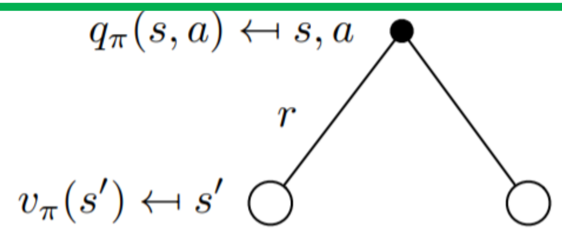
状态-动作值函数Q(s,a) $Q(s,a) = E[\,G_t \mid S_t = s, A_t = a\,]$

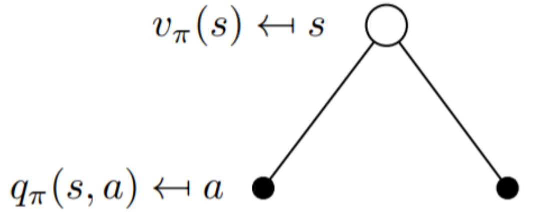
状态值函数V(s) $V(s) = E[\,G_t \mid S_t = s\,]$

Bellman Equation $Q_\pi(s,a) = E[R_{t+1} + \gamma Q(S_{t+1}) \mid S_t = s, A_t = a]$

$V(s) = E[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s]$



$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_\pi(s')$$

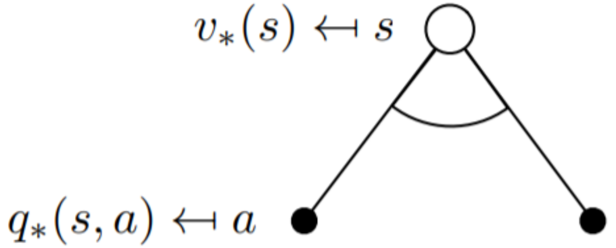


$$V_\pi(s) = \boxed{\sum_{a \in A} \pi(a \mid s)} Q_\pi(s,a)$$

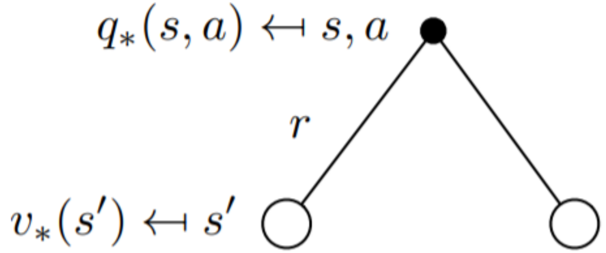
$V^*(s) = \max V_\pi(s)$

$Q^*(s,a) = \max Q_\pi(s,a)$

最优值只有一种决策，故取1



$$Q_*(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_*(s')$$



$$V_*(s) = \max Q_*(s,a) \quad , a \in A$$

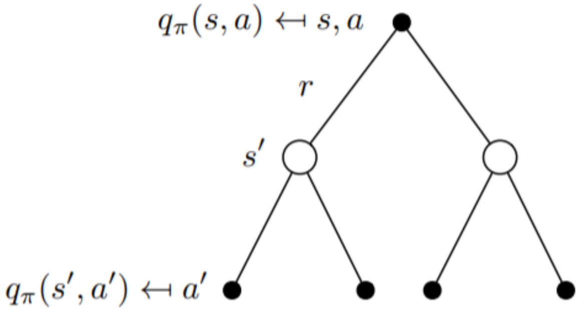
状态-动作值函数Q(s,a) $Q(s,a) = E[G_t \mid S_t = s, A_t = a]$

状态值函数V(s) $V(s) = E[G_t \mid S_t = s]$

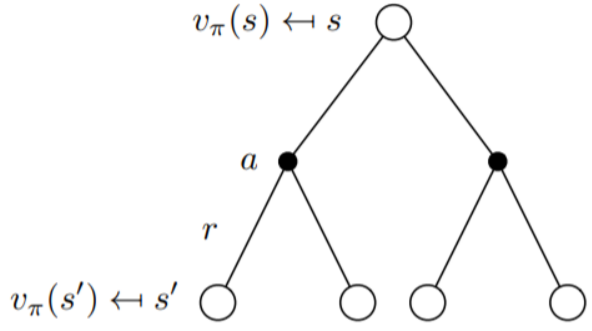
Bellman Equation

$$Q_\pi(s,a) = E[R_{t+1} + \gamma Q(S_{t+1}) \mid S_t = s, A_t = a]$$

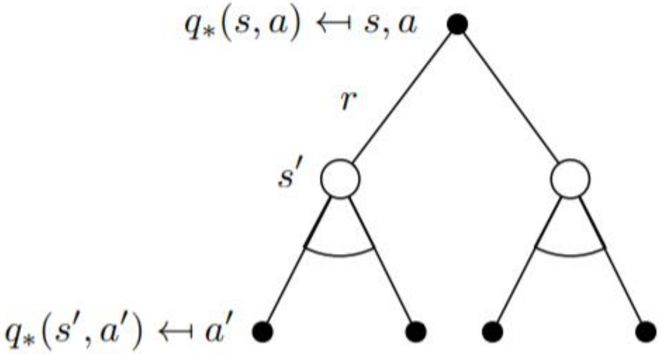
$$V(s) = E[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s]$$



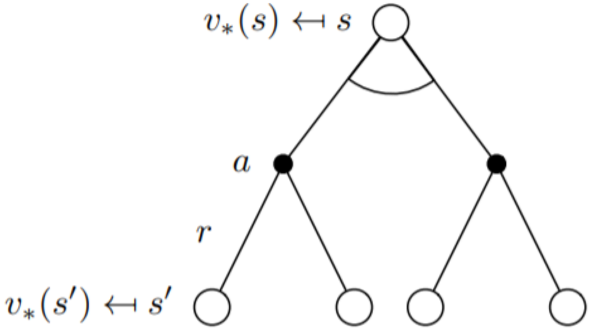
$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) \boxed{\sum_{a \in A} \pi(a' \mid s')} Q_\pi(s',a')$$



$$V_\pi(s) = \boxed{\sum_{a \in A} \pi(a \mid s)} [R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_\pi(s')]$$



$$Q_*(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) max Q_*(s',a'), a' \in A$$



