

N

Natural logarithm function (自然对数函数), 185
 Natural mergesort (自然合并排序), 285
 Natural order (自然次序), 337
 Negative cost cycle (负权重的环)
 See Negative cycle (见 Negative cycle)
 Negative cycle (负权重的环), 668-670, 677-681
 Nested class (嵌套类), 159
 Network flow (网络流量)
 See Maxflow problem (见 Maxflow problem)
 new(), 67
 Newton's method (牛顿迭代法), 23
 NFA. See Nondeterministic finite-state automata (非确定有限状态自动机, 见 Nondeterministic finite-state automata)
 Node data type (Node 数据类型), 159
 bag (背包), 155
 binary search tree (二叉查找树), 398
 Huffman trie (霍夫曼单词查找树), 828
 linked list (链表), 142
 queue (队列), 151
 red-black BST (红黑二叉查找树), 433
 R-way trie (R 向单词查找树), 734
 stack (栈), 149
 ternary search trie (三向单词查找树), 747
 Nondeterminism (非确定), 794
 Turing machine (图灵机), 914

Nondeterministic finite-state automata (非确定有限状态自动机), 794-799
 NP, 912
 NP-complete (NP-完全), 917-918
 Null link (空链接), 396
 null literal (null 字面量), 112-113

O

Object (对象), 67-74
 See also Object-oriented programming (另见 Object-oriented programming)
 behavior (行为), 67, 73
 identity (身份), 67, 73
 memory usage of (对象的内存使用), 201
 state (对象的状态), 67, 73
 Object-oriented programming (面向对象编程), 64-119
 arrays of objects (对象数组), 72
 creating an object (创建对象), 67
 declaring an object (声明对象), 67
 encapsulation (封装), 96
 inheritance (继承), 100
 instance (实例), 73
 instantiate an object (实例化对象), 67
 invoke instance method (调用实例方法), 68
 objects (对象), 67-74
 objects as arguments (对象作为参数), 71
 objects as return values (对象作为返回值), 71
 reference (引用), 67
 subtyping (子类型), 100
 using objects (使用对象), 69
 Odd-length cycle in a graph (图中长度为奇数的环), 562
 OOP. See Object-oriented programming (OOP, 见 Object-oriented programming)
 Operations research (运筹学), 349
 Optimization problem (最优化问题), 913
 Ordered symbol table (有序符号表), 366-369
 floor and ceiling (向上取整和向下取整函数), 367
 minimum and maximum (最小元素和最大元素), 367
 ordered array (有序数组), 378
 range query (范围查找), 368
 rank and selection (排名和选择), 367
 red-black BST (红黑二叉查找树), 446
 Order of growth (增长数量级), 179
 Order-of-growth classifications (增长数量级的分类), 186-188
 Order-of-growth hypothesis (增长数量级的猜想), 180
 Order statistic (顺序统计), 345

binary search tree (二叉查找树), 406
 ordered symbol table (有序符号表), 367
 quicksort (快速排序), 345-347
 Orphaned object (孤儿对象), 104, 137
 Out data type (Out 数据类型), 41, 83
 Outdegree of a vertex (顶点的出度), 566
 Output. See Input and output (输出, 见 Input and output)
 Overflow (溢出), 51
 Overloading (过载)
 constructor (构造函数), 84
 static method (静态方法), 24
 Overriding a method (重载方法), 66, 101

P

P complexity class (*P*-复杂性类), 914
 $P=NP$ question ($P=NP$ 问题), 916
 Page data type (Page 数据类型), 870
 Palindrome (回文), 81, 783
 Parallel arrays (平行的数组)
 linear probing (线性探测), 471
 ordered symbol table (有序符号表), 378
 sorting (排序), 357
 Parallel edge (平行边), 518, 566, 612, 640
 Parallel job scheduling (并行任务调度), 663-667
 Parallel precedence-constrained scheduling (优先级限制下的并行任务调度), 663, 904
 Parameterized type. See Generics (参数化类型, 见 Generics)
 Parent-link representation (父链接形式)
 breadth-first search tree (广度优先搜索树), 539
 depth-first search tree (深度优先搜索树), 535
 minimum spanning tree (最小生成树), 620
 shortest-paths tree (最短路径树), 640
 union-find (union-find 算法), 225
 Parsing (解析)
 an arithmetic expression (算术表达式), 128
 a regular expression (正则表达式), 800-804
 Particle data type (Particle 数据类型), 860
 Partitioning algorithm (切分算法), 290
 2-way (二向切分), 288
 3-way (Bentley-McIlroy) (Bentley-McIlroy 三向切分), 306
 3-way (Dijkstra) (Dijkstra 三向切分), 298
 median-of-3 (三取样切分), 296, 305
 median-of-5 (五取样切分), 305
 selection (基于切分的选择算法), 346-347
 Partitioning item (切分元素), 290
 Pass by reference (按引用传递), 71
 Pass by value (按值传递), 24, 71
 Path. See Longest paths
 See also Shortest paths (路径, 见 Longest paths, 另见 Shortest paths)
 augmenting (增广), 891
 Hamiltonian (哈密尔顿), 913, 920
 in a digraph (在有向图中), 567
 in a graph (在图中), 519
 length of (长度), 519, 567
 simple (简单), 519, 567
 Path compression (路径压缩), 231
 Pattern matching
 See Regular expression (模式匹配, 见 Regular expression)
 Perfect hash function (完美散列函数), 480
 Performance. See Propositions (性能, 见命题)
 Permutation (排列)
 Kendall-tau distance (Kendall's tau 距离), 356
 random (随机排列), 168
 ranking (排名), 345
 sorting (排序), 354
 Phone book (电话黄页), 492
 Picture data type (Picture 数据类型), 814
 Piping (管道), 40
 Point data type (Point 数据类型), 77
 Pointer, 111. See also Reference (指针, 另见 Reference)
 Safe (安全指针), 112
 Pointer sort (指针排序), 338
 Poisson approximation (泊松近似), 466
 Poisson distribution (泊松分布), 466
 Polar angle (极角), 356
 Polar coordinate (极坐标), 77
 Poly-time reduction (多项式时间问题的相互归约), 916
 Pop operation (出栈操作), 127
 Postfix notation (后缀记法), 162
 Postorder traversal (后序遍历)
 of a digraph (对于有向图), 578
 reverse (取反), 578
 Power law (幂次规则), 178
 Pratt, V. R., 759
 Precedence-constrained scheduling (优先级限制下的任务调度), 574-575
 Precedence order (优先级次序)
 arithmetic expressions (算术表达式), 13
 regular expressions (正则表达式), 789
 Prefix-free code (前缀码), 826-827
 compression (压缩), 829
 expansion (压缩的展开), 828
 Huffman (霍夫曼), 833

