第1章

Chapter 1

引言：某些典型的问题

1.1 第一个问题：稳定匹配

作为开始，我们先看一个算法问题，这个问题对我们将要强调的许多主题都做了非常好的说明. 它来自于某些非常自然的实际问题，从中我们得到关于问题的清晰简洁的形式化陈述. 求解这个问题的算法也非常清晰，而我们的主要工作是证明它的正确性，并且对于为得到一个解所占用的时间量给出一个可接受的界. 这个问题——稳定匹配问题——本身有着几个起源.

1.1.1 问题

稳定匹配问题一部分起源于 1962 年，当时 David Gale 和 Lloyd Shapley，两个数理经济学家问了这个问题：能否设计一个学校的入学或工作招聘处理过程，使得它是自强化的？他们的这句话意味着什么呢？

为提出问题，让我们首先简略地想一下这种情况. 可能有一组朋友，都是在学院主修计算机科学专业的学生，开始申请到公司暑期实习. 申请过程的关键是两类不同的参与者：公司(雇主)和学生(申请人)之间的相互影响. 每个申请人对公司有一个优先排序，一旦申请人来到公司，每个公司对它的申请人也构成一个优先排序. 基于这些优先排序，公司对他们的某些申请人发出录用通知书，而申请人则选择某个通知书来接受，于是人们开始进入暑期实习.

Gale 和 Shapley 考虑了这种处理在缺少任何强制保持现状的机制下可能要出几类问题. 例如，假设你的朋友 Raj 刚接受了一个大电信公司 CluNet 的暑期工作. 一个小的新兴的公司 WebExodus，它做最后的决策时慢了几拍，几天以后它也打电话给 Raj，也给了他一个暑期的工作. 现在，也许是由于这种随意的、什么都能做的氛围的吸引，Raj 实际上更愿意去 WebExodus 而不是 CluNet. 于是这个新的发展也可能使他取消接受 CluNet 的录用，而去 WebExodus. 由于突然少了一个暑期实习者，因此 CluNet 向待聘的一个申请人提供一项工作，这个人马上取消他早先从一家软件巨人 Babelsoft 接受的录用，那么情况开始连续失控.

情况看起来似乎不妙，但从另一方面来看，这还不算太坏. 假如 Raj 的朋友 Chelsea 已经预定去 Babelsoft，她刚听说了 Raj 的事. 于是，她打电话给 WebExodus 的人说："你知道，我暑假真的愿意去你们那里，而不是 Babelsoft". 他们发现这很可信，接着看了 Chelsea 的申请，感到他们更愿意聘用她，而不是实际上被安排在 WebExodus 暑期实习的另一个学生. 在这种情况下，如果 WebExodus 不是那种顾虑多的公司，它可能找到某种适当的方式取消对另一个学生的录用，转而聘用 Chelsea.

类似这样，情况可能乱成一团. 许多人——申请人和雇主双方——都对这种处理和结果感到不快. 究竟什么地方出问题了？一个基本的问题是这个过程不是自强化的——如果允许人们以他们自己的兴趣行事，那么它就有毁掉的风险.

我们可能更喜欢下面比较稳定的情况，在这种情况下兴趣本身就能防止录用通知被取消和重寄. 考虑另一个学生，他已被安排在 CluNet 暑期实习，但是他打电话给 WebExodus 并且表示更愿意为他们工作. 在这种情况下，鉴于录用通知已经被接受，他们可以回答："不，实际情况是我们宁愿要已经接受的每个学生，所以恐怕我们对你帮不上忙."或者考虑一个雇主，他处心积虑想招来那些排名在前的申请人，而这些人去了其他地方，结果每个人都告诉他，"不，我在这里非常满意". 在这种情况下，所有的结果都是稳定的，不再有进一步可行的幕后交易.

