哈尔滨理工大学

实　验　报　告

课程名称：人工智能基础

预习报告	过程及记录	结果分析	实验总结	总评
教师签名：				

班　级 [redacted]

学　号 [redacted]

姓　名 [redacted]

指导教师 史云玲

教务处 印制

实验室安全管理个人注意事项

1.进入实验室工作、实验和研究人员必须进行实验室安全承诺，务必遵守学校及实验室各项规章制度和仪器设备操作规程。

2.熟悉紧急情况下的逃离路线和紧急应对措施，清楚急救箱、灭火器材、紧急洗眼装置和冲淋器的位置。

3.进行实验操作时，在做好个人防护的同时，要根据实验风险需要选择合适的实验个体防护用品。使用前应确认其使用范围、有效期及完好性等，熟悉其使用、维护和保养方法。

不得在实验室吸烟、饮食、储存食品、饮料等个人生活物品；不得做与实验、研究无关的事情。

4.触电事故特点：

4.1 被电击会导致人身伤害，甚至死亡；

4.2 短路有可能导致爆炸和火灾；

4.3 电弧或火花会点燃物品或者引燃具有爆炸性的物料；

4.4 冒失地开启或操作仪器设备可能导致仪器设备的损坏，使身体受伤；

4.5 电器过载会令其损坏、短路或燃烧。

5.触电事故的预防：

5.1 检查电线、插座和插头，一旦发现损坏要立即更换。

5.2 仪器设备开机前要先熟悉该仪器设备的操作规程，确认状态完好后方可接通电源。

5.3 当手、脚或身体沾湿或站在潮湿的地上时，切勿启动电源开关或接触电器用具。

6.触电事故应急措施：

6.1 使触电者脱离电源：应立即切断电源，可以采用关闭电源开关，用干燥木棍挑开电线或拉下电闸。救护人员注意穿上绝缘靴或站在干燥木板上，尽快使伤员脱离电源。

6.2 检查伤员：触电者脱离电源后，应迅速将其移到通风干燥的地方仰卧，并立即检查伤员情况。

6.3 急救并求医：根据受伤情况确定处理方法，对心跳、呼吸停止的，立即就地采用人工心肺复苏方法抢救，并及时拨打 120 急救电话。应坚持不懈地做心肺复苏，直到医生到达。

上述注意事项请仔细阅读后签字确认！

参加实验人员：_____（签名）

日　　期：　　年　　月　　日

实验名称	实验一 状态空间搜索	时间	2023 年 11 月 11 日
		地点	
同实验者		班组	[redacted]

一、实验预习（准备）报告

1、实验目的

状态空间表示法是以状态和操作符为基础的一种基于解答空间的问题表示和求解方法，是人工智能基本的知识表示方法之一，也是进一步学习状态空间搜索策略的基础. 本实验通过农夫过河问题，强化学生对知识表示的理解与应用，为人工智能后续环节的课程奠定基础。

2、实验相关原理及内容

- 实验原理

1. 状态空间： 在人工智能中，问题的解空间可以被表示为状态空间，其中每个状态代表问题的一个可能解。状态之间的转移由问题的规则决定。

2. 搜索算法： 状态空间搜索通过应用搜索算法来寻找问题的解。搜索算法从初始状态开始，根据问题的规则，逐步探索状态空间直到找到目标状态。

3. 深度优先搜索： 是一种基本的搜索算法，它从根节点开始，沿着树的深度探索尽可能远的节点。当节点的所有子节点都被探索过后，搜索返回到父节点，继续探索未被访问的兄弟节点。

- 实验内容

问题描述： 农夫过河问题是一个经典的状态空间问题。农夫需要带一匹狼、一只羊和一棵白菜过河，但船只能容纳农夫和另外一样东西。如果农夫不在场，狼会吃掉羊，羊也会吃掉白菜。由上述的实验内容可知，本次实验是一个经典的农夫过河问题，利用状态空间搜索对其进行寻找最优的解决方案。

算法设计思想：①状态表示： 使用状态对象表示问题的当前状态，包含农夫、狼、羊和白菜的位置信息。这里用 False 表示在右岸，True 表示在左岸，初始状态是都在右岸为 False。②目标状态：定义一个目标状态，表示问题的解。在这个问题中，目标状态是所有物品都在河的对岸。也就是目标状态是都在左岸为 True。③移动操作：设计合法的移动操作，即规定每一步农夫可以选择带一样东西过河或者独自回到原岸。④深度优先搜索：使用深度优先搜索算法在状态空间中寻找合法的过河路径。

源代码实现：使用 Python 语言编写程序，定义状态对象、移动操作和深度优先搜索函数。执行搜索算法，并输出找到过河路径的过程。

3、实验方法及步骤设计

①问题描述：确定问题背景和要解决的问题，即农夫过河问题。理解问题的规则，包括农

夫、狼、羊和白菜的过河约束。

②状态空间设计：定义问题的状态，即包含农夫、狼、羊和白菜的位置信息的状态对象。这里用列表 status[]来存储他们的状态对象，其中 status[0]存放的是农夫的状态，status[1]存放的是狼的状态，status[2]存放的是羊的状态，status[3]存放白菜的状态。并规定状态值 False 表示在右岸，True 表示在左岸，也就是布尔值为 false 说明相应的人（物）未通过河流，布尔值为 true 代表对应人（物）已经过河了。根据题目的意思可设他们初始的状态为 status[4] = [false, false, false, false]，则问题的目标状态为[True,True,True,True]。

③移动操作设计：定义合法的移动操作，即规定每一步农夫可以选择带一样东西过河或者独自回到原岸。自定义一个判断状态是否合理的函数，根据问题描述中的“如果农夫不在场，狼会吃掉羊，羊也会吃掉白菜。”来制定合理条件。

④深度优先搜索算法设计：编写深度优先搜索算法，以递归方式在状态空间中搜索合法的过河路径。在搜索过程中，维护访问过的状态集合，避免重复搜索。并且每生成一个有效新状态都判断是否达到最终目标状态。如果到达则输出整个过程。

⑤实验程序设计：使用 Python 语言编写程序，实现状态对象、移动操作和深度优先搜索函数。编写主程序，执行搜索算法，并输出找到的过河路径。

⑥测试及验证：运行实验程序，验证搜索算法是否能够找到安全的过河路径。

⑦结果分析：分析搜索结果，检查路径是否满足问题规则，即保证在没有农夫的情况下不会发生狼吃羊或羊吃白菜的情况。

⑧实验报告撰写和总结：撰写实验报告，包括实验的背景、目的、方法、步骤、程序源代码、实验结果及分析等内容。强调状态空间搜索的基本原理和算法在解决问题中的应用。和同学讨论实验过程中遇到的问题、解决方案以及可能的改进。总结实验的收获，思考状态空间搜索在其他问题中的应用。

