第三章：马尔科夫决策过程 （Finite Markov Decision Processes）

介绍（Introduction）

在本次实验中，熟练掌握马尔可夫过程、马尔可夫奖励过程和马尔可夫决策过程。

涉及核心概念回顾： 参考课堂上ppt

评分标准如下：

- 测试1-1 （10分）
- 测试1-2（10分）
- 测试1-3（10分）
- 测试1-4 (10分)
- 测试1-5 （10分）
- 测试2-1 （10分）
- 测试2-2（10分）
- 测试2-3（10分）
- 测试2-4 （10分）
- 测试2-5 （10分）

主题1： 马尔可夫奖励过程（Markov Reward Processes）

In [19]:

```
# 导入需要使用的库
import random
import numpy as np
random.seed(1234)
np.random.seed(1234)
```

测试1-1：阅读下图状态转移图，写出状态转移概率矩阵和奖励函数（向量表示）。

In [20]:

```
STATE_SPACE=["s_1", "s_2", "s_3", "s_4", "s_5", "s_6"]
# 定义状态转移概率矩阵P
# TODO: 基于上图，替换下面x为所需要的概率数值
P = [
    [0.9, 0.1, 0.0, 0.0, 0.0, 0.0],
    [0.5, 0.0, 0.5, 0.0, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.6, 0.0, 0.4],
    [0.0, 0.0, 0.0, 0.0, 0.3, 0.7],
    [0.0, 0.2, 0.3, 0.5, 0.0, 0.0],
    [0.0, 0.0, 0.0, 0.0, 0.0, 1.0],
```

```
]
# 对上述二位数组用矩阵进行封装
P = np.array(P)
# 定义奖励函数R
# TODO: 基于上图，替换下面x为所需要的奖励数值
R = {"s_1": -1, "s_2": -2, "s_3": -2, "s_4": 10, "s_5": 1, "s_6": 0}
# 定义折扣因子gamma，请设置x为一个小于1的数值
gamma = 0.5

 # 封装MRP
MRP = (P, R, gamma)
```

测试1-2： 基于上述状态概率转移矩阵，编写一个函数，采样相应轨迹(episode)

要求：对下述函数填写注释

In [21]:
```
def sample_trajectory(start_state, probabilities, max_len=1000):
    # NOTE:使用起始状态初始化轨迹
    episode = [start_state + 1]
    # NOTE:将当前状态设置为起始状态
    state = start_state
    # NOTE:循环直到到达最终的状态或者是达到最大轨迹长度
    while state != 5 and len(episode) < max_len:
        # NOTE: 根据上面的转移矩阵选择下一个状态
        next_state = random.choices(range(len(probabilities[state])), probabilities[stat
        # NOTE: 将下一个状态添加到轨迹中
        episode.append(next_state + 1)
        # NOTE: 更新当前状态
        state = next_state
    return episode #返回轨迹向量
```

测试1-3： 基于状态转移概率，采样轨迹（其实你可以采样无数条轨迹。） 要求：从s_2状态为起始状态，采样轨迹长度不能超过5

In [22]:
```
P, _, _ = MRP
# 调用sample_trajectory，从状态s2开始采样一个轨迹，最大长度为5
# TODO: 请注意，这里的状态是从0开始编号的，所以s2对应的是状态1
episode = sample_trajectory(1,P,5)
# TODO: 打印采样的轨迹
print(episode)
```

```
[2, 3, 4, 5, 4]
```

测试1-4： 针对上述采样的轨迹，计算出初始状态的回报

基于上述轨迹，计算s_2的回报（Return） 提示：$G_t = R_{t+1} + \gamma G_{t+1}$

In [23]:
```
# 给定一条序列,计算从某个索引（起始状态）开始到序列最后（终止状态）得到的回报
def compute_return(start_index, episode, gamma):
    G = 0
    for i in reversed(range(start_index, len(episode))):
        G = gamma * G + R['s'+'_'+str((episode[i]))]
    return G

start_index = 0
# 注意: episode是上面采样的轨迹
G = compute_return(start_index, episode, gamma)
print("根据本序列计算得到回报为：%s。" % G)
```

```
根据本序列计算得到回报为：0.25。
```

思考： 其实上述轨迹，有的状态可能会被visit多次，如何计算。

答：可以根据每次访问该状态时获得的奖励以及之后的**折扣因子**来计算每次访问该状态的**回报**，然后将所有这些回报**累加**。这样可以考虑到状态被多次访问的情况，而不仅仅是考虑第一次访问时的奖励。

测试1-5：MRP状态价值函数计算

- 一个状态的期望回报，即从这个状态出发，未来累积奖励的期望。
- 所有状态的价值组成价值函数。 </div> 我们将状态价值函数写成

$$\mathbf{v}(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}[R_t + \gamma G_{t+1}|S_t = s]$$

基于贝尔曼方程直接计算状态价值

$$\mathbf{v}(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s)\mathbf{v}(s')$$