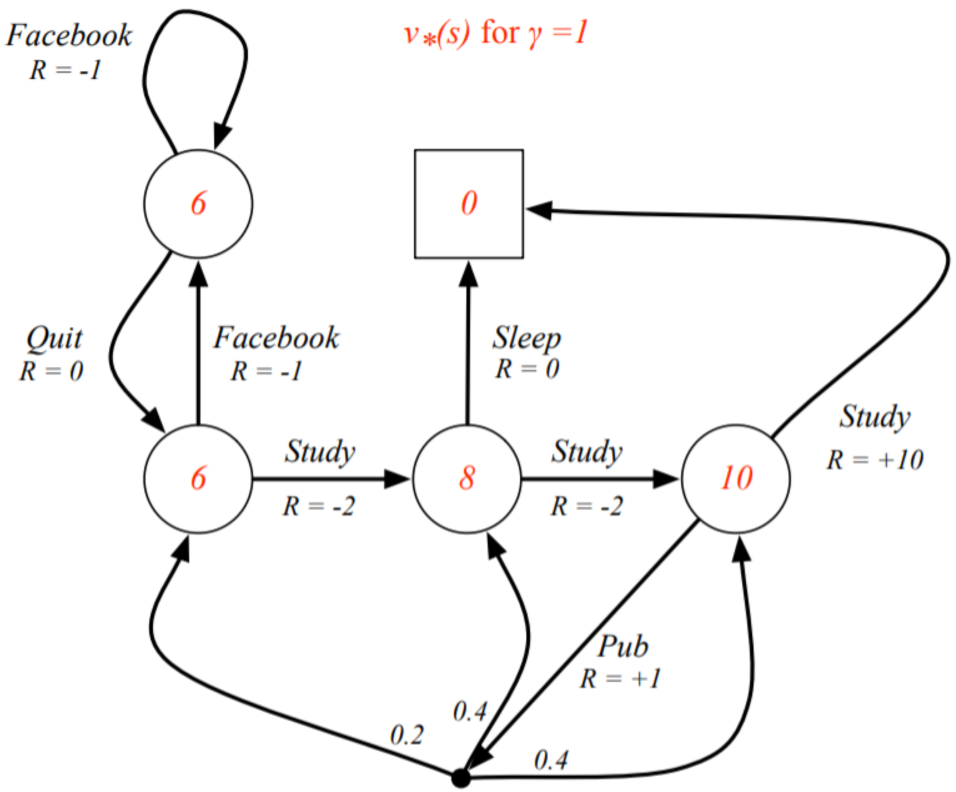
$$V_*(s) = max R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_*(s')]$$