4、实验设备仪器选择及材料

一台笔记本电脑、pycharm 软件、A4 记录纸、截屏软件、键盘、鼠标等。

5、实验注意事项（包括实验过程、仪器设备、个人操作等方面）

（1）在编写程序时对变量的取名要有意义，能够清楚的知道其意思；

（2）使用 PyCharm 2020.1 x64 软件编写程序时，出现错误要及时修改以免错误越来越多；

（3）及时保存编写的程序，以防电脑突然死机；

（4）在进行电脑编程时，注意正确的使用电脑防止触电危险。

6、实验思考题解答

对于本实验来说利用状态空搜索法进行求解，其实就是首先设定状态变量及确定值域；再确定状态组，分别列出初始状态集和目标状态集；之后再定义并确定操作集；然后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；最后当状态数量不是很大时，按问题的

有序元组画出状态空间图，依照状态空间图搜索求解。

二、实验过程及记录

通过分析题意，我们可以理解重点在于：穷举状态，同时仍需判断动作的有效性。设定一个有四个元素的列表表示狼，羊，菜，农夫的状态。status = [0, 0, 0, 0]，这个元素对应的状态为 0 时，代表这个元素在此岸，反之在彼岸。

我们需要做的就是将这个状态由[0,0,0,0]变成[1,1,1,1]、也就是让所有的元素全部过河。但是为了避免狼吃掉羊，羊吃掉菜。我们对可行的中间状态还要做一定的设置，然后用枚举法的方式列举出来。直到有一个方案能成功让[0,0,0,0]变成[1,1,1,1]为止。下图 1 是我在编写程序之前对这个问题模拟的解答过程：

A1 fx

	A	B	C	D	E	F
		wolf	lamb	vegetables	farmer	
0		0	0	0	0	
1		0	1	0	1	
2		0	1	0	0	
3		1	1	0	1	
4		1	0	0	0	
5		1	0	1	1	
6		1	0	1	0	
7		1	1	1	1	

图 1 算法模拟过程

开始根据状态空间搜索的算法逻辑编程，测试结果如下图 2 所示：

```
实验一：状态空间搜索
D:\softwares\anaconda3\python.exe D:\softwares_saving\pycharm\deep_study_1\人工智能实验\实验一：状态空间搜索.py
scheme: 1
[[False, False, False, False], [True, False, True, False], [False, False, True, False], [True, True, True, False], [False, True, Fals
scheme: 2
[[False, False, False, False], [True, False, True, False], [False, False, True, False], [True, False, True, True], [False, False, Fals
histoy_status: [[False, False, False, False]]
finish search, find 2 scheme
```

图 2 测试过程

由于本实验是建立在解决具体问题上，因此我觉得问题答案的显示清晰易懂很重要，即使不是编写程序的人一看运行结果应该就可以明白问题的求解过程，所以我对答案显示进行了优化，力图使它们更直白，优化后的效果如下图所示：

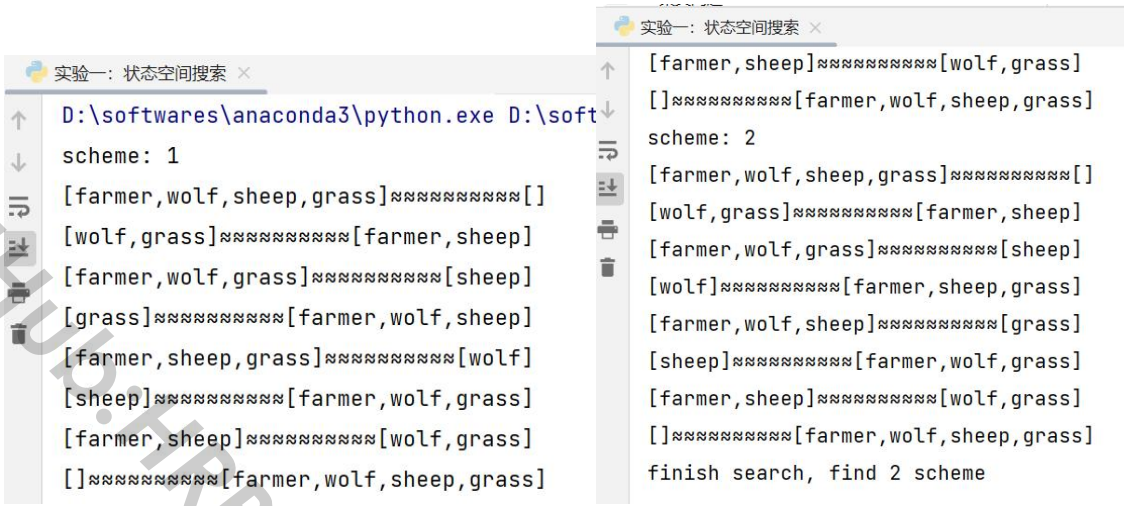

图 3 优化后的结果

三、实验结果分析

由实验结果图 3 可知，本次实验共有两种方式分别是：

方式 1：1）农夫带羊过去；2）农夫自己回来；3）农夫带狼过去；

4）农夫带羊回来；5）农夫带草过去；6）农夫自己回来；7）农夫带羊过去。

方案 2：1）农夫带羊过去；2）农夫自己回来；3）农夫带草过去；

农夫带羊回来；5）农夫带狼过去；6）农夫自己回来；7）农夫带羊过去。

因此对于本次实验的农夫过河问题可选择两种方法，这两种方法都需要七步来实现所有目标都到河对岸。

对于这个实验，由于深度优先搜索是一种盲目的搜索算法，它只是简单地按照一定规则递归地搜索状态空间直到找到目标状态或者搜索完整个状态空间。因此，会存在多种解决方案。

两种方案都采用了农夫先带着羊过河，确保在对岸时不会发生狼吃羊的情况。两者的相同点在于选择了相同的起始步骤，并且都考虑到了在农夫不在场的情况下可能发生的风险。不同点在于搜索的先后顺序，即在状态空间中的不同路径。方式 1 在第三步选择带狼过河，而方式 2 在第三步选择带白菜过河。这导致了两者在安排农夫、狼、羊和白菜过河的具体顺序上的不同。

下面用算法流程图 4 来说明我在解决这个具体问题时，使用状态搜索法的设计思想，首先是定义问题中可能的状态，每个状态表示农夫、狼、羊和白菜的位置。例如，每个状态可以用一个四元组表示：(农夫位置，狼位置，羊位置，白菜位置)。再确定问题的初始状态和目标状态。并且由于题意可以去定义合法的移动操作。接着去定义状态之间的转移规则。确定每个合法动作对应的状态转移，并确保转移后的状态是合法的。然后选择广度优先搜索（BFS），在状态空间中进行搜索，不断尝试合法的动作，直到找到满足目标状态的路径。在搜索过程中，使用剪枝策略避免重复访问已经探索过的状态，提高搜索效率。在找到目标状态时，记录路径，即从初始状态到目标状态的每一步动作，作为最终的解决方案。

