

### 3.6.

#### a.

**状态：**地图的着色情况，每个相邻地区的颜色不相同  
**初始状态：**地图没有着色  
**行动：**对地图的某个区域进行着色  
**转移模型：**在对某个区域进行着色后，当前状态变为在之前着色情况基础上该区域着色的状态  
**目标测试：**地图是否每个区域都着色且相邻两个区域的着色不相同  
**路径耗散：**每次行动耗散为 1

#### b.

**状态：**猴子和箱子的位置  
**初始状态：**最开始时猴子和箱子的位置  
**行动：**猴子对箱子的移动，叠放，攀爬  
**转移模型：**猴子对箱子做出行动后，箱子和猴子的状态变为现态  
**目标测试：**猴子是否能够够到香蕉  
**路径耗散：**猴子搬箱子爬箱子等行动的体力消耗

#### c.

**状态：**记录的处理进程和当前的处理结果  
**初始状态：**未对记录进行处理的状态  
**行动：**对单条记录进行处理并在处理错误时输出错误信息  
**转移模型：**在处理了某条信息后转到那条信息已经处理结束的状态  
**目标测试：**是否输出了出错的记录  
**路径耗散：**每次处理记录的耗散为 1

#### d.

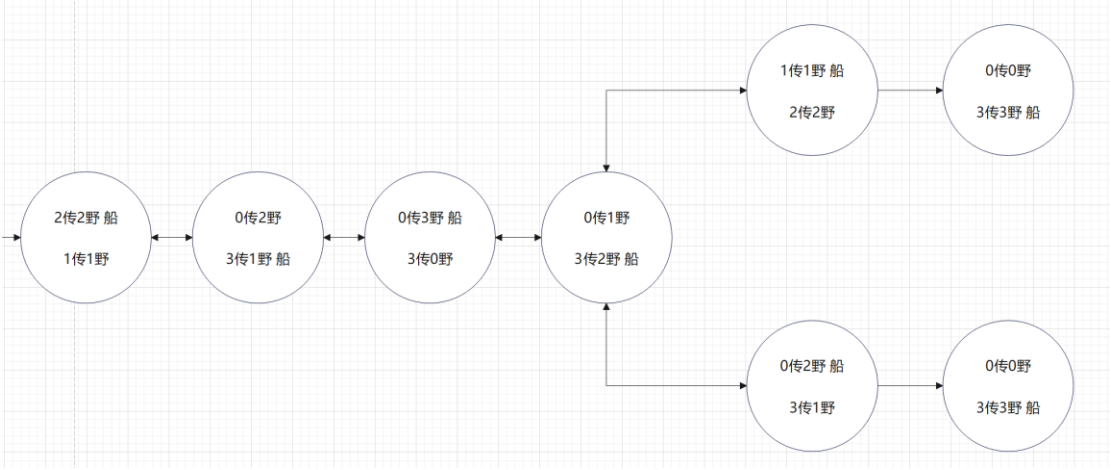
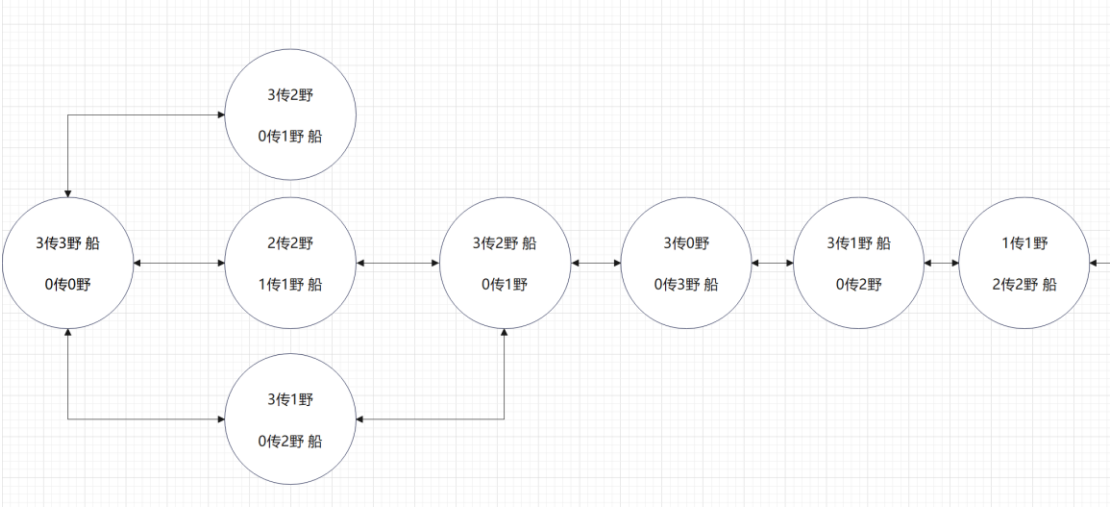
**状态：**三个水壶中的水的水量  
**初始状态：**三个水壶中的水量均为 0  
**行动：**使用放液嘴对水壶进行装满，倒空，从一个壶倒进另一个等操作  
**转移模型：**在运用放液嘴进行操作后，当前状态转移到操作后的水壶中的水量状态  
**目标测试：**某一个壶中是否有 1 加仑的水  
**路径耗散：**每次使用放液壶的耗散为 1

