1. The missionaries and cannibals problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem formulation from analytical viewpoint (Amarel, 1968).

a. Formulate the problem precisely, making only those distinctions necessary to ensure avalid solution. Draw a diagram of the complete state space.

（1）设定状态变量及确定值域。

为了建立这个问题的状态空间，设左岸传教士数为m，则$m=\{0,1,2,3\}$；对应右岸的传教士数为$3-m$；左岸的野人数为c，则有$c=\{0,1,2,3\}$，对应右岸野人数为$3-c$；左岸船数为b，设左岸有船的情况为 0，无船的情况为 1，故有$b=\{0,1\}$，对应右岸的船数为$1-b$。

（2）确定状态组，分别列出初始状态集和目标状态集。

问题的状态可以用一个三元数组来描述，以左岸的状态来标记，即$S_k=$（m,c,b），右岸的状态可以不必标出。m 为左岸传教士人数，c 为左岸野人人数，b 左岸的船数目。

初始状态一个：$S_0=(3,3,1)$，初始状态表示全部成员在河的左岸；目标状态也只一个：$S_g=(0,0,0)$，表示全部成员从河左岸渡河完毕。

（3）定义并确定操作集；

仍然以河的左岸为基点来考虑，把船从左岸划向右岸定义为P_{ij}操作。

其中,第一下标i表示船载的传教士数,第二下标j表示船载的野人数；

同理，从右岸将船划回左岸称之为Q_{ij}操作，下标的定义同前。则共有 10 种操作，操作集为

$$F=\{P_{01},\ P_{10},\ P_{11},\ P_{02},\ P_{20},\ Q_{01},\ Q_{10},\ Q_{11},\ Q_{02},\ Q_{20}\}$$

（4）估计全部状态空间数，并尽可能列出全部状态空间或予以描述之；

在这个问题世界中，$S_0=(3,3,1)$为初始状态，$S_{31}=S_g=(0,0,0)$为目标状态。全部的可能状态共有 32 个，如表所示，m为左岸传教士人数，c为左岸野人人数，b左岸的船数目。

表 1 全部的状态空间

状态	m, c, b	状态	m, c, b	状态	m, c, b	状态	m, c, b
S_0	3 3 1	S_8	~~1 3 1~~	S_{16}	~~3 3 0~~	S_{24}	~~1 3 0~~
S_1	3 2 1	S_9	~~1 2 1~~	S_{17}	3 2 0	S_{25}	~~1 2 0~~
S_2	3 1 1	S_{10}	1 1 1	S_{18}	3 1 0	S_{26}	1 1 0
S_3	~~3 0 1~~	S_{11}	~~1 0 1~~	S_{19}	3 0 0	S_{27}	~~1 0 0~~
S_4	~~2 3 1~~	S_{12}	0 3 1	S_{20}	~~2 3 0~~	S_{28}	~~0 3 0~~
S_5	2 2 1	S_{13}	0 2 1	S_{21}	2 2 0	S_{29}	0 2 0
S_6	~~2 1 1~~	S_{14}	0 1 1	S_{22}	~~2 1 0~~	S_{30}	0 1 0
S_7	~~2 0 1~~	S_{15}	~~0 0 1~~	S_{23}	~~2 0 0~~	S_{31}	0 0 0

