3.算法设计与分析

本章主要研究如何设计算法以及如何评价一个算法（算法复杂度）

两个重要想法：放缩和几何改进 scaling and geometric improvement argument

一些用于分析和简化算法的工具：potential function method、parameter balancing technique、dynamic programming、binary search

search algorithms

3.2 复杂性分析

本书中算法的衡量指标——时间

三种衡量算法性能的方法：1.实证分析 2.平均案例分析（为问题实例选择一个概率分布，统计分析解决问题的时间）3.最坏情况分析

网络问题的大小取决于问题的表述方式

算法的时间复杂度函数是问题大小的函数，规定了算法所用时间的上限。

对于大 O 表示法：1.表示算法运行时间的上界 2.仅表示运行时间中最重要的项 3.假设基本数学运算所用的时间相同

相似性假设：如果数字是指数级的，那么运行时间是有误导性的，为了避免这种情况，假设最大弧成本 C 和最大弧容量 U 是有界的，C=O（n^k）和 U=O（n^k）

多项式和指数时间算法：多项式时间算法（复杂度为 n、m、logc 和 logu 的多项式函数），指数时间算法（不是多项式时间算法的算法，如 O（nC），0（2^n），O（n!），和 O（$n^{\log n}$）等），伪多项式算法（如 O（m+nC）和 O（mC）），对于满足相似性假设的问题，伪多项式时间算法变为多项式时间算法，通常多项式时间算法的性能优于指数时间算法

计算机硬件能力的提高对多项式时间算法的影响是巨大的，但对指数时间算法的问题解决能力的影响微乎其微

需要注意的是，将最坏情况下的复杂性视为平均复杂性的同义词可能会导致错误的结论

大Ω表示法：规定算法的下界

大θ表示法：算法同时是 O（f（n））和Ω（f（n））的特殊情况

势函数和分摊复杂度：势函数是一种属性，通过不同的操作能够改变势函数，势函数能够表示一组操作的最坏情况。通常，我们通过对其他类型的步数使用已知的界限来限定一种类型的步数。分摊复杂度是指一组操作中的每一个分摊的平均复杂度

参数平衡：计算复杂度时微分法可能不太好解决，这时候可以用参数平衡法寻找近似解，

假设我们选择 k*，使得 f（n，m，k*）=g（n，m，k*）。设λ*=f（n，m，k*）+g（n，m，k*）。对于任何 k<k*，

$$f(n, m, k) + g(n, m, k) \geq f(n, m, k) \geq f(n, m, k^*) = \lambda^*/2.$$

第二个不等式来自函数 f（n，m，k）在 k 中单调递减的事实。类似地，对于任何 k>k*，

$$f(n, m, k) + g(n, m, k) \geq g(n, m, k) \geq g(n, m, k^*) = \lambda^*/2. \qquad (3.2)$$

表达式（3.1）和（3.2）意味着对于任何 k，

$$f(n, m, k) + g(n, m, k) \geq \lambda^*/2.$$

3.3 开发多项式算法

几何改进算法：具有几何收敛速度的网络算法是多项式时间算法，即每次迭代向目标结果的改进量是固定百分比

由于 H 是最大可能的改进，并且每个目标函数值都是整数，因此算法必须在 O（(log H）/α）内终止迭代。

放缩法：从简单问题出发，一步步逼近复杂问题

动态规划：通过求解与原问题相同性质的小规模问题逐渐逼近最终问题

二进制搜索：二分法

3.4 搜索算法

搜索算法试图找到网络中满足特定属性的所有节点，搜索算法的应用包括（1）在网络中查找特定节点能通过有向路径到达的所有节点，(2) 在网络中查找可以沿着有向路径到达特定节点 t 的所有节点，（3）识别网络的所有连接节点，以及（4）确定给定网络是否为二分图。

拓扑排序：能够找出这样一种排序使得节点 1，2，…，n 对于每个弧（i，j）∈A，都有 i<j。

搜索算法的基本思想是利用标记的方法，从源节点出发逐步标记能够沿着有向路径到达的节点，这种算法中的逐步包括两种策略，分别是先进先出的广度优先算法和后进先出的深度优先算法