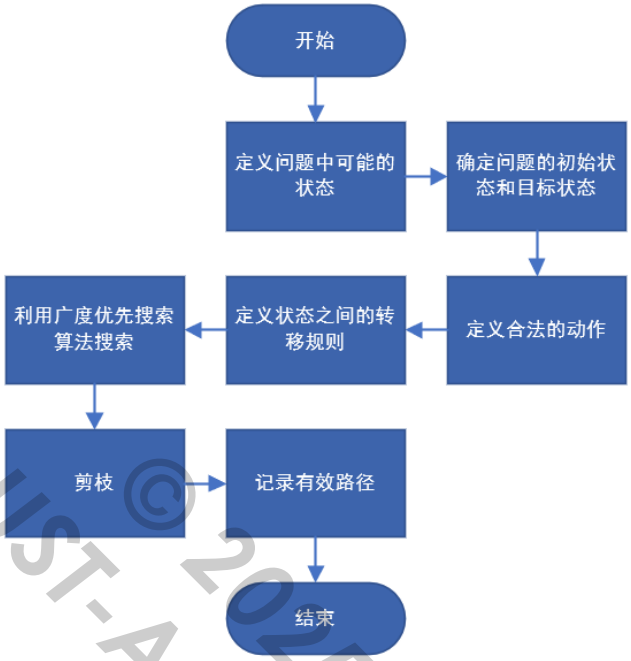

图 4 实验算法设计思想

四、实验总结（200-300 字）

在实验过程中，我遇到了一个问题，即广度优先搜索可能会导致多个不同的解决方案。在分析两种方案时，我发现它们在搜索路径的顺序上有所不同，这引发了一个问题：不同运行时可能产生不同的解。为解决这一问题，我引入了一个排序机制，确保每次搜索时都按照一定规则选择状态，使得解决方案更具一致性。这种方式能够更好地控制搜索的路径，提高了算法的可预测性。

结论上，通过状态空间搜索算法，我成功找到了两种合法的过河方案。实验强调了问题建模的重要性，清晰定义状态和规则是解决复杂问题的关键。在体会中，我意识到灵活应用搜索算法可以应对不同问题，同时解决方案的优化需要更深入地研究不同的算法和启发式方法。为了进一步优化项目，我考虑引入其他搜索算法和更智能的剪枝策略，以提高算法的效率和性能。这个实验为我提供了深刻的学习经验，使我更熟悉状态空间搜索算法的应用和优化。

附录 1--实验 1 源代码：

```
'''
False 表示在右岸，True 表示在左岸
初始状态是都在右岸为 False,目标状态是都在左岸为 True
status[0]--->存放人的状态（在左岸还是右岸）
status[1]--->存放狼的状态（在左岸还是右岸）
status[2]--->存放羊的状态（在左岸还是右岸）
status[3]--->存放白菜的状态（在左岸还是右岸）
history_status 用来记录每走过的步骤
'''
#实验一：状态空间搜索->农夫过河问题.py
name = ["farmer", "wolf", "sheep", "grass"]
scheme_count = 0
'''
题目要求是：在没有人在场的情况下，狼会吃掉羊，羊也会吃掉白菜
不合理的状态有两种：
如果人不在（status[0]!=status[2]）
狼和羊在一起（status[1]==status[2]）
羊和白菜在一起（status[2]==status[3]）
'''
#判断状态是否合理函数
def judge_status_reasonable(status):
    if status[2] == status[3]:
        if status[0]!=status[2]:
            return False
    if status[1] == status[2]:
        if status[0]!=status[2]:
            return False
    return True
'''
    以人为主角，来想想该带谁过河
还有一种情况是人一个人过河，不带东西
先看谁和人在不同的一侧，这样就不用考虑了（status[0]! =status[i]）
如果发现谁和人在同一侧，就带它过河，然后判断状态是否合理
如果合理就生成了一个新状态(记录这个步骤)，不合理就返回一个空的状态
'''
#生成下一个状态
def create_new_status(status):
    new_status_step = []
    for i in range(0,4):
        if status[0] != status[i] :
            continue  #如果在不同的一侧，就不用带过河了，不用考虑
        else:
```

```
            new_status =[status[0],status[1],status[2],status[3]]  #把当前状态提取出来
            new_status[0] = not new_status[0]   #农夫的新状态
            new_status[i] = new_status[0]      #东西的新状态
            #print("new_status:",i,new_status)
            #判断状态是否合理
            if judge_status_reasonable(new_status):
                new_status_step.append(new_status)
    return new_status_step
def search(history_status_list):
    global scheme_count
    current_status = history_status_list[len(history_status_list)-1]
    #print("current_status:",current_status)
    new_status_list = create_new_status(current_status)
    #print("new_status_list:",new_status_list)
    for new_status in new_status_list :
            if new_status in history_status_list : continue  #如果与之前的状态重复了就不要
            #如果没有重复
            history_status_list.append(new_status)
            if (new_status[0] and new_status[1] and new_status[2] and new_status[3]) : #如果到达了最终的状态
                scheme_count+=1    #说明有一种方法了
                print("scheme:",scheme_count)
                #print(history_status_list)
                print_history_status(history_status_list)
            else :
                search(history_status_list)
            history_status_list.pop()
'''
可视化结果
'''
def readable_status(status, is_across):
    result = ""
    for i in range(0,4):
        if status[i] == is_across:
            if len(result) != 0:
                result += ","
            result += name[i]
    return "[" + result + "]"  #字符串"["
#打印结果
def print_history_status(history_status):
    for status in history_status:
        print(readable_status(status, False) + "==========" + readable_status(status, True))

if __name__ == "__main__":
```

```
    # 初始状态
    status = [False, False, False, False]
    # 目标状态
    history_status = [status]
    #new_status_list = create_new_status([True, False, True, False])
    #print( new_status_list)
    search(history_status)
    #print("histoy_status:",history_status)
    print("finish search, find " + str(scheme_count) + " scheme")
```

实验名称	实验二 A*算法实验九宫格重排	时间	2023 年 11 月 11 日
		地点	
同实验者		班组	

一、实验预习（准备）报告

1、实验目的

熟悉和掌握启发式搜索的定义、估价函数，并利用 A*算法求解 N 数码问题，理解求解流程和搜索顺序。A*算法是人工智能领域重要的启发式搜索算法之一。本实验通过九宫格重排问题，强化学生对 A*算法的理解与应用，为人工智能后续环节的课程奠定基础。