注：删除线为不需要考虑的状态空间。

~~红色~~表示：左岸野人会袭击传教士的情况，即 $m<c$，且$m\neq 0, S_4, S_8, S_9, S_{20}, S_{24}, S_{25}$

~~绿色~~表示： S_{15}左岸无人却有船，S_{16}去右岸时，船未携带人。

~~黄色~~表示：右岸野人会袭击传教士的情况，即$3-m<3-c$，$S_6, S_7, S_{11}, S_{22}, S_{23}, S_{27}$。

~~蓝色~~表示：由于S_9，S_{25} 不可能出现，所以S_3不可能出现，由于S_9，S_{25}不可能出现，所以S_{28}能出现。

b. Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

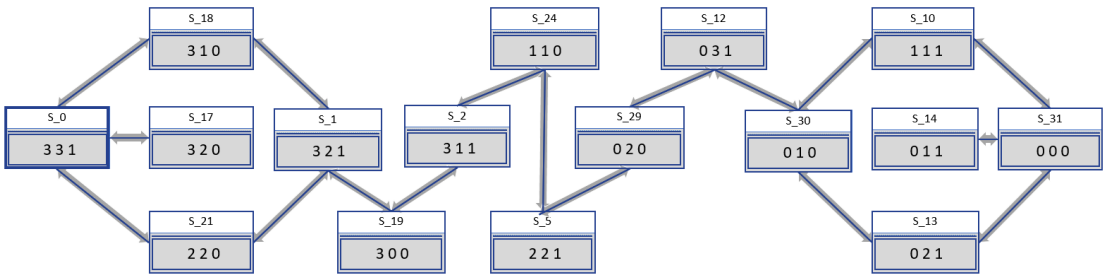

图 1 状态空间图

可用状态 16 种，考虑到任何一条从 S0 到达 S31 的路径都是该问题的解，最短路径共有 4 条。

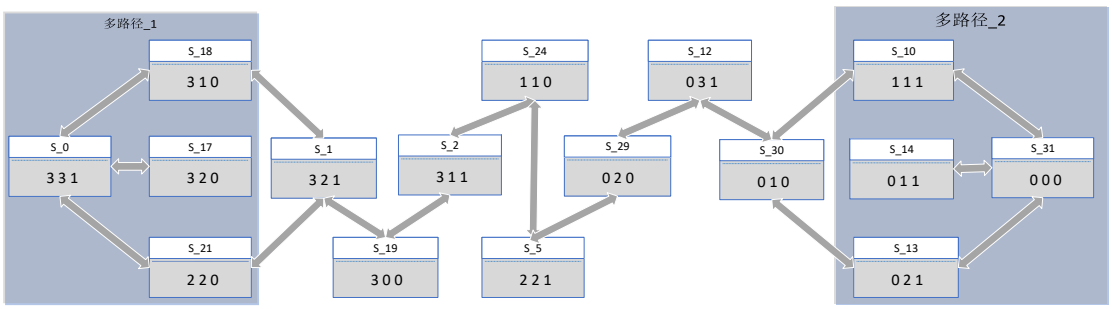

图 2 考虑多路径的状态空间图

c. Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

根据条件：右岸->左岸之间可去 1 人，右岸->左岸可去 2 人，同时两个可逆，遍历出符合条件的状态空间，并对这些状态空间进行图的建立，事实上比起上题中列举所有可能情景按条件删除，以上这个条件进行点的列举不会考虑~~绿色，蓝色，~~中提到的结点，当c和m的值进行改变时，具有更高的普适性。在图建立后用 DFS 深度优先遍历搜索算法进行搜索遍历，即最优解法，深度优先算法过程见图 3 图 4。当探索最短路径时，不需要确认已经重复的节点，若确认已经重复的节点，探索的路径就不是最短路径。