因此这就是 Gale 和 Shapley 问的问题：给定一组在雇主和申请人之间的优先权，我们能否把申请人分配给雇主，以使得对每个雇主 E，以及没分配为 E 工作的每个申请人 A，至少下面两种情况之一成立？

(i) E 对他所接受的每个申请人比 A 更满意；或者

(ii) A 对他目前的情况比其为雇主 E 工作更满意.

如果这种情况成立，结果是稳定的：个人兴趣将防止任何申请人/雇主进行幕后交易.

Gale 和 Shapley 对这一问题继续提出了一个引人注目的算法解答，不久我们就会讨论它. 在此之前，请注意这不是**稳定匹配问题**的唯一起源. 原来在 Gale 和 Shapley 工作的十年前，具有同样潜在的动机，已经在一个类似的过程中使用过国家医生匹配计划，将医生与医院进行匹配. 对此他们并不知情. 的确，这一系统至今仍在使用，几乎没做什么改变.

这证实了问题的基本吸引力. 从本书的观点看，为导出某些基本的组合定义以及在这些定义之上的算法，这个问题为我们提供了第一个好的论域.

问题形式化 了解这个概念的本质有助于使得这个问题尽可能的清晰. 公司和申请人的世界包含了某些混乱的不均匀性. 每个申请人正在找一家公司，而每家公司正在找多个申请人；此外，可能存在着比暑期工作的有效位置更多(或者在某些时候更少)的申请人. 最后，每个申请人通常不会申请每一家公司.

至少在初始阶段，排除这些复杂因素，得到一个"露骨的"问题描述是有益的. n 个申请人中的每个人对 n 个公司中的每个公司提出申请，每个公司想接受单一的申请人. 我们将看到这样做可以保留问题固有的基本特点；特别是我们对这个简化描述的解将被直接推广到更一般的情况.

沿着 Gale 和 Shapley 的思路，我们观察到这个特殊情况可以被看做一个系统设计问题，通过这个设计使得 n 个男人和 n 个女人中的每个人最终都成婚了. 我们的问题自然地模拟了两种"性别"——申请人和公司——在这种情况下，我们正在考虑的是，每个人都在寻求

与恰好一个相反性别的人配对[①].

于是，考虑 n 个男人的集合 $M=\{m_1,m_2,\cdots,m_n\}$ 和 n 个女人的集合 $W=\{w_1,w_2,\cdots,w_n\}$. 令 $M\times W$ 表示所有可能的形如 (m,w) 的有序对的集合，其中 $m\in M,w\in W$. 一个匹配 S 是来自 $M\times W$ 的有序对的集合，并且具有下述性质：每个 M 的成员和每个 W 的成员至多出现在 S 的一个有序对中. 一个完美匹配 S' 是具有下述性质的匹配：M 的每个成员和 W 的每个成员恰好出现在 S' 的一个对里.

匹配和完美匹配是贯穿这本书频繁出现的术语；在广泛的算法问题建模领域中它们都会自然地出现. 此刻，一个完美匹配仅仅对应于一种男人和女人配对的方式，以这种方式，每个人最终与某个人结婚，且没有人与多个人结婚——既没有单身的，也没有一夫多妻或一妻多夫的情况.

现在我们在这个背景下增加优先的概念. 每个男人 $m\in M$ 对所有的女人排名，如果 m 给 w 的排名高于 w'，我们就说 m 偏爱 w 超过 w'. 我们将把 m 的按顺序的排名作为他的优先表，但我们不允许排名中出现并列. 类似地，每个女人也对所有的男人排名.

给定一个完美匹配 S，它可能出错吗？按照我们在雇主和申请人方面初始的动机，我们应该担心下述情况：(如图 1.1 所示)在 S 中存在两个对 (m,w) 和 (m',w')，它们具有 m 更偏爱 w' 而不爱 w，且 w' 更偏爱 m 而不爱 m' 的性质. 在这种情况下，没有什么能阻止 m 和 w' 放弃他们当前的伴侣并且转过来走到一起，婚姻的集合不再是自强化的. 我们说这样的对 (m,w') 是一个相对于 S 的不稳定因素：(m,w') 不属于 S，但是 m 和 w' 双方都偏爱另一方而不爱他们在 S 中的伴侣.