### 3.9.

#### a.

**状态：**两岸传教士和野人的数量（传教士在两岸的数量都不少于野人数量）以及船的位置（在船的哪一侧）  
**初始状态：**三个传教士和三个野人和船都在河的一侧  
**行动：**利用船将传教士和野人在两岸间运输，保证在船上，两岸的传教士都不少于野人的数量  
**转移模型：**船运输后的两岸的传教士和野人的数量和船的位置作为当前状态  
**目标测试：**传教士和野人是否都到达岸的另一侧  
**路径耗散：**船每摆渡一次耗散为 1

状态空间图：



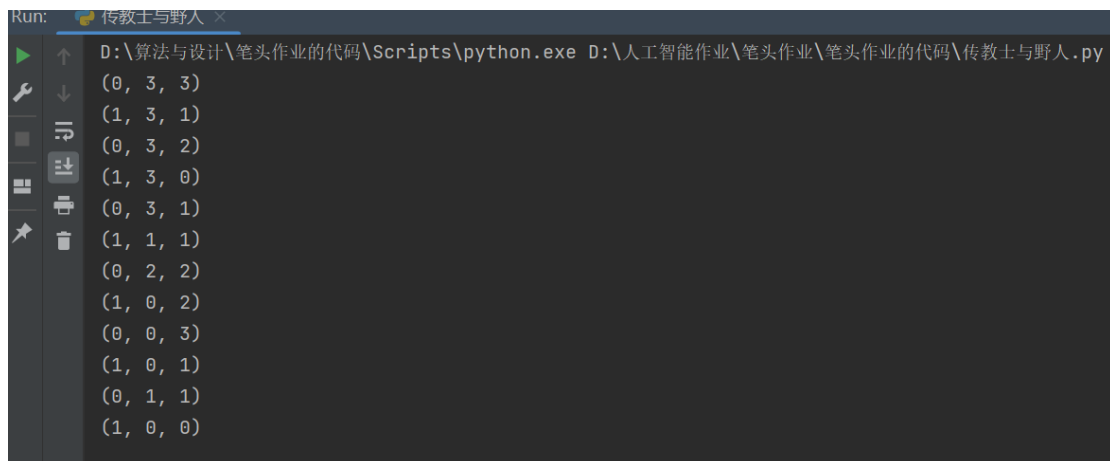
b.

