**程序报告**

学号： 2211999 姓名：邢清画

1. **问题重述**

====================================================================

**斑马问题**：5 个不同国家（英国、西班牙、日本、意大利、挪威）且工作各不相同（油漆工、摄影师、外交官、小提琴家、医生）的人分别住在一条街上的 5 所房子里，

每所房子的颜色不同（红色、白色、蓝色、黄色、绿色），每个人都有自己养的不同宠物（狗、蜗牛、斑马、马、狐狸），喜欢喝不同的饮料（矿泉水、牛奶、茶、橘子汁、咖啡）。

根据以下提示，得出哪所房子里的人养斑马，哪所房子里的人喜欢喝矿泉水。

1.英国人住在红色的房子里

2.西班牙人养了一条狗

3.日本人是一个油漆工

4.意大利人喜欢喝茶

5.挪威人住在左边的第一个房子里

6.绿房子在白房子的右边

7.摄影师养了一只蜗牛

8.外交官住在黄房子里

9.中间那个房子的人喜欢喝牛奶

10.喜欢喝咖啡的人住在绿房子里

11.挪威人住在蓝色的房子旁边

12.小提琴家喜欢喝橘子汁

13.养狐狸的人所住的房子与医生的房子相邻

14.养马的人所住的房子与外交官的房子相邻

1. **设计思想**

====================================================================

演绎推理（Deductive Reasoning）是由一般到特殊的推理方法。本题使用python逻辑编程kanren库，将题目中的条件形式化为多个表达式，加入约束到kanren的集合中，然后利用kanren内置的run求解即可得到问题的解。

优化方法**：**分析和优化每条逻辑规则，避免不必要的约束和复杂性；适当合并规则，减少求解过程中的搜索深度和广度；减少逻辑变量的数量、减少逻辑规则的复杂度；添加异常处理机制，以处理一些不常见的情况。

1. **代码内容**

====================================================================

from kanren import run, eq, membero, var, conde

from kanren.core import lall # lall包用于定义规则

import time

###############################################################################

#### 可在此处定义自己所需要用到的自定义函数(可选) ####

#### 提示：定义左邻近规则left(), 定义右邻近规则right(),定义邻近规则next() ####

###############################################################################

# #

def right(q, p, list):

return membero((p, q), zip(list, list[1:]))

def next(q, p, list): # 相邻可以只使用单侧相对位置作参考，pq其中一个必在另一个右侧

return conde([right(q, p, list)], [right(p, q, list)])

# #

###############################################################################

################# 非必要性工作 ##################

###############################################################################

class Agent:

"""

推理智能体.

"""

def \_\_init\_\_(self):

"""

智能体初始化.

"""

self.units = var()

self.rules\_zebraproblem = None

self.solutions = None

def define\_rules(self):

"""

定义逻辑规则.

"""

self.rules\_zebraproblem = lall(

(eq, (var(), var(), var(), var(), var()), self.units),

# 各个unit房子包含五个成员属性: (国家，工作，饮料，宠物，颜色)

##############################################################################

#### 请在以下区域中添加逻辑规则，感受逻辑约束问题 ####

#### 输出：五条房子匹配信息('英国人', '油漆工', '茶', '狗', '红色') ####

##############################################################################

# #

# 示例：基于问题信息可以提炼出，有人养斑马，有人喜欢和矿泉水等信息

(membero, ('英国人', var(), var(), var(), '红色'), self.units), # 英国人住在红房子里

(membero, ('西班牙人', var(), var(), '狗', var()), self.units), # 西班牙人养了一条狗

(membero, ('日本人', '油漆工', var(), var(), var()), self.units), # 日本人是一个油漆工

(membero, ('意大利人', var(), '茶', var(), var()), self.units), # 意大利人喝茶。

(eq, (('挪威人', var(), var(), var(), var()), var(), var(), var(), var()), self.units), # 挪威人住在左边的第一个房子里