2、实验相关原理及内容

- 实验原理

1. 启发式搜索： A*算法结合了广度优先搜索和启发式搜索的优点。它利用估价函数评估每个节点的代价，并选择最有可能导致最优解的节点进行拓展。启发式搜索中的估价函数为：$f(n) = g(n) + h(n)$。其中 f(n)为从起始点通过结点 n 至目标点的估价函数，g(n)为从起始结点至目标结点的实际成本，以及 h(n)为从 n 至目标结点的最优路径的估算费用。

2. 估价函数： 为了评估状态的好坏，需要设计一个估价函数。在九宫格问题中，估价函数可以是当前状态与目标状态之间的曼哈顿距离（各数字在水平和垂直方向上的距离之和）也可以是棋子数（不在正确位置的个数）。

3. A*算法：

①A*算法思想

是一种静态路网中求解最短路径最有效的直接搜索方法，也是解决许多搜索问题的有效算法。算法中的距离估算值与实际值越接近，最终搜索速度越快。

②A*算法流程

初始化：将初始状态加入待扩展的节点集合，并将其代价设为 0。

循环直至找到目标状态：

从待扩展节点集合中选取代价最小的节点进行扩展。

根据规定的操作规则生成相邻的节点。

计算每个相邻节点的代价（路径代价 + 估计到目标的距离）。

将合法的相邻节点加入待扩展的节点集合。

找到目标状态后，通过回溯路径找到解。

- 实验内容

问题描述： 给定九宫格的初始状态，要求在有限步的操作内，使其转化为目标状态，且所得的解式代价最小解（即移动的步数最小）。如图 1 所示：

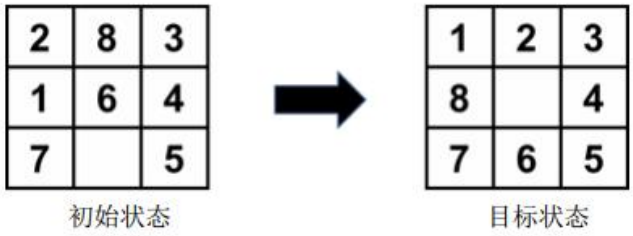

图 1 问题要求示意图

算法基础设计思想：①根据输入的信息获取九宫格的初始状态和目标状态。②设计合适的估价函数，用于评估状态之间的距离。③实现 A*算法，通过迭代选择合适的节点进行状态拓展，直至找到最优解。④记录重排过程，即途径的状态，以及每一步的移动。

3、实验方法及步骤设计

①问题建模： 将九宫格重排问题抽象成状态空间图，定义状态表示、操作规则和目标状态。

②初始化： 将初始状态设为当前状态，设置初始代价为 0。

③估价函数设计： 定义估价函数，例如使用错棋子数，评估当前状态与目标状态之间的距离。

④A*算法设计流程：将初始状态加入待扩展节点集合，初始代价设为 0。循环以下步骤直至找到目标状态：从待扩展节点集合中选择代价最小的节点进行扩展。生成相邻的节点，并计算它们的代价。将合法的相邻节点加入待扩展的节点集合。检查是否找到目标状态，如果是则停止搜索。

⑤路径记录： 找到目标状态后，通过回溯路径找到解，即途径的状态，以及每一步的移动。

⑥输出结果： 输出重排过程，即途径的状态，展示每一步的移动。

⑦实验验证： 验证算法得到的解是否满足问题要求，即移动步数最小且能够从初始状态到达目标状态。

⑧优化策略： 根据实验结果，考虑是否需要进一步优化算法，例如调整估价函数或引入其他启发式方法。

⑧实验报告撰写和总结：撰写实验报告，包括实验的背景、目的、方法、步骤、程序源代码、实验结果及分析等内容。总结实验结果，分析算法的效率和可行性。如果存在改进的空间，提出可能的优化方案。

4、实验设备仪器选择及材料

一台笔记本电脑、pycharm 软件、A4 记录纸、截屏软件、键盘、鼠标等。

5、实验注意事项（包括实验过程、仪器设备、个人操作等方面）

（1）确保提供初始状态和目标状态的正确输入数据。检查输入数据的合法性，确保它们

符合九宫格的规则。

（2）设计合适的估价函数对 A*算法的性能至关重要。确保估价函数能够有效地评估状态之间的距离，有助于提高搜索效率。

（3）在实验中需要详细记录路径，包括途径的状态和每一步的移动。确保输出结果的可读性和准确性。

（4）确保使用的计算机设备和编程工具能够正常运行。

6、实验思考题解答

思考题：为什么选择 A 算法来解决九宫格重排问题？A 算法有哪些优势？

在最开始做实验之前我脑海里一直在想这个问题，等做完答案后我对这个问题已经有了答案。A*算法是一种启发式搜索算法，通过综合考虑实际代价和启发式估价函数，能够在保证最优解的情况下提高搜索效率。其优势包括能够找到最优解、广泛适用于不同问题、可调整的启发式函数等。

二、实验过程及记录

通过分析题意，我们可以理解：如果我们去移动数字，那么对于此问题来说会非常复杂，故我们选择移动空格。这样我们就将问题简化为空格的移动，空格移动的状态只有 4 种：上、下、左、右，这四种状态会使问题变得简单。

根据 A*算法的定义，若要使用 A*算法解决九宫格重排问题，首先要定义一下 f(n)、g(n)和 h(n)。其中：f(n)指估计从初始状态到目标状态的代价；g(n)指从初始状态到当前状态的实际代价；h(n)指当前状态与目标状态的错位数。以下图 2 为一个测试过程：