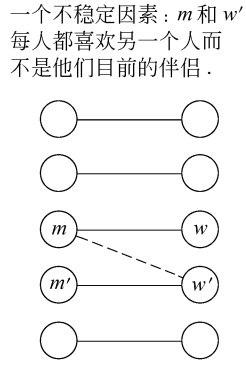

图 1.1 具有不稳定元素 (m,w') 的完美匹配 S

那么我们的目标就是一个不含有不稳定因素的婚姻集合. 我们说一个匹配 S 是稳定的，如果(i)它是完美的，并且(ii)不存在相对于 S 的不稳定因素. 马上头脑中出现两个问题：

- 对每组优先表是否存在一个稳定匹配？
- 给定一组优先表，如果存在稳定匹配，我们能够有效地把它构造出来吗？

某些例子 为说明这些定义，考虑下面两个非常简单的**稳定匹配问题**的例子.

第一个例子，假设我们有两个男人的集合 $\{m,m'\}$ 和两个女人的集合 $\{w,w'\}$. 优先表如下：

m 更偏爱 w 而不爱 w'.

m' 更偏爱 w 而不爱 w'.

w 更偏爱 m 而不爱 m'.

w' 更偏爱 m 而不爱 m'.

---

① Gale 和 Shapley 也考虑了同性的**稳定匹配问题**，其中只有一种单一性别. 这也由相关的应用所推动，但是已被证明在技术层面上是相当不同的. 鉴于我们这里正在考虑的申请人-雇主的应用，我们将关注具有两个性别的问题.

如果我们直观上考虑这组优先表，它描述了完全的一致性：男人在女人的顺序上一致，且女人在男人的顺序上也一致. 存在一个由(m,w)和(m',w')对组成的唯一的稳定匹配. 另一个由(m,w')和(m',w)组成的完美匹配不是稳定匹配，因为(m,w)对构成了相对于这个匹配的不稳定因素(m 和 w 双方都想离开他们各自的伴侣而组成一对).

下一个例子，情况更复杂一些. 假设优先表是：

m 更偏爱 w 而不爱 w'.

m'更偏爱 w'而不爱 w.

w 更偏爱 m'而不爱 m.

w'更偏爱 m 而不爱 m'.

在这种情况下会发生什么？两个男人的优先表完全互相协调(他们把不同的女人排为第一)，两个女人的优先表同样完全互相协调. 但是男人的优先表与女人的优先表完全冲突.

在第二个例子里，存在两个不同的稳定匹配. 由(m,w)和(m',w')对组成的匹配是稳定的，因为两个男人已是最满意了，因此没有人想离开他们匹配的伴侣. 而由(m',w)和(m,w')对组成的匹配也是稳定的，这是由于两个女人也是最满意这一互补的理由. 这是我们向下进行时要记住的一个要点——对于一个实例可能有多于一个的稳定匹配.

1.1.2 设计算法

我们现在来证明对于每组男人和女人中的优先表，存在一个稳定匹配. 进一步，我们证明这一点的方法也将回答上面问到的第二个问题：我们将给出一个有效的算法，它以这些优先表为输入并且构造出一个稳定匹配.

让我们考虑推动这个算法的某些基本思想.

- 初始，每个人都是未婚的. 假设一个未婚的男人 m 选择了他的优先表上排名最高的女人 w，并且向她求婚. 我们能够立刻声明(m,w)将是最后的稳定匹配中的一对吗？不一定：因为在将来的某个时候，女人 w 偏爱的男人 m'可能向她求婚. 另一方面，对 w 来说，立刻拒绝 m 可能是危险的；她可能还没有接收到来自她的优先表上排名像 m 那样高的某个人的求婚. 于是，一种自然的想法是使这个(m,w)对进入一种中间状态——约会.
- 假设我们现在处在某种状态，某些男人和女人是自由的——没有约会——且某些是有约会的. 下一步看起来可能是这样. 任意一个自由的男人 m 选择他还没有求过婚的最高排名的女人 w，且向她求婚. 如果 w 也是自由的，那么 m 和 w 就成为约会状态. 否则，w 已经在与某个其他的男人 m'约会. 在这种情况下，她来决定 m 和 m'哪个人在她的优先表中的排名更高；排名更高的男人变成与 w 约会而另一个人变成自由的.
- 最后，当没有一个人是自由的，算法将结束；此刻所有的约会将被宣告为最后的结果，且将返回最终的完美匹配.

