第二章　马尔科夫决策过程

求解强化学习问题可以理解为如何最大化个体在与环境交互过程中获得的累积奖励。环境的动力学特征确定了个体在交互时的状态序列和即时奖励，环境的状态是构建环境动力学特征所需要的所有信息。当环境状态是完全可观测时，个体可以通过构建马尔科夫决策过程来描述整个强化学习问题。有时候环境状态并不是完全可观测的，此时个体可以结合自身对于环境的历史观测数据来构建一个近似的完全可观测环境的描述。从这个角度来说，几乎所有的强化学习问题都可以被认为或可以被转化为马尔科夫决策过程。正确理解马尔科夫决策过程中的一些概念和关系对于正确理解强化学习问题非常重要。

2.1　马尔科夫过程

在一个时序过程中，如果 $t+1$ 时刻的状态仅取决于 t 时刻的状态 S_t 而与 t 时刻之前的任何状态都无关时，则认为 t 时刻的状态 S_t 具有**马尔科夫性** (Markov property)。若过程中的每一个状态都具有马尔科夫性，则这个过程具备马尔科夫性。具备了马尔科夫性的随机过程称为**马尔科夫过程** (Markov process)，又称马尔科夫链 (Markov chain)。马尔科夫过程中的每一个状态 S_t 记录了过程历史上所有相关的信息，而且一旦 S_t 确定了，那么历史状态信息 $S_1 \dots S_{t-1}$ 对于确定 S_{t+1} 均不再重要，可有可无。

描述一个马尔科夫过程的核心是状态转移概率矩阵：

$$P_{ss'} = \mathbb{P}\left[S_{t+1} = s' | S_t = s\right] \tag{2.1}$$

公式 (2.1) 中的状态转移概率矩阵定义了从任意一个状态 s 到其所有后继状态 s' 的状态转

移概率：

$$P = \text{from} \begin{bmatrix} & \overset{\text{to}}{} & \\ P_{11} & \cdots & P_{1n} \\ \vdots & & \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \tag{2.2}$$

其中，矩阵 P 中每一行的数据表示从某一个状态到所有 n 个状态的转移概率值。每一行的这些值加起来的和应该为 1。

通常使用一个元组 $\langle S, P \rangle$ 来描述马尔科夫过程，其中 S 是有限数量的状态集，P 是状态转移概率矩阵。

图 2.1 描述了一个假想的学生学习一门课程的马尔科夫过程。在这个随机过程中，学生需要顺利完成三节课并且通过最终的考试来完成这门课程的学习。当学生处在第一节课中时，会有 50% 的几率拿起手机浏览社交软件信息，另有 50% 的几率完成该节课的学习进入第二节课。一旦学生在第一节课中浏览手机社交软件信息，则有 90% 的可能性继续沉迷于浏览，而仅有 10% 的几率放下手机重新听讲第一节课。学生处在第二节课的时有 80% 的几率听完第二节课顺利进入到第三节课的学习中，也有 20% 的几率因课程内容枯燥或难度较大而休息或者退出。学生在学习第三节课内容后，有 60% 的几率通过考试继而 100% 的进入休息状态，也有 40% 的几率因为过于兴奋而出去娱乐泡吧，随后可能因为忘掉了不少学到的东西而分别以 20%,40% 和 50% 的概率需要重新返回第一、二、三节课中学习。

上图中，我们使用内有文字的空心圆圈来描述学生可能所处的某一个状态。这些状态有：第一节课（C1）、第二节课（C2）、第三节课（C3）、泡吧中（Pub）、通过考试（Pass）、浏览手机（FB）、以及休息退出（Sleep）共 7 个状态，其中最后一个状态是终止状态，意味着学生一旦进入该状态则永久保持在该状态，或者说该状态的下一个状态将 100% 还是该状态。连接状态的箭头表示状态转移过程，箭头附近的数字表明着发生箭头所示方向状态转移的概率。

假设学生现处在状态“第一节课（C1）”中，我们按照马尔科夫过程给出的状态转移概率可以得到若干学生随后的状态转化序列。例如下面的这 4 个序列都是可能存在的状态转化序列：

- C1 - C2 - C3 - Pass - Sleep
- C1 - FB - FB - C1 - C2 - Sleep
- C1 - C2 - C3 - Pub - C2 - C3 - Pass - Sleep
- C1 - FB - FB - C1 - C2 - C3 - Pub - C1 - FB - FB - FB - C1 - C2 - C3 - Pub - C2 - Sleep

从符合马尔科夫过程给定的状态转移概率矩阵生成一个状态序列的过程称为**采样**（sample）。采样将得到一系列的状态转换过程，本书我们称为**状态序列** (episode)。当状态序列的最后一个

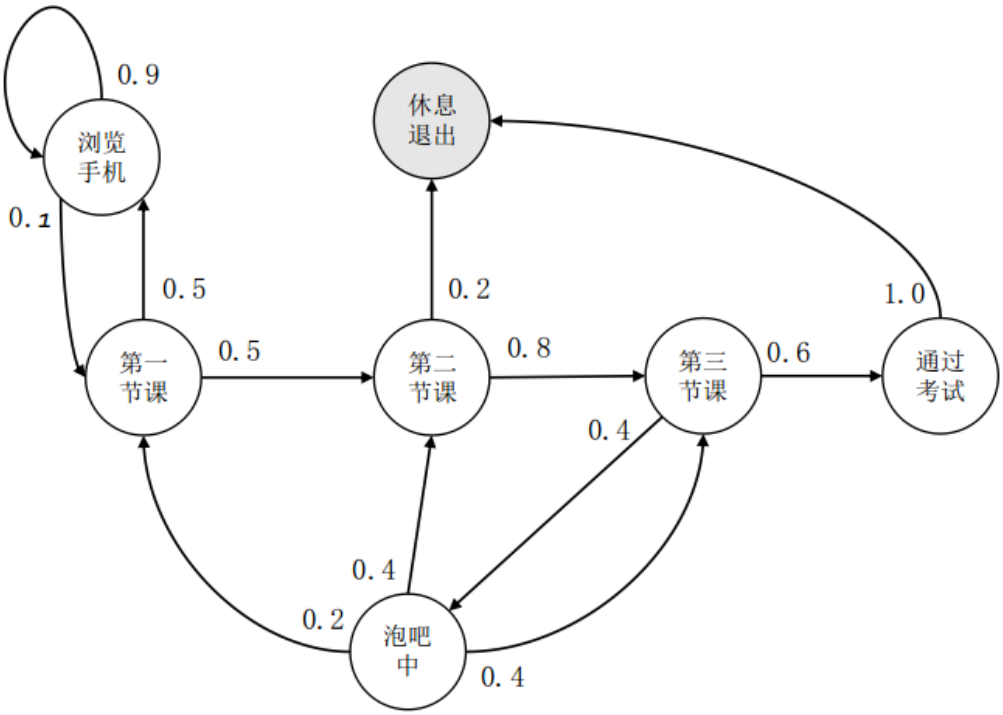


图 2.1: 学生马尔科夫过程

状态是终止状态时，该状态序列称为**完整的**状态序列 (complete episode)。本书中所指的状态序列大多数指的都是完整的状态序列。

2.2 马尔科夫奖励过程

马尔科夫过程只涉及到状态之间的转移概率，并未触及强化学习问题中伴随着状态转换的奖励反馈。如果把奖励考虑进马尔科夫过程，则成为**马尔科夫奖励过程** (Markov reward process, MRP)。它是由 $\langle S, P, R, \gamma \rangle$ 构成的一个元组，其中：