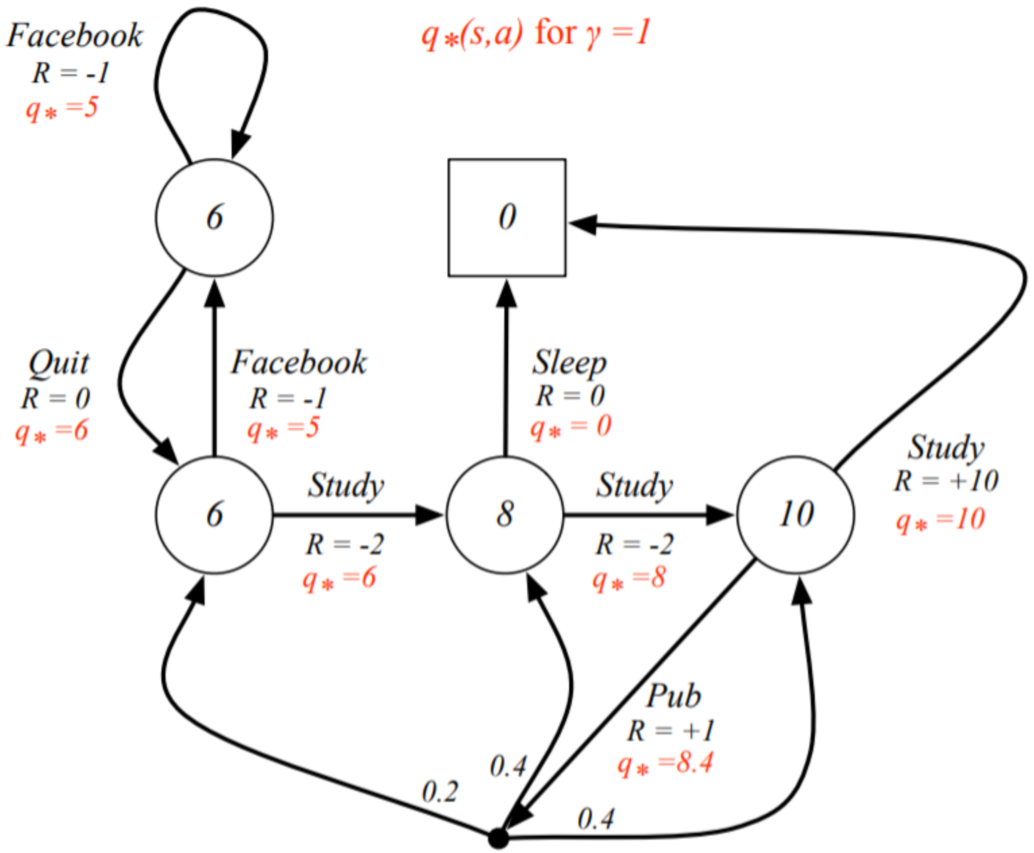
Bellman Optimality Equation

最优值函数$V^*(s), Q^*(s,a)$

$V^*(s) = max\ V_\pi(s)$，对于所有的策略，找到一个策略使得V值函数最大

$Q^*(s,a) = max\ Q_\pi(s,a)$，对于所有的策略，找到一个策略使得Q值函数最大





思考下**如何计算**？

Value Iteration:寻找最优策略

初始化，将所有的V值设置为0

$k = 0$ $\qquad \forall V_0(s) = 0$

输入值：S, A, P, r, γ

值迭代

for $k = 1:H$

贝尔曼最优方程

$$Q_{k+1}(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_k(s')$$

$$V_{k+1}(s) = max Q_{k+1}(s,a)$$

$$k = k + 1$$

优化函数：$min\ |V_k(s) - V_{k+1}(s)|$

V值收敛时

$$V_*(s) = max\ Q_{k+1}(s,a)$$

$$\pi^* = argmax\ Q_{k+1}(s,a)$$

输入值：S, A, P, r, γ

优化函数：$min\,|V_k(s) - V_{k+1}(s)|$

Value Iteration:寻找最优策略

初始化，将所有的V值设置为0

$$k = 0 \qquad \forall V_0(s) = 0$$

值迭代

$$\text{for } k = 1:H$$

贝尔曼最优方程

$$Q_{k+1}(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a,s) V_k(s')$$

$$V_{k+1}(s) = max Q_{k+1}(s,a)$$

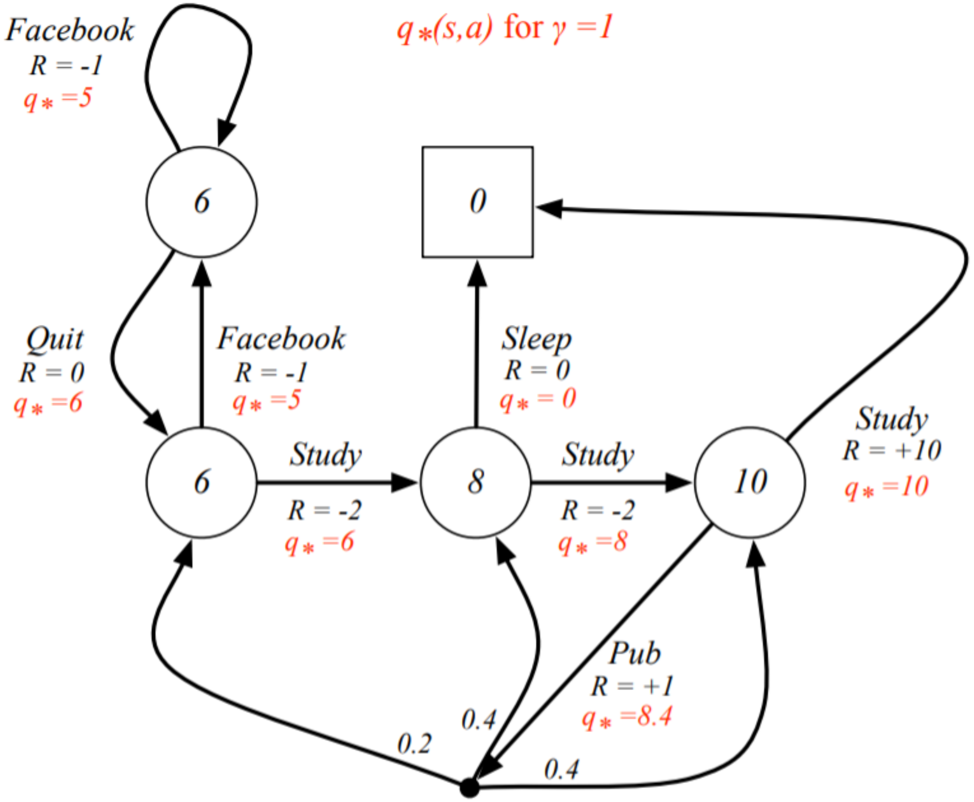
$$k = k + 1$$

V值收敛时

$$V_*(s) = max\ Q_{k+1}(s,a)$$

$$\pi^* = argmax\ Q_{k+1}(s,a)$$

$q_*(s,a)$ for $\gamma = 1$



$$Q_{11}(s,a) = -1 + V_1$$
$$Q_{12}(s,a) = 0 + V_2$$
$$V_1(s) = max\ Q_{11,12}(s,a)$$

$$Q_{21}(s,a) = -1 + V_1$$
$$Q_{22}(s,a) = -2 + V_3$$
$$V_2(s) = max\ Q_{21,22}(s,a)$$

$$Q_{31}(s,a) = 0 + V_5$$
$$Q_{32}(s,a) = -2 + V_4$$
$$V_3(s) = max\ Q_{31,32}(s,a)$$

$$Q_{41}(s,a) = 1 + 0.2V_2$$
$$Q_{42}(s,a) = 1 + 0.4V_3$$
$$Q_{43}(s,a) = 1 + 0.4V_4$$
$$Q_{44}(s,a) = 10 + V_5$$
$$V_4(s) = max\ Q_{41,42,43,44}(s,a)$$

Value Iteration:寻找最优策略

输入值：S, A, P, r, γ

优化函数：$min\ |V_k(s) - V_{k+1}(s)|$

初始化，将所有的V值设置为0

$$k = 0 \qquad \forall V_0(s) = 0$$

值迭代

$$\text{for } \ k = 1{:}H$$

贝尔曼最优方程

$$Q_{k+1}(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_k(s')$$

$$V_{k+1}(s) = max Q_{k+1}(s,a)$$

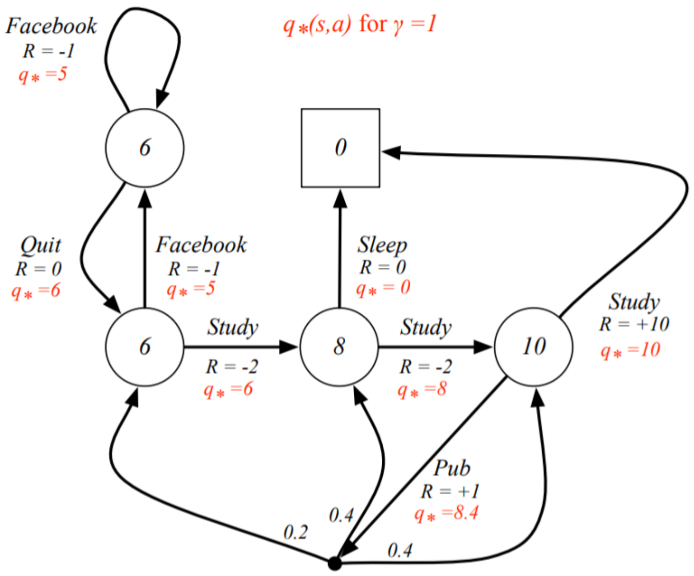
$$k = k + 1$$

V值收敛时

$$V_*(s) = max\ Q_{k+1}(s,a)$$

$$\pi^* = argmax\ Q_{k+1}(s,a)$$

V_1	0	0	-1	-2	6	6
V_2	0	-1	-2	6	6	6
V_3	0	0	8	8	8	8
V_4	0	10	10	10	10	10
V_5	0	0	0	0	0	0



$$Q_{11}(s,a) = \text{-}1 + V_1$$
$$Q_{12}(s,a) = 0 + V_2$$
$$V_1(s) = max\ Q_{11,12}(s,a)$$

$$Q_{21}(s,a) = -1 + V_1$$
$$Q_{22}(s,a) = -2 + V_3$$
$$V_2(s) = max\ Q_{21,22}(s,a)$$

$$Q_{31}(s,a) = 0 + V_5$$
$$Q_{32}(s,a) = -2 + V_4$$
$$V_3(s) = max\ Q_{31,32}(s,a)$$

$$Q_{41}(s,a) = 1 + 0.2V_2$$
$$Q_{42}(s,a) = 1 + 0.4V_3$$
$$Q_{43}(s,a) = 1 + 0.4V_4$$
$$Q_{44}(s,a) = 10 + V_5$$
$$V_4(s) = max\ Q_{41,42,43,44}(s,a)$$