这里是 Gale-Shapley 算法的具体描述，用图 1.2 描写了算法的一种状态.

初始所有的 $m \in M$ 和 $w \in W$ 都是自由的
While 存在男人 m 是自由的且还没对每个女人都求过婚
  选择这样一个男人 m
  令 w 是 m 的优先表中 m 还没求过婚的最高排名的女人
  If w 是自由的 then
    (m, w) 变成约会状态
  Else w 当前与 m' 约会
    If w 是更偏爱 m' 而不爱 m then
      m 保持自由
    Else w 更偏爱 m 而不爱 m',
      (m, w) 变成约会状态
      m' 变成自由
    Endif
  Endif
Endwhile
输出已约会对的集合 S.

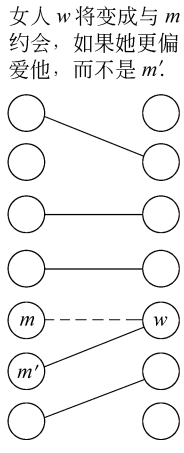

图 1.2 当自由男人 m 向女人 w 求婚时，G-S 算法的一个中间状态

一个令人感兴趣的事情是，尽管 G-S 算法叙述起来相当简单，但是也不是立刻就能看出它返回一个稳定匹配，或者甚至一个完美匹配. 我们现在就通过一系列中间的事实来证明这一点.

1.1.3 算法的分析

首先考虑女人 w 在算法执行时的看法. 此刻暂时还没有一个人向她求过婚，她是自由的. 然后，男人 m 可能向她求婚，并且她变成约会状态，随着时间继续，她可能收到另外一些求婚，且只接受比她的伴侣的排名更高的那些. 于是，我们发现下述事实：

命题 1.1 *w 从接受对她的第一次求婚开始保持约会状态，且她正在约会的一系列伴侣(依照她的优先表)变得越来越好.*

男人 m 在算法执行中的看法是相当不一样. 直到向他的表中最高排名的女人求婚之前他都是自由的；在这次求婚时，他可能或者不可能变成约会状态. 随着时间继续，他可能在自由和约会状态之间交替，但是保持下面的性质.

命题 1.2 *m 求过婚的一系列女人(依照他的优先表)变得越来越差.*

现在我们证明这个算法终止，并且给出终止所需要的最大迭代次数的界.

定理 1.3 *G-S 算法在至多 n^2 次 While 循环的迭代之后终止.*

证 正如我们这里打算做的，一个给出算法运行时间上界的有用策略是找出进展的度量标准. 即我们找出某种精确的方法说这个算法执行的每一步使得它离终止更接近.

在当前这个算法的情况下，每次迭代由某个男人向一个以前没有求过婚的女人(唯一的一次)求婚构成. 于是，如果令 $P(t)$ 表示到迭代 t 结束 m 已经向 w 求过婚的那些(m,w)对的集合，我们看到，对所有的 t，$P(t+1)$的大小严格大于 $P(t)$的大小. 但是总计只存在 n^2 个可能的男人和女人的对，因此 $P(\cdot)$的值在算法的整个进程中至多可能增加 n^2 次. 从而得到至多可能存在 n^2 次迭代. ■

关于前面的事实和它的证明有两点值得注意. 第一点，根据确定的优先表，算法的某些执行有可能包含着接近于 n^2 次的迭代，因此这个分析与最佳可能的结果差距不大. 第二点，有许多量不像算法的进展度量表现得那样好，因为它们在每次迭代中不是严格增加的. 例如，单个自由人的数目，可能从一次迭代到下一次迭代保持常数，同样可能的还有参与约会的对数. 于是，这些量不能按照前一段的方式直接用来给出最大可能迭代次数的上界.