- optimal (最优的), 833
- reading and writing (读和写), 834-835
- trie representation (单词查找树表示), 827
- Preorder traversal (前序遍历)
 - of a digraph (对于无向图), 578
 - of a trie (对于单词查找树), 834
- Prime number (素数), 23, 774, 785
- Primitive data type (原始数据类型), 11-12
 - memory usage of (内存使用), 200
 - wrapper type (封装类型), 102
- Primitive type (原始数据类型)
 - versus reference type (及引用类型), 110
- Prim, R., 628
- Prim's algorithm (Prim 算法), 350, 616-623
 - eager (即时实现), 620-623
 - lazy (延时), 616-619
- Priority queue (优先队列), 308-335
 - binary heap (二叉堆), 313-322
 - change priority (改变优先级), 321
 - delete (删除元素), 321
 - Dijkstra's algorithm (Dijkstra 算法), 652
 - Fibonacci heap (斐波纳契堆), 628
 - Huffman compression (霍夫曼压缩), 830
 - index priority queue (索引优先队列), 320-321
 - linked-list (链表), 312
 - multiway heap (多叉堆), 319
 - ordered array (有序数组), 312
 - Prim's algorithm (Prim 算法), 616
 - reductions (归约), 345
 - remove the minimum (删除最小元素), 321
 - stability (稳定性), 356
 - unordered array (无序数组), 310
- private access modifier (private 访问修饰符), 84
- Probabilistic algorithm. See Randomized algorithm (演算法, 见 Randomized algorithm)
- Probe (探测), 471
- Problem size (问题规模), 173
- Programs (程序)
 - Accumulator, 93
 - AcyclicLP, 661
 - AcyclicSP, 660
 - Arbitrage, 680
 - Average, 39
 - Bag, 155
 - BellmanFordSP, 674
 - BinaryDump, 814
 - BinarySearch, 47
 - BinarySearchST, 379, 381, 382
 - BlackFilter, 491
 - BoyerMoore, 772
 - BreadthFirstPaths, 540
 - BST, 398, 399, 407, 409, 411
 - BTreeSET, 872
 - Cat, 82
 - CC, 544
 - CollisionSystem, 863-864
 - Count, 699
 - Counter, 89
 - CPM, 665
 - Cycle, 547
 - Date, 91, 103, 247
 - DeDup, 490
 - DegreesOfSeparation, 555
 - DepthFirstOrder, 580
 - DepthFirstPaths, 536
 - DepthFirstSearch, 531
 - Digraph, 569
 - DijkstraAllPairsSP, 656
 - DijkstraSP, 655
 - DirectedCycle, 577
 - DirectedDFS, 571
 - DirectedEdge, 642
 - DoublingTest, 177
 - Edge, 610
 - EdgeWeightedDigraph, 643
 - EdgeWeightedGraph, 611
 - Evaluate, 129
 - Event, 861
 - Example, 245
 - FileIndex, 501
 - FixedCapacityStack, 135
 - FixedCapacityStackOfStrings, 133
 - Flips, 70
 - FlipsMax, 71
 - FlowEdge, 896
 - FordFulkerson, 898
 - FrequencyCounter, 372
 - Genome, 819-820
 - Graph, 526
 - GREP, 804
 - Heap, 324
 - HexDump, 814
 - Huffman, 836
 - Insertion, 251
 - KMP, 768
 - KosarajuSCC, 587
 - KruskalMST, 627
 - KWIC, 881