S 是一个有限状态集

P 是集合中状态转移概率矩阵： $P_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]]$

R 是一个奖励函数:$R_s = \mathbb{E}[R_{t+1} | S_t = s]$

γ 是一个衰减因子:$\gamma \in [0, 1]$

图 2.2 在图 2.1 的基础上在每一个状态旁增加了一个奖励值，表明到达该状态后（或离开该状态时）学生可以获得的奖励，如此构成了一个学生马尔科夫奖励过程。

学生到达每一个状态能获得多少奖励不是学生自己能决定的，而是由充当环境的授课老师

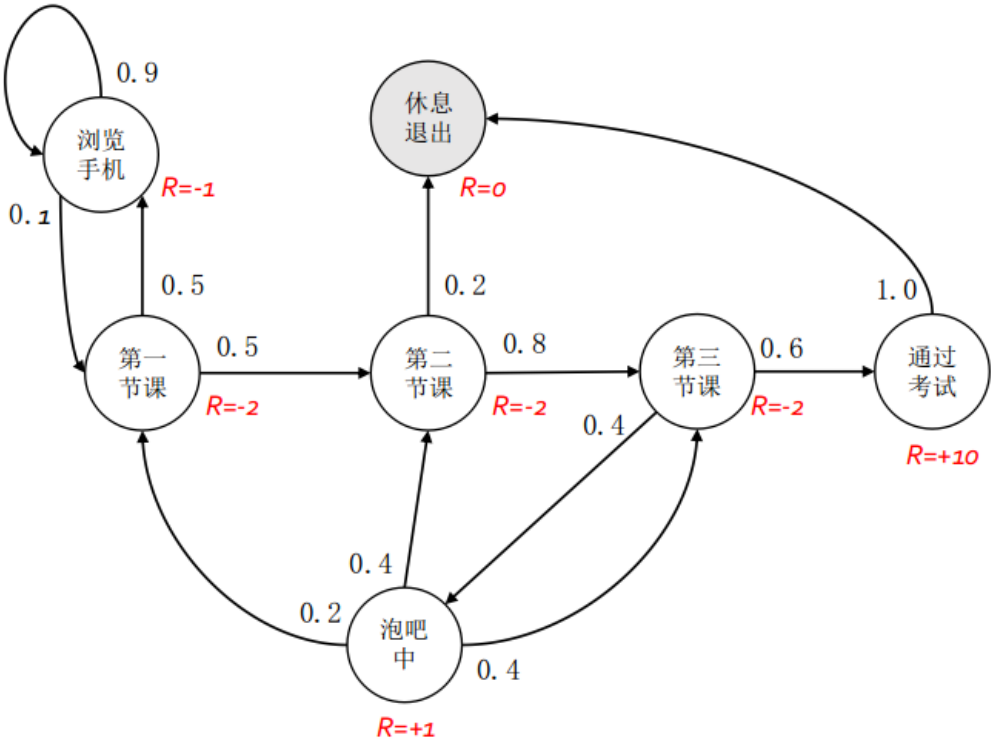


图 2.2: 学生马尔科夫奖励过程

或教务部门来确定，从强化学习的角度来说，奖励值由环境动力学确定。在该学生马尔科夫奖励过程中，授课老师的主要目的是希望学生能够尽早的通过考试，因而给了“考试通过”这个状态以正的较高的奖励（+10），而对于过程中的其它状态多数给的是负奖励。虽然设定状态“泡吧中”的奖励为 +1，但由于状态“泡吧中”随后的三个可能状态获得的奖励都低于 −1，因而可以认为授课教师并不十分赞在完成“第三节课”后出去泡吧。从学生的角度来说，学生的目标是在学习一门课程的过程中获得尽可能多的累积奖励，对于这个例子来说，也就是尽早的到达“考试通过”这个状态进而进入“睡觉休息”这个终止状态，完成一个完整的状态序列。在强化学习中，我们给这个累计奖励一个新的名称“收获”。

收获（return）是一个马尔科夫奖励过程中从某一个状态 S_t 开始采样直到终止状态时所有奖励的有衰减的之和。数学表达式如下：

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

收获有时也被翻译为回报。从式 (2.3) 中可以看出，收获是对应于状态序列中的某一时刻的状态的，计算从该状态开始直至结束还能获得的累积奖励。在一个状态序列中，不同时刻的状态

一般对应着不同的收获。从该式中我们还可以看出，收获并不是后续状态的奖励的直接相加，而是引入了一个取值范围在 $[0,1]$ 间的衰减系数 γ。引入该系数使得后续某一状态对当前状态收获的贡献要小于其奖励。这样设计从数学上可以避免在计算收获时因陷入循环而无法求解，从现实考虑也反映了远期奖励对于当前状态具有一定的不确定性，需要折扣计算。当 γ 取 0 时，表明某状态下的收获就是当前状态获得的奖励，不考虑后续状态，这属于“短视”行为，当 γ 取 1 时，表明将考虑所有的后续状态，属于有“长远眼光”的行为。求解实际问题时，模型构建者可根据实际问题的特点来设定 γ 值。

下文给出了学生马尔科夫过程中四个状态序列的开始状态“第一节课”的收获值的计算，选取 S_1 =“第一节课”，$\gamma = 0.5$。

- C1 C2 C3 Pass Sleep

$G_1 = -2 + (-2) * 1/2 + (-2) * 1/4 + 10 * 1/8 + 0 * 1/16 = -2.25$

- C1 FB FB C1 C2 Sleep

$G_1 = -2 + (-1) * 1/2 + (-1) * 1/4 + (-2) * 1/8 + (-2) * 1/16 + 0 * 1/32 = -3.125$

- C1 C2 C3 Pub C2 C3 Pass Sleep

$G_1 = -2 + (-2) * 1/2 + (-2) * 1/4 + 1 * 1/8 + (-2) * 1/16 + ... = -3.41$

- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

$G_1 = -2 + (-1) * 1/2 + (-1) * 1/4 + (-2) * 1/8 + (-2) * 1/16 + (-2) * 1/32 + ... = -3.20$

可以认为，收获间接给状态序列中的每一个状态设定了一个数据标签，反映了某状态的重要程度。由于收获的计算是基于一个状态序列的，从某状态开始，根据状态转移概率矩阵的定义，可能会采样生成多个不同的状态序列，而依据不同的状态序列得到的同一个状态的收获值一般不会相同。如何评价从不同状态序列计算得到的某个状态的收获呢？此外，一个状态还可能存在于一个状态序列的多个位置，可以参考学生马尔科夫过程的第四个状态序列中的状态“第一节课”，此时在一个状态序列下同一个状态可能会有不同的收获，如何理解这些不同收获的意义呢？不难看出收获对于描述一个状态的重要性还存在许多不方便的地方，为了准确描述一个状态的重要性，我们引入状态的“价值”这个概念。

价值（value**）** 是马尔科夫奖励过程中状态收获的期望。ta 数学表达式为：

$$v(s) = \mathbb{E}\left[G_t | S_t = s\right] \tag{2.4}$$

从式 (2.4) 可以看出，一个状态的价值是该状态的收获的期望，也就是说从该状态开始依据状态转移概率矩阵采样生成一系列的状态序列，对每一个状态序列计算该状态的收获，然后对该状态的所有收获计算平均值得到一个平均收获。当采样生成的状态序列越多，计算得到的平均收获就越接近该状态的价值，因而价值可以准确地反映某一状态的重要程度。

如果存在一个函数，给定一个状态能得到该状态对应的价值，那么该函数就被称为**价值函数**（value function）。价值函数建立了从状态到价值的映射。

从状态的价值的定义可以看出，得到每一个状态的价值，进而得到状态的价值函数对于求解强化学习问题是非常重要的。但通过计算收获的平均值来求解状态的价值不是一个可取的办法，因为一个马尔科夫过程针对一个状态可能可以产生无穷多个不同的状态序列。

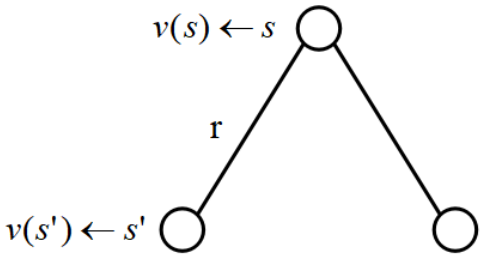
我们对价值函数中的收获按照其定义进行展开：

$$\begin{aligned} v(s) &= \mathbb{E}\left[G_t|S_t=s\right] \\ &= \mathbb{E}\left[R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots|S_t=s\right] \\ &= \mathbb{E}\left[R_{t+1}+\gamma(R_{t+2}+\gamma R_{t+3}+\ldots)|S_t=s\right] \\ &= \mathbb{E}\left[R_{t+1}+\gamma G_{t+1}|S_t=s\right] \\ &= \mathbb{E}\left[R_{t+1}+\gamma v(S_{t+1})|S_t=s\right] \end{aligned}$$