让我们确定在算法终止时得到的集合 S 事实上是一个完美匹配. 为什么这一点不是显而易见的呢？本质上，我们必须证明没有男人可以在他的优先表结束前“落空”；While 循环终止的唯一方式就是不存在自由的男人. 在这种情况下，约会的那些对的集合的确是一个完美匹配.

因此我们必须证明的主要部分就是下面的命题.

命题 1.4 *如果 m 在算法执行的某点是自由的，那么存在一个他还没有向她求过婚的女人.*

证 假设存在一个点，此刻 m 是自由的，但是已经向每个女人都求过婚了. 那么根据命题 1.1，n 个女人中的每个人在这个时间点上是约会状态. 由于约会的对的集合构成一个匹配，在这个时间点上一定也有 n 个约会的男人. 但是，总共只有 n 个男人，且 m 不在约会，因此这是矛盾. ■

命题 1.5 *终止时返回的集合 S 是一个完美匹配.*

证 约会的对的集合总是构成一个匹配. 我们假设算法终止时有自由的男人 m. 在终止时一定是这种情况，m 已经向每个女人求过婚，否则 While 循环不会结束. 但是，这与命题 1.4 矛盾，这个命题说不可能存在一个自由的且向每个女人都求过婚的男人. ■

最后，我们证明算法的主要性质——即它导致一个稳定匹配.

定理 1.6 *考虑 GS 算法的一次执行，它返回一个对的集合 S. 集合 S 是一个稳定匹配.*

证 我们在命题 1.5 已经看到 S 是一个完美匹配，于是，为证明 S 是稳定匹配，我们将假设存在一个相对于 S 的不稳定因素并且得出一个矛盾. 正如早些时候定义的，这样一个不稳定因素涉及 S 中的两个对，即(m,w)和(m',w')，它们具有性质：

m 偏爱 w' 而不爱 w，且

w'偏爱 m 而不爱 m'.

在算法的执行中产生 S，根据定义，m 的最后一次求婚是向 w 求婚. 现在我们问：在这次执行中某个更早的时间点，m 向 w'求过婚吗？如果他没有，那么在 m 的优先表上 w 一定比 w'出现得更高，这与我们假设 m 更偏爱 w'而不爱 w 矛盾. 如果他求过婚，那么他被 w'拒绝是由于 w'更偏爱某个其他的男人 m''，而不爱 m. m'是 w'最后的伴侣，因此或者 $m''=m'$，或者根据命题 1.1，w'更偏爱她最后的伴侣 m'而不爱 m''. 每种情况都与我们假设 w'更偏爱 m 而不爱 m'矛盾.

从而证明 S 是稳定匹配. ■

1.1.4 推广

由定义稳定匹配的概念开始，我们刚证明了 G-S 算法事实上构造了一个稳定匹配. 现在考虑某些深一层的关于 G-S 算法的行为以及它与不同稳定匹配性质的关系问题.

作为开始，我们先回顾早些时候看过的一个存在多个稳定匹配的例子. 把要点重复一下，这个例子的优先表如下：

m 更偏爱 w 而不爱 w'.

m'更偏爱 w'而不爱 w.

w 更偏爱 m'而不爱 m.

w'更偏爱 m 而不爱 m'.

现在，在 G-S 算法的任何一次执行中，m 将变成与 w 约会，m'将变成与 w'约会（也许按照其他的顺序），并且情况将在那里停止不变. 于是，由对（m'，w）和（m，w'）组成的另一个稳定匹配通过这种男人求婚的 G-S 算法的一次执行是不可能达到的. 另一方面，如果我们按照女人求婚运行算法，它是可能达到的. 在双方都超过两个人的较大的例子中，我们甚至可能得到更大的稳定匹配的集合，其中许多稳定匹配不可能由任何自然的算法得到.

