人工智能原理与技术第四周作业

2153067 王灏博

问题形式化描述

> 通过研究题面信息，可以用一系列状态来抽象描述当前的情形，即用一系列三元组，第一个元素表示当前岸边的传教士人数m，第二个元素表示当前岸边的野人人数n，第三个元素表示当前船停靠的位置b（1表示在当前岸边，0表示在目标岸边），来表示。不难分析出，符合条件（即，在任何一边，如果有野人，传教士人数m>=野人人数n）的三元组分别为((3,3,1),(3,2,1),(3,1,1),(3,0,1),(2,2,1),(1,1,1),(0,3,1),(0,2,1),(0,1,1)(3,2,0),(3,1,0),(3,0,0),(2,2,0),(1,1,0),(0,3,0), (0,2,0),(0,1,0),(0,0,0))共18种状态。需要智能体完成的目标可以抽象为从其中的初始状态通过一系列动作找到目标位置，并且评估这一路径的代价。

1、初始状态

> 根据题目描述，智能体一开始的初始状态是(3,3,1)。

2、动作

> 智能体能够执行的操作可以这样考虑：
> 当b=1（船停在当前岸边时）：
> 1)m=m-1,n=n-1,b=0
> 2)m=m-2,b=0
> 3)n=n-2,b=0
> 4)m=m-1,b=0
> 5)n=n-1,b=0
>
> 当b=0（船停在目标岸边时）：
> 6)m=m+1,n=n+1,b=1
> 7)m=m+2,b=1
> 8)n=n+2,b=1
> 9)m=m+1,b=1
> 10)n=n+1,b=1
>
> 如果没有特殊规定，每次动作的代价假设为1，到达的下一个状态就按照动作规定完成（如(3,3,1)执行了第1)个动作时，状态变为(2,2,0)）。
> 要注意每一次的动作都应该进入一个合法的状态，否则就不能执行这样的动作。
> 同时，注意到如果一个状态如果在搜索的过程中经历了两次，就会陷入死循环，所以要注意排除重复的状态。

3、目标测试

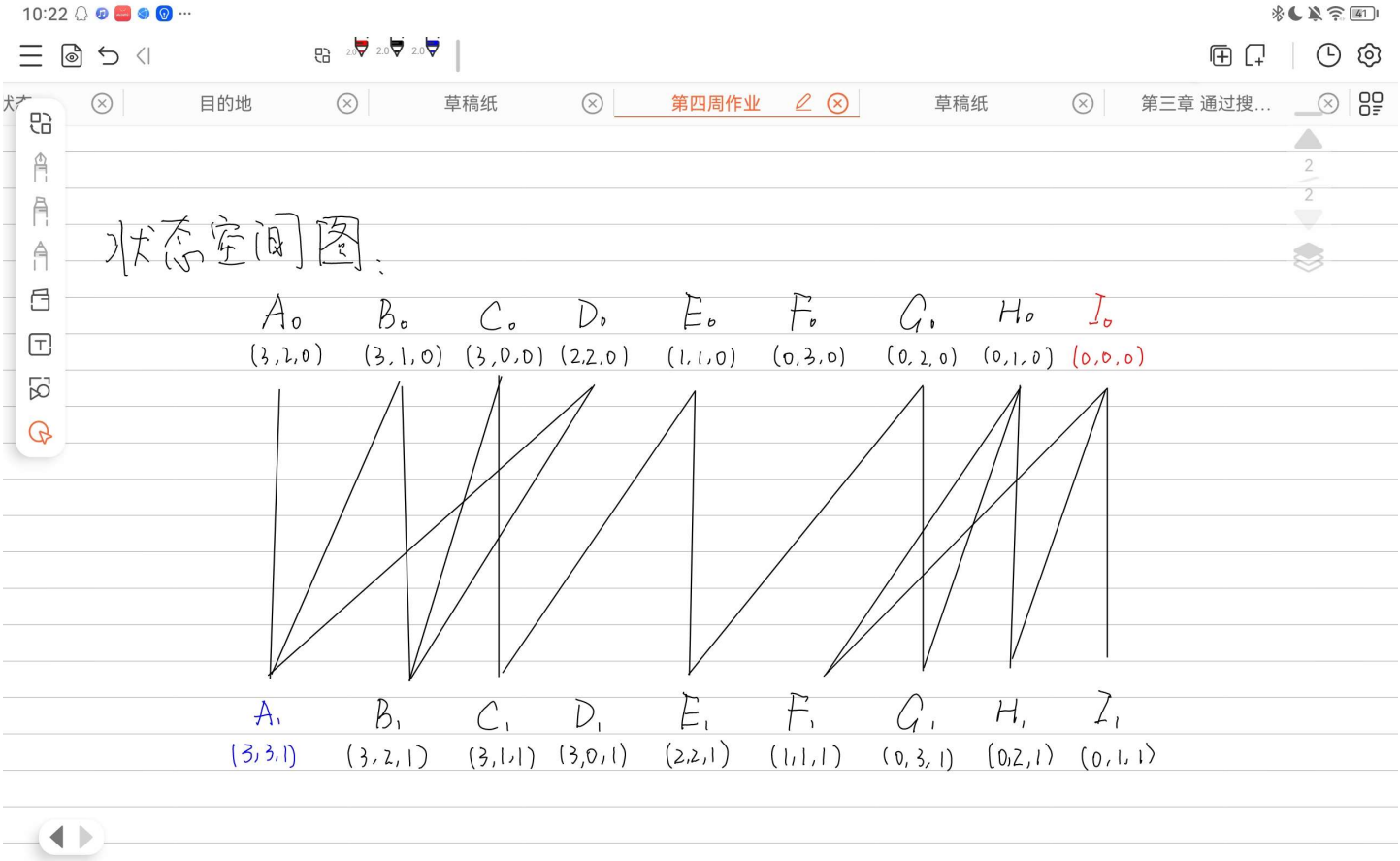
> 题目的目标是把三个传教士和三个野人全部运到对岸，所以目标状态应该是(0,0,0)，如果在搜索的过程中找到了这个状态，就将这个路径作为一种解法返回。