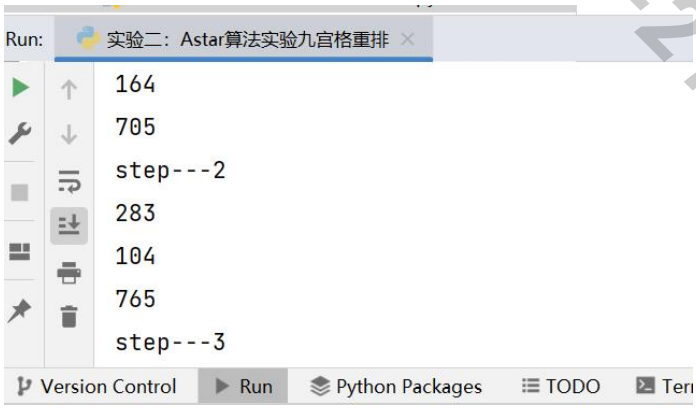

图 2 测试过程记录

三、实验结果分析

以图 1 的初始状态和目标状态为例，输入下图所示的状态：

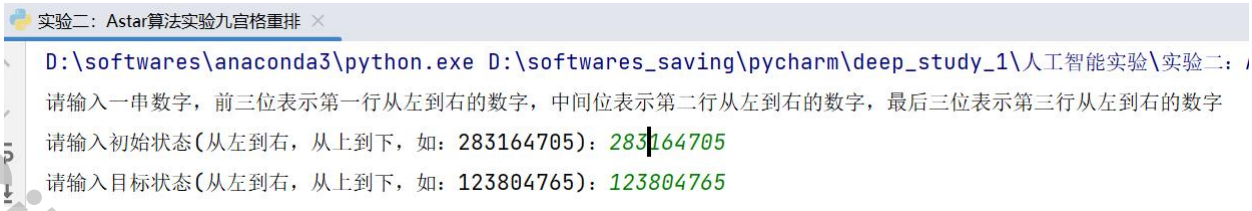

图 3 输入测试

得到如下图 4 所示的结果，可以看出通过图形展示清晰地展示了每一步的移动过程，有助于更好地理解和解决过程。在上面这个例子中，九宫格的状态空间相对较小，A*算法能够在短时间内找到最优解。但对于更大的状态空间，需要考虑算法的复杂度。通过比较不同可能的解，经过我自己的验算验证，我确认所得解是最优解，的确为移动步数最小的解。

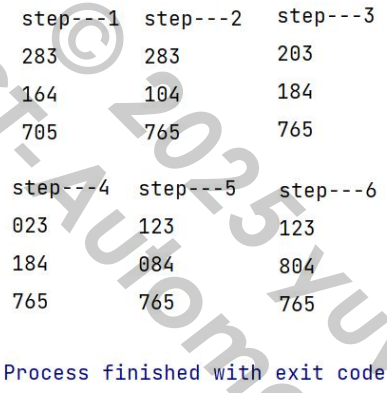

图 4 测试输出结果

当输入的初始状态不能到达目标状态时，会输出“无法到达”的提示。当输入的初始状态与目标状态相同时，会输出“一致”的提示。如下图所示。

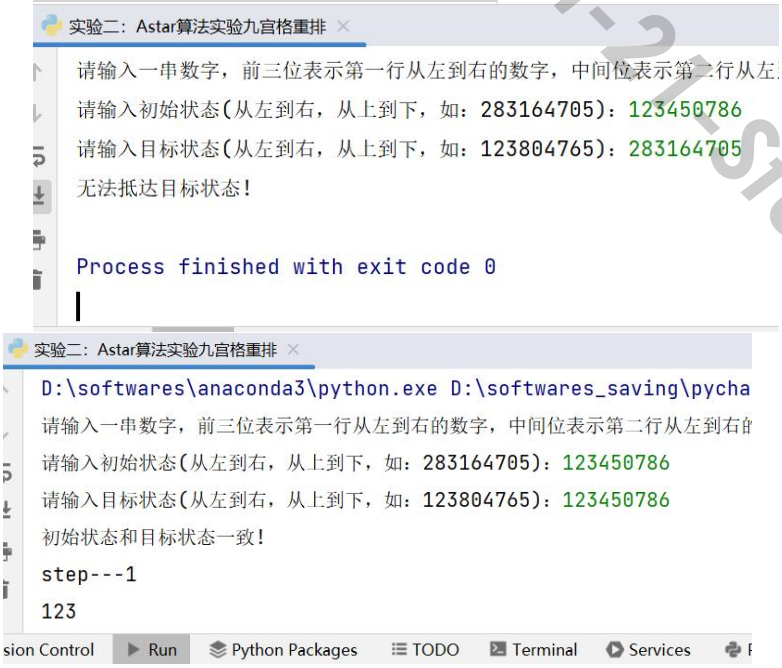

图 5 异常时的提示消息

下面用算法流程图 4 来说明我在解决这个具体问题时，使用 A*算法的设计思想，首先定

义九宫格的状态表示方式，例如将每个数字在九宫格中的位置编码成一个状态，再确定初始状态和目标状态，作为搜索的起点和终点。确定合法的操作规则，即在每一步格子限制的情况下允许的状态转移，通常是数字的上、下、左、右移动。设计估价函数，用于评估当前状态与目标状态之间的距离。在九宫格问题中，可以使用错棋子数作为估价函数，然后使用 A*算法框架，初始化初始状态并将其加入待扩展节点集合，代价设为 0。循环进行以下步骤：①从待扩展节点集合中选择代价最小的节点进行扩展。②生成相邻的节点，并计算它们的代价（路径代价 + 估计到目标的距离）。③将合法的相邻节点加入待扩展的节点集合。④检查是否找到目标状态，如果是则停止搜索。在找到目标状态后，通过回溯路径找到解，即途径的状态，以及每一步的移动。

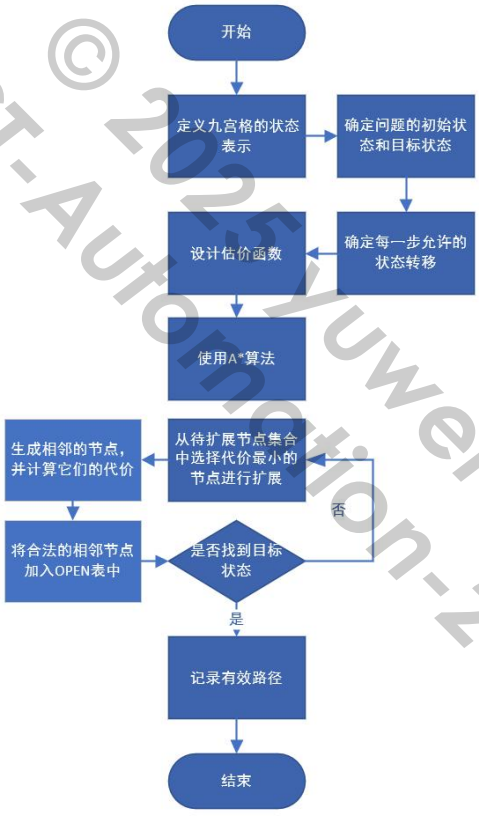

图 6 实验算法设计思想

五、实验总结（200-300 字）

在进行 A*算法解决九宫格重排问题的实验过程中，我遇到了一个关键问题：在状态扩展时，如何高效地管理和维护待扩展节点集合，以提高搜索效率。待扩展节点集合的管理涉及到选择最小代价节点和检查相邻节点是否已经在集合中的操作。在状态空间较大的情况下，这可能导致集合的查找操作变得耗时，影响算法性能。为了解决这个问题，我采用了字典来管理待扩展节点集合和列表存储代价函数。这样能够保证每次选择代价最小的节点，避免了遍历整个集合的开销。这种方式有效地提高了 A*算法的搜索效率。