最终得到：

$$v(s)=\mathbb{E}\left[R_{t+1}+\gamma v(S_{t+1})|S_t=s\right] \tag{2.5}$$

上式中，根据马尔科夫奖励过程的定义，R_{t+1} 的期望就是其自身，因为每次离开同一个状态得到的奖励都是一个固定的值。而下一时刻状态价值的期望，可以根据下一时刻状态的概率分布得到。如果用 s' 表示 s 状态下一时刻任一可能的状态：



那么上述方程可以写成：

$$v(s)=R_s+\gamma\sum_{s'\in S}P_{ss'}v\left(s'\right) \tag{2.6}$$

上式称为马尔科夫奖励过程中的**贝尔曼方程**（Bellman equation），它提示一个状态的价值由该状态的奖励以及后续状态价值按概率分布求和按一定的衰减比例联合组成。

图 2.3 根据奖励值和衰减系数的设定给出了学生马尔科夫奖励过程中各状态的价值，并对状态“第三节课”的价值进行了验证演算。读者可以根据上述方程对其它状态的价值进行验证。

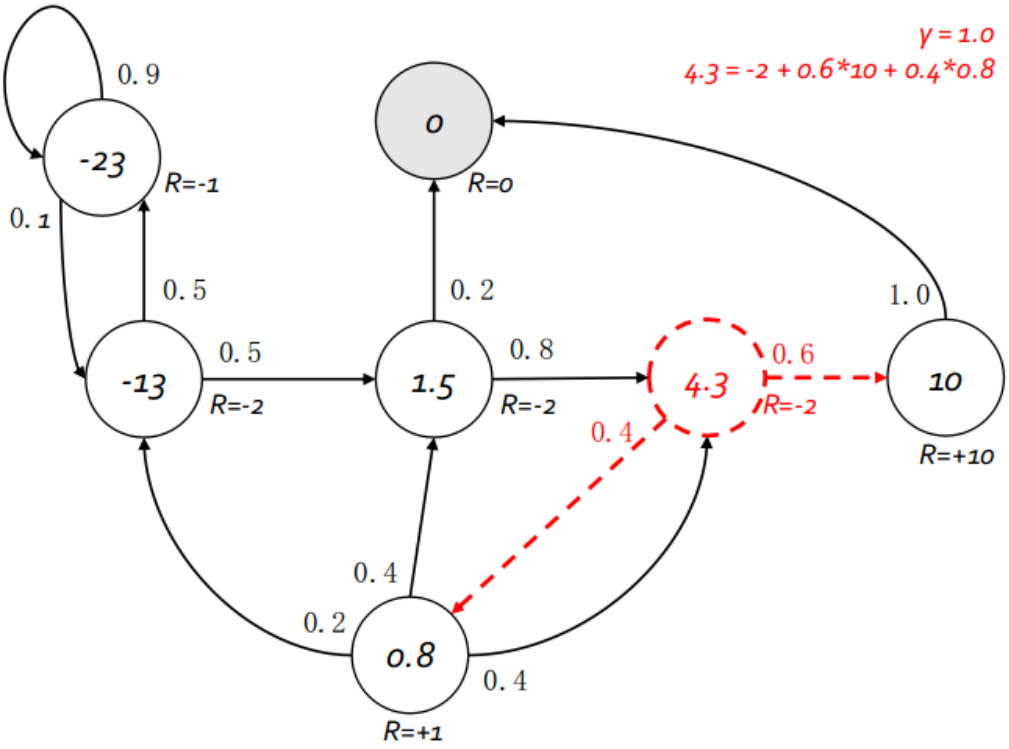


图 2.3: 学生马尔科夫奖励过程的价值

上述 Bellman 方程可以写成如下矩阵的形式：

$$v = R + \gamma P v \tag{2.7}$$

它表示：

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots P_{1n} \\ \vdots & \\ P_{n1} & \cdots P_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} \tag{2.8}$$

理论上，该方程可以直接求解：

$$\begin{aligned} v &= R + \gamma P v \\ (1 - \gamma P) v &= R \\ v &= (1 - \gamma P)^{-1} R \end{aligned}$$

计算这类问题的时间复杂度是 $O(n^3)$，其中 n 是状态的数量。这意味着，对于学生马尔科夫奖励过程这类状态数比较少的小规模问题，直接求解是可行的，但如果问题涉及到的状态数量增多，这种解法就不现实了。本书的后续章节会陆续介绍其它行之有效的求解方法。

在强化学习问题中，如果个体知道了每一个状态的价值，就可以通过比较后续状态价值的大小而得到自身努力的方向是那些拥有较高价值的状态，这样一步步朝着拥有最高价值的状态进行转换。但是从第一章的内容我们知道，个体需要采取一定的行为才能实现状态的转换，而状态转换又与环境动力学有关。很多时候个体期望自己的行为能够到达下一个价值较高的状态，但是它并不一定能顺利实现。这个时候个体更需要考虑在某一个状态下采取从所有可能的行为方案中选择哪个行为更有价值。要解释这个问题，需要引入马尔科夫决策过程、行为、策略等概念。

2.3 马尔科夫决策过程

马尔科夫奖励过程并不能直接用来指导解决强化学习问题，因为它不涉及到个体行为的选择，因此有必要引入马尔科夫决策过程。**马尔科夫决策过程**（Markov decision process, MDP）是由 $\langle S, A, P, R, \gamma \rangle$ 构成的一个元组，其中：

S 是一个有限状态集

A 是一个有限行为集

P 是集合中基于行为的状态转移概率矩阵：$P_{ss'}^{a} = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$

R 是基于状态和行为的奖励函数：$R_s^a = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$

γ 是一个衰减因子：$\gamma \in [0, 1]$

图 2.4 给出了学生马尔科夫决策过程的状态转化图。图中依然用空心圆圈表示状态，增加一类黑色实心圆圈表示个体的行为。根据马尔科夫决策过程的定义，奖励和状态转移概率均与行为直接相关，同一个状态下采取不同的行为得到的奖励是不一样的。此图还把 Pass 和 Sleep 状态合并成一个终止状态；另外当个体在状态“第三节课”后选择”泡吧”这个动作时，将被环境按照动力学特征分配到另外三个状态。请注意，学生马尔科夫决策过程示例虽然与之前的学生马尔科夫奖励过程示例有许多相同的状态，但两者还是有很大的差别，建议读者将这两个示例完全区分开来理解。

马尔科夫决策过程由于引入了行为，使得状态转移矩阵和奖励函数与之前的马尔科夫奖励过程有明显的差别。在马尔科夫决策过程中，个体有根据自身对当前状态的认识从行为集中选择一个行为的权利，而个体在选择某一个行为后其后续状态则由环境的动力学决定。个体在给定状态下从行为集中选择一个行为的依据则称为**策略** (policy)，用字母 π 表示。策略 π 是某一状态下