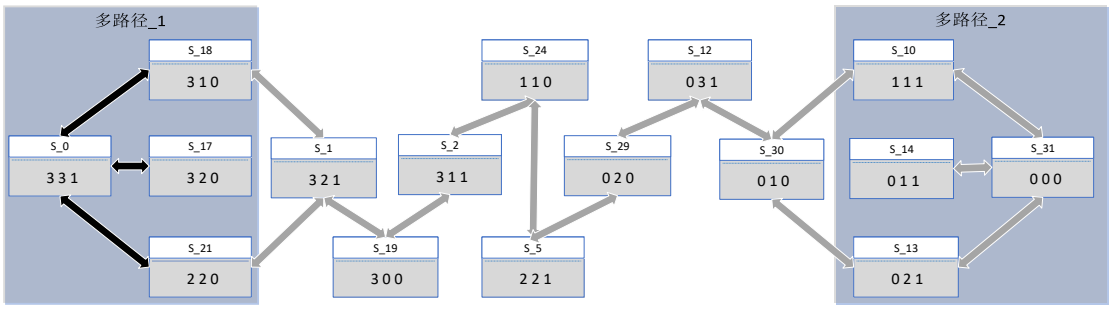

图 3 深度优先搜索算法开始情形

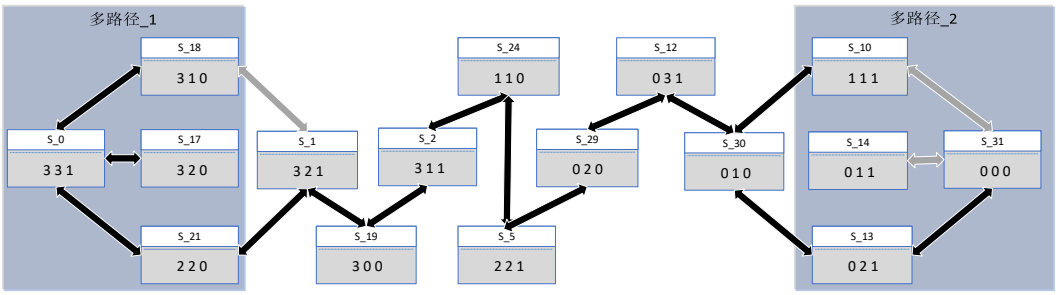

图 4 深度优先搜索算法结束情形

图 5 图 6 为基于 python 中 networkx 包的实验结果，从图上可发现除找出最优路径之外，还有 3 种同等的最优路径，图 6 主要是修改 m 和 c 的实验结果拓展，某些情况无最短路径生成，应该加以限制。由于 b 修改会导致先前设定的条件(右岸->左岸之间可去 1 人，右岸->左岸可去 2 人，同时两个可逆)改变，这种情况加入判断语句进行控制即可，暂时没有进一步完善。代码部分见附录。