在实验过程中，我深刻认识到算法的实现不仅仅依赖于算法本身的设计，还受到数据结构和数据管理方式的影响。正确选择和优化数据结构可以显著提升算法性能。在进一步的项目优化方面，我考虑引入更复杂的启发式函数来改进估价过程，以及进一步优化算法参数。此外，可以考虑并行计算等策略，以加速算法在大规模状态空间中的搜索过程。过对实验中的问题及解决方案的深入思考，我获得了对 A*算法和实验设计的更深层次的理解，为未来的算法研究和应用提供了有益的经验。

附录 2--实验 2 源代码：

```
def solve_reserve(node):
    Sum = 0
    for i in range(1, 9):
        num = 0
        for j in range(0, i):
            if node[j] > node[i] and node[i] != '0':
                num = num + 1
        Sum += num
    return Sum

def fhn(node):
    global goal
    hn = 0
    for i in range(0, 9):
        if node[i] != goal[i]:  #判断是否和目标状态一样
            hn += 1
    return hn

'''
选择 open 表中最小的估价函数值对应的状态
'''
def min_fn(opened):
    find_value_temp = {}    #定义一个字典来临时存取序列和对应的估价函数
    for node in opened:  #把 open 表中的每个序列都取出来
        find_value_temp[node] = Fn[node]    #拿到估价函数
    node_min = min(find_value_temp, key=find_value_temp.get)
    return node_min
'''
对这一分支的 node 状态的子结点进行扩展
'''
def expand_node(node):
    global expand
    tnode = []
    state = node.index("0")   #把父节点放入 open 表中
    elist = expand[state]
    j = state
    for i in elist:
        j = state
        if i > j:
            i, j = j, i
        re = node[:i] + node[j] + node[i + 1:j] + node[i] + node[j + 1:]
```

```
            tnode.append(re)
        return tnode
    '''
    输出的结果按照一定的格式打印
    '''
    def print_process(result):
        for i in range(len(result)):
            print("step---" + str(i + 1))
            print(result[i][:3])
            print(result[i][3:6])
            print(result[i][6:])

    if __name__ == "__main__":
        # expand 中存放着九宫格着每个位置能移动的状态
        #上下左右，在序列里移动的下标
        expand = {0: [1, 3], 1: [0, 2, 4], 2: [1, 5],3: [0, 4, 6],
                  4: [3, 1, 5, 7], 5: [4, 2, 8],
                  6: [3, 7], 7: [6, 4, 8], 8: [7, 5]}
        Gn = {}  # 用字典来存储移动的步数，key 是序列，value 是 hn=gn+hn
        Fn = {}
        open = []   #open 表，用来存放还没有拓展的节点
        close = []  #open 表，用来存放已经拓展的节点
        PARENT = {}  # 父结点

        print("请输入一串数字，前三位表示第一行从左到右的数字，中间位表示第二行从左到右的数字，最后三位表示第三行从左到右的数字")
        start = input("请输入初始状态(从左到右，从上到下，如：283164705)：")
        goal = input("请输入目标状态(从左到右，从上到下，如：123804765)：")
        #判断初始状态与目标状态是否一致
        if start == goal:
            print("初始状态和目标状态一致!")
        # 判断初始状态能否达到目标状态
        #首先需要判断序列有没有解，两个状态有没有解，通过他们的逆序数的奇偶来判断，同为奇或同为偶就有解
        if (solve_reserve(start) % 2) == (solve_reserve(goal) % 2):
             #能抵达
            PARENT[start] = -1  # 初始状态父结点为-1
            Gn[start] = 0  # 初始结点的 g(n) 函数值为 0
            Fn[start] = Gn[start] + fhn(start)  # 求初始的估价函数值
            open.append(start)  # 将初始结点存入 open 表
            while open:
                current = min_fn(open)  # 选择估价函数值最小的状态
                del Fn[current]
                open.remove(current)  # 将要遍历的结点取出 open 表
                if current == goal:
```

```
                break
            if current not in close:
                close.append(current)  # 存入 close 表
                Tnode = expand_node(current)  # 扩展子结点
                for node in Tnode:
                    #若子结点在 open 和 close 表中都未出现，则存入 open 表
                    #并求出对应的估价函数值

                    if node not in open and node not in close:
                        Gn[node] = Gn[current] + 1
                        Fn[node] = Gn[node] + fhn(node)
                        PARENT[node] = current
                        open.append(node)
                    else:
                        # 若子结点已经在 open 表中，则判断估价函数值更小的一个路径
                        if node in open:
                            fn = Gn[current] + 1 + fhn(node)
                            if fn < Fn[node]:
                                Fn[node] = fn
                                PARENT[node] = current

        result = []  # 用来存放路径
        result.append(current)
        while PARENT[current] != -1:
            current = PARENT[current]
            result.append(current)
        result.reverse()  # 逆序就是所运行的过程
        print_process(result)  # 按格式输出结果
    else:
        print("无法抵达目标状态!")
```

实验名称	实验三 遗传算法求函数最大值	时间	2023 年 11 月 11 日
		地点	
同实验者		班组	

一、实验预习（准备）报告

1、实验目的

熟悉和掌握遗传算法原理，用遗传算法求给定函数的最大值。本实验通过模拟自然进化过程搜索最优解的方法，加深学生对遗传算法原理的理解和应用。

2、实验相关原理及内容

- 实验原理

1. 遗传算法：遗传算法（Genetic Algorithm，GA）最早是由美国的 John holland 于 20 世纪 70 年代提出,该算法是根据大自然中生物体进化规律而设计提出的。是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。该算法通过数学的方式,利用计算机仿真运算,将问题的求解过程转换成类似生物进化中的染色体基因的交叉、变异等过程。在求解较为复杂的组合优化问题时,相对一些常规的优化算法,通常能够较快地获得较好的优化结果。

2.求解函数最大值：在函数最大值求解中，遗传算法通过模拟生物进化的过程，通过交叉、变异等操作，生成新一代个体，并筛选适应度高的个体，逐渐趋向于全局最优解。

- 实验内容

问题描述： 输入： $f(x) = \cos(5x) - \sin(3x) + 10, x \in [1,7]$，取种群大小 20，搜索精度 0.0001，交叉概率 0.6，变异概率 0.1，遗传 20 代。

输出：用遗传算法确定出的最大值。

算法设计思想：①编码方案： 选择一种适合问题的编码方式，将问题的解表示为个体的基因。例如，对于优化函数的问题，可以使用二进制、实数等编码。

②初始化种群： 随机生成初始个体，构成初始种群。每个个体对应一个可能的解。

③适应度评估： 计算每个个体的适应度，即目标函数在该个体对应解下的取值。适应度越高，个体越有可能被选择。

④选择操作： 根据适应度选择个体，高适应度个体被选中的概率较大。可以使用轮盘赌选择、锦标赛选择等方法。

⑤交叉操作： 选择一对个体进行基因交叉，生成新的个体。交叉点的选择方式可以是随机选择或固定选择。

⑥变异操作： 对新生成的个体进行基因变异，引入随机扰动，以增加搜索空间的广度。

⑦生成新种群： 将原种群和新生成的个体组成新的种群。

⑧重复操作： 重复进行适应度评估、选择、交叉、变异等步骤，直至满足停止条件（例如达到最大迭代次数或适应度达到一定阈值）。

⑨输出结果： 输出遗传算法收敛后的最优解，即使目标函数达到最大值的个体。

3、实验方法及步骤设计

①选择优化目标，定义适应度函数。
②选择编码方式，初始化种群。
③进行适应度评估，计算每个个体的适应度。
④进行选择、交叉、变异等遗传算法操作，生成新一代种群。
⑤重复步骤 3 和 4，直至满足停止条件。
⑥输出收敛后的最优解及相应的最大值。