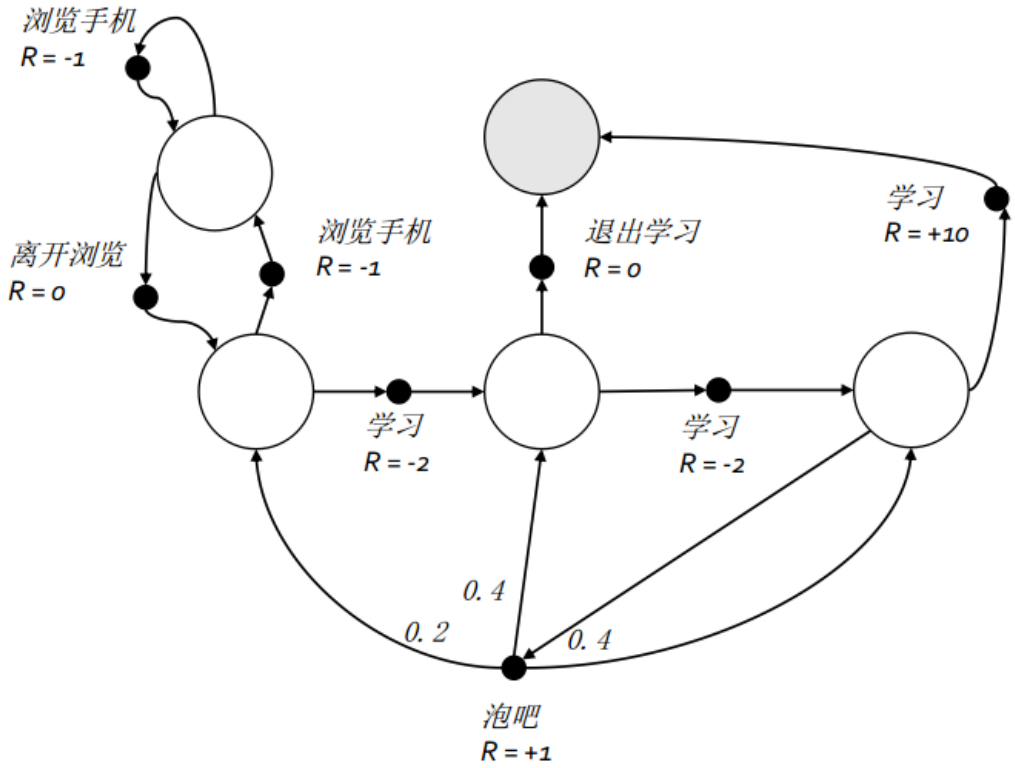


图 2.4: 学生马尔科夫决策过程

基于行为集合的一个概率分布：

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right] \tag{2.9}$$

在马尔科夫决策过程中，策略仅通过依靠当前状态就可以产生一个个体的行为，可以说策略仅与当前状态相关，而与历史状态无关。对于不同的状态，个体依据同一个策略也可能产生不同的行为；对于同一个状态，个体依据相同的策略也可能产生不同的行为。策略描述的是个体的行为产生的机制，是不随状态变化而变化的，被认为是静态的。

随机策略是一个很常用的策略，当个体使用随机策略时，个体在某一状态下选择的行为并不确定。借助随机策略，个体可以在同一状态下尝试不同的行为。

当给定一个马尔科夫决策过程：$M = \langle S, A, P, R, \gamma \rangle$ 和一个策略 π，那么状态序列 $S_1, S_2, \ldots$ 是一个符合马尔科夫过程 $\langle S, P_\pi \rangle$ 的采样。类似的，联合状态和奖励的序列 $S_1, R_2, S_2, R_3, \ldots$ 是

一个符合马尔科夫奖励过程 $\langle S, P_\pi, R_\pi, \gamma \rangle$ 的采样，并且在这个奖励过程中满足下面两个方程：

$$\begin{aligned} P^\pi_{s,s'} &= \sum_{a \in A} \pi(a|s) P^a_{ss'} \\ R^\pi_s &= \sum_{a \in A} \pi(a|s) R^a_s \end{aligned} \tag{2.10}$$

上述公式体现了马尔科夫决策过程中一个策略对应了一个马尔科夫过程和一个马尔科夫奖励过程。不难理解，同一个马尔科夫决策过程，不同的策略会产生不同的马尔科夫（奖励）过程，进而会有不同的状态价值函数。因此在马尔科夫决策过程中，有必要扩展先前定义的价值函数。

定义：价值函数 $v_\pi(s)$ 是在马尔科夫决策过程下基于策略 π 的状态价值函数，表示从状态 s 开始，遵循当前策略 π 时所获得的收获的期望：

$$v_\pi(s) = \mathbb{E}\left[G_t | S_t = s\right] \tag{2.11}$$

同样，由于引入了行为，为了描述同一状态下采取不同行为的价值，我们定义一个基于策略 π 的行为价值函数 $q_\pi(s, a)$，表示在遵循策略 π 时，对当前状态 s 执行某一具体行为 a 所能的到的收获的期望:

$$q_\pi(s, a) = \mathbb{E}\left[G_t | S_t = s, A_t = a\right] \tag{2.12}$$

行为价值（函数）绝大多数都是与某一状态相关的，所以准确的说应该是状态行为对价值(函数)。为了简洁，本书统一使用行为价值（函数）来表示状态行为对价值（函数），而状态价值（函数）或价值（函数）多用于表示单纯基于状态的价值（函数）。

定义了基于策略 π 的状态价值函数和行为价值函数后，依据贝尔曼方程，我们可以得到如下两个贝尔曼期望方程：

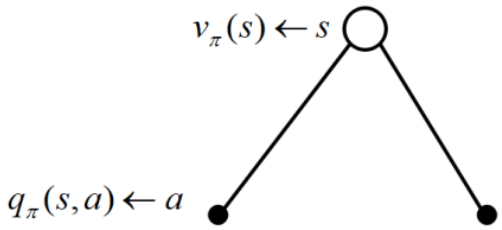
$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right] \tag{2.13}$$

$$q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right] \tag{2.14}$$

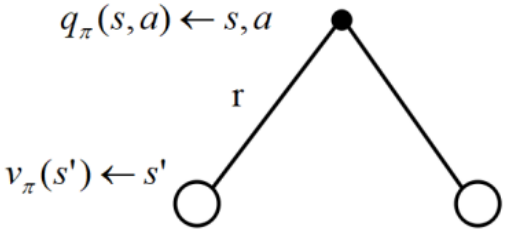
由于行为是连接马尔科夫决策过程中状态转换的桥梁，一个行为的价值与状态的价值关系紧密。具体表现为一个状态的价值可以用该状态下所有行为价值来表达：

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s, a) \tag{2.15}$$

类似的，一个行为的价值可以用该行为所能到达的后续状态的价值来表达：

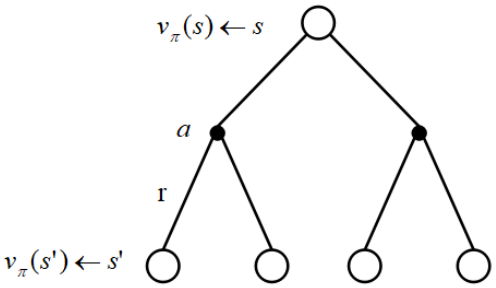


$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \tag{2.16}$$



如果把上二式组合起来，可以得到下面的结果：

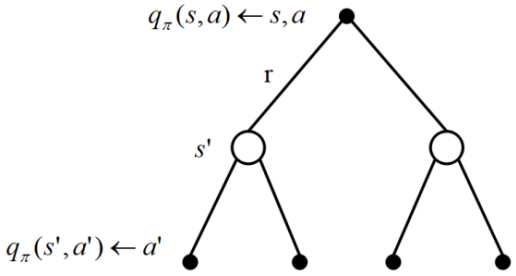
$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right) \tag{2.17}$$



或：

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s',a') \tag{2.18}$$

图 2.5 给出了一个给定策略下学生马尔科夫决策过程的价值函数。每一个状态下都有且仅有



2 个实质可发生的行为，我们的策略是两种行为以均等 (各 0.5) 的概率被选择执行，同时衰减因子 $\gamma = 1$。图中状态“第三节课”在该策略下的价值为 7.4，可以由公式 (2.17) 计算得出。