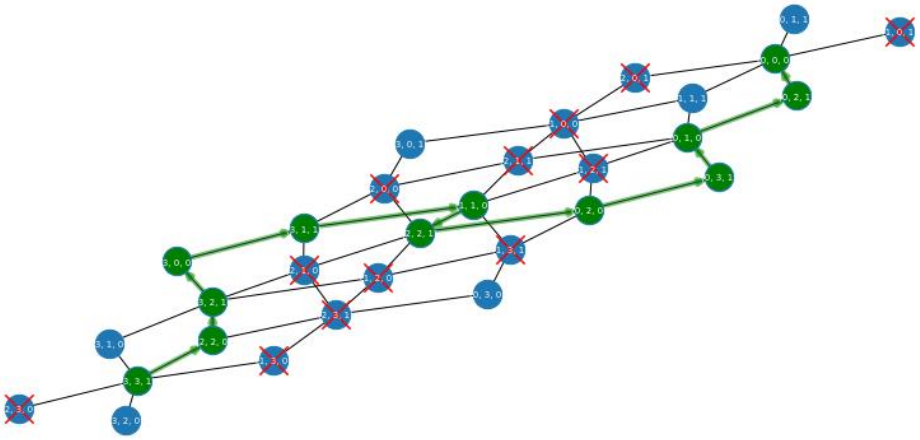

图 5 m=3,c=3,b=2 实验结果

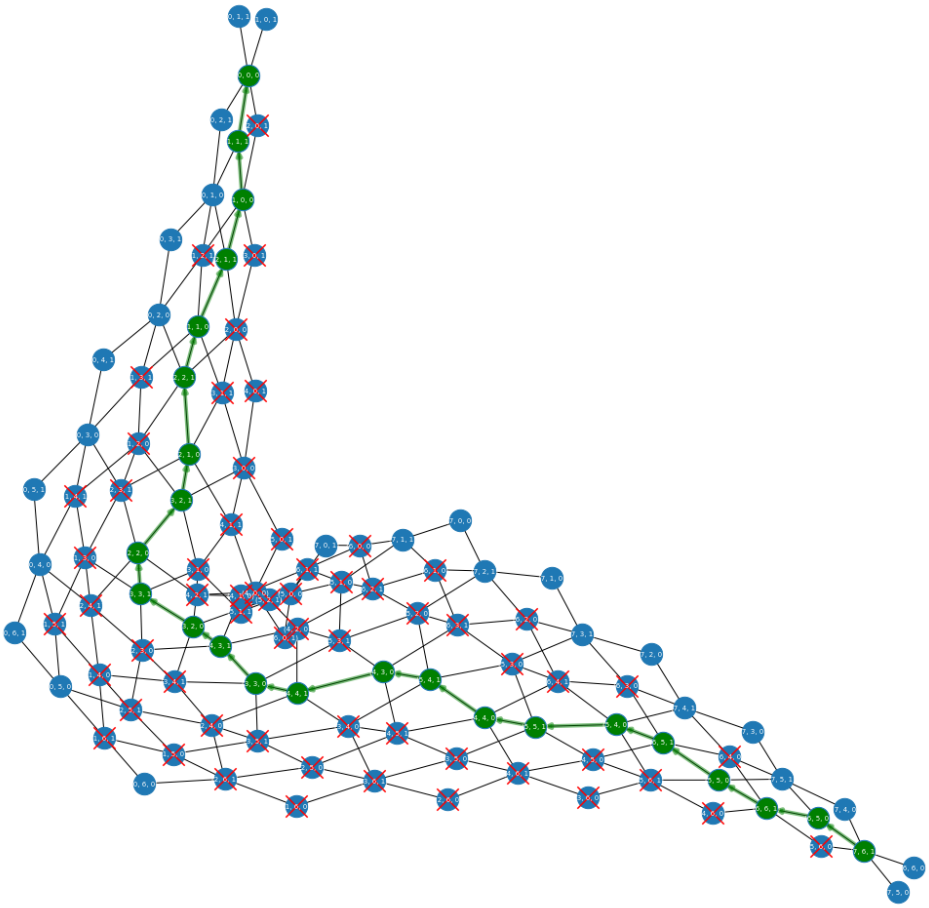

图 6 图 6 m=7,c=6,b=2 实验结果

d. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

这类问题的求解是通过搜索找到一个从初始状态到目标状态的最短路径，而无法用传统的数学方法进行求解。当行为数量不是很大时，按问题的有序元组可以画出状态空间图。但是当行为数量很大的时候，仅仅靠人力进行穷举会花费很多资源。

2. Define in your own words the following terms: state, state space, search tree, search node, goal, action, transition model, and branching factor.

State：状态是问题在不同时期或条件下的形态，即描述某一类事物在不同时刻所处于的信息状况，有初始状态（问题提出的状态）和目标状态（问题预期的状态）之分。
state space： 状态空间可以形式化地定义为初始状态、动作、转换模型。是问题在不同时期或条件下的所有状态的有机组成。例如可抽象为一个三元序组$< \boldsymbol{S}, \boldsymbol{F}, \boldsymbol{G} >$，其中$\boldsymbol{S}$表示问题的全部初始状态的集合，$\boldsymbol{F}$表示操作的集合，$\boldsymbol{G}$表示目标状态的集合。
search tree: 以有向图表示的问题状态空间。
search node: 将每个状态的表征作为一个黑盒子，即不考虑其内部结构。
goal: 设定一个目标状态，即达到这个目标则终止。
Action: 描述智能体可执行的动作。
transition model: 描述每个动作做什么。
branching factor: 可提供给智能体的行为数量。例如在搜索树中可表示为每一层选择下一层的选择方式的数量。

附录

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 12 18:54:11 2022

@author: dobby
"""

import networkx as nx
import matplotlib.pyplot as plt

WALKABLE = 'walkable'
PARENT = 'parent'
VISITED = 'visited'

#3维网格图
def my_graph(m,c,b):
    plt.subplots(1,1,figsize=(20,20))
    G=nx.Graph()
    #若b发生变化则以下结点的生成规则需要更改
    G.add_edges_from([
    ((x,y,0), (x,y+1,1))
    for x in range(m+1)
    for y in range(c)
    ] + [
    ((x,y,0), (x+1,y,1))
    for x in range(m)
    for y in range(c+1)
    ])#右岸->左岸之间可去1人
    G.add_edges_from([
    ((x,y,0), (x+1,y+1,1))
    for x in range(m)
    for y in range(c)
    ] + [
    ((x,y,0), (x+2,y,1))
    for x in range(m-1)
    for y in range(c+1)
    ] + [
    ((x,y,0), (x,y+2,1))
    for x in range(m+1)
    for y in range(c-1)
    ])#右岸->左岸可去2人
    pos = nx.spring_layout(G, iterations=100)
    nx.draw_networkx(G, pos=pos,
```