4、路径代价

> 如果没有特殊要求，通过完成搜索后返回的解法中的操作次数作为其代价；如果每步动作有一定的权值，也可以在搜索的过程中求和每一步的代价，最后返回。

状态空间图和搜索树

状态空间图



搜索树

搜索算法设计

代码设计

> 代码设计分为两部分，即搜索到第一个解就返回和搜索所有解法后返回，其python代码贴于文档末尾。

搜索序列和解序列

> 将所有的状态分别编码为:
> A0,B0,C0,D0,E0,F0,G0,H0,I0
> A1,B1,C1,D1,E1,F1,G1,H1,I1

搜索序列

> 找到第一个解就返回:
> A1,D0,B1,C0,C1,E0,E1,G0,G1,H0,F1,I0
> 找到所有解:
> A1,D0,B1,C0,C1,E0,E1,G0,G1,H0,F1,I0,H1,I0,B0,B0,B1,C0,C1,E0,E1,G0,G1,H0,F1,I0,H1,I0

解序列

> 找到第一个解就返回:
> A1,D0,B1,C0,C1,E0,E1,G0,G1,H0,F1,I0
> 找到所有解:
> A1,D0,B1,C0,C1,E0,E1,G0,G1,H0,F1,I0

> A1,D0,B1,C0,C1,E0,E1,G0,G1,H0,H1,I0
> A1,B0,B1,C0,C1,E0,E1,G0,G1,H0,F1,I0
> A1,B0,B1,C0,C1,E0,E1,G0,G1,H0,H1,I0

附：源代码

```
#搜索找到第一个解
statesAvailable1=((3,3,1),(3,2,1),(3,1,1),
                  (3,0,1),(2,2,1),(1,1,1),
                  (0,3,1),(0,2,1),(0,1,1)
                  )
statesAvailable0=((3,2,0),(3,1,0),(3,0,0),
                  (2,2,0),(1,1,0),(0,3,0),
                  (0,2,0),(0,1,0),(0,0,0)
                  )
def depthFirstSearch(startState:tuple,path:list,visitedStates:set):
    possibleStates=[]
    if path[-1]==(0,0,0) and (0,0,0) in visitedStates:
        print(path)
        return None
    if startState[2]==1:
        for i in range(0,5):
            m=0
            c=0
            b=0
            if i==0:
                m=startState[0]-1
                n=startState[1]-1
            elif i==1:
                m=startState[0]-2
                n=startState[1]
            elif i==2:
                m=startState[0]
                n=startState[1]-2
            elif i==3:
                m=startState[0]-1
                n=startState[1]
            elif i==4:
                m=startState[0]
                n=startState[1]-1
            stateTemp=(m,n,b)
            if stateTemp in statesAvailable0 and stateTemp not in visitedStates :
                visitedStates.add(stateTemp)
                possibleStates.append(stateTemp)
                path.append(stateTemp)
                depthFirstSearch(stateTemp,path,visitedStates)
                path.pop()
    else:
        for i in range(0,5):
            m=0
            c=0
            b=1
            if i==0:
                m=startState[0]+1
                n=startState[1]+1
            elif i==1:
                m=startState[0]+2
                n=startState[1]
            elif i==2:
                m=startState[0]
                n=startState[1]+2
            elif i==3:
                m=startState[0]+1
                n=startState[1]
            elif i==4:
```

```
                m=startState[0]
                n=startState[1]+1
            stateTemp=(m,n,b)
            if stateTemp in statesAvailable1 and stateTemp not in visitedStates :
                visitedStates.add(stateTemp)
                possibleStates.append(stateTemp)
                path.append(stateTemp)
                depthFirstSearch(stateTemp,path,visitedStates)
                path.pop()
    return None