4、实验设备仪器选择及材料

一台笔记本电脑、pycharm 软件、A4 记录纸、截屏软件、键盘、鼠标等。

5、实验注意事项（包括实验过程、仪器设备、个人操作等方面）

（1）在编写程序时对变量的取名要有意义，能够清楚的知道其意思；

二、实验过程及记录

首先把目标函数的函数曲线画出来，大致预估最大值对应的 x 值取值区间，这将有助于确定合适的初始群体，并且检验最后结果的正确性。目标函数的函数曲线如下图 1 所示：图中红色的点代表随机生成的个体，初始群体的个数为 20。可以看出该函数的最大值大概在 12 附近。

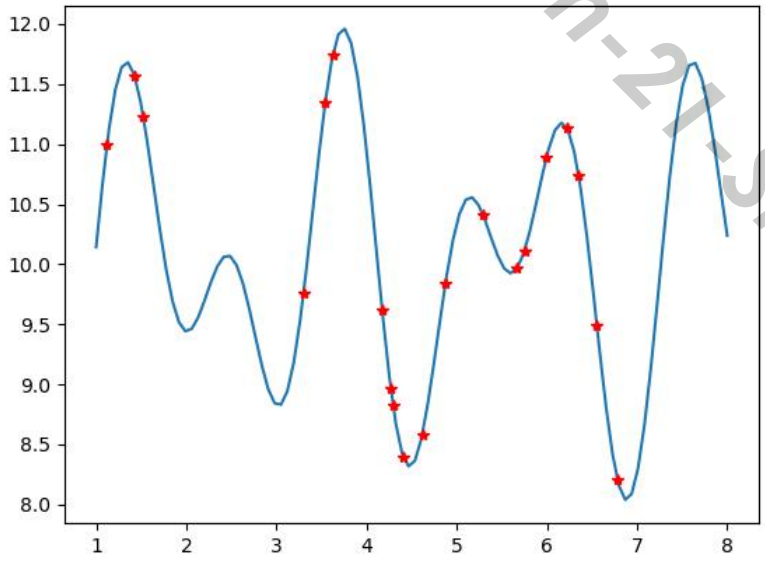

图 1 目标函数的函数曲线

为了验证编码和解码函数编写的是否正确，在编写实验的过程中，把样本数据编码解码前后打印出来，便于理解和排查错误。如果出现了问题可以在对应的函数里检查逻辑和修改。如下图 2 的测试结果所示，可以看到这两个过程得到的数据是无误的。并且我还通过了函数绘制来展现编码前和解码后的数据，如图 3 所示。原点表示编码后的，星号表示解码后的。

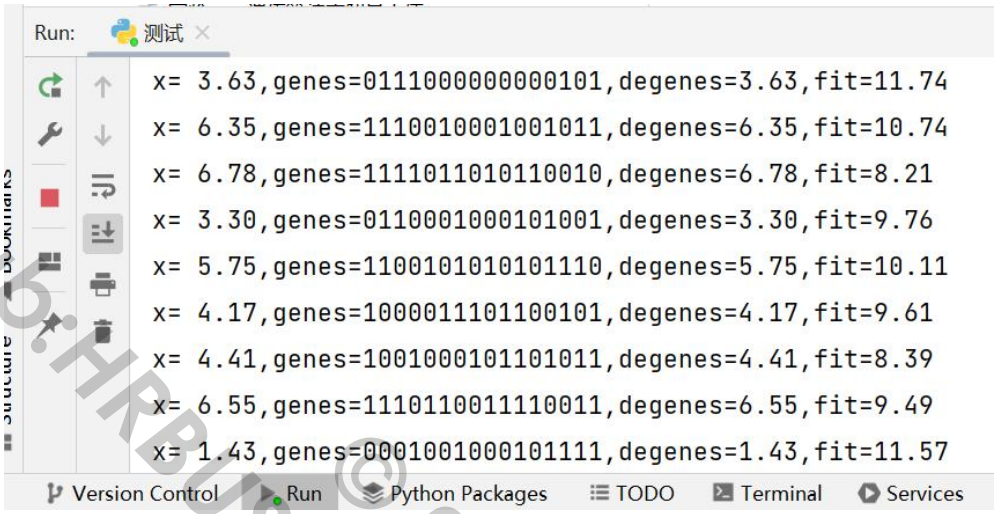

图 2 编码和解码功能测试

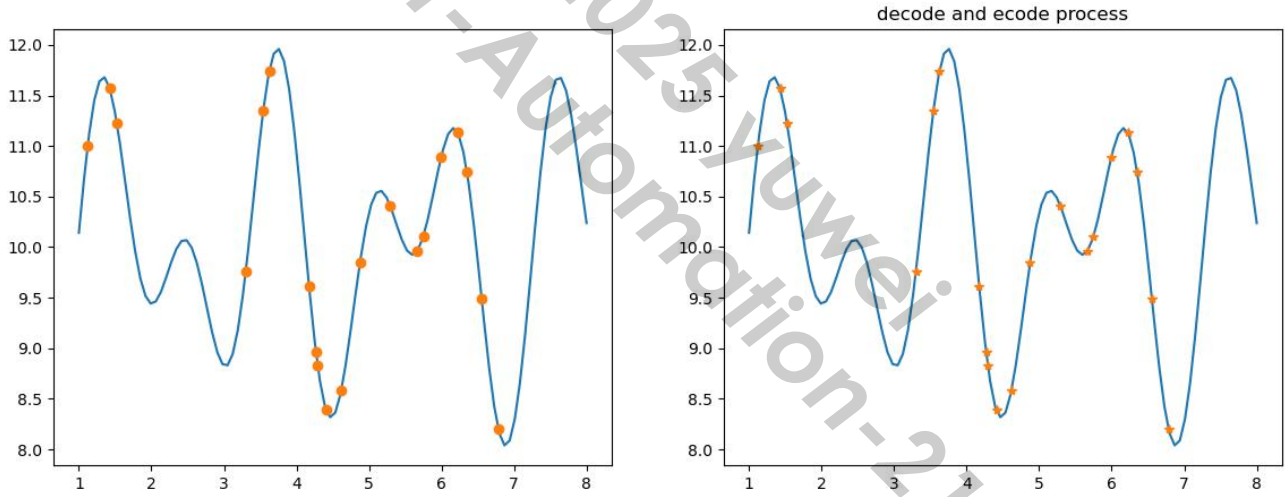

图 3 编码和解码曲线对比

由于题目指定迭代次数为 20 次，为了更直观地看到在遗传过程中个体的变化，我设置了每 10 代就显示一次结果，图 4 和图 5 分别展示的是迭代 10 次和 20 次时个体的分布图像。