- LazyPrimMST, 619
- LinearProbingHashST, 470
- LookupCSV, 495
- LookupIndex, 499
- LRS, 880
- LSD, 707
- LZW, 842, 844
- MaxPQ, 318
- Merge, 271, 273
- MergeBU, 278
- MSD, 712
- Multiway, 322
- NFA, 799, 802
- PictureDump, 814
- PrimMST, 622
- Queue, 151
- Quick, 289, 291
- Quick3string, 720
- Quick3way, 299
- RabinKarp, 777
- RedBlackBST, 439
- ResizingArrayQueue, 140
- ResizingArrayStack, 141
- Reverse, 127
- RLE, 824
- Rolls, 72
- Selection, 249
- SeparateChainingHashST, 465
- SequentialSearchST, 375
- SET, 489
- Shell, 259
- SortCompare, 256
- SparseVector, 503
- Stack, 149
- StaticSETofInts, 99
- Stats, 125
- Stopwatch, 175
- SuffixArray, 883
- SymbolGraph, 552
- ThreeSum, 173
- ThreeSumFast, 190
- TopM, 311
- Topological, 581
- Transaction, 340
- TransitiveClosure, 593
- TrieST, 737-741
- TST, 747
- TwoColor, 547
- TwoSumFast, 189
- UF, 221
- VisualAccumulator, 95
- WeightedQuickUnionUF, 228
- WhiteFilter, 491
- Whitelist, 99
- Properties (性质), 180
 - 3-sum, 180
 - Boyer-Moore algorithm (Boyer-Moore 算法), 773
 - insertion sort (插入排序), 255
 - quicksort (快速排序), 343
 - Rabin-Karp algorithm (Rabin-Kar 算法), 778
 - red-black BST (红黑二叉查找树), 445
 - selection sort (选择排序), 255
 - separate-chaining (投链法), 467
 - shellsort (希尔排序), 262
 - versus proposition (性质与命题), 183
- Propositions (命题), 182
 - 2-3 search tree (2-3 树), 429
 - 3-sum, 182
 - 3-way quicksort (三向快速排序), 301
 - 3-way string quicksort (3-向字符串快速排序), 723
 - arbitrage (套汇), 681
 - B-tree (B-树), 871
 - Bellman-Ford, 671, 673
 - binary heap (二叉堆), 319
 - binary search (二分查找), 383
 - BST (二叉查找树), 403-404, 412
 - breadth-first search (广度优先搜索), 541
 - brute substring search (暴力子字符串查找), 761
 - complete binary tree (完全二叉查找树), 314
 - connected components (连通分量), 546
 - Cook-Levin theorem (Cook-Levin 定理), 918
 - critical path method (关键路径法), 666
 - cut property (切分定理), 606
 - DFS (深度优先搜索), 531, 537, 570
 - Dijkstra's algorithm (Dijkstra 算法), 652, 654
 - flow conservation (流量守恒), 893
 - Ford-Fulkerson (Ford-Fulkerson 算法), 900-901
 - generic shortest-paths (通用最短路径), 651
 - greedy MST algorithm (贪心最小生成树算法), 607
 - heapsort (堆排序), 323, 326
 - Huffman algorithm (霍夫曼算法), 833
 - index priority queue (索引优先队列), 321
 - insertion sort (插入排序), 250, 252
 - integer programming (整数规划), 917
 - key-indexed counting (键索引计数法), 705
 - Knuth-Morris-Pratt, 769
 - Kosaraju's algorithm (Kosaraju 算法), 588, 590
 - Kruskal's algorithm (Kruskal 算法), 624, 625

linear-probing hash table (线性探测散列表), 475
 linear programming (线性规划), 908
 longest paths in DAG (有向无环图中的最长路径), 661
 longest repeated substring (最长重复子字符串), 885
 LSD string sort (低位优先的字符串排序), 706, 709
 maxflow-mincut theorem (最大流-最小切分定理), 894
 maxflow reductions (最大流消减), 906
 mergesort (合并排序), 272, 279, 282
 MSD string sort (高位优先的字符串排序), 717, 718
 negative cycles (负权重环), 669
 parallel job scheduling with relative deadlines (相对最后期限限制下的并行任务调度问题), 667
 particle collision (粒子的相互碰撞), 865
 Prim's algorithm (Prim 算法), 616, 618, 623
 quick-find algorithm (quick-find 算法), 223
 quicksort (快速排序), 293-295
 quick-union algorithm (quick-union 算法), 226
 red-black BST (红黑二叉查找树), 444, 447
 regular expression (正则表达式), 799, 804
 R-way trie (R-向单词查找树), 742, 743, 744
 selection sort (选择排序), 248
 separate-chaining (拉链法), 466, 475
 sequential search (顺序查找), 376
 shortest paths in DAG (有向无环图中的最短路径), 658
 shortest-paths optimality (最短路径最优性条件), 650
 shortest paths reductions (最短路径归约), 905
 sorting lower bound (排序下界), 280, 300
 sorting reductions (排序问题), 903
 suffix array (后缀数组), 882
 ternary search trie (三向单词查找树), 749, 751
 topological order (拓补排序), 578, 582
 universal compression (通用压缩), 816
 weighted quick-union (加权 quick-union 算法), 229
 protected modifier (protected 修饰符), 110
 Protein folding (蛋白质折叠), 920
 public access modifier (public 访问修饰符), 110
 Pushdown stack (下压栈), 127
 See also Stack data type (另见 Stack data type)
 Push operation (入栈操作), 127