if __name__ == '__main__':
    startState=(3,3,1)
    path=[startState]
    visitedStates=set()
    visitedStates.add(startState)
    depthFirstSearch(startState,path,visitedStates)
```

```
#搜索找到所有解
statesAvailable1=((3,3,1),(3,2,1),(3,1,1),
                  (3,0,1),(2,2,1),(1,1,1),
                  (0,3,1),(0,2,1),(0,1,1)
                  )
statesAvailable0=((3,2,0),(3,1,0),(3,0,0),
                  (2,2,0),(1,1,0),(0,3,0),
                  (0,2,0),(0,1,0),(0,0,0)
                  )
def depthFirstSearch(startState:tuple,path:list,visitedStates:set,paths:list):
    possibleStates=[]
    if startState[2]==1:
        for i in range(0,5):
            m=0
            c=0
            b=0
            if i==0:
                m=startState[0]-1
                n=startState[1]-1
            elif i==1:
                m=startState[0]-2
                n=startState[1]
            elif i==2:
                m=startState[0]
                n=startState[1]-2
            elif i==3:
                m=startState[0]-1
                n=startState[1]
            elif i==4:
                m=startState[0]
                n=startState[1]-1
            stateTemp=(m,n,b)
            if stateTemp==(0,0,0):
                path.append(stateTemp)
                paths.append(path[:])
                path.pop()
                continue
            if stateTemp in statesAvailable0 and stateTemp not in path :
                visitedStates.add(stateTemp)
                possibleStates.append(stateTemp)
                path.append(stateTemp)
                depthFirstSearch(stateTemp,path,visitedStates,paths)
                path.pop()
    else:
        for i in range(0,5):
            m=0
            c=0
            b=1
            if i==0:
                m=startState[0]+1
                n=startState[1]+1
            elif i==1:
                m=startState[0]+2
                n=startState[1]
            elif i==2:
                m=startState[0]
                n=startState[1]+2
            elif i==3:
                m=startState[0]+1
                n=startState[1]
            elif i==4:
```

```
                m=startState[0]
                n=startState[1]+1
            stateTemp=(m,n,b)
            if stateTemp==(0,0,0):
                path.append(stateTemp)
                paths.append(path[:])
                path.pop()
                continue
            if stateTemp in statesAvailable1 and stateTemp not in path :
                visitedStates.add(stateTemp)
                possibleStates.append(stateTemp)
                path.append(stateTemp)
                depthFirstSearch(stateTemp,path,visitedStates,paths)
                path.pop()
    return None

if __name__ == '__main__':
    startState=(3,3,1)
    path=[startState]
    paths=[]
    visitedStates=set()
    visitedStates.add(startState)
    depthFirstSearch(startState,path,visitedStates,paths)
    print(paths)
```