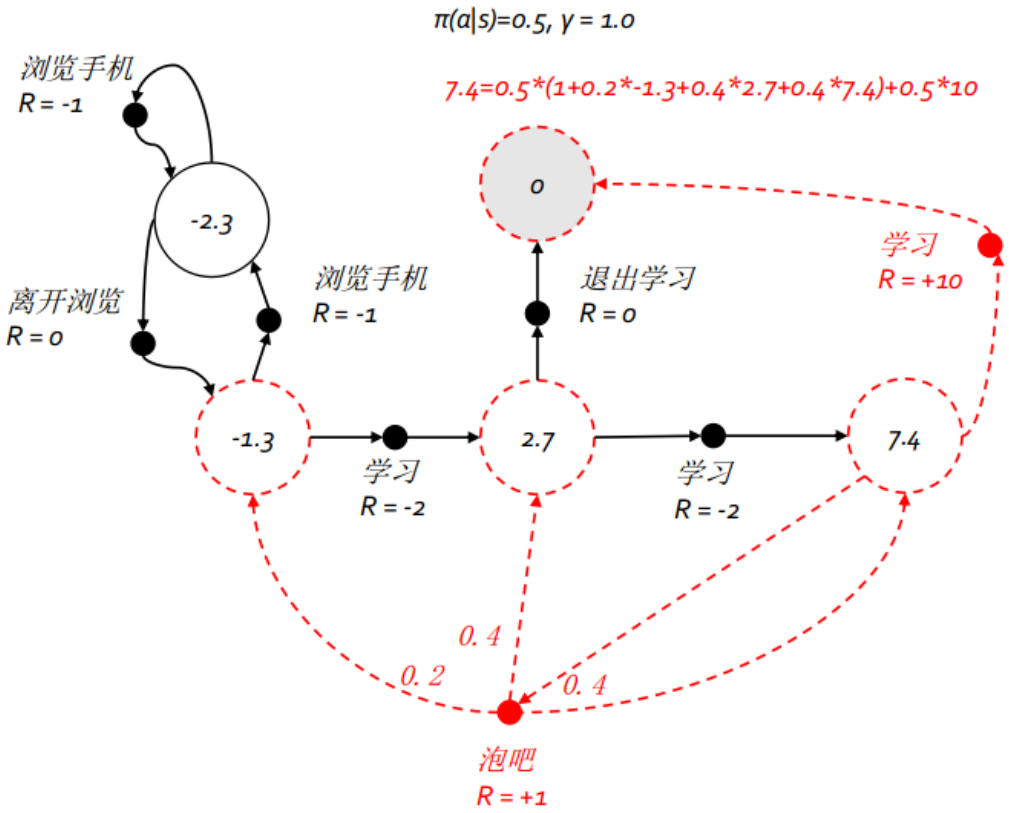


图 2.5: 基于给定策略的学生马尔科夫决策过程价值函数

给定策略 π 下的 MDP 问题可以通过其衍生的 MRP 和 P 来求解。不同的策略可以得到不同的价值函数，这些价值函数之间的差别有什么意义？是否存在一个基于某一策略的价值函数，在该策略下每一个状态的价值都比其它策略下该状态的价值高？如果存在如何找到这样的价值函数？这样的价值函数对应的策略又是什么策略？为了回答这些问题，我们需要引入最优策略、最优价值函数等概念。

解决强化学习问题意味着要寻找一个最优的策略让个体在与环境交互过程中获得始终比其它策略都要多的收获，这个最优策略用 π^* 表示。一旦找到这个最优策略 π^*，那么就意味着该强化学习问题得到了解决。寻找最优策略是一件比较困难的事情，但是可以通过比较两个不同策略的优劣来确定一个较好的策略。

定义：**最优状态价值函数**（optimal value function）是所有策略下产生的众多状态价值函数中的最大者：

$$v_* = \max_{\pi} v_\pi(s) \tag{2.19}$$

定义：**最优行为价值函数**（optimal action-value function）是所有策略下产生的众多行为价值函数中的最大者：

$$q_*(s,a) = \max_{\pi} q_\pi(s,a) \tag{2.20}$$

定义：**策略 π 优于 π'**$(\pi \geqslant \pi')$，如果对于有限状态集里的任意一个状态 s，不等式：$v_\pi(s) \geqslant v_{\pi'}(s)$ 成立。

存在如下的结论：对于任何马尔科夫决策过程，存在一个最优策略 π_* 优于或至少不差于所有其它策略。一个马尔科夫决策过程可能存在不止一个最优策略，但最优策略下的状态价值函数均等同于最优状态价值函数：$v_{\pi_*}(s) = v_*(s)$；最优策略下的行为价值函数均等同于最优行为价值函数：$q_{\pi_*}(s,a) = q_*(s,a)$。

最优策略可以通过最大化最优行为价值函数 $q_*(s,a)$ 来获得：

$$\pi_*(a|s) = \begin{cases} 1 & \text{如果 } a = \underset{a \in A}{argmax}\ q_*(s,a) \\ 0 & \text{其它情况} \end{cases} \tag{2.21}$$

该式表示，在最优行为价值函数已知时，在某一状态 s 下，对于行为集里的每一个行为 a 将对应一个最优行为价值 $q_*(s,a)$，最优策略 $\pi_*(a|s)$ 将给予所有最优行为价值中的最大值对应的行为以 100% 的概率，而其它行为被选择的概率则为 0，也就是说最优策略在面对每一个状态时将总是选择能够带来最大最优行为价值的行为。这同时意味着，一旦得到 $q_*(s,a)$，最优策略也就找到了。因此求解强化学习问题就转变为了求解最优行为价值函数问题。

拿学生马尔科夫决策过程来说，图 2.6 用粗虚箭头指出了最优策略，同时也对应了某个状态下的最优行为价值。

在学生马尔科夫决策过程例子中，各状态以及相应行为对应的最优价值可以通过回溯法递推计算得到。其中，状态 s 的最优价值可以由下面的贝尔曼最优方程得到：

$$v_*(s) = \max_{a} q_*(s,a) \tag{2.22}$$

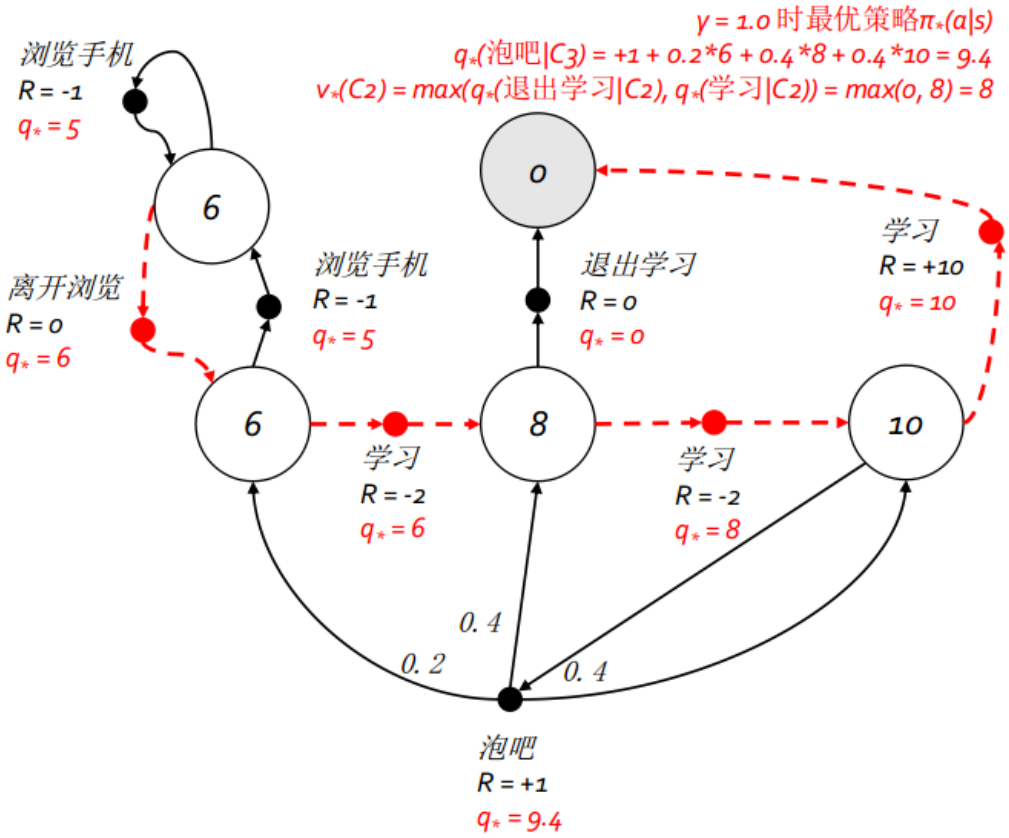
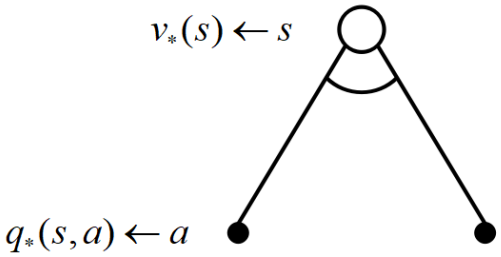