这个例子说明在 G-S 算法中的某种偏爱男人的“不公平性”. 如果男人的优先表完全协调（他们全都列出不同的女人作为他们的第一选择），那么在 G-S 算法的所有运行中所有男人最终都与他们的第一选择匹配，而与女人的优先表无关. 如果女人的优先表完全与男人的优先表冲突（正像在这个例子中的情况），那么得到的稳定匹配对女人是最差的. 因此，这个简单的优先表集合简洁地概括了某人将注定以不幸结束的世界：如果男人求婚，女人就是不幸的；如果女人求婚，那么男人就是不幸的.

现在让我们更详细地分析 G-S 算法，试着理解这种“不公平”现象是多么普遍.

作为开始，我们的例子强化下述观点：G-S 算法实际上是有特定条件的：只要存在一个自由的男人，就允许我们选出任一自由的男人产生下一次求婚. 不同的选择指定了算法的不同的执行；要小心，这就是为什么我们在命题 1.6 说“考虑 G-S 算法的一次执行，它返回一个对的集合 S”而不是说“考虑由 G-S 算法返回的集合 S”.

于是，我们遇到另一个非常自然的问题：G-S 算法所有的执行得到同样的匹配吗？这是在计算机科学的许多背景下产生的一类问题：我们有一个异步运行的算法，具有不同的独立分量实现那些可以用复杂方式交叉的动作，我们想知道这种异步操作在最终结果会引发多少可变性. 考虑一类非常不同的例子，独立分量可能不是男人和女人，而是机翼活动部

分的电子部件;它们行动的异步影响可能就很大了.

在当前的背景下,将会看到对我们问题的回答是异乎寻常的清楚:G-S 算法的所有执行将得到同样的匹配.我们现在就证明这个结果.

所有的执行得到同样的匹配 有几种可能的方式证明这样一个论述.其中许多将导致相当复杂的论证.原来对我们最简单、最有益的方法将是给出所得匹配的唯一特征,然后证明所有的执行将导致具有这个特征的匹配.

这个特征是什么?我们将证明每个男人将以拥有具体意义下的"最佳伴侣"来结束(回顾前面,如果所有的男人都偏爱不同的女人,这是真的).首先,如果存在一个稳定匹配包含了(m,w)对,我们就说女人 w 是男人 m 的有效伴侣.如果 w 是 m 的有效伴侣,且没有别的在 m 的排名中比 w 更高的女人是他的有效伴侣,那么 w 就是 m 的最佳有效伴侣.我们用 $best(m)$ 表示 m 的最佳有效伴侣.

现在让我们用 S^* 表示对的集合 $\{(m,best(m)):m\in M\}$.我们将证明下面的事实.

定理 1.7 G-S 算法的每次执行都得到集合 S^*.

这个陈述在几个层面上是令人惊奇的.首先,先不管是唯一的稳定匹配,正如被定义的,根本没有理由相信 S^* 是一个匹配.为什么两个男人有相同最佳有效伴侣的情况最终不会发生?第二点,这个结果证明 G-S 算法对每个男人同时给出了最佳可能的结果;不存在稳定匹配使得其中任何男人可以有希望做得更好.最后,通过证明在 G-S 算法中求婚的顺序完全不影响最终的结果来回答我们上面的问题.

不管所有这些,这个证明没有这么难.

证 使用反证法.让我们假设 G-S 算法的某次执行 ε 得到了匹配 S,其中某个男人与一个不是他最佳有效伴侣的女人配对.因为男人求婚依照优先的递减次序,这就意味着在算法的 ε 执行期间,某个男人被一个有效伴侣拒绝过.于是考虑算法的 ε 执行期间某个男人,比如说 m,被一个有效伴侣 w 拒绝的第一个时刻.此外,因为男人求婚依照优先的递减次序,这是第一次发生这样的拒绝,w 一定是 m 的最佳有效伴侣 $best(m)$.

m 被 w 拒绝已经发生了,或者可能由于 m 提出求婚而 w 对已有约会更好感而被拒绝,或者 w 停止她与 m 的约会而倾心于一个更好的求婚者.但是,此刻无论哪种方式 w 都构成或继续与她更青睐的一个男人 m' 的约会.