(right, # 绿房子在白房子的右边

(var(), var(), var(), var(), '白色'),

(var(), var(), var(), var(), '绿色'),

self.units),

(membero, (var(), '摄影师', var(), '蜗牛', var()), self.units), # 摄影师养了一只蜗牛

(membero, (var(), '外交官', var(), var(), '黄色'), self.units), # 外交官住在黄房子里

(eq, (var(), var(), (var(), var(), '牛奶', var(), var()), var(), var()), self.units), # 中间那个房子的人喜欢喝牛奶

(membero, (var(), var(), '咖啡', var(), '绿色'), self.units), # 喜欢喝咖啡的人住在绿房子里

(next, ('挪威人', var(), var(), var(), var()),

(var(), var(), var(), var(), '蓝色'), self.units), # 挪威人住在蓝房子旁边。

(membero, (var(), '小提琴家', '橘子汁', var(), var()), self.units), # 小提琴家喜欢喝橘子汁

(next, (var(), '医生', var(), var(), var()), # 养狐狸的人所住的房子与医生的房子相邻

(var(), var(), var(), '狐狸', var()), self.units),

(next, (var(), '外交官', var(), var(), var()), # 养马的人所住的房子与外交官的房子相邻

(var(), var(), var(), '马', var()), self.units),

(membero, (var(), var(), var(), '斑马', var()), self.units), # 有人养斑马

(membero, (var(), var(), '矿泉水', var(), var()), self.units) # 有人喝水

# #

##############################################################################

################# 完成后请记得提交作业 #################

##############################################################################

)

def solve(self):

"""

规则求解器(请勿修改此函数).

return: 斑马规则求解器给出的答案，共包含五条匹配信息，解唯一.

"""

self.define\_rules()

self.solutions = run(0, self.units, self.rules\_zebraproblem)

return self.solutions

agent = Agent()

solutions = agent.solve()

# 提取解释器的输出

output = [house for house in solutions[0] if '斑马' in house][0][4]

print('\n{}房子里的人养斑马'.format(output))

1. **实验结果**

====================================================================

绿色房子里的人养斑马

黄色房子里的人喜欢喝矿泉水

('挪威人', '外交官', '矿泉水', '狐狸', '黄色')

('意大利人', '医生', '茶', '马', '蓝色')

('英国人', '摄影师', '牛奶', '蜗牛', '红色')

('日本人', '油漆工', '咖啡', '斑马', '绿色')

('西班牙人', '小提琴家', '橘子汁', '狗', '白色')

1. **总结**

====================================================================

斑马问题的成功解决不仅加深了我对逻辑编程范式的理解，也提高了我解决实际问题的能力，同时熟悉了python的编程格式，初步学会利用kanren库去解决问题。通过这种方式，我更加直观地理解问题的本质，了解了逻辑编程与命令式编程的区别，同时能够依据给定的事实以及规则编写代码，解决逻辑约束问题（CLP），从而在未来的项目中更加有效地应用逻辑编程技术。

1. 可能的改进方向：

优化逻辑规则：逻辑规则的定义方式直接影响求解过程的效率。通过优化和简化逻辑规则，可能会减少求解时的计算量，提高求解速度。例如，某些规则如果能合并或重构，可能会减少搜索空间的大小。

使用高效的数据结构：在底层实现中，使用更高效的数据结构来存储和管理逻辑规则和变量，可能会减少内存使用并加快搜索速度。例如利用哈希表来快速查找和更新变量的约束条件。

1. 遇到的困难

斑马问题涉及多个变量和约束条件，将这些条件转化为逻辑规则时，规则之间的相互依赖关系需要仔细处理，否则逻辑上会出现冲突。

错误不直接明显，隐藏在逻辑规则中。

1. 提升性能的方法

分析和优化每条逻辑规则，避免不必要的约束和复杂性。适当合并规则，减少求解过程中的搜索深度和广度。