Value Iteration:编程实现

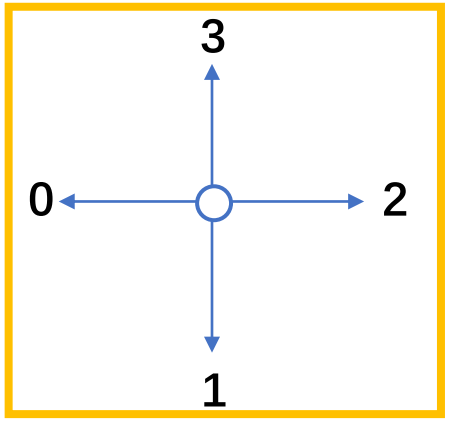
Gym框架：FrozenLake游戏

FrozenLake游戏的**状态空间**很简单，是一个 4 × 4 的矩阵，每个位置有一个字母:

- S，Starting Point，起始位置，安全
- F，Frozen Surface，冻结的湖面，安全
- H，Hole，洞，来到这个位置游戏失败，得-1分
- G，Goal，游戏终点，来到这个位置，得1分

```
SFFF
FHFH
FFFH
HFFG
```

```
0   1   2   3
4   5   6   7
8   9   10  11
12  13  14  15
```



游戏的**动作空间**也很简单。只有四个option，分别是0,1,2,3，代表朝四个方向运动。当然，游戏中向左运动的命令不一定100%被执行，而是有一定概率“被风吹到”其他地方，增加了游戏的随机性。

需要注意的是，在左侧边界（矩阵的最左列）是没法向左移动的，有一定概率（较大）保持原地，有较小概率**向下移动**；在上侧边缘（矩阵的第一行）没法向上走，较大概率原地不动，较小概率**向右移动**，右侧和下侧亦然。

Value Iteration:编程实现

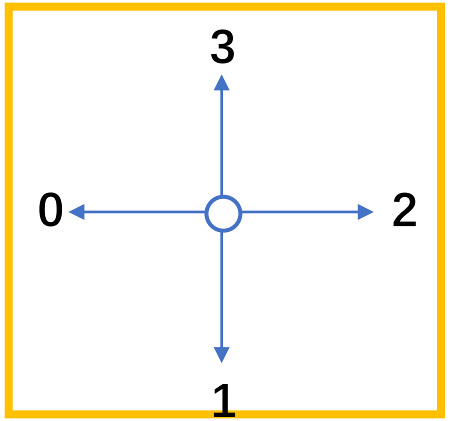
Gym框架：FrozenLake游戏

FrozenLake游戏的**状态空间**很简单，是一个 4×4 的矩阵，每个位置有一个字母：

- S， Starting Point，起始位置，安全
- F， Frozen Surface，冻结的湖面，安全
- H， Hole，洞，来到这个位置游戏失败，得-1分
- G， Goal，游戏终点，来到这个位置，得1分

```
SFFF
FHFH
FFFH
HFFG
```

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



左 env.P[0][0]

[(0.33, 0, 0.0, False),
(0.33, 0, 0.0, False),
(0.33, 4, 0.0, False)]

- 2/3概率原地不动，1/3概率向下移动

下 env.P[0][1]

[(0.33, 0, 0.0, False),
(0.33, 4, 0.0, False),
(0.33, 1, 0.0, False)]

- 1/3概率原地不动，1/3概率向下、右移动

右 env.P[0][2]

[(0.33, 4, 0.0, False),
(0.33, 1, 0.0, False),
(0.33, 0, 0.0, False)]

- 1/3概率原地不动，1/3概率向下、右移动

上 env.P[0][3]

[(0.33, 1, 0.0, False),
(0.33, 0, 0.0, False),
(0.33, 0, 0.0, False)]

- 2/3概率原地不动，1/3概率向下

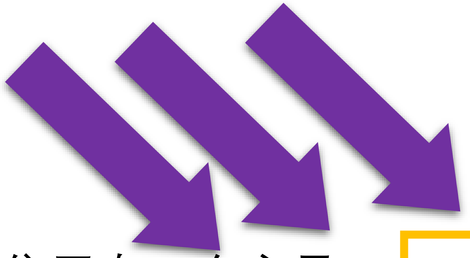
$P(s' \mid a, s)$

(状态转移概率, 下一个状态, reward,游戏是否结束)

Value Iteration:编程实现

Gym框架：FrozenLake游戏

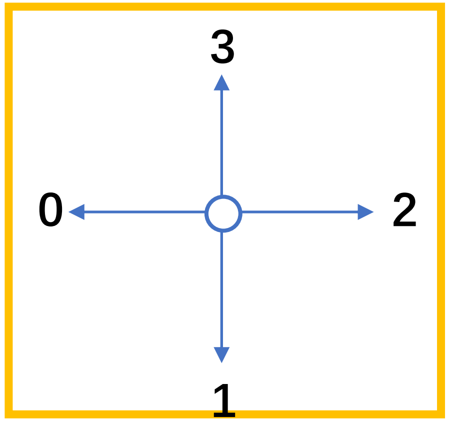
风向



FrozenLake游戏的**状态空间**很简单，是一个 4 × 4 的矩阵，每个位置有一个字母：

- S，Starting Point，起始位置，安全
- F，Frozen Surface，冻结的湖面，安全
- H，Hole，洞，来到这个位置游戏失败，得-1分
- G，Goal，游戏终点，来到这个位置，得1分

```
SFFF    0  1  2  3
FHFH    4  5  6  7
FFFH    8  9  10 11
HFFG    12 13 14 15
```



左 env.P[0][0]

[(0.33, 0, 0.0, False),
(0.33, 0, 0.0, False),
(0.33, 4, 0.0, False)]

- 2/3概率原地不动，1/3概率向下移动

右 env.P[0][2]

[(0.33, 4, 0.0, False),
(0.33, 1, 0.0, False),
(0.33, 0, 0.0, False)]

- 1/3概率原地不动，1/3概率向下、右移动

下 env.P[0][1]

[(0.33, 0, 0.0, False),
(0.33, 4, 0.0, False),
(0.33, 1, 0.0, False)]

- 1/3概率原地不动，1/3概率向下、右移动

上 env.P[0][3]

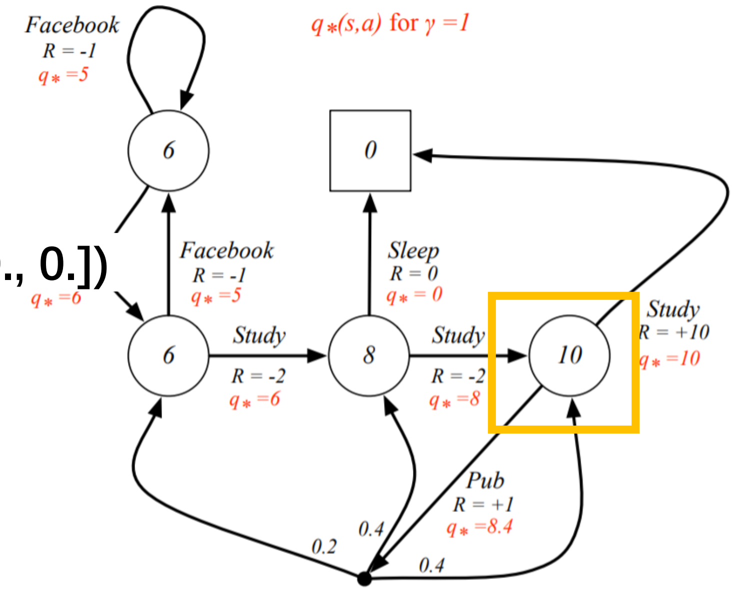
[(0.33, 1, 0.0, False),
(0.33, 0, 0.0, False),
(0.33, 0, 0.0, False)]