Q

Quadratic running time (平方级别的运行时间), 186
 Quantum computer (量子计算机), 911
 Queue data type (Queue 数据类型)
 analysis of (分析), 198

API, 126
 circular linked list (环形链表), 165
 linked-list (链表), 150-151
 resizing-array (可调整大小的数组), 140
 Quick-find algorithm (Quick-find 算法), 222-223
 Quicksort (快速排序), 288-307
 2-way partitioning (二向切分), 290
 3-way partitioning (三向切分), 298-301
 3-way string (三向字符串), 719
 analysis of (分析), 293-295
 and binary search trees (与二叉查找树), 403
 duplicate keys (重复元素), 292
 median-of-3 (三取样切分), 296, 305
 median-of-5 (五取样切分), 305
 nonrecursive (非递归), 306
 random shuffle (随机打乱), 292
 Quick-union (Quick-union 算法), 224-227
 path compression (路径压缩 Quick-union 算法), 231
 weighted (加权 Quick-union 算法), 227-230

R

Rabin-Karp algorithm (Rebin-Karp 算法), 774-778
 Rabin, M. O., 759
 Radius of a graph (图的半径), 559
 Radix (基数), 700
 Radix sorting. See String sorting (基数排序法, 见 String sorting)
 Random bag data type (随机背包数据类型), 167
 Randomized algorithm (随机化算法), 198
 Las Vegas (拉斯维加斯), 778
 Monte Carlo (蒙特卡洛), 776
 quicksort (快速排序), 290, 307
 Rabin-Karp algorithm (Rabin-Karp 算法), 776
 3-way string quicksort (三向字符串快速排序), 722
 Random number (随机数), 30-32
 Random queue data type (随机队列数据类型), 168
 Random string model (随机字符串模型), 716-717
 Range query (范围查找)
 binary search tree (二叉查找树), 412
 ordered symbol table (有序符号表), 368
 Rank (排名)
 binary search (二分查找), 25, 378-381
 binary search tree (二叉查找树), 408, 415
 ordered symbol table (有序符号表), 367
 suffix array (后缀数组), 879
 Reachability (可达性), 570-572, 590
 Reachable vertex (可达顶点), 567

- Recurrence relation
 binary search (二分查找), 383
 mergesort (合并排序), 272
 quicksort (快速排序), 293
- Recursion (递归), 25
 See also Base case (另见 Base case);
 See also Recursion (另见 Recursion)
- binary search (二分查找), 25, 380
 binary search tree, 401
 depth-first search (深度优先搜索), 531
 Euclid's algorithm (欧几里德算法), 4
 Fibonacci numbers (斐波纳契数), 57
 mergesort (合并排序), 272
 quicksort (快速排序), 289
- Red-black BST (红黑二叉查找树), 432–447
 and 2-3 search tree (与 2-3 查找树), 432
 analysis of (分析红黑 BST), 444–447
 color flip (颜色转换), 436
 color representation (颜色表示), 433
 defined (定义), 432
 delete the maximum (删除最大元素), 454
 delete the minimum (删除最小元素), 453
 deletion (删除元素), 441–443, 455
 implementation (实现), 439
 insertion (插入), 437–439
 left-leaning (左链接), 432
 perfect black balance (完美黑色平衡), 432
 rotation (旋转), 433–434
 search (查找), 432
- Redirection (重定向), 40
- Reduction (问题归约), 903–909
 defined (定义), 903
 polynomial-time (多项式时间), 916
 linear programming (线性规划), 907–909
 maxflow (最大流量), 905–907
 priority queue (优先队列), 345
 shortest-paths (最短路径), 904–905
 sorting (排序), 344–347, 903–904
- Reference (引用), 67
- Reference type (引用类型), 64
- Reflexive relation (自反关系), 102, 216, 247, 584
- Regular expression (正则表达式), 82, 788
 building an NFA (构造 NFA), 800–804
 closure operation (闭包操作), 789
 concatenation operation (连接操作), 789
 defined (定义), 790
 epsilon-transition (ϵ -转换), 795
 match transition (匹配转换), 795
 nondeterministic finite-state automaton (非确定有限状态自动机), 794–799
 or operation (或操作), 789
 parentheses (括号), 789
 $\backslash s^+$, 82
 shortcuts (缩略写法), 791
 simulating an NFA (NFA 的模拟), 797–799
- Rehashing (重新散列), 474
- Relation (关系)
 antisymmetric (反对称性), 247
 equivalence (等价性), 102, 216, 584
 reflexive (自反性), 102, 216, 247, 584
 symmetric (对称性), 102, 216, 584
 total order (完整的比较序列), 247
 transitive (传递性), 102, 216, 247, 584
- Residual network (剩余网络), 895–897
- Resizing array (可调整大小的数组), 136–137
 binary heap (二叉堆), 320
 hash table (散列表), 474–475
 queue (队列), 140
 stack (栈), 136
- Return value (返回值), 22
- Reverse postorder traversal (逆后序遍历), 578
- Reverse (反向, 逆向)
 a linked list (将链表反转), 165–166
 an array (颠倒数组元素的顺序), 21
 array iterator (数组反向迭代器), 139
 with a stack (将栈中的元素逆序排列), 127
- Reverse-delete algorithm (逆向删除算法), 633
- Reverse graph (反图), 586
- Reverse postorder (逆后序), 578
- Ring buffer data type (环形缓冲区数据类型), 169
- RLE. See Run-length encoding
- Robson, J., 412
- Rooted tree (一棵树), 640
- Rotation in a BST (BST 中的旋转操作), 433–434, 452
- Run-length encoding (游程编码), 822–825
- Running time (运行时间), 172–173
 analysis of (分析运行时间), 176
 constant (常数级别的运行时间), 186
 cubic (立方级别的运行时间), 186
 doubling ratio (倍率), 192
 exponential (指数级别的运行时间), 186
 inner loop (内循环), 180
 linear (线性级别的运行时间), 186
 logarithmic (对数级别的运行时间), 186
 measuring (测量准确的运行时间), 174
 order of growth (运行时间的增长数量级), 179
 quadratic (立方级别的运行时间), 186
 tilde approximation (近似), 178–179

