

**人工智能实验报告**

题 目 修道士和野人问题

专 业 人工智能

学　　 号

姓 名

# 一 背景简介/问题描述

设有3个修道士和3个野人来到河边，打算用一条船从河的左岸渡到河的右岸。但该船每次只能装载两个人，在任何岸边野人的数目都不得超过修道士的人数，否则修道士就会被野人吃掉。假设野人服从任何一种过河安排，请问如何规划过河计划才能把所有人都安全地渡过河去。

# 二 算法介绍

**A\*算法介绍**：

初始化待考察节点集合（Openlist）和已考察节点集合（Closedlist）；

将起点加入Openlist中，并设置其代价为0；

1.如果Openlist不为空，则从Openlist中选取代价最小的节点n：

2.如果节点n为终点，则：从终点开始逐步追踪父节点，一直达到起点；

返回找到的结果路径，算法结束；

3.如果节点n不是终点，则：将节点n从Openlist中删除，并加入Closedlist中；遍历节点n所有的邻近节点：

如果邻近节点m在Closedlist中，则：跳过，选取下一个邻近节点；

如果邻近节点m不在Openlist中，则：设置节点m的父节点为节点n；

计算节点m的代价（包括距离起点的历史代价和距离终点的预计代价）；将节点m加入Openlist中；

## 2.1 状态空间的定义

输入M,N,K 分别代表初始传教士,野人,船容量

(m,c,b) 为状态空间

m表示左岸的传教士个数

c代表左岸的野人个数

b代表船的状态,1在左岸,0在右岸

m,的范围为0 − M

c的范围为0 − C

b的范围为(0,1)

目标值为(0,0,0)

## 2.2 需要满足的规律：

当船上传教士数量>0时,野人数目<=传教士数目

野人数目+传教士数目<船的容量

野人数目!=传教士数目!=0

## 2.3启发式函数的定义

f ( n ) = d ( n ) + w ( n )

其中d ( n )为搜索树的深度,也可以理解为当前是第几轮循环

w ( n )为当前状态到目标状态的实际最小费用的估计值,

在野人传教士问题中,定义为

m + c − K ∗ b f ( n )

# 三 算法实现

## 3.1 ：使用python语言

import queue

import numpy as np

import prettytable as pt

M = int(input("请输入传教士的人数：")) # 传教士

C = int(input("请输入野人的人数:")) # 野人

K = int(input("请输入船的最大容量："))

class Node:

def \_\_init\_\_(self, m, c, b, parent, step):

self.data = np.array([m, c, b])

self.m = m # m:左岸传教士数量

self.c = c # c:左岸野人数量

self.b = b # b:船的状态 b=1->船在左岸;b=0->船在右岸

self.step = step

self.f\_loss = self.step + m + c - K \* b # f(n)

self.parent = parent

def sorte\_by\_floss():

tmp\_open = opened.queue.copy()

length = len(tmp\_open)

# 排序,从小到大,当一样的时候按照step的大小排序

for i in range(length):

for j in range(length):

if tmp\_open[i].f\_loss < tmp\_open[j].f\_loss:

tmp = tmp\_open[i]

tmp\_open[i] = tmp\_open[j]

tmp\_open[j] = tmp

if tmp\_open[i].f\_loss == tmp\_open[j].f\_loss:

if tmp\_open[i].step > tmp\_open[j].step:

tmp = tmp\_open[i]

tmp\_open[i] = tmp\_open[j]

tmp\_open[j] = tmp

opened.queue = tmp\_open

'算法'

opened = queue.Queue() # open表

start\_node = Node(M, C, 1, None, 0) # 初始状态节点

end\_data = np.array([0, 0, 0])

opened.put(start\_node)

closed = {}

def refresh\_open(now\_node):

'''

:param now\_node: 当前的节点

:return:

'''

tmp\_open = opened.queue.copy() # 复制一份open表的内容

for i in range(len(tmp\_open)):

'''这里要比较一下node和now\_node的区别,并决定是否更新'''

data = tmp\_open[i]

now\_data = now\_node.data

if (data == now\_data).all():

data\_f\_loss = tmp\_open[i].f\_loss

now\_data\_f\_loss = now\_node.f\_loss

if data\_f\_loss <= now\_data\_f\_loss:

return False

else:

print('')

tmp\_open[i] = now\_node

opened.queue = tmp\_open # 更新之后的open表还原

return True

tmp\_open.append(now\_node)

opened.queue = tmp\_open # 更新之后的open表还原

return True

def data\_to\_int(num):

value = 0

for i in num:

value = value \* 10 + i

return value

def is\_safe(node):