- 2/3概率原地不动，1/3概率向下

$P(s' \mid a, s)$

(状态转移概率，下一个状态,reward,游戏是否结束)

Value Iteration:编程实现



Value Table 16×1 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

updated_value_table = np.copy(value_table)

Updated Value Table 16×1

Q Value 16×1

```
Q_value.append(np.sum(next_states_rewards))
```

$$Q_{41}(s,a) = 1 + 0.2V_2$$
$$Q_{42}(s,a) = 1 + 0.4V_3$$
$$Q_{43}(s,a) = 1 + 0.4V_4$$

类似于学生问题，将此时选用0动作的Q值加起来，1，2，3也是这样。

$$0.2 \times (1 + V_2) + 0.4 \times (1 + V_3) + 0.4 \times (1 + V_4) = Q_{publish}$$

```
next_states_rewards.append((trans_prob * (reward_prob + gamma * updated_value_table[next_state])))
```

$$V_*(s) = max R_s^a + \gamma \sum_{s \in s'} P(s' \mid a,s) V_*(s')]$$

其实是一回事，这个概率放在里面和放在外面都一样

$$= max[\sum_{s \in s'} P(s' \mid a,s) R_s^a + \gamma \sum_{s \in s'} P(s' \mid a,s) V_*(s')]] = max \sum_{s \in s'} P(s' \mid a,s)[R + \gamma V_*(s')]]$$

1 Value Iteration: 值迭代函数,找到收敛的V值

```
def value_iteration(env, gamma = 0.8, no_of_iterations = 2000):
    value_table = np.zeros(env.observation_space.n)
    threshold = 1e-20
    for i in range(no_of_iterations):
        updated_value_table = np.copy(value_table)
        for state in range(env.observation_space.n):
            Q_value = []
            for action in range(env.action_space.n):
                next_states_rewards = []
                for next_sr in env.P[state][action]:
                    trans_prob, next_state, reward_prob, _ = next_sr
                    next_states_rewards.append((trans_prob * (reward_prob + gamma * updated_value_table[next_state])))
                Q_value.append(np.sum(next_states_rewards))
            value_table[state] = max(Q_value)
        if (np.sum(np.fabs(updated_value_table - value_table)) <= threshold):
            print ('Value-iteration converged at iteration# %d.' %(i+1))
            break
    return value_table
```

value_table 长度为16

state=0,1,···15

Q_value 长度为4

action=0,1,2,3

next_sr：（状态转移概率，下一个状态,reward值,游戏是否结束）

$$V_*(s) = \sum_{s\in s'} P(s' \mid a, s)\,[R + \gamma V_*(s')]$$

计算某一动作的Q值，需要将所有状态转移后的V值求和

用最大的Q值来更新该状态的V值 $V_{k+1}(s) = max Q_{k+1}(s, a)$

2 Value Iteration: V值收敛后选择策略

```
def extract_policy(value_table, gamma = 1.0):
    policy = np.zeros(env.observation_space.n)
    for state in range(env.observation_space.n):
        Q_table = np.zeros(env.action_space.n)
        for action in range(env.action_space.n):
            for next_sr in env.P[state][action]:
                trans_prob, next_state, reward_prob, _ = next_sr
                Q_table[action] += (trans_prob * (reward_prob + gamma * value_table[next_state]))
        policy[state] = np.argmax(Q_table)

    return policy
```

确定的策略

Policy 长度为16, 表示处于状态t时应该采取的最佳动作是上/下/左/右

Q_table 长度为4

通过Q值来选择策略，而不是V值

在计算了最佳V值后，遍历每一个状态下采取不同动作后的Q值，确定了该状态下的策略是采取什么样的动作。遍历16个状态后，确定最终的策略。

Value Iteration: 检验效果如何

```
def get_score(env, policy, episodes=1000):
  misses = 0
  steps_list = []
  for episode in range(episodes):
    observation = env.reset()
    steps=0
    while True:
      action = policy[observation]
      observation, reward, done, _ = env.step(action)
      steps+=1
      if done and reward == 1:
        steps_list.append(steps)
        break
      elif done and reward == 0:
        print("You fell in a hole!")
        misses += 1
        break
  print('----------------------------------------------')
  print('You took an average of {:.0f} steps to get the frisbee'.format(np.mean(steps_list)))
  print('And you fell in the hole {:.2f} % of the times'.format((misses/episodes) * 100))
  print('----------------------------------------------')
```

env.step:(观测值，奖励，游戏是否结束，转移概率)

根据此时的动作，根据环境预设的情况转移到另一个状态。

optimal_policy array([1., 3., 2., 3., 0., 0., 0., 0., 3., 1., 0., 0., 0., 2., 1., 0.])

optimal_value_function

```
SFFF  0  1  2  3
FHFH  4  5  6  7
FFFH  8  9  10 11
HFFG  12 13 14 15
```

array([0.015, 0.015 , 0.027 , 0.015,
 0.026, 0. , 0.070, 0. ,
0.058, 0.134, 0.197 , 0. ,
 0. , 0.247 , 0.544, 0.])

Policy Iteration:寻找最优值

初始化策略π_0和$V(s)$

$k=0$ $\forall V_0(s)$， $\forall \boldsymbol{\pi_0}$

输入值：$\boldsymbol{S, A, P, r, \gamma}$

$\boldsymbol{for}\ \boldsymbol{\pi_K}=1:N$

策略评估

$\boldsymbol{for}\ k=1:n$

$$V_{k+1}(s)=\sum_{a\in A}\pi(a\mid s)[R_s^a+\gamma\sum_{s\in s\prime}P(s'\mid a,s)V_k(s')]$$

$k=k+1$

计算确定策略下的当前动作的$\boldsymbol{V}$值

优化函数 $\boldsymbol{min|V_k(s)-V_{k+1}(s)|}$

策略改进

$$Q_{k+1}(s,a)=R_s^a+\gamma\sum_{s\in s\prime}P(s'\mid a,s)V_k(s')$$

$$V_{k+1}(s)=max Q_{k+1}(s,a),\quad \pi=\pi_K, K\in[1:N]$$

若得到的新策略记为$\boldsymbol{\pi_K}$，用该策略进行下一次评估

计算所有策略下所有动作的$\boldsymbol{Q}$值

判断 $\boldsymbol{if\ \pi_{K+1}=\pi_K}$?