Run-time error. See Error; See also Exception (运行时间错误, 见 Error, 另见 Exception)

R-way trie (R 向单词查找树), 730–744

- Alphabet (字母表), 741
- analysis of (分析), 742–743
- collecting keys (查找所有键), 738
- deletion (删除), 740
- insertion (插入), 734
- longest prefix (最长前缀), 739
- memory usage of (内存使用空间), 744
- one-way branching (单向分支), 744–745,
- representation (表示), 734
- search (查找), 732–733
- wildcard match (通配符匹配), 739

S

Safe pointer (安全指针), 112

Sample mean (采样期望值), 30

Samplesort (取样排序), 306

Sample standard deviation (采样标准差), 30

Sample variance (采样), 30

Scheduling (调度)

- critical-path method (关键路径法), 664–666
- load-balancing problem (负载均衡问题), 349
- LPT first (最大优先), 349
- parallel precedence-constrained (优先级限制下的并行), 663–667
- precedence constraint (优先级限制), 574–575
- relative deadlines (相对最后期限), 666
- SPT first (最小优先), 349

Scientific method (科学方法), 172

Scope of a variable (变量作用域), 14, 87

Search hit (查找命中), 376

Searching (查找), 360–513. See also Symbol table (另见 Symbol table)

Search miss (查找未命中), 376

Search problem (搜索问题), 912

Sedgewick, R., 298

Selection (选择), 345

- binary search tree (二叉查找树), 406
- ordered symbol table (有序符号表), 367
- suffix array (后缀数组), 879

Selection client (选择用例), 249

Selection sort (选择排序), 248–249

Self-loop (自环), 518, 566, 612, 640

Separate-chaining (拉链法), 464–468

Sequential allocation (顺序存储), 156

Sequential search (顺序查找), 374–377

Set data type (Set 数据类型), 489–491

Shannon entropy (香农信息量), 300–301

Shellsort (希尔排序), 258–262

Shortest ancestral path (最短先导路径), 598

Shortest augmenting path (最短增广路径), 897

Shortest path (最短路径), 638

Shortest paths problem (最短路径问题), 638–693

- all-pairs (顶点对), 656
- arbitrage detection (套汇检测), 679–681
- Bellman-Ford (Bellman-Ford 算法), 668–678
- bitonic (双调), 689
- bottleneck (瓶颈), 690
- certification (验证), 651
- critical edge (关键边), 690
- Dijkstra's algorithm (Dijkstra 算法), 652–657
- edge relaxation (边放松), 646–647
- edge-weighted DAG (加权有向无环图), 658–667
- generic algorithm (通用算法), 651
- ineligible edge (无效边), 646
- in Euclidean graphs (在欧拉图中), 656
- monotonic (单调), 689
- negative cycle (负权重环), 669
- Negative cycle detection (负权重环检测), 670
- negative weights (负权重), 668–681
- optimality conditions (最优条件), 650
- parent-link (父结点的链接), 640
- reduction (问题归约), 904–905
- shortest-paths tree (最短路径树), 640
- single-source (单点), 639, 654
- source-sink (给定两点), 656
- undirected graph (无向图), 654
- vertex relaxation (顶点放松), 648

Shortest-processing-time-first rule (最小优先法则), 349, 355

short primitive data type (short 原始数据类型), 13

Shuffling (打乱)

- a linked list (打乱链表), 286
- an array (打乱数组), 32
- quicksort (打乱快速排序结果), 292

Side effect (副作用), 22, 108

Signature (签名)

- instance method (实例方法签名), 86
- static method (静态方法签名), 22

Simple digraph (简单有向图), 567

Simple graph (简单图), 518

Simplex algorithm (单纯形法), 909

Single-source problems (单点问题)