定理 3.3。在广度优先搜索树中，从源节点 s 到任何节点 i 的树路径是“最短路径”（即，在连接这两个节点的所有路径中包含最少数量的弧）。

定理 3.4 在深度优先搜索树中（a）如果节点 j 是 i 的后代，则 order（j）>order（i）。

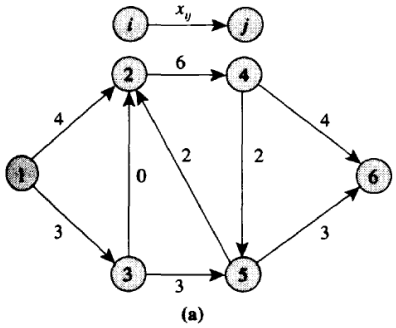
（b） 任何节点的所有后代都是按顺序连续排列的。

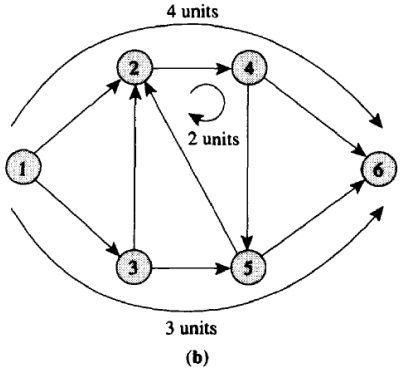
反向搜索算法：识别网络中通过有向路径可以从给定节点到达的所有节点，只需在搜索算法的基础上做一点小调整：从终点出发，搜索传入弧

拓扑序算法：首先计算所有节点的 indegree，形成一个由 indegree 为零的所有节点组成的集合列表。在每次迭代中，从列表中选择一个节点 i，从 i 出发的每个弧（i，j）∈A（i），都将节点 j 的 indegree 减少 1 个单位，如果节点 j 的 indegree 变为零，我们将节点 j 添加到集合列表中。其核心思想在于在迭代中将标签分配给节点时，节点只能有传出弧

3.5 流分解算法

在描述网络流问题时，可以采用两种等效的建模方法中的任何一种：定义弧上的流或者定义路径和循环上的流。





从路径觉和循环角度建立的模型用另一个术语-e（i）替换了节点 i 的供需 b（i）；我们称 e（i）为节点的不平衡（为什么）

弧（i，j）上的流 Xij 等于包含该弧的所有路径 P 和循环 W 的流 f（P）和 f（W）之和。如果弧（i，j）包含在路径 P 中，则 δij（P）等于 1，否则为 0。同样，如果循环 W 中包含弧（i，j），则 δ（W）等于 1，否则为 0。可以得到

$$x_{ij} = \sum_{P \in \mathcal{P}} \delta_{ij}(P) f(P) + \sum_{W \in \mathcal{W}} \delta_{ij}(W) f(W).$$

定理 3.5（流分解定理）。每个路径和循环流都有一个唯一的非负弧流表示。相反，每个非负弧流 x 可以表示为具有以下两个特性的路径和循环流（尽管不一定唯一）：

（a） 每一条有向正流路径连接一个亏损节点和一个过剩节点。

（b） 最多 n+m 路径和循环具有非零流；其中，最多 m 个循环具有非零流。

定理 3.6。循环 circulation x 可以表示为沿最多 m 个定向循环的循环流。

增广循环：如果通过增大循环周围的正流量 f（W），流仍然是可行的，则循环 W（不一定有方向）称为关于流 x 的增广循环。增加的流量增加了循环中前向弧上的流量，减少了循环中后向弧上的流量。因此，如果每个向前弧（i，j）的 Xij<Uij，每个向后弧（i，j）的 Xij>0，则循环 W 是 G 中的增广循环。如果弧（i，j）是循环 W 中的前向弧，则定义 δij（W）等于 1；如果弧（i，j）是循环 W 中的后向弧，则定义 δij（W）等于-1；否则定义 δij（W）等于 0。这样可以得到一个增广周期的成本$c(W) = \sum_{(i,j) \in W}$，如果在一个周期中增加 1 个单位的流量，那么一个增加周期的成本代表了一个可行解的成本的变化。沿循环 W 增加 f（W）单位的流量成本变化为 c（W）f（W）。

定理 3.7（增广循环定理）。设 x 和 x^0 是网络流问题的任意两个可行解。那么 x 等于 x^0 加上 G（x^0）中最多 m 个有向循环上的流。此外，x 的成本等于 x^0 的成本加上这些扩充循环上的流的成本。

定理 3.8（负循环最优性定理）。当且仅当剩余网络 G（x*）不包含负费用有向循环时，最小费用流问题的可行解 x*是最优解。