因为 w 是 m 的有效伴侣,存在一个包含 (m,w) 对的稳定匹配 S'.现在我们要问:在这个匹配中 m' 与谁配对?假设是女人 $w'\neq w$.

因为 m 被 w 拒绝是在 ε 执行中一个男人被一个有效伴侣的第一次拒绝,在 ε 执行中 m' 开始与 w 约会的这一点,m' 一定还没有被任何有效伴侣拒绝过.由于他求婚依照优先表的递减顺序进行,并且由于 w' 显然是 m' 的一个有效伴侣,一定是 m' 更偏爱 w 而不爱 w'.但是我们已经看到 w 更偏爱 m' 而不爱 m,因为在 ε 执行中她拒绝了 m 而倾心于 m'.由于 $(m',w)\notin S'$,从而得到 (m',w) 是 S' 中的一个不稳定因素.

这与我们声称 S' 是稳定的相矛盾,因此与我们的初始假设冲突. ■

对男人而言,G-S 算法是理想的.不幸的是对女人就不能这样说了.如果存在一个稳定匹配包含 (m,w) 对,就说 m 是对女人 w 的有效伴侣.如果 m 是 w 的有效伴侣,且没有 w 排

名中比 m 更低的男人是她的有效伴侣，我们说 m 是 w 的最差的有效伴侣.

命题 1.8 在稳定匹配 S^* 中每个女人与她最差的有效伴侣配对.

证 假设在 S^* 中存在(m,w)对使得 m 不是 w 的最差的有效伴侣.那么存在稳定匹配 S'，其中 w 与比 m 更不喜欢的男人 m'配对.在 S'中，m 与一个女人 $w'\neq w$ 配对；因为 w 是 m 的最佳有效伴侣，w'是 m 的有效伴侣，我们看到 m 宁愿要 w 而不是 w'.

但是由此得到(m,w)是 S'中的一个不稳定因素，与声称 S'是稳定的矛盾，因此与我们的初始假设矛盾. ■

于是我们发现，上述男人优先表与女人优先表冲突的简单例子，暗示了一种非常一般的现象：对于任何输入，在 G-S 算法中求婚的一方(根据他们的优先表)以最佳可能的稳定匹配结束，而没有求婚的另一方相对地却以最差可能的稳定匹配结束.

1.2 五个典型问题

稳定匹配问题为我们提供了一个丰富的算法设计过程的例子.对于许多问题，这个过程涉及了一些重要的步骤：以充分数学化的精确性简洁地表达这个问题，以便我们可以问一个具体的问题并且开始思考求解它的算法；针对这个问题设计一个算法；通过证明它是正确的并且给出一个运行时间的界以确定算法的效率来分析算法.

在实践中是通过一些基础设计技术的帮助来完成这个高级策略的，这些基础设计技术对估计问题固有的复杂性并且形式化表达求解它的算法方面是非常有用的.由于在任何领域，逐渐熟悉这些设计技术是一个渐进的过程；但是一个人只要有经验就可以开始按照属于某种可以确认的类型来识别问题，并且意识到问题陈述中的微妙变化对它的计算效率可能有着巨大的影响.

那么，为了开始这个讨论，有用的是要挑选一些在算法研究中我们将遇到的典型的重要问题：清晰简洁表示的问题，在一般化的层次上彼此类似，但在它们的难点和使用的方法类型上有着大的区别.前三个问题用一系列逐渐复杂的算法技术可以被有效地求解.第四个问题被认为不可能用任何有效算法求解，作为这种例子，它标出了我们讨论中的一个重要的转折点；第五个问题暗示了一个被认为甚至是更难的问题类.