Policy Iteration:寻找最优值

输入值：S, A, P, r, γ

初始化策略π_0和$V(s)$

$k=0$ $\qquad \forall V_0(s), \quad \forall \pi_0$

$for\ \pi_K = 1:N$

策略评估

计算确定策略下的当前动作的V值

$for\ k = 1:n$

$$V_{k+1}(s) = \sum_{a \in A} \pi(a \mid s)[R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_k(s')]$$

$k = k+1$ $\qquad$ 优化函数 $min|V_k(s) - V_{k+1}(s)|$

策略改进

计算所有策略下所有动作的Q值

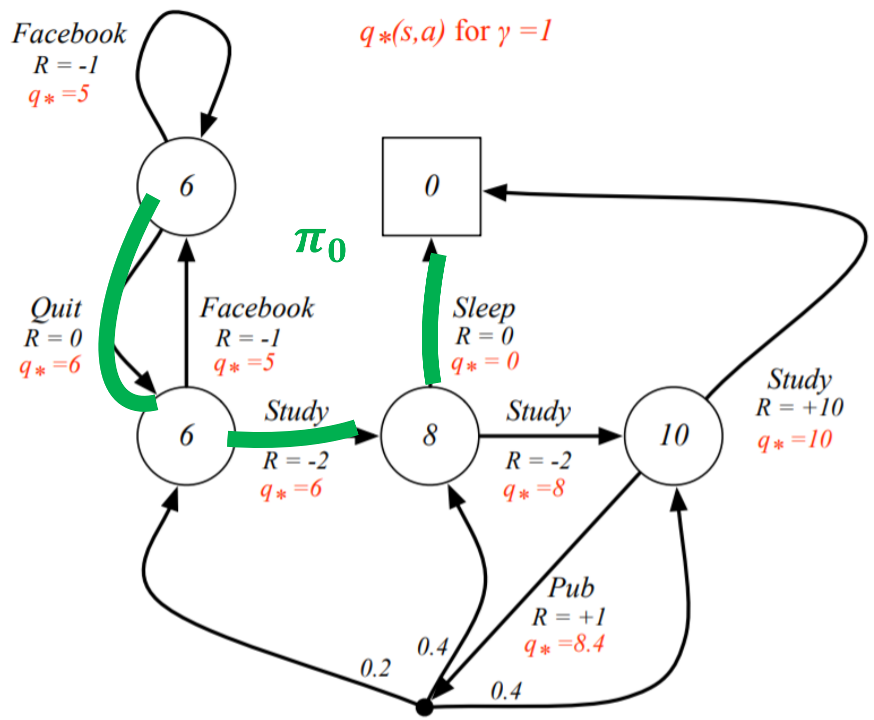
$$Q_{k+1}(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_k(s')$$

判断 $if\ \pi_{K+1} = \pi_K$?

$$V_{k+1}(s) = max Q_{k+1}(s,a), \quad \pi = \pi_K, K \in [1:N]$$

若得到的新策略记为π_K，用该策略进行下一次评估

V_1	6	6	4	-2	-2
V_2	6	4	−2	-2	-2
V_3	6	0	0	0	0
V_4	6	6	6	6	6
V_5	0	0	0	0	0



Policy Iteration:寻找最优值

输入值：S, A, P, r, γ

初始化策略π_0和$V(s)$

$k = 0$　　　$\forall V_0(s)$,　$\forall \pi_0$

$for\ \pi_K = 1{:}N$

策略评估

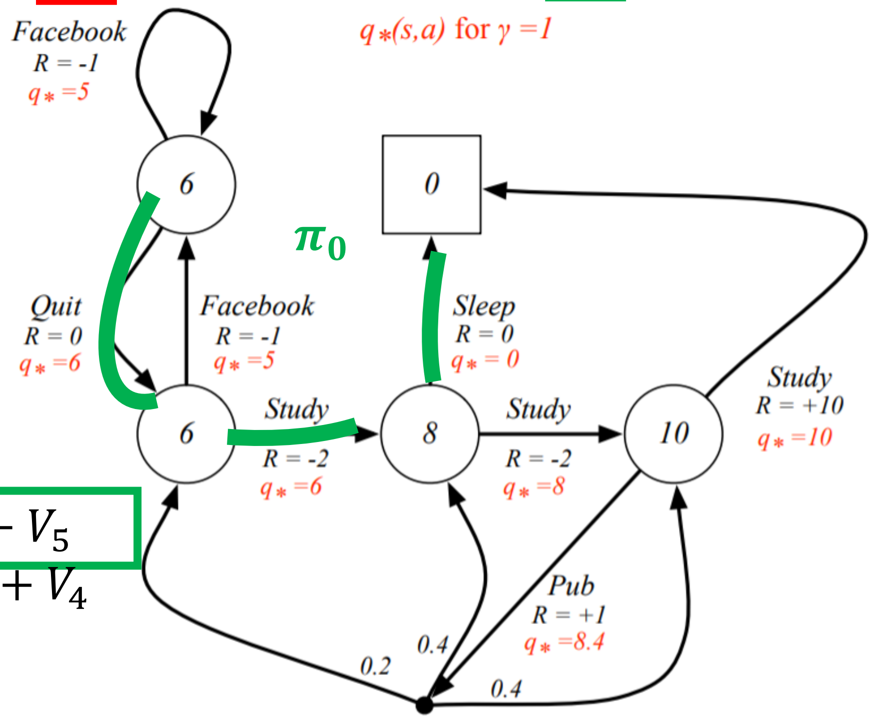
计算确定策略下的当前动作的V值

$for\ k = 1{:}n$

$$V_{k+1}(s) = \sum_{a \in A} \pi(a \mid s)[R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_k(s')]$$

$k = k + 1$　　优化函数 $\min |V_k(s) - V_{k+1}(s)|$

V_1	$\boxed{6}$	6	4	-2	-2
V_2	6	4	−2	-2	-2
V_3	6	0	0	0	0
V_4	6	6	6	6	6
V_5	0	0	0	0	0