图 2.6: 学生马尔科夫决策过程最优策略

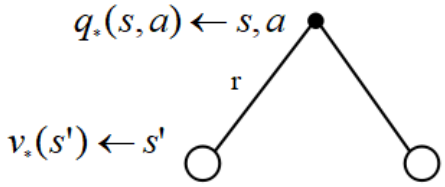


公式 (2.22) 表示：一个状态的最优价值是该状态下所有行为对应的最优行为价值的最大值。这不难理解，对于图 2.6 学生示例中的状态“第三节课”，可以选择的行为有“学习”和“泡吧”两个，其对应的最优行为价值分别为 10 和 9.4，因此状态“第三节课”的最优价值就是两者中最大的 10。

由于一个行为的奖励和后续状态并不由个体决定，因此在状态 s 时选择行为 a 的最优行为

价值将不能使用最大化某一可能的后续状态的价值来计算。它由下面的贝尔曼最优方程得到：

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \tag{2.23}$$

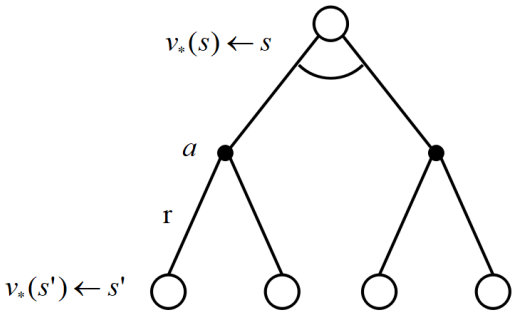


公式 (2.23) 表示：一个行为的最优价值由两部分组成，一部分是执行该行为后环境给予的确定的即时奖励，另一部分则由所有后续可能状态的最优状态价值按发生概率求和乘以衰减系数得到。

同样在学生示例中，考虑学生在“第三节课”时选择行为“泡吧”的最优行为价值时，先计入学生采取该行为后得到的即时奖励 $+1$，学生选择了该行为后，并不确定下一个状态是什么，环境根据一定的概率确定学生的后续状态是“第一节课”、“第二节课”还是“第三节课”。此时要计算“泡吧”的行为价值势必不能取这三个状态的最大值，而只能取期望值了，也就是按照进入各种可能状态的概率来估计总的最优价值，具体表现为 $6*0.2+8*0.4+10*0.4=8.4$，考虑到衰减系数 $\gamma=1$ 以及即时奖励为 $+1$，因此在第三节课后采取泡吧行为的最优行为价值 9.4。

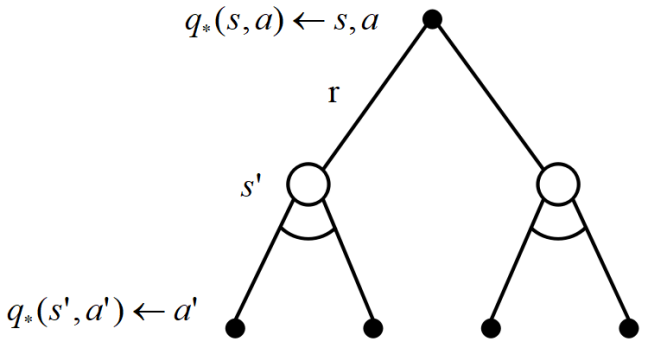
可以看出，某状态的最优价值等同于该状态下所有的行为价值中最大者，某一行为的最优行为价值可以由该行为可能进入的所有后续状态的最优状态价值来计算得到。如果把二者联系起来，那么一个状态的最优价值就可以通过其后续可能状态的最优价值计算得到：

$$v_*(s) = \max_a \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \right) \tag{2.24}$$



类似的，最优行为价值函数也可以由后续的最优行为价值函数来计算得到：

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \underset{a'}{max}\ q_*(s',a') \tag{2.25}$$



贝尔曼最优方程不是线性方程，无法直接求解，通常采用迭代法来求解，具体有价值迭代、策略迭代、Q 学习、Sarsa 学习等多种迭代方法，后续几章将陆续介绍。

2.4 编程实践——学生马尔科夫决策示例

本章的编程实践环节将以学生马尔科夫奖励和决策过程两个示例为核心，通过编写程序并观察程序运行结果来加深对马尔科夫奖励过程、马尔科夫决策过程、收获和价值、贝尔曼期望方程和贝尔曼最优方程等知识的理解。首先我们将讲解如何对一个马尔科夫奖励过程进行建模，随后利用我们建立的模型来计算马尔科夫奖励过程产生的状态序列中某一状态的收获，并通过直接求解线性方程的形式来获得马尔科夫奖励过程的价值函数。在对马尔科夫决策过程进行建模后，我们将编写各种方法实现贝尔曼期望方程给出的基于某一策略下的状态价值和行为价值的关系；同时验证在给出最优价值函数的情况下最优状态价值和最优行为价值之间的关系。

2.4.1 收获和价值的计算

图 2.2 给出了定义学生马尔科夫奖励过程所需要信息。其中状态集 S 有 7 个状态，状态转换概率如果用矩阵的形式则将是一个 $7*7$ 的矩阵，奖励函数则可以用 7 个标量表示，分别表示离开某一个状态得到的即时奖励值。在 Python 中，可以使用列表 (list) 或字典 (dict) 来表示集合数据。在本例中，为了方便计算，我们使用数字索引 $0-6$ 来表示 7 个状态，用一个列表嵌套列表的方式来表示状态转移概率矩阵，同时为了方便说明和理解，我们使用两个字典来建立数字对应的状态与其实际状态名的双向映射关系。实现这些功能，我们的代码可以这样写：

```python
import numpy as np # 需要用到numpy包

num_states = 7
# {"0": "C1", "1":"C2", "2":"C3", "3":"Pass", "4":"Pub", "5":"FB", "6":"Sleep"}
i_to_n = {} # 索引到状态名的字典
i_to_n["0"] = "C1"
i_to_n["1"] = "C2"
i_to_n["2"] = "C3"
i_to_n["3"] = "Pass"
i_to_n["4"] = "Pub"
i_to_n["5"] = "FB"
i_to_n["6"] = "Sleep"

n_to_i = {} # 状态名到索引的字典
for i, name in zip(i_to_n.keys(), i_to_n.values()):
    n_to_i[name] = int(i)

#     C1   C2   C3  Pass Pub   FB  Sleep
Pss = [ # 状态转移概率矩阵
    [ 0.0, 0.5, 0.0, 0.0, 0.0, 0.5, 0.0 ],
    [ 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.2 ],
    [ 0.0, 0.0, 0.0, 0.6, 0.4, 0.0, 0.0 ],
    [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 ],
    [ 0.2, 0.4, 0.4, 0.0, 0.0, 0.0, 0.0 ],
    [ 0.1, 0.0, 0.0, 0.0, 0.0, 0.9, 0.0 ],
    [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 ]
]

Pss = np.array(Pss)
# 奖励函数，分别于状态对应
rewards = [-2, -2, -2, 10, 1, -1, 0]
gamma = 0.5 # 衰减因子
```

至此，学生马尔科夫奖励过程就建立了。接下来我们要建立一个函数用来计算一状态序列中某一状态的收获。由于收获值是针对某一状态序列里的某一状态的，因此我们给传递给这个方法的参数需要有一个马尔科夫链、要计算的状态、以及衰减系数值。使用公式 (2.3) 来计算，代码如下：

```
def compute_return(start_index = 0,
                   chain = None,
                   gamma = 0.5) -> float:
    '''计算一个马尔科夫奖励过程中某状态的收获值
    Args:
        start_index 要计算的状态在链中的位置
        chain 要计算的马尔科夫过程
        gamma 衰减系数
    Returns:
        retrn 收获值
    '''
    retrn, power, gamma = 0.0, 0, gamma
    for i in range(start_index, len(chain)):
        retrn += np.power(gamma, power) * rewards[n_to_i[chain[i]]]
        power += 1
    return retrn
```

我们定义一下正文中的几条以 S_1 为起始状态的马尔科夫链，并使用刚才定义的方法来验证最后一条马尔科夫链起始状态的收获值：

```
chains =[
    ["C1", "C2", "C3", "Pass", "Sleep"],
    ["C1", "FB", "FB", "C1", "C2", "Sleep"],
    ["C1", "C2", "C3", "Pub", "C2", "C3", "Pass", "Sleep"],
    ["C1", "FB", "FB", "C1", "C2", "C3", "Pub", "C1", "FB",\
     "FB", "FB", "C1", "C2", "C3", "Pub", "C2", "Sleep"]
]

compute_return(0, chains[3], gamma = 0.5)
# 将输出：-3.196044921875
```