- connectivity (连通性), 556

directed paths (有向路径), 573
 longest paths in DAG (有向无环图中的最长路径), 661
 paths (路径), 534
 reachability (可达性), 570
 shortest directed paths (最短有向路径), 573
 shortest paths in undirected graphs (无向图中的最短路径), 654, 904
 shortest paths (最短路径), 538, 639
 Social network (社交网络), 517
 Software cache (缓存), 391, 451, 462
 Sollin, M., 628
 Sorting (排序), 242–359
 See also String sorting (另见字符串排序)
 3-way quicksort (三向快速排序), 298–301
 binary search tree (二叉查找树), 412
 certification (认证), 246, 265
 Comparable, 246–247
 compare-based (基于比较的排序算法), 279
 complexity of (排序复杂性), 279–282
 cost model (排序的成本模型), 246
 entropy-optimal (平均信息量最优的排序), 296–301
 extra memory (额外的内存使用), 246
 heapsort (堆排序), 323–327
 indirect (间接排序), 286
 in-place (原地排序), 246
 insertion sort (插入排序), 250–252
 inversion (反向排序), 252
 lower bound (下界), 279–282, 306
 mergesort (合并排序), 270–288
 partially-sorted array (部分有序的数组), 252
 pointer (指针), 338
 primitive types (原始数据类型), 343
 quicksort (快速排序), 288–307
 reduction (归约问题), 903–904
 reductions (归约), 344–347
 selection sort (选择排序), 248–250
 shellsort (希尔排序), 258–262
 stability (稳定性), 341
 suffix array (后缀数组), 875–885
 system sort (系统排序), 343
 Source-sink shortest paths (给定两点的最短路径), 656
 Spanning forest (生成森林), 520
 Spanning tree (生成树), 520, 604
 Sparse graph (稀疏图), 520
 Sparse matrix (稀疏矩阵), 510
 Sparse vector (稀疏向量), 502–505
 Specification problem (说明书问题), 97
 SPT. See Shortest paths tree;

See also Shortest-processing-time-first rule (最短路径树, 见 Shortest paths tree, 另见 Shortest-processing-time-first rule)
 st-cut (st- 切分), 892
 st-flow (st- 流量配置), 888
 st-flow network (st- 流量网络), 888
 Stability (稳定性), 341, 355
 insertion sort (插入排序), 341
 key-indexed counting (键索引计数法), 705
 LSD string sort (低位优先的字符串排序), 706
 mergesort (合并排序), 341
 priority queue (优先队列), 356
 Stack data type (栈数据类型), 127
 analysis of (分析栈), 198, 199
 array implementation (实现保存在栈中的数组), 132
 fixed-capacity (定容栈), 132–133
 generic (泛型), 134
 iteration (迭代), 138–140
 linked-list (链表), 147–149
 resizing array (可调整大小的数组), 136
 Standard deviation (标准差), 30
 Standard drawing (标准图像), 36, 42–45
 Standard input (标准输入), 36, 39
 Standard libraries (标准库), 30
 Draw, 82–83
 In, 41, 82–83
 Out, 41, 82–83
 StdDraw, 43
 StdIn, 39
 StdOut, 37
 StdRandom, 30
 StdStats, 30
 Stopwatch, 174–175
 Standard output (标准输出), 36, 37–38
 Static method (静态方法), 22–25
 argument (参数), 22
 defining a (定义静态方法), 22
 invoking a (调用静态方法), 22
 overloaded (重载静态方法), 24
 pass by value (按值传递), 24
 recursive (递归), 25
 return statement (返回语句), 24
 return value (返回值), 22
 side effect (副作用), 22, 24
 signature (签名), 22
 Static variable (静态变量), 113
 Statistics (统计学)
 chi-square (卡方检验), 483
 median (中位数), 345

- minimum and maximum (最小值和最大值), 30
- order (顺序), 345
- sample mean (采样期望值), 30, 125
- sample standard deviation (采样标准差), 30
- sample variance (采样方差), 30, 125
- StdDraw library, 43
- StdIn library, 39
- StdOut library, 37
- StdRandom library, 30
- StdStats library, 30
- Steque data type (Steque 数据类型), 167, 212
- Stirling's approximation (斯特灵公式), 185
- Stopwatch data type (Stopwatch 数据类型), 174–175
- String data type (字符串数据类型), 34, 80–81
 - API, 80
 - characters (字符), 696
 - charAt() method (charAt() 方法), 696
 - concatenation (字符串的连接), 34, 697
 - conversion (字符串类型转换), 102
 - immutability (不可变性), 696
 - indexing (索引), 696
 - indexOf() method (indexOf() 方法), 779
 - length (字符串长度), 696
 - length() method (length() 方法), 696
 - literal (字面量), 34
 - memory usage of (内存使用), 202
 - + operator (+ 运算符), 80, 697
 - substring extraction (提取子字符串), 696
 - substring() method (substring() 方法), 696
- String processing (字符串处理), 80–81, 694–851
 - data compression (数据压缩), 810–851
 - regular expression (正则表达式), 788
 - sorting (排序), 702–729
 - substring search (子字符串查找), 758–785
 - suffix array (后缀数组), 875–885
 - tries (单词查找树), 730–757
- String search. See Substring search; See also Trie (字符串查找, 见 Substring search, 另见 Trie)
- String sorting (字符串排序), 702–729
 - 3-way quicksort (三向快速排序), 719–723
 - key-indexed counting (键索引计数法), 703
 - LSD string sort (低位优先的字符串排序), 706–709
 - MSD string sort (高位优先的字符串排序), 710–718
- Strong component (强连通分量), 584
- Strong connectivity (强连通性), 584–591
- Strongly connected component. See Strong component (强连通的分量, 见 Strong component)
- Strongly connected relation (强连通的关系), 584
- Strongly typed language (强类型语言), 14
- Subclass (子类), 101
- Subgraph (子图), 519
- Sublinear running time (次线性运行时间), 716, 779
- Substring extraction (子字符串提取)
 - memory usage of (内存使用), 202–204
 - substring() method (substring() 方法), 696
- Substring search (子字符串查找), 758–785
 - Boyer-Moore (Boyer-Moore 算法), 770–773
 - brute-force (暴力查找), 760–761
 - indexOf() method (indexOf() 方法), 779
 - Knuth-Morris-Pratt, 762–769
 - Rabin-Karp, 774–778
- Subtyping (子类型), 100
- Suffix array (后缀数组), 875–885
- Suffix array data type (后缀数组数据类型), 879
- Suffix-free code (后缀码), 847
- Superclass (父类), 101
- Symbol digraph (符号有向图), 581
- Symbol graph (符号图), 548–555
- Symbol table (符号表), 360–513
 - 2-3 search tree (2-3 查找树), 424–431
 - API, 363, 366
 - associative array (关联数组), 363
 - balanced search tree (平衡查找树), 424–457
 - binary search (二分查找), 378–384
 - binary search tree (二叉查找树), 396–423
 - B-tree (B-树), 866–874
 - cost model (成本模型), 369
 - defined (符号表的定义), 362
 - duplicate key policy (重复元素), 363
 - floor and ceiling (向上取整和向下取整), 367
 - hash table (散列表), 458–485
 - insertion (插入), 362
 - key equality (等值键), 365
 - lazy deletion (延时删除), 364
 - linear-probing (线性探测), 469–474
 - minimum and maximum (最小元素和最大元素), 367
 - null value (null 值), 364
 - ordered (有序符号表), 366–369
 - ordered array (有序数组), 378
 - range query (范围查找), 368
 - rank and selection (排名和选择), 367
 - red-black BST (红黑二叉查找树), 432–447
 - R-way trie (R 向单词查找树), 732–745
 - search (查找), 362
 - separate-chaining (拉链法), 464–468
 - sequential search (顺序查找), 374
 - string keys (字符串键), 730–757
 - ternary search trie (三向单词查找树), 746–751