提示：

$$\mathbf{v} = R + \gamma P \mathbf{v}$$

$$(I - \gamma P)\mathbf{v} = R$$

$$\mathbf{v} = (I - \gamma P)^{-1} R$$

基于上述贝尔曼方程，计算MRP状态价值函数。

In [30]:

```python
# TODO: 请在下面编写代码，计算状态价值函数
def compute_state_value(P, R, gamma, STATE_SPACE):
    """
    计算MDP中的状态价值函数
    """
    # 提示：可以使用np.linalg.inv()来计算矩阵的逆，使用np.eye()来生成单位矩阵
    # np.dot()来计算矩阵乘法
    # TODO: 生成单位矩阵
    n = len(STATE_SPACE)
    I = np.eye(n,n)
    # TODO: 计算价值函数，使用公式V = (I - gamma * P)^(-1) * R
    RR = []
    for state in STATE_SPACE:
        RR.append(R[state])
    V = np.dot(np.linalg.inv(I - gamma * P),RR)
    return {s: V[i] for i, s in enumerate(STATE_SPACE)}

P, R, gamma = MRP
value = compute_state_value(P, R, gamma, STATE_SPACE)
# STEP 5: 输出解析解value
print("MRP中每个状态价值分别为\n", value)
# 输出结果如下
# MRP中每个状态价值分别为
#  [[-2.01950168]
#  [-2.21451846]
#  [ 1.16142785]
#  [10.53809283]
#  [ 3.58728554]
#  [ 0.        ]]
```

```
MRP中每个状态价值分别为
 {'s_1': -2.0195016779238815, 's_2': -2.214518457162695, 's_3': 1.161427849273102, 's_4': 10.538092830910342, 's_5': 3.587285539402281, 's_6': 0.0}
```

主题2： 有限马尔可夫决策过程（finite Markov Decision Processes）

以下gridWorld定义的MDP环境。

基于上述Grid信息，定义MDP环境（为了方便理解，我们假设奖励本身为确定量，非随机变量）：

$$< \mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma >$$

- $\mathcal{S}$: 状态空间
- $\mathcal{A}$: 动作空间
- γ折扣因子
- $R(s, a)$: 奖励函数，即奖励取决于状态s和动作a
- $\mathcal{P}(s'|s, a)$为状态转移函数，为在状态s下执行动作a后到达s'的概率。

测试2-1： 基于上图，定义马尔可夫决策过程。

In [63]:

```
# TODO: 定义状态空间("s_A", "s_B", "s_C", "s_D")和动作空间("Up", "Down", "Left", "Right")
# 替换None为所需要的代码
ACTION_SPACE = ["Up", "Down", "Left", "Right"]
STATE_SPACE = ["s_A", "s_B", "s_C", "s_D"]

P = {f"{s}-{a}-{next_s}": 0.0 for s in STATE_SPACE for a in ACTION_SPACE for next_s in S
# 设置给定位置为相应的概率
P["s_A-Up-s_A"] =  1.0
P["s_A-Down-s_C"] =  1.0
P["s_A-Left-s_A"] = 1.0
P["s_A-Right-s_B"] = 1.0
# TODO: 设置其他位置的概率
P["s_B-Up-s_B"] =  1.0
P["s_B-Down-s_D"] =  1.0
P["s_B-Left-s_A"] = 1.0
P["s_B-Right-s_B"] = 1.0

P["s_C-Up-s_A"] =  1.0
P["s_C-Down-s_C"] =  1.0
P["s_C-Left-s_C"] = 1.0
P["s_C-Right-s_D"] = 1.0

P["s_D-Up-s_B"] =  1.0
P["s_D-Down-s_D"] =  1.0
P["s_D-Left-s_C"] = 1.0
P["s_D-Right-s_D"] = 1.0
# TODO: 定义奖励函数, 替换None为所需要的奖励数值
rewards = {
    "s_A": {"Up": 0, "Down": 0, "Left": 0, "Right": 5},
    "s_B": {"Up": 5, "Down": 0, "Left": 0, "Right": 5},
    "s_C": {"Up": 0, "Down": 0, "Left": 0, "Right": 0},
    "s_D": {"Up": 5, "Down": 0, "Left": 0, "Right": 0},
}
# 折扣因子为0.7
gamma = 0.7
# 定义MDP
MDP = {
    "P": P,
    "rewards": rewards,
    "gamma": gamma
}
```

测试2-2： 定义一个随机策略$\pi(a|s)$，在上述环境中每个cell上, 可执行的动作为{上，下，左，右}，概率分别为25%。

In [54]:
```python
# 可用字典定义随机策略
# TODO: 请替换下面的None为所需要的代码
policy = {
    "s_A": {"Up": 0.25, "Down": 0.25, "Left": 0.25, "Right": 0.25},
    "s_B": {"Up": 0.25, "Down": 0.25, "Left": 0.25, "Right": 0.25},
    "s_C": {"Up": 0.25, "Down": 0.25, "Left": 0.25, "Right": 0.25},
    "s_D": {"Up": 0.25, "Down": 0.25, "Left": 0.25, "Right": 0.25},
}
```