$Q_{11}(s,a) = -1 + V_1$

$\boxed{Q_{12}(s,a) = 0 + V_2}$

$Q_{21}(s,a) = -2 + V_1$

$\boxed{Q_{22}(s,a) = -2 + V_3}$

$\boxed{Q_{31}(s,a) = 0 + V_5}$

$Q_{32}(s,a) = -2 + V_4$

Policy Iteration:寻找最优值

输入值：S, A, P, r, γ

初始化策略π_0和$V(s)$

$k = 0$ $\qquad \forall V_0(s), \quad \forall \pi_0$

$for\ \pi_K = 1:N$

策略改进

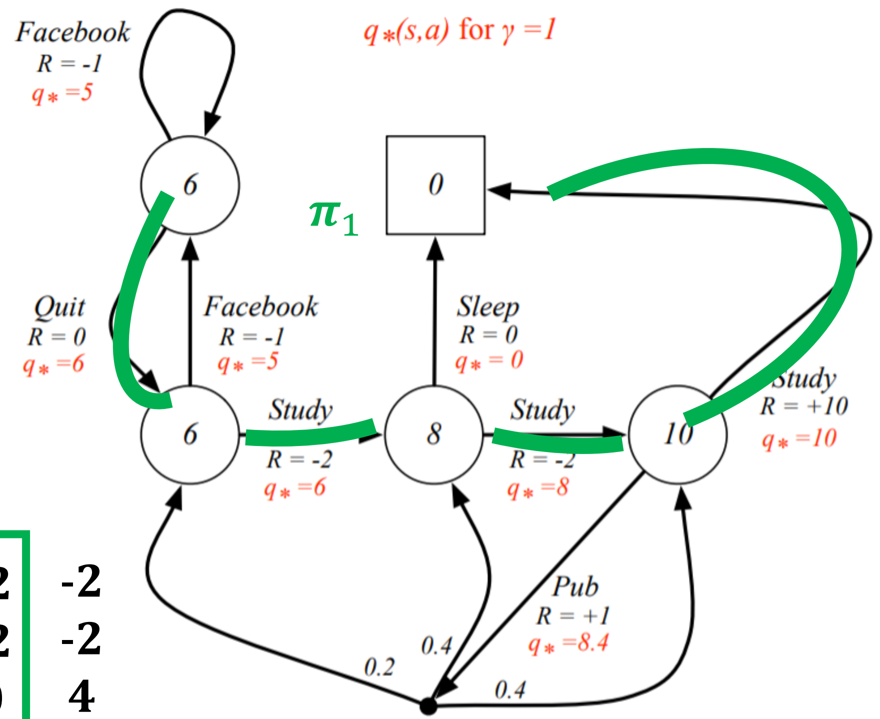
计算所有策略下所有动作的Q值

$$Q_{k+1}(s,a) = R_s^a + \gamma \sum_{s \in s'} P(s' \mid a, s) V_k(s')$$

$$V_{k+1}(s) = max Q_{k+1}(s,a), \quad \pi = \pi_K, K \in [1:N]$$

判断 $if\ \pi_{K+1} = \pi_K$?

若得到的新策略记为π_K，用该策略进行下一次评估



V_1	-2	-2
V_2	-2	-2
V_3	0	4
V_4	6	10
V_5	0	0

$$Q_{11}(s,a) = -1 + V_1$$
$$Q_{12}(s,a) = 0 + V_2$$
$$V_1(s) = max\ Q_{11,12}(s,a)$$

$$Q_{21}(s,a) = -1 + V_1$$
$$Q_{22}(s,a) = -2 + V_3$$
$$V_2(s) = max\ Q_{21,22}(s,a)$$

$$Q_{31}(s,a) = 0 + V_5$$
$$Q_{32}(s,a) = -2 + V_4$$
$$V_3(s) = max\ Q_{31,32}(s,a)$$

$$Q_{41}(s,a) = 1 + 0.2V_2$$
$$Q_{42}(s,a) = 1 + 0.4V_3$$
$$Q_{43}(s,a) = 1 + 0.4V_4$$
$$Q_{44}(s,a) = 10 + V_5$$
$$V_4(s) = max\ Q_{41,42,43,44}(s,a)$$

新的策略π_1，故再进行一次迭代

Policy Iteration:寻找最优值

输入值：S, A, P, r, γ

初始化策略π_0和$V(s)$

$k = 0$　　$\forall V_0(s)$，　$\forall\, \pi_0$

$for\ \pi_K = 1{:}N$

策略评估

计算确定策略下的当前动作的V值

$for\ k = 1{:}n$

$$V_{k+1}(s) = \sum_{a\in A} \pi(a \mid s)[R_s^a + \gamma \sum_{s\in s'} P(s' \mid a, s) V_k(s')]$$