trie (单词查找树), 730-757
 unordered linked list (无序链表), 374
 Symmetric order (树的对称性), 396
 Symmetric relation (对称关系), 102, 216, 584
 Szpankowski, W., 882

T

Tail vertex (顶点的尾), 566
 Tale of Two Cities (双城记), 371
 Tandem repeat (串联重复查找), 784
 Tarjan, R. E., 590, 628
 Terminal window (终端窗口), 10, 36
 Ternary search trie (三向单词查找树), 746-751
 alphabet (字母表), 750
 analysis of (分析), 749
 collecting keys (查找所有键), 750
 deletion (删除), 750
 insertion (插入), 746
 one-way branching (单向分支), 751, 755
 prefix match (匹配前缀), 750
 search (查找), 746
 wildcard match (通配符匹配), 750
 Theseus (忒修斯), 530
 this reference (this 引用), 87
 Threading (线性符号表), 420
 Tilde notation (近似), 178, 206
 Time-driven simulation (时间驱动模拟), 856
 Timing a program (为应用程序计时), 174-175
 Top-down 2-3-4 tree (自顶向下的 2-3-4 树), 441
 Top-down mergesort (自顶向下的合并排序), 273
 Topological sort (拓补排序), 574-583
 depth-first search (深度优先搜索), 578
 queue-based algorithm (基于队列的算法), 599
 toString() method (toString() 方法), 66, 102
 Total order (完整的比较序列), 247
 Transaction data type (事务数据类型), 78-79
 compare(), 340
 compareTo(), 266, 337
 hashCode(), 462
 Transitive closure (传递闭包), 592
 Transitive relation (传递关系), 102, 216, 247, 584
 Transpose a matrix (转置矩阵), 56
 Tree (树)
 2-3 search tree (2-3 查找树)
 See 2-3 search tree (见 2-3 search tree)
 binary. See Binary tree (二进制. 见 Binary tree)
 binary search tree (二叉查找树)
 See Binary search tree (见 Binary search tree)
 balanced search tree. See Balanced search tree (平衡查找树. 见 Balanced search tree)
 binomial (二项树), 237
 depth of a node (树中的节点深度), 226
 height of (树的高度), 226
 inorder traversal (中序遍历), 412
 min spanning tree. See Minimum spanning tree (最小生成树. 见 Minimum spanning tree)
 parent-link (父结点的链接), 535, 539
 preorder traversal (前序遍历), 834
 rooted (根结点), 640
 size (树的大小), 226
 spanning tree. See Spanning tree (生成树. 见 Spanning tree)
 undirected graph (无向图), 520
 union-find (union-find 算法), 224-226
 Tremaux exploration (Tremaux 搜索), 530
 Triangular sum (等差数列之和), 185
 Trie (单词查找树), 730-757
 See also R-way trie;
 See also Ternary search trie (见 R-way trie, 另见 Ternary search trie)
 collecting keys (查找所有键), 731
 Lempel-Ziv-Welch, 840
 longest prefix match (匹配最长前缀), 731, 842
 one-way branching (单向分支), 744-745, 751, 755
 prefix-free code (前缀代码), 827
 preorder traversal (前序遍历), 834
 reading and writing (读和写), 834-835
 wildcard match (通配符匹配), 731
 Tufte plot (Tufte 图), 456
 Tukey ninther, 306
 Turing, A., 910
 Turing machine (图灵机), 910
 Church-Turing thesis (丘奇-图灵论题), 910
 computability (可计算性), 910
 nondeterministic (非确定性的), 914
 universality (普遍性), 910
 Type conversion (类型转换), 13
 Type erasure (类型擦除), 158
 Type parameter (类型参数), 122, 134