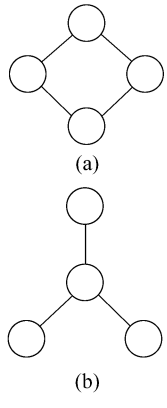

图 1.3 (a)和(b)都描述了四个结点的图

这些问题是自包含的，并且全都来自于计算应用.虽然使用图的术语将有助于讨论其中某些问题.而图在早期计算机科学课程中是一个共同的话题，我们将在第 3 章以适当的深度引入图的概念.由于它们强大的表达能力，我们也将在全书中推广使用它们.至于这里的讨论，只不过把图 G 看做一组个体之间二元关系的一种编码方式就足够了.于是，G 由一对集合

(V,E)——结点集合 V 和边的集合 E 构成，每条边与两个结点“相交”. 于是我们把一条边 $e\in E$ 表示成 V 的两个元素的子集：$e=\{u,v\}$，$u,v\in V$，其中称 u 和 v 为 e 的端点. 我们将图画成图 1.3 中典型的图形，每个结点画成一个小圆圈，每条边画成一条与其两个端点相交的线段.

现在让我们回到五个典型问题的讨论.

1.2.1 区间调度

考虑下述非常简单的调度问题. 你有某种资源——它可能是一个报告厅、一台超级计算机或者一台电子显微镜——许多人需要在某个时间段使用这个资源. 一个需求采用这种形式：我能从时刻 s 开始直到时刻 f 预订这个资源吗？我们假设每个时刻至多一个人使用这个资源. 一个调度员想接受这些需求的一个子集，而拒绝所有其他需求，使得被接受的需求时间上不重叠. 目标就是使得被接受的需求数目最大.

更形式的说法是，有标记为 $1,2,\cdots,n$ 的 n 个需求，每个需求 i 由开始时间 s_i 和结束时间 f_i 指定. 当然，对所有的 i，我们有 $s_i<f_i$. 如果两个需求 i 和 j 所要求的区间是不重叠的，这就是说，或者需求 i 比需求 j 处在一个更早的时间区间($f_i\leqslant s_j$)，或者需求 i 比需求 j 处在一个更迟的时间区间($f_j\leqslant s_i$)，那么它们是相容的. 更一般地说，如果每对需求 $i,j\in A$，$i\neq j$，都是相容的，那么需求的子集 A 是相容的. 目标就是选择一个最大的相容的需求子集.

图 1.4 给出了区间调度问题的一个实例. 注意只存在一个大小为 4 的相容集合，并且这就是最大的相容集合.

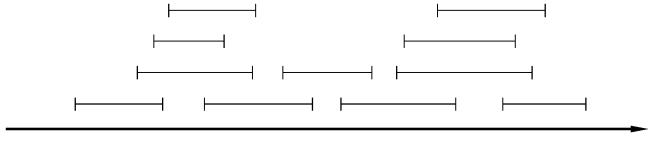

图 1.4 区间调度问题的一个实例

我们马上就可以看到这个问题可以用一个非常自然的算法求解，这个算法按照某种启发式的方法把一组需求排序，然后“贪心”地对需求进行一轮处理，选出最大的相容子集. 这是一类典型的贪心算法，我们将考虑把它用到不同的问题上——这种方法以一种近视的原则，一次处理一段输入，看上去似乎没有长远的打算. 当一个贪心算法被证明能够对问题的所有实例都得到最优解，它通常是相当的令人惊讶. 这样一个简单的方法可能是最优的，我们从这一事实一般地可以获悉有关这个基本问题结构的某些特征.

1.2.2 带权的区间调度

在**区间调度问题**中我们的目标是使得同时考虑的需求数目达到最大. 现在，更一般地假设每个需求区间 i 有一个相联系的值或者权 $v_i>0$；我们可以把它描述为：如果对第 i 个人的需求做出安排，我们从他或她那里挣到的钱数. 我们的目标将是找一个总价值最大的相容的区间子集.