读者可以修改参数来验证其它收获值。

接下来我们将使用矩阵运算直接求解状态的价值，编写一个计算状态价值的方法如下：

```
def compute_value(Pss, rewards, gamma = 0.05):
    '''通过求解矩阵方程的形式直接计算状态的价值
    Args:
```

```
        P 状态转移概率矩阵 shape(7, 7)
        rewards 即时奖励 list
        gamma 衰减系数
    Return
        values 各状态的价值
    '''
    #assert(gamma >= 0 and gamma <= 1.0)
    # 将rewards转为numpy数组并修改为列向量的形式
    rewards = np.array(rewards).reshape((-1,1))
    # np.eye(7,7)为单位矩阵，inv方法为求矩阵的逆
    values = np.dot(np.linalg.inv(np.eye(7,7) - gamma * Pss), rewards)
    return values

values = compute_value(Pss, rewards, gamma = 0.99999)
print(values)
# 将输出：
# [[-12.54296219]
# [  1.4568013 ]
# [  4.32100594]
# [ 10.        ]
# [  0.80253065]
# [-22.54274676]
# [  0.        ]]
```

在利用矩阵的逆直接求解状态价值时，本示例的状态转移概率矩阵 Pss 的设置使得当 $\gamma = 1$ 时使得需要计算的矩阵的逆恰好不存在，因而我们给 γ 一个近似于 1 的值，计算得到的状态价值取一位小数后与图 2.3 显示的状态值相同。

2.4.2 验证贝尔曼方程

本节将使用学生马尔科夫决策过程的例子中的数据，如正文中图 2.4 所示，图中的状态数变成了 5 个，为了方便理解，我们把这五个状态分别命名为：‘浏览手机中’，‘第一节课’，‘第二节课’，‘第三节课’，‘休息中’；行为总数也是 5 个，但具体到某一状态则只有 2 个可能的行为，这 5 个行为分别命名为：‘浏览手机’，‘学习’，‘离开浏览’，‘泡吧’，‘退出学习’。与马尔科夫奖励过程不同，马尔科夫决策过程的状态转移概率与奖励函数均与行为相关。本例中多数状态下某行为将以 100% 的概率到达一个后续状态，但除外在状态‘第三节课中’选择‘泡吧’行为。在对该马尔科夫决策过程进行建模时，我们将使用字典这个数据结构来存放这些概率和奖励

数据。我已经事先写好了一些工具方法来操作字典，包括根据状态和行为来生成一个字典的键、显示和读取相关字典内容等，读者可以在本节的最后找到这些代码。我们要操作的字典除了记录了状态转移概率和奖励数据的两个字典外，还将设置一个记录状态价值的字典以及下文会提及的一个策略字典。具体如下：

```
# 导入工具函数：根据状态和行为生成操作相关字典的键，显示字典内容
from utils import str_key, display_dict
# 设置转移概率、奖励值以及读取它们方法
from utils import set_prob, set_reward, get_prob, get_reward
# 设置状态价值、策略概率以及读取它们的方法
from utils import set_value, set_pi, get_value, get_pi

# 构建学生马尔科夫决策过程
S = ['浏览手机中','第一节课','第二节课','第三节课','休息中']
A = ['浏览手机','学习','离开浏览','泡吧','退出学习']
R = {} # 奖励Rsa字典
P = {} # 状态转移概率Pss'a字典
gamma = 1.0 # 衰减因子
# 根据学生马尔科夫决策过程示例的数据设置状态转移概率和奖励，默认概率为1
set_prob(P, S[0], A[0], S[0]) # 浏览手机中 - 浏览手机 -> 浏览手机中
set_prob(P, S[0], A[2], S[1]) # 浏览手机中 - 离开浏览 -> 第一节课
set_prob(P, S[1], A[0], S[0]) # 第一节课 - 浏览手机 -> 浏览手机中
set_prob(P, S[1], A[1], S[2]) # 第一节课 - 学习 -> 第二节课
set_prob(P, S[2], A[1], S[3]) # 第二节课 - 学习 -> 第三节课
set_prob(P, S[2], A[4], S[4]) # 第二节课 - 退出学习 -> 退出休息
set_prob(P, S[3], A[1], S[4]) # 第三节课 - 学习 -> 退出休息
set_prob(P, S[3], A[3], S[1], p = 0.2) # 第三节课 - 泡吧 -> 第一节课
set_prob(P, S[3], A[3], S[2], p = 0.4) # 第三节课 - 泡吧 -> 第一节课
set_prob(P, S[3], A[3], S[3], p = 0.4) # 第三节课 - 泡吧 -> 第一节课

set_reward(R, S[0], A[0], -1) # 浏览手机中 - 浏览手机 -> -1
set_reward(R, S[0], A[2],  0) # 浏览手机中 - 离开浏览 -> 0
set_reward(R, S[1], A[0], -1) # 第一节课 - 浏览手机 -> -1
set_reward(R, S[1], A[1], -2) # 第一节课 - 学习 -> -2
set_reward(R, S[2], A[1], -2) # 第二节课 - 学习 -> -2
set_reward(R, S[2], A[4],  0) # 第二节课 - 退出学习 -> 0
set_reward(R, S[3], A[1], 10) # 第三节课 - 学习 -> 10
set_reward(R, S[3], A[3], +1) # 第三节课 - 泡吧 -> -1

```

```
MDP = (S, A, R, P, gamma)
```

至此，描述学生马尔科夫决策过程的模型就建立好了。当该 MDP 构建好之后，我们可以调用显示字典的方法来查看我们的设置是否正确：

```
print("----状态转移概率字典（矩阵）信息:----")
display_dict(P)
print("----奖励字典（函数）信息:----")
display_dict(R)
# 将输出如下结果:
# ----状态转移概率字典（矩阵）信息:----
# 第三节课_学习_休息中:  1.00
# 第三节课_泡吧_第三节课:  0.40
# 浏览手机中_浏览手机_浏览手机中:  1.00
# 第一节课_浏览手机_浏览手机中:  1.00
# 第三节课_泡吧_第二节课:  0.40
# 第三节课_泡吧_第一节课:  0.20
# 第一节课_学习_第二节课:  1.00
# 第二节课_学习_第三节课:  1.00
# 第二节课_退出学习_休息中:  1.00
# 浏览手机中_离开浏览_第一节课:  1.00
#
# ----奖励字典（函数）信息:----
# 第三节课_学习:  10.00
# 第一节课_学习:  -2.00
# 浏览手机中_离开浏览:  0.00
# 第二节课_学习:  -2.00
# 第二节课_退出学习:  0.00
# 第三节课_泡吧:  1.00
# 第一节课_浏览手机:  -1.00
# 浏览手机中_浏览手机:  -1.00
```

一个 MDP 中状态的价值是基于某一给定的策略的，要计算或验证该学生马尔科夫决策过程，我们需要先指定一个策略 π，这里考虑使用均一随机策略，也就是在某状态下所有可能的行为被选择的概率相等，对于每一个状态只有两种可能性为的该学生马尔科夫决策过程来说，每个可选行为的概率均为 0.5。初始条件下所有状态的价值均设为 0。与状态转移概率和奖励函数一