测试2-3： 基于上述策略，请计算MDP中每个状态的奖励。

我们可以将策略的动作选择进行边缘化（marginalization)，就可以得到没有动作的 MRP 了。具体来说，对于某一个状态，我们根据策略所有动作的概率进行加权，得到的奖励和就可以认为是一个 MRP 在该状态下的奖励，即

$$R(s) = \sum_{a \in \mathcal{A}} \pi(a|s) R(s,a)$$

In [64]:
```python
def compute_reward(MDP, STATE_SPACE):
    """
    计算MDP中的奖励函数
    """
    # TODO: 请在下面编写代码
    # 将None替换为所需要的代码
    P, R, gamma = MDP["P"],MDP["rewards"],MDP["gamma"]
    R_state = {s: 0 for s in STATE_SPACE}
    for state in STATE_SPACE:
        for i in ACTION_SPACE:
            # TODO: 计算状态奖励函数
            for next_state in STATE_SPACE:
            # 将None替换为所需要的代码
                R_state[state] += policy[state][i]*P[f"{state}-{i}-{next_state}"] *R[sta
    return R_state
# 计算奖励函数
R_MRP = compute_reward(MDP, STATE_SPACE)
print("reward: ", R_MRP)
```

```
reward:  {'s_A': 1.25, 's_B': 2.5, 's_C': 0.0, 's_D': 1.25}
```

测试2-4： 请计算状态转移概率

我们计算采取动作的概率与使s转移到的s'概率的乘积，再将这些乘积相加，其和就是一个 MRP 的状态s从转移至s'的概率

$$\mathcal{P}'(s,s') = \sum_{a \in \mathcal{A}} \pi(a|s) P(s'|s,a)$$

In [67]:
```python
# 状态转移矩阵
def cal_state_transition(MDP):
    """
    计算MDP中的状态转移矩阵
    """
    # TODO: 请在下面编写代码
    # 将None替换为所需要的代码
    P, R, gamma = MDP["P"],MDP["rewards"],MDP["gamma"]
    # 初始化状态转移矩阵
    n = len(STATE_SPACE)
    P_prim = np.zeros((n, n))
    for i, s in enumerate(STATE_SPACE):
        for j, next_s in enumerate(STATE_SPACE):
            for a in ACTION_SPACE:
                # TODO: 计算状态转移概率
```

```
                # 将None替换为所需要的代码
                # 基于公式P(s'|s, a) = sum(pi(a|s) * P(s'|s, a))
                P_prim[i, j] += policy[s][a] *P[f"{s}-{a}-{next_s}"]
    return P_prim
# 调用cal_state_transition函数，计算状态转移矩阵
# 替换None为所需要的代码
P_MRP = cal_state_transition(MDP)
print(P_MRP)
```

```
[[0.5  0.25 0.25 0.  ]
 [0.25 0.5  0.   0.25]
 [0.25 0.   0.5  0.25]
 [0.   0.25 0.25 0.5 ]]
```

测试2-5： 我们通过对动作进行margalization, 实现了将MDP转换成MRP，现在请调用函数，计算MDP的状态价值函数。

In [68]:

```
# 计算状态价值函数
# TODO: 请在下面编写代码
# Hint: 用到MRP中的函数
value_MDP = compute_state_value(P_MRP,R_MRP,gamma,STATE_SPACE)
value_MDP
```

Out[68]:

```
{'s_A': 4.166666666666665,
 's_B': 6.0897435897435885,
 's_C': 2.2435897435897427,
 's_D': 4.166666666666666}
```

总结

请写一下你对马尔可夫过程、马尔可夫奖励过程，马尔可夫决策过程的异同点。

马尔可夫过程：很单纯，仅仅就是转移

指的是马尔可夫性质的**随机过程**，**未来状态只跟当前状态有关，而与过去的其他状态无关**。

马尔可夫奖励过程：多了个“奖励”

每个状态都与一个**即时奖励**有关。该过程描述了每个状态获得奖励的情况下，整体获得最大奖励的期望过程，可以评估某个过程策略的好坏成都

马尔可夫决策过程：又多了个“动作”

引入**智能体**概念，智能体在每个状态下选择**动作**，且所选择的动作会影响下一个状态的转移和即使奖励。用来求智能体在不确定环境中做出决策最大化长期奖励期望。

相同点

三个过程都涉及**状态空间的变化**（转移）过程

不同点

①马尔可夫过程和奖励过程描述了状态到状态的**转移概率**，而决策过程引入了**动作**概念，状态转移概率与决策动作相关联

②马尔可夫奖励过程和决策过程 都与**奖励**有关，每个状态可能获得一个奖励，而单纯的马尔可夫过程**不涉及奖励**计算

③马尔可夫过程 和 奖励过程 不涉及**智能体**概念，只描述状态间的变化和奖励