

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 人工智能实验 | | | | |
| 实验编号： | 实验三 | | | | |
| 实验名称： | 传教士野人过河问题 | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 18级计算机科学与技术（创新班） | | | |
| 实验日期： | 2020.10.20 | | | | |
| 实验室： | 2060402 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 实验目的

【传教士和食人者问题】

在河的左岸有3个传教士、1条船和3个食人者，传教士们想用这条船将所有的成员运到河的右岸，但是受到以下条件的限制：

（1）船每次最多只能装2个乘客（传教士和食人者都会划船）；

（2）在任何岸边，如果食人者数目超过传教士则传教士将被食人者吃掉。

（3）假定食人者会服从任何一种过河安排。

只输出一个解，但是输出一个解后会继续运行，如果想要得到全部解，需要稍加修改输出部分，本代码使用迭代法，深度优先策略。

# 二、实验要求

## 完成实验后请填写实验报告并上交。要求在实验报告中必须记录实验中遇到的问题及其问题解决方案。

# 三、实验内容

代码：

import operator

#判断是否是目标状态

def isgoal(node):

if node[0]==0 and node[1]==0 and node[2]==0:

return True

else:

return False

#判断是否是合法状态

def islegal(node):

if 0<=node[0]<=3 and 0<=node[1]<=3:

if node[0]==node[1] or node[0]==3 or node[0]==0:

return True

else:

return False

else:

return False

#判断是否在closed表中

def isclosed(node):

global closed

flag=0

for n in closed:

if operator.eq(n,node):

flag=1

break

if flag==1:

return True

else:

return False

#判断是否在opened表中

def isopen(node):

global opened

flag=0

for n in opened:

if operator.eq(n,node):#因为在python3中已经不存在python2中的cmp函数，用operator模块代替

flag=1

break

if flag==1:

return True

else:

return False

#将结点扩展，求子结点

#按照五条可能的操作规则：(1,0),(0,1),(1,1),(2,0),(0,2)

#返回一个包含有五个元组的元组，元组可以作为函数返回值

def Expand(node):

b=node[2]

node1=()

node2=()

node3=()

node4=()

node5=()

tnode=()

if b==1:

node1=(node[0]-1,node[1],0)#注意node是元组，不允许修改，也就是说node1[0]=node[0]-1，这样的写法是错误的，所以直接赋值即可

node2=(node[0],node[1]-1,0)

node3=(node[0]-1,node[1]-1,0)

node4=(node[0]-2,node[1],0)

node5=(node[0],node[1]-2,0)

else:

node1=(node[0]+1,node[1],1)

node2=(node[0],node[1]+1,1)

node3=(node[0]+1,node[1]+1,1)

node4=(node[0]+2,node[1],1)

node5=(node[0],node[1]+2,1)

tnode=(node1,node2,node3,node4,node5)

return tnode

#将合法的，且不在closed表中（即还没有被访问过，或者不是之前的有效路径中的已经被删除的）的子结点添加到opened表中

#使用的insert方法在opened表的表头存入结点，使得opened列表相当于栈的功能，后进先出，实现深度优先搜索

def Toopen(tnode):

global opened

for m in tnode:

if islegal(m) and isclosed(m)==False:

opened.insert(0,m)

#将合法的，且不在closed表中（即没有被访问过,或者不是之前的有效路径中的已经被删除的）的子结点和得到的对应的操作添加到result字典中

#注意必须要求不在closed表中，否则字典会将之前的值覆盖，如果没有这个条件就会发生“即便这个结点不是有效路径的，也会将其和对应操作存进来，覆盖掉之前有效结点的值”的情况

def Toresult(ltnode):

global result

if isclosed(ltnode[0])==False and islegal(ltnode[0]):

result[ltnode[0]]=(1,0)

if isclosed(ltnode[1])==False and islegal(ltnode[1]):

result[ltnode[1]]=(0,1)

if isclosed(ltnode[2])==False and islegal(ltnode[2]):

result[ltnode[2]]=(1,1)

if isclosed(ltnode[3])==False and islegal(ltnode[3]):

result[ltnode[3]]=(2,0)

if isclosed(ltnode[4])==False and islegal(ltnode[4]):

result[ltnode[4]]=(0,2)

#判断closed表中的结点的子结点是否在opened表中或在closed表中

#即当结点已被访问，不是目标状态，且不存在子结点或子结点下不存在有效路径，将其从closed表中删除

#因为最后closed表中存放的会是一条有效路径，所以需要将无效结点删除

def cino():

global closed

for k in closed:

flag1=0

for x in Expand(k):

if isopen(x) or isclosed(x):

flag1=1

if flag1==0 :

closed.remove(k)

node=(3,3,1)#初始结点状态

opened=[]

closed=[]

result={}#使用字典结构，可以利用键值对来存放结点状态和对应的操作，而且，即便是已经访问过，后因为路径无效又被删除的结点，在另一条路径被重新访问时，也可以覆盖原来的值（也就是操作）

opened.insert(0,node)

while opened:

nodet=opened.pop(0)#pop操作直接删除指定位置的元素并返回值，取出最开始的元素，后进先出，实现深度优先

if isgoal(nodet):

closed.append(nodet)

for i in range(0,len(closed)):

if (i+1)<len(closed):

print("step",i+1," 过河前(传教士，食人者，船的位置)：",closed[i]," 过河（传教士，食人者）：",result[closed[i+1]]," 过河后(传教士，食人者，船的位置)：",closed[i+1])

#已经搜索到目标结点，输出closed表中的有效路径，并从result字典中寻找状态对应的操作并输出

break

if isclosed(nodet)==False:

closed.append(nodet)#若不在closed表中，将其添加进来

snode=Expand(nodet)#求该结点的子结点，用元组返回

Toopen(snode)#将有效子结点添加到opened表中

Toresult(snode)#将有效结点和对应操作存入result字典中

cino()#将closed表中的无效结点删除

运行截图：

