使用 Qlearning 实现汉诺塔求解

主要思路：将汉诺塔中的各个状态进行编码，然后计算出各个状态之间相互转移时的 R 矩阵和 Q 矩阵，然后给定任一状态，根据 Q 矩阵即可找到通向目标的路径。

首先需要定义汉诺塔的各个状态，这里使用如'AAA'来表示状态，三个字母分别表示最小盘，中盘和最大盘在 A,B,C 的哪个塔中，'AAA'表示三个盘都在 A 塔中，并且呈小，中，大盘依次向下摆放，共选取状态如下：

```
states={
1:'AAA',
2:'BAA',
3:'CAA',
4:'BCA',
5:'CBA',
6:'CCA',
7:'BBA',
8:'CCB',
9:'BBC',
10:'ACB',
11:'BCB',
12:'CBC',
13:'CAC',
14:'ABC',
15:'CBC',
16:'ACC',
17:'CCC'
}
```

然后需要确定 R 矩阵和 Q 矩阵。R 矩阵代表奖励矩阵，状态之间能够转移，则赋为 0，如果不能转移则赋值为-1，能到达目标状态的状态转移赋值为 100。

这里将各个状态之间的转移默认设置为-1，然后找出那些有效的转移赋值为 0，最后将能到达目标状态的状态转移赋值为 100。在找有效的转移时，有两个汉诺塔状态。分别对三个盘子进行判断，看每个盘子进行转移后，第一个状态是否等于第二个状态，若等于则转移有效。小盘子能够转移的条件：无论何时，小盘子在最上面，能够转移，需要转移到另外的两个塔上。中盘子转移条件：小盘子没有在中盘子上面，中盘子转移到当前没有小盘子也没有中盘子的塔上。大盘子转移条件：中盘子和小盘子在一个塔上，大盘子在另一个塔上，大盘子从当前塔转移到另一个空塔上。

确定 R 矩阵的代码如下：

```
for i in range(17):
    i += 1
    for j in range(17):
        j += 1
        if i!=j:
            a=''
            b=''
            for a in ['A', 'B', 'C']:
                for k in ['A', 'B', 'C']:
                    if states[i][0]==a and k!=a and (k+states[i][1]+states[i][2])==states[j]:
                        R[i - 1][j - 1] = 0
            if states[i][1]!=states[i][0]:
                for a in ['A','B','C']:
                    for k in ['A','B','C']:
                        if states[i][0]==a and k!=a and k!=states[i][1] and (states[i][0]+k+states[i][2])==states[j]:
                            R[i - 1][j - 1] = 0
            if states[i][1]==states[i][0] and states[i][2]!=states[i][0]:
                for a in ['A', 'B', 'C']:
                    if a!=states[i][0] and a!=states[i][2] and (states[i][0]+states[i][1]+a)==states[j]:
                        R[i - 1][j - 1] = 0
R[15][16]=100
R[16][16]=100
```

R 矩阵的结果如下：

```
[-1,  0,  0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[0, -1,  0,  0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[0,  0, -1, -1,  0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[-1,  0, -1, -1, -1,  0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[-1, -1,  0, -1, -1, -1,  0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1,  0, -1, -1, -1,  0, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1,  0, -1, -1, -1,  0, -1, -1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1,  0, -1, -1, -1,  0,  0, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1,  0, -1, -1, -1, -1,  0, -1,  0,  0, -1, -1]
[-1, -1, -1, -1, -1, -1, -1,  0, -1, -1,  0, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1,  0, -1,  0, -1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1, -1,  0, -1, -1, -1,  0,  0, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  0, -1, -1,  0, -1, -1]
[-1, -1, -1, -1, -1, -1, -1, -1,  0, -1, -1,  0, -1, -1,  0,  0, -1]
[-1, -1, -1, -1, -1, -1, -1, -1,  0, -1, -1, -1,  0,  0, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  0, -1, -1, 100]
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  0, 100]
```

然后需要根据 R 矩阵确定 Q 矩阵。只需要对各个状态的各个行动，根据 Q 矩阵的公式进行更新就可以了。给定某一状态更新其各行动的 Q 矩阵的值的函数如下：

```
def QLearning(state):
    curAction = None
    for action in range(17):
        if(R[state][action] == -1):
                Q[state, action]=0
        else:
                curAction = action
                Q[state,action]=R[state][action]+GAMMA * getMaxQ(curAction)
```

然后可以执行大量的周期，每个周期中，为每个状态调用 QLearning 函数即可。

```
count=0
while count<1000:
    for i in range(17):
        QLearning(i)
    count+=1
```

Q 矩阵的结果如下：

```
[[  0.  20.  26.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.]
 [ 20.   0.  26.  16.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.]
 [ 20.  20.   0.   0.  32.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.]
 [  0.  20.   0.   0.   0.  13.   0.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.]
 [  0.   0.  26.   0.   0.   0.  40.   0.   0.   0.   0.   0.
    0.   0.   0.   0.   0.]
 [  0.   0.   0.  16.   0.   0.   0.  10.   0.   0.   0.   0.
    0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.  32.   0.   0.   0.  51.   0.   0.   0.
    0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.  13.   0.   0.   0.   8.   8.   0.
    0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.  40.   0.   0.   0.   0.  51.
    0.  64.  51.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.  10.   0.   0.   8.   0.
    0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.  10.   0.   8.   0.   0.
    0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.  51.   0.   0.   0.
   40.  64.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  51.
    0.   0.  51.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.  51.   0.   0.  51.
    0.   0.  51.  80.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.  51.   0.   0.   0.
   40.  64.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.  64.   0.   0. 100.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
    0.   0.   0.  80. 100.]]
```

有了 Q 矩阵，给定任意状态，可以根据 Q 矩阵来找到一条通向目标的路径。每次从 Q 矩阵中找到当前状态下，值最大的行动，然后根据行动更新状态即可。打印从状态 1 到状态 17 的路径代码如下。

```
i=1                                    # get the road to the target
print('states:%d '%i+states[i])
while i!=17:
    for j in range(17):
        if Q[i-1][j]==max(Q[i-1]):
            i=j+1
            print('states:%d '%i+states[i])
```

结果如下：

```
states:1 AAA
states:3 CAA
states:5 CBA
states:7 BBA
states:9 BBC
states:14 ABC
states:16 ACC
states:17 CCC
```