因为要求求出最优解，所以我采用宽度优先搜索，代码如下图：

```
1 class state:
2     def __init__(self, ship, missionaries, savages):
3         self.ship = ship
4         self.missionaries = missionaries
5         self.savages = savages
6     def __eq__(self, other):
7         if isinstance(other, state):
8             return self.ship == other.ship and self.missionaries == other.missionaries and self.savages == other.savages
9         return False
10    def __hash__(self):
11        return hash((self.ship, self.missionaries, self.savages))
12    def trans(self, missionaries, savages):
13        if self.ship == 0:
14            if missionaries > self.missionaries:
15                return 0
16            if savages > self.savages:
17                return 0
18            if (self.missionaries - missionaries) < (self.savages - savages) and (self.missionaries - missionaries) != 0:
19                return 0
20            if (3 - self.missionaries + missionaries) < (3 - self.savages + savages) and self.missionaries - missionaries != 3:
21                return 0
22            self.missionaries = self.missionaries - missionaries
23            self.savages = self.savages - savages
24            self.ship = 1
25            return 1
26        elif self.ship == 1:
27            if missionaries > 3 - self.missionaries:
28                return 0
29            if savages > 3 - self.savages:
30                return 0
31            if (3 - self.missionaries - missionaries) < (3 - self.savages - savages) and (3 - self.missionaries - missionaries) != 0:
32                return 0
33            if (self.missionaries + missionaries) < (self.savages + savages) and 3 - self.missionaries - missionaries != 3:
34                return 0
35            self.missionaries = self.missionaries + missionaries
36            self.savages = self.savages + savages
37            self.ship = 0
38            return 1
```

```
40
41
42 def BFS(primary):
43     thislist = [primary] # BFS队列
44     thisdict = {primary: [(primary.ship, primary.missionaries, primary.savages)]} # 路径
45     thisset = {primary} # 是否访问过
46     method = [(1, 0), (0, 1), (2, 0), (0, 2), (1, 1)]
47     while 1:
48         if len(thislist) == 0:
49             break
50         now = thislist.pop(0)
51         list = thisdict[now].copy()
52         flag = 0
53         for x in method:
54             temp = state(now.ship, now.missionaries, now.savages)
55             f = temp.trans(x[0], x[1])
56             g = temp in thisset
57             if f and g == False:
58                 thislist.append(temp) # 该状态入队
59                 thisset.add(temp) # 标记为已访问
60                 thisdict[temp] = list.copy()
61                 thisdict[temp].append((temp.ship, temp.missionaries, temp.savages)) # 记录路径
62                 if temp.savages == 0 and temp.missionaries == 0:
63                     flag = 1
64                     break
65         if flag == 1:
66             break
67
68     n = len(thislist)
69     if n == 0:
70         print("no way")
71     else:
72         temp = thislist.pop()
73         for i in thisdict[temp]:
74             print(i)
75
76 primary = state(0, 3, 3)
77 BFS(primary)
```

运行结果如下图：



```
Run: 传教士与野人
D:\算法与设计\笔头作业的代码\Scripts\python.exe D:\人工智能作业\笔头作业\笔头作业的代码\传教士与野人.py
(0, 3, 3)
(1, 3, 1)
(0, 3, 2)
(1, 3, 0)
(0, 3, 1)
(1, 1, 1)
(0, 2, 2)
(1, 0, 2)
(0, 0, 3)
(1, 0, 1)
(0, 1, 1)
(1, 0, 0)
```

对于这个问题是要检查重复状态的，因为相邻两个合法状态可以无止境的来回转换，所以为了规避这种情况，需要检查重复状态

**c.**

- ①状态转化的条件比较特殊
- ②在不排除非法情况时状态很多
- ③需要一个合适的搜索算法

### 3.21

**a.**宽度优先搜索可以看作是状态转移的代价按照层数递增的一致代价搜索，这样在每次考虑当前路径代价时则优先考虑层数较浅的，知道本层访问完毕再访问下一层。

**b.**深度优先搜索可以看作是启发式函数的值随着层数的增加减少的最佳优先函数，这样在扩展下一个状态的时候会优先扩展下一层的状态。

**c.**一致代价搜索可看作是启发式函数恒为 0 的 A\*搜索，根据定义即可知。

### 3.25

当  $w$  取 1 和 0 时是最优的；

$w=0$ ：一致代价搜索

$w=1$ ：A\*搜索

$w=2$ ：贪婪最佳优先搜索