```
                font_size=7,
                font_color='white',
                node_size=500,
                width=1)
START = (m,c,1)
GOAL = (0,0,0)

road_closed_nodes = dummy_nodes(G,m,c,b)

nx.draw_networkx_nodes(
    G, pos,
    nodelist=road_closed_nodes,
    node_size=500,
    node_color="red",
    node_shape="x",
    label='x'
)

dfs(G, START, GOAL)# 基于栈实深度优先遍历搜索
try:
    path = find_path_by_parent(G, START, GOAL)
except:
    print("该种情况没有可行路径")
    return;
print('path', path)

nx.draw_networkx_nodes(
    G, pos,
    nodelist=path,
    node_size=400,
    node_color="green",
    node_shape='o',
)



path_edges = []
for i in range(len(path)):
    if (i + 1) == len(path):
        break
    path_edges.append((path[i], path[i + 1]))

print('path_edges', path_edges)

# 把 path 着色加粗重新描边
```

```
    G2=G.to_directed()
    nx.draw_networkx_edges(G2, pos,
                           edgelist=path_edges,
                           width=4,
                           alpha=0.5,
                           edge_color="g")

    plt.axis('off')
    plt.show()


# 基于栈的深度优先遍历搜索
def dfs(G, START, GOAL):
    for n in G.nodes():
        G.nodes[n]['visited'] = False#初始所有点都是未被访问的

    stack = []  # 用列表当作一个栈，只在栈顶操作（数组的第 1 个位置）
    stack.append(START)
    close_list = []
    while True:
        if len(stack) == 0:
            break

        print('-----')
        print('stack-', stack)

        visit_node = stack[0]
        G.nodes[visit_node]['visited'] = True
        print('访问', visit_node)

        if visit_node == GOAL:
            break

        close_list.append(visit_node)#已访问的节点

        count = 0
        neighbors = nx.neighbors(G, visit_node)#寻找 visit_node 周围的结点
        for node in neighbors:#依次遍历 visit_node 周围的节点
            visited = G.nodes[node][VISITED]
            try:
                walkable = G.nodes[node][WALKABLE]#如果存在之前设定的 WALKABLE 则为阻碍点
            except:
                walkable = True
```

```
            if (visited) or (node in stack) or (node in close_list) or (not walkable):
                continue
 #如果结点被访问，或者在将要访问的 stack 中，或者在已被记录的最短路径 close_list 中，或者是阻碍点 则跳过

            G.nodes[node][PARENT] = visit_node#设定该找到的节点 node 的父节点为 visit_node，改变 G
            stack.append(node)#将 node 压入 stack
            count = count + 1#计数

        if count == 0:#该节点下没有节点被压入
            print(visit_node, '尽头')
            del (stack[0])
            print('弹出', visit_node)

        print('stack--', stack)#显示进入下次循环的 stack

    return stack


def find_path_by_parent(G, START, GOAL):
    t = GOAL
    path = [t]
    is_find = False
    while not is_find:
        for n in G.nodes(data=True):#G.nodes 第一个是结点的数据，第二个是字典存储的结点属性
            if n[0] == t:
                parent = n[1][PARENT]#找 Node 中 parent 属性存储的信息
                path.append(parent)

                if parent == START:#找到开始结点
                    is_find = True
                    break

                t = parent

    path.reverse()#从 GOAL->START 需要翻转
    return path

def dummy_nodes(G,m,c,b):

    list1=[];
    #list1.append((m,c,0))#去右岸时，船未携带人 不需要考虑本不在生成点中
    #list1.append((0,0,1))#左岸无人却有船  不需要考虑本不在生成点中
    for i in range(m+1):
        for j in range(c+1):
```

```
            if(i<j and i!=0):#左岸野人会袭击传教士的情况
                p=(i,j,0)
                q=(i,j,1)
                list1.append(p)
                list1.append(q)

            if(m-i<c-j and m-i!=0):#右岸野人会袭击传教士的情况
                p=(i,j,0)
                q=(i,j,1)
                list1.append(p)
                list1.append(q)
    for i in list1:
        G.nodes[i][WALKABLE] = False

    return list1

if __name__ == '__main__':
    m=7#自定义修道士数
    c=6#自定义野人数
    b=2
    my_graph(m,c,b)
```