$k = k + 1$　　优化函数 $min|V_k(s) - V_{k+1}(s)|$

$Q_{11}(s,a) = -1 + V_1$

$\boxed{Q_{12}(s,a) = 0 + V_2}$

$Q_{21}(s,a) = -1 + V_1$

$\boxed{Q_{22}(s,a) = -2 + V_3}$

$Q_{31}(s,a) = 0 + V_5$

$\boxed{Q_{32}(s,a) = -2 + V_4}$

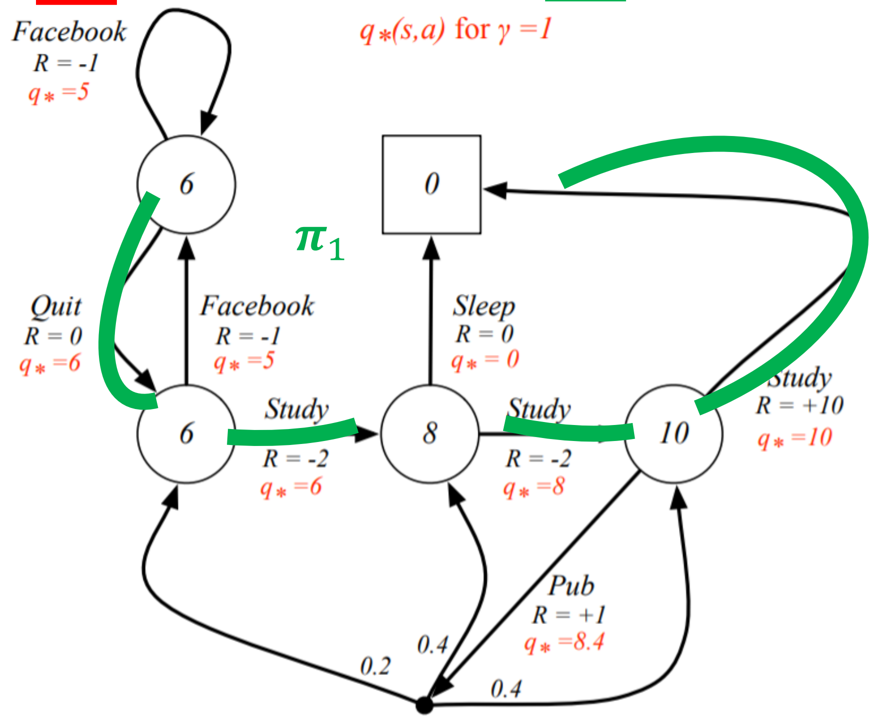
$Q_{41}(s,a) = 1 + 0.2V_2$

$Q_{42}(s,a) = 1 + 0.4V_3$

$Q_{43}(s,a) = 1 + 0.4V_4$

$\boxed{Q_{44}(s,a) = 10 + V_5}$

V_1	−2	−2	2	6	6
V_2	−2	2	6	6	6
V_3	4	8	8	8	8
V_4	10	10	10	10	10
V_5	0	0	0	0	0



新的策略π_2 和上次策略路径相同，迭代结束。

1 Policy Iteration: 根据策略计算值函数,策略评估

```
def compute_value_function(policy, gamma=1.0, threshold = 1e-20):
    value_table = np.zeros(env.nS)
    while True:
        updated_value_table = np.copy(value_table)
        for state in range(env.nS):
            action = policy[state]
            value_table[state] = sum([ trans_prob * (reward_prob + gamma * updated_value_table[next_state])
              for next_sr in env.P[state][action]:
                    trans_prob, next_state, reward_prob, _ = next_sr
        if (np.sum((np.fabs(updated_value_table - value_table))) <= threshold):
            break

    return value_table
```

$$V_*(s) = \sum_{s \in s'} P(s' \mid a, s)\,[R + \gamma V_*(s')]$$

选定了一个策略, $\boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{s})$

在确定的策略下，查看下一个时刻的状态奖励

2 Policy Iteration: V值收敛后选择策略

```
def extract_policy(value_table, gamma = 1.0):
    policy = np.zeros(env.observation_space.n)
    for state in range(env.observation_space.n):
        Q_table = np.zeros(env.action_space.n)
        for action in range(env.action_space.n):
            for next_sr in env.P[state][action]:
                trans_prob, next_state, reward_prob, _ = next_sr
                Q_table[action] += (trans_prob * (reward_prob + gamma * value_table[next_state]))
        policy[state] = np.argmax(Q_table)

    return policy
```

确定的策略

Policy 长度为16, 表示处于状态t时应该采取的最佳动作是上/下/左/右

Q_table 长度为4

通过Q值来选择策略，而不是V值

在计算了最佳V值后，遍历每一个状态下采取不同动作后的Q值，确定了该状态下的策略是采取什么样的动作。遍历16个状态后，确定最终的策略。

3 Policy Iteration: 策略改进

```
def policy_iteration(env,gamma = 1.0, no_of_iterations = 200000):
    gamma = 1.0
    random_policy = np.zeros(env.observation_space.n)

    for i in range(no_of_iterations):
        new_value_function = compute_value_function(random_policy, gamma)
        new_policy = extract_policy(new_value_function, gamma)
        if (np.all(random_policy == new_policy)):
            print('Policy-Iteration converged at step %d.'%(i+1))
            break
        random_policy = new_policy

    return new_policy
```

策略评估

策略改进

MC + Policy Iteration:编程实现

Gym框架：Blackjack游戏

- First Visit Monte Calo，每局游戏只记录第一次访问s_t后的累计回报。
- Every Visit Monte Calo，每局游戏记录每次访问s_t后的累计回报。
- **2~10**为对应分数，**JQK**为10分，**A**可当作1分或者11分

$$Q^{\pi}(s,a) \approx \hat{Q}^{\pi}(s,a) = \frac{1}{N}\sum_{n=1}^{N} G(\tau^{(n)}_{s_0=s,a_0=a})$$

通过不停的模拟，从一个状态出发直到结束时所得的分数来计算累计回报。
最后取平均值得到了V值。

状态空间：闲家分数，庄家分数，是否将Ace当作11来计算

动作空间：Hit，Stand，即叫牌和不叫牌。

```
def sample_policy(observation):
    '''
    如果当前牌面值>=20，就不再叫牌（Stand）
    如果当前牌面值<20,继续叫牌（Hit）
    '''
    # observation len=3,分别是闲家分数、庄家分数、是否将A当做11
    score, dealer_score, usable_ace = observation
    return 0 if score >= 20 else 1
```

MC+Policy Iteration:编程实现

Gym框架：Blackjack游戏

状态空间：闲家分数，庄家分数，是否将Ace当作11来计算

动作空间：Hit，Stand，即叫牌和不叫牌。

```
def generate_episode(policy, env):
    '''
    玩一局游戏，收集状态信息、动作信息和reward
    '''
    states, actions, rewards = [], [], []
    observation = env.reset()
    while True:
        states.append(observation)
        # 根据策略函数，确定采取的动作
        action = sample_policy(observation)
        actions.append(action)
        observation, reward, done, info = env.step(action)
        rewards.append(reward)
        if done:
            break
    return states, actions, rewards
```

(7, 1, False)

((14, 1, False), 0.0, False, {})\
通过实验来得到此时的观测值。

MC+Policy Iteration:编程实现

Gym框架：Blackjack游戏

状态空间：闲家分数，庄家分数，是否将Ace当作11来计算

动作空间：Hit，Stand，即叫牌和不叫牌。

```
def first_visit_mc_prediction(policy, env, n_episodes):

    value_table = defaultdict(float)
    N = defaultdict(int)
    for _ in range(n_episodes):
        states, _, rewards = generate_episode(policy, env)
        returns = 0
        for t in range(len(states) - 1, -1, -1):      如8,7,6…,0
            R = rewards[t]
            S = states[t]
            returns += R
            if S not in states[:t]:
                N[S] += 1
                value_table[S] += (returns - value_table[S]) / N[S]
    return value_table
```

$$
\begin{aligned}
\hat{Q}_N^{\pi}(s,a) &= \frac{1}{N}\sum_{n=1}^{N} G(\tau^{(n)}_{s0=s,a0=a}) \\
&= \frac{1}{N}\left(G(\tau^{(N)}_{s0=s,a0=a}) + \sum_{n=1}^{N-1} G(\tau^{(n)}_{s0=s,a0=a})\right) \\
&= \frac{1}{N}\left(G(\tau^{(N)}_{s0=s,a0=a}) + (N-1)\frac{1}{N-1}\sum_{n=1}^{N-1} G(\tau^{(n)}_{s0=s,a0=a})\right) \\
&= \frac{1}{N}\left(G(\tau^{(N)}_{s0=s,a0=a} + (N-1)\hat{Q}_{N-1}^{\pi}(s,a)\right) \\
&= \hat{Q}_{N-1}^{\pi}(s,a) + \frac{1}{N}\left(G(\tau^{(N)}_{s0=s,a0=a} - \hat{Q}_{N-1}^{\pi}(s,a)\right)
\end{aligned}
$$

((14, 1, False), 0.0, False, {})