样，策略与价值也各一个字典来维护。在编写代码时，我们使用 Pi(或 pi) 来代替 π。该段代码如下：

```
# S = ['浏览手机中','第一节课','第二节课','第三节课','休息中']
# A = ['继续浏览','学习','离开浏览','泡吧','退出学习']
# 设置行为策略：pi(a|.) = 0.5
Pi = {}
set_pi(Pi, S[0], A[0], 0.5) # 浏览手机中 - 浏览手机
set_pi(Pi, S[0], A[2], 0.5) # 浏览手机中 - 离开浏览
set_pi(Pi, S[1], A[0], 0.5) # 第一节课 - 浏览手机
set_pi(Pi, S[1], A[1], 0.5) # 第一节课 - 学习
set_pi(Pi, S[2], A[1], 0.5) # 第二节课 - 学习
set_pi(Pi, S[2], A[4], 0.5) # 第二节课 - 退出学习
set_pi(Pi, S[3], A[1], 0.5) # 第三节课 - 学习
set_pi(Pi, S[3], A[3], 0.5) # 第三节课 - 泡吧

print("----状态转移概率字典（矩阵）信息:----")
display_dict(Pi)
# 初始时价值为空，访问时会返回0
print("----状态转移概率字典（矩阵）信息:----")
V = {}
display_dict(V)
# 将输出如下结果：
# ----状态转移概率字典（矩阵）信息:----
# 第二节课_学习： 0.50
#  第三节课_学习： 0.50
# 第二节课_退出学习： 0.50
# 浏览手机中_浏览手机： 0.50
# 第三节课_泡吧： 0.50
# 第一节课_浏览手机： 0.50
# 第一节课_学习： 0.50
# 浏览手机中_离开浏览： 0.50
#
# ----状态转移概率字典（矩阵）信息:----
#
```

下面我们将编写代码计算在给定 MDP 和状态价值函数 V 的条件下如何计算某一状态 s 时某行为 a 的价值 $q(s,a)$，依据是公式 (2.16)。该计算过程不涉及策略。代码如下：

```
def compute_q(MDP, V, s, a):
    '''根据给定的MDP, 价值函数V, 计算状态行为对s,a的价值qsa
    '''
    S, A, R, P, gamma = MDP
    q_sa = 0
    for s_prime in S:
        q_sa += get_prob(P, s,a,s_prime) * get_value(V, s_prime)
        q_sa = get_reward(R, s,a) + gamma * q_sa
    return q_sa
```

依据公式 (2.15), 我们可以编写如下的方法来计算给定策略 Pi 下如何计算某一状态的价值:

```
def compute_v(MDP, V, Pi, s):
    '''给定MDP下依据某一策略Pi和当前状态价值函数V计算某状态s的价值
    '''
    S, A, R, P, gamma = MDP
    v_s = 0
    for a in A:
        v_s += get_pi(Pi, s,a) * compute_q(MDP, V, s, a)
    return v_s
```

至此, 我们就可以验证学生马尔科夫决策过程中基于某一策略下各状态以及各状态行为对的价值了。不过在验证之前, 我们先给出在均一随机策略下该学生马尔科夫决策过程的最终状态函数。下面的两个方法将完成这个功能。在本章中, 对下面这段代码不作要求, 读者可以在学习了第三章内容后再来理解这段代码:

```
# 根据当前策略使用回溯法来更新状态价值, 本章不做要求
def update_V(MDP, V, Pi):
    '''给定一个MDP和一个策略, 更新该策略下的价值函数V
    '''
    S, _, _, _, _ = MDP
    V_prime = V.copy()
    for s in S:
        #set_value(V_prime, s, V_S(MDP, V_prime, Pi, s))
        V_prime[str_key(s)] = compute_v(MDP, V_prime, Pi, s)
    return V_prime
```

```

# 策略评估，得到该策略下最终的状态价值。本章不做要求
def policy_evaluate(MDP, V, Pi, n):
    '''使用n次迭代计算来评估一个MDP在给定策略Pi下的状态价值，初始时价值为V
    '''
    for i in range(n):
        V = update_V(MDP, V, Pi)
        #display_dict(V)
    return V

V = policy_evaluate(MDP, V, Pi, 100)
display_dict(V)
# 将输出如下结果：
# 第一节课： -1.31
# 第三节课： 7.38
# 浏览手机中： -2.31
# 第二节课： 2.69
# 休息中： 0.00
```

我们来计算一下在均一随机策略下，状态“第三节课”的最终价值，写入下面两行代码：

```
v = compute_v(MDP, V, Pi, "第三节课")
print("第三节课在当前策略下的最终价值为:{:.2f}".format(v))
# 将输出如下结果：
# 第三节课在当前策略下的最终价值为:7.38
```

可以看出该结果与图 2.5 所示的结果相同。读者可以修改代码验证其它状态在该策略下的最终价值。

不同的策略下得到个状态的最终价值并不一样，特别的，在最优策略下最优状态价值的计算将遵循公式 (2.22)，此时一个状态的价值将是在该状态下所有行为价值中的最大值。我们编写如下方法来实现计算最优策略下最优状态价值的功能：

```
def compute_v_from_max_q(MDP, V, s):
    '''根据一个状态的下所有可能的行为价值中最大一个来确定当前状态价值
    '''
    S, A, R, P, gamma = MDP
```

```
    v_s = -float('inf')
    for a in A:
        qsa = compute_q(MDP, V, s, a)
        if qsa >= v_s:
            v_s = qsa
    return v_s
```

下面这段代码实现了在给定 MDP 下得到最优策略以及对应的最优状态价值的一种办法，同样这段代码可以在学习了第三章内容之后再来仔细理解：

```
def update_V_without_pi(MDP, V):
    '''在不依赖策略的情况下直接通过后续状态的价值来更新状态价值
    '''
    S, _, _, _, _ = MDP
    V_prime = V.copy()
    for s in S:
        #set_value(V_prime, s, compute_v_from_max_q(MDP, V_prime, s))
        V_prime[str_key(s)] = compute_v_from_max_q(MDP, V_prime, s)
    return V_prime

# 价值迭代，本章不作要求
def value_iterate(MDP, V, n):
    '''价值迭代
    '''
    for i in range(n):
        V = update_V_without_pi(MDP, V)
    return V

V = {}
# 通过价值迭代得到最优状态价值及
V_star = value_iterate(MDP, V, 4)
display_dict(V_star)
# 将输出如下结果：
# 第一节课： 6.00
# 第三节课： 10.00
# 浏览手机中： 6.00
# 第二节课： 8.00
# 休息中： 0.00
```

上段代码输出的个状态的最终价值与图 2.6 所示结果相同。有了最优状态价值，我们可以依据公式 (2.23) 计算最优行为价值，我们将不必再为此编写一个方法，之前的方法 compute_q 就可以完成这个功能。使用下面的代码来验证在状态“第三节课”时选择“泡吧”行为的最优价值：

```
# 验证最优行为价值
s, a = "第三节课", "泡吧"
q = compute_q(MDP, V_star, "第三节课", "泡吧")
print("在状态{}选择行为{}的最优价值为:{:.2f}".format(s,a,q))
# 将输出结果：
# 在状态第三节课选择行为泡吧的最优价值为:9.40
```

本章的编程实践到此结束。下面的代码是马尔科夫决策过程一开始导入的那些方法，这些方法保存在与之前代码同一文件夹下，文件名为“utils.py”。

```
def str_key(*args):
    '''将参数用"_"连接起来作为字典的键，需注意参数本身可能会是tuple或者list型，
    比如类似((a,b,c),d)的形式。
    '''
    new_arg = []
    for arg in args:
        if type(arg) in [tuple, list]:
            new_arg += [str(i) for i in arg]
        else:
            new_arg.append(str(arg))
    return "_".join(new_arg)

def set_dict(target_dict, value, *args):
    target_dict[str_key(*args)] = value

def set_prob(P, s, a, s1, p = 1.0): # 设置概率字典
    set_dict(P, p, s, a, s1)

def get_prob(P, s, a, s1): # 获取概率值
    return P.get(str_key(s,a,s1), 0)

```

```
22 def set_reward(R, s, a, r): # 设置奖励值
23     set_dict(R, r, s, a)
24
25 def get_reward(R, s, a): # 获取奖励值
26     return R.get(str_key(s,a), 0)
27
28 def display_dict(target_dict): # 显示字典内容
29     for key in target_dict.keys():
30         print("{}:  {:.2f}".format(key, target_dict[key]))
31     print("")
32
33 def set_value(V, s, v): # 设置价值字典
34     set_dict(V, v, s)
35
36 def get_value(V, s): # 获取价值字典
37     return V.get(str_key(s), 0)
38
39 def set_pi(Pi, s, a, p = 0.5): # 设置策略字典
40     set_dict(Pi, p, s, a)
41
42 def get_pi(Pi, s, a): # 获取策略（概率）值
43     return Pi.get(str_key(s,a), 0)
```