'判断当前的节点 两岸是否安全'

m, c, b = node.data

if m < c and m != 0: # 左岸的野人人数不得超过传教士,除非传教士为0

return False

if M - m < C - c and m != M: # 右岸的野人人数不得超过传教士,除非传教士为0

return False

if m < 0 or c < 0: # 人数不得小于0

return False

if m >= c or M - m >= C - c:

return True

def method\_a\_function():

while len(opened.queue) != 0:

node = opened.get()

if (node.data == end\_data).all():

return node

closed[data\_to\_int(node.data)] = 1 # 取出的点加入closed表中

for i in range(M + 1): # 上船传教士

for j in range(C + 1): # 上船野人

# 船上非法情况

if i + j == 0 or i + j > K or (i != 0 and i < j):

continue

# a=a+1

if node.b == 1: # 当前船在左岸，下一状态统计船在右岸的情况

child\_node = Node(node.m - i, node.c - j, 0, node, node.step + 1)

else: # 当前船在右岸，下一状态统计船在左岸的情况

child\_node = Node(node.m + i, node.c + j, 1, node, node.step + 1)

if data\_to\_int(child\_node.data) not in closed:

# 更新一下open表,如果当前的状态已经在open表,选择性添加一下

if is\_safe(child\_node.data):

refresh\_open(child\_node)

'''为open表进行排序,根据其中的f\_loss值'''

sorte\_by\_floss()

return None

result\_node = method\_a\_function()

def output\_result(node):

all\_node = [node]

for i in range(node.step):

father\_node = node.parent

all\_node.append(father\_node)

node = father\_node

return list(reversed(all\_node))

def cal\_tb\_row(node, option):

data = node.data

m, c, b = data

return [node.step, m, c, option[node.step], M - m, C - c]

def cal\_option(node\_list):

option = []

for i in range(len(node\_list) - 1):

now\_m, now\_c, now\_b = node\_list[i].data

next\_m, next\_c, next\_b = node\_list[i + 1].data

mr = abs(next\_m - now\_m)

cr = abs(next\_c - now\_c)

mr = f'{mr} missionary' if mr else ''

cr = f'{cr} cannibals ' if cr else ''

status = 'turn right,' + mr + ' ' + cr if now\_b == 1 else 'turn left,' + mr + ' ' + cr

option.append(status)

option.append('success!')

return option

def out\_put():

node\_list = output\_result(method\_a\_function())

option = cal\_option(node\_list)

tb = pt.PrettyTable()

field\_names = ['step', 'm\_left', 'c\_left', 'option', 'm\_right', 'c\_right']

tb.field\_names = field\_names

for node in node\_list:

tb.add\_row(cal\_tb\_row(node, option))

print(tb)

out\_put()

# 四 讨论及结论

## 4.1 结果

以3传教士3野人一船最多载2人为例，结果如下，可以保证任意时刻都满足题干要求：

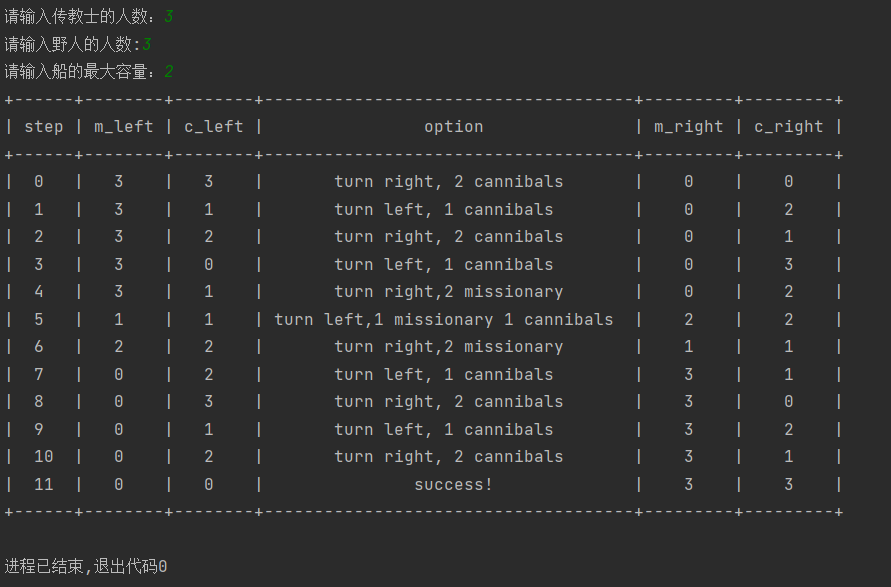


图2 3传教士3野人一船最多载2人结果示例，其中m表示传教士，c表示野人，option表示操作。

## 4.2 结论

本次作业通过查询资料了解了修道士和野蛮人问题的由来和含义，然后采用回溯算法来完成了这个项目的主体部分，简要输出结果的同时，拓展了更多数量问题和启发式算法。

**参考文献**

[1]<https://blog.csdn.net/jjsd__gulugulu/article/details/127843486>

[2]<https://blog.csdn.net/qq_43063093/article/details/106057923>

[3]<https://blog.csdn.net/weixin_44001813/article/details/104891315>