U

Undecidability (不可判定性), 97, 817
 Undirected graph (无向图)
 acyclic (无环), 520

adjacency-lists (邻接表), 524
 adjacency-matrix (邻接矩阵), 524
 adjacency-sets (邻接集), 527
 adjacent vertex (邻接顶点), 519
 articulation point (关节点), 562
 biconnected (双向连通的), 562
 bipartite (二分的), 521, 546–547, 562
 breadth-first search (广度优先搜索), 538–542
 bridge (桥), 562
 center (中点), 559
 connected (连通的), 519
 connected component (连通分量), 519
 connected to relation (连通关系), 519
 connectivity (连通性), 534, 543–546
 cycle (环), 519
 cycle detection (环检测), 546–547
 defined (定义), 518
 degree (度), 519
 dense (稠密), 520
 depth-first search (深度优先搜索), 530–533
 diameter (直径), 559
 edge (边), 518
 edge-connected (边连通的), 562
 edge-weighted (加权)
 See Edge-weighted graph (见 Edge-weighted graph)
 Euler tour (欧拉回路), 562
 forest (森林), 520
 girth (周长), 559
 Hamilton tour (哈密尔顿回路), 562
 interval graph (区间图), 564
 isomorphism (同构), 561
 multigraph (多重图), 518
 odd cycle detection (长度为奇数的环检测), 562
 parallel edge (平行边), 518
 path (路径), 519
 radius (半径), 559
 self-loop (自环), 518
 simple (简单), 518
 simple cycle (简单环), 519, 567
 simple path (简单路径), 519
 single-source connectivity (单点连通性), 556
 single-source paths (单点路径), 534
 single-source shortest paths (单点最短路径), 538
 spanning forest (生成森林), 520
 spanning tree (生成树), 520
 sparse (稀疏), 520
 subgraph (子图), 519
 tree (树), 520
 two-colorability (两种颜色着色), 546–547, 562

vertex (顶点), 518

weighted (权重)

See Edge-weighted graph (见 Edge-weighted graph)

Unicode (Unicode 编码), 696

Uniform hashing (均匀散列), 463

Union-find (union-find 算法), 216–241

depth-first search (深度优先搜索), 546

binomial tree (二叉树), 237

Boruvka's algorithm (Boruvka 算法), 636

dynamic connectivity (动态连通性), 216

forest-of-trees (森林), 225

Kruskal's algorithm (Kruskal 算法), 625

parent-link (父链接), 225

path compression (路径压缩), 231, 237

quick-find (quick-find 算法), 222–223

quick-union (quick-union 算法), 224–227

weighted quick-find (加权 quick-find 算法), 236

weighted quick-union (加权 quick-union 算法), 227–231

weighted quick-union by height (根据高度加权的

quick-union 算法), 237

weighted quick-union with path compression (使用路径压缩的加权 quick-union 算法), 237

Uniquely decodable code (解码方式唯一的编码), 826

Unit testing (单元测试), 26

Universal data compression (通用数据压缩), 816

Universality (通用性), 910

Upper bound (上界), 206, 207, 281

V

Value type parameter (值类型参数)

symbol table (符号表), 361

trie (单词查找树), 730

Variable (变量), 10

Variable-length code (变长编码), 826

Variance (共变), 30

Vector data type (向量数据类型), 106

Vertex (顶点)

adjacent (邻接), 519

connected to relation (连通关系), 519

degree of (度), 519

eccentricity (离心率), 559

head and tail (头和尾), 566

indegree and outdegree (入度和出度), 566

reachable (可达), 567

source (点), 528

Vertex cover problem (顶点覆盖问题), 920

Vertex relaxation (顶点放松), 648

Virtual terminal (虚拟终端), 10
 Vyssotsky's algorithm (Vyssotsky 算法), 633

W

Web search (网络搜索), 496
 Weighted digraph.
 See Edge-weighted digraph (加权有向图, 见 Edge-weighted digraph)
 Weighted edge (加权边), 604, 638
 Weighted external path length (加权外部路径长度), 832
 Weighted graph. See Edge-weighted graph (加权图, 见 Edge-weighted graph)
 Weighted quick-union (加权 quick-union 算法), 227–231
 Weighted quick-union with path compression (使用路径压缩的加权 quick-union 算法), 237

Weiner, P., 884
 Welch, T., 839
 while loop (while 循环), 15
 Whitelist filter (白名单过滤器), 8, 48–49, 99, 491
 Wide interface (宽接口), 160, 557
 Wildcard character (通配符), 791
 Wildcard match (通配符匹配), 750
 Worst-case guarantee (对最坏情况下的性能保证), 197
 Wrapper type (封装类型), 102, 122

Z

Ziv, J., 839
 Zero-based indexing (起始索引是 0), 53
 Zipf's law (Zipf 法则), 393

数字图书馆
 推广工程
 PDG