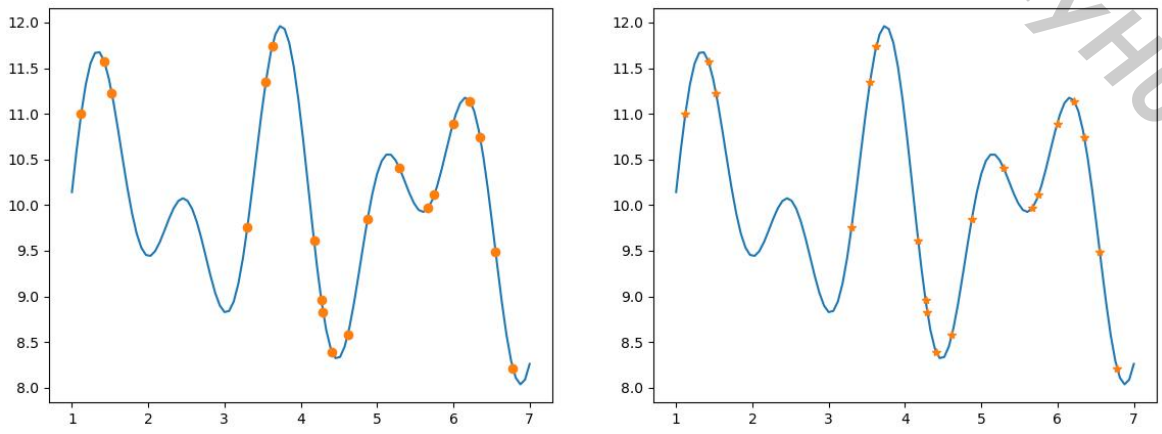

图 4 迭代 10 次时个体的分布

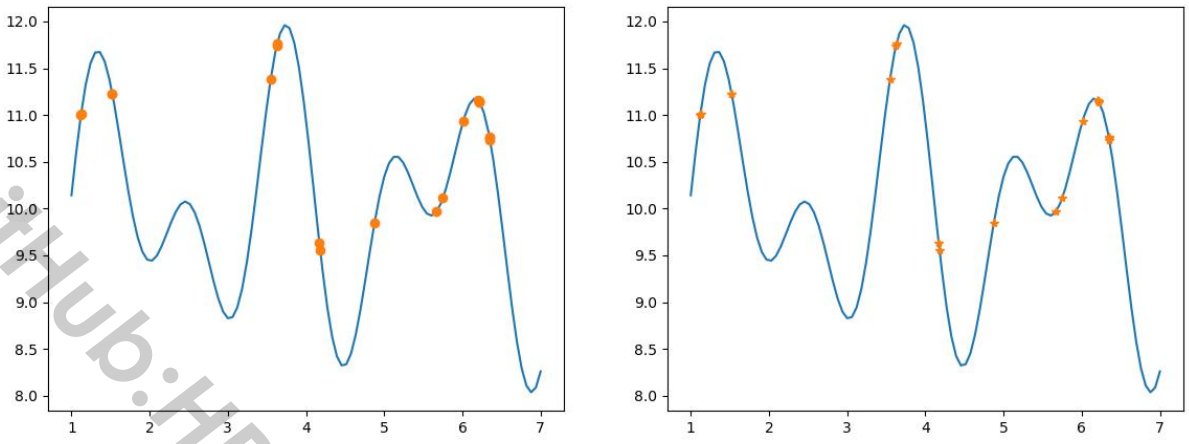

图 5 迭代 20 次时个体的分布

三、实验结果分析

由实验结果可知，通过 20 次迭代后，函数 f（x）= cos(5x) − sin(3x) +10 在 x ∈[1,7] 区间时，最大值是 max: 11.743212874689457，这与我们最开始预估计会离 12 很近的猜测结果是一致的，表明实验是可行的。

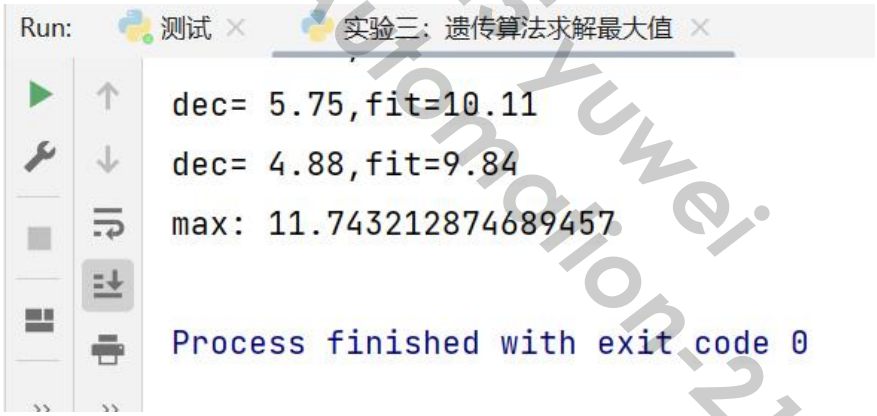

图 6 最后的输出结果

除此之外我还用 matlab 实现了本实验，目标函数曲线和适应度曲线如下图所示：

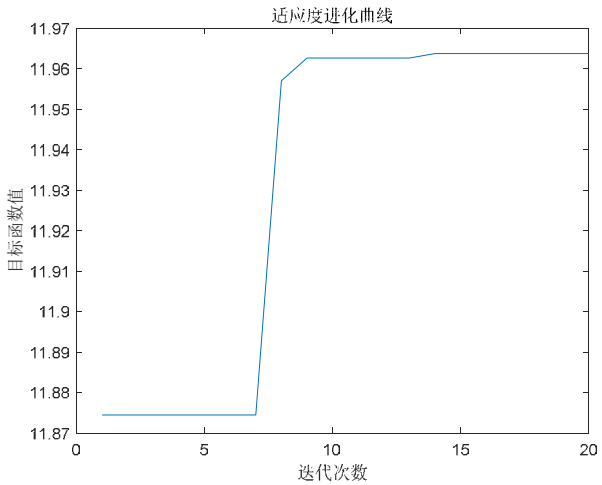

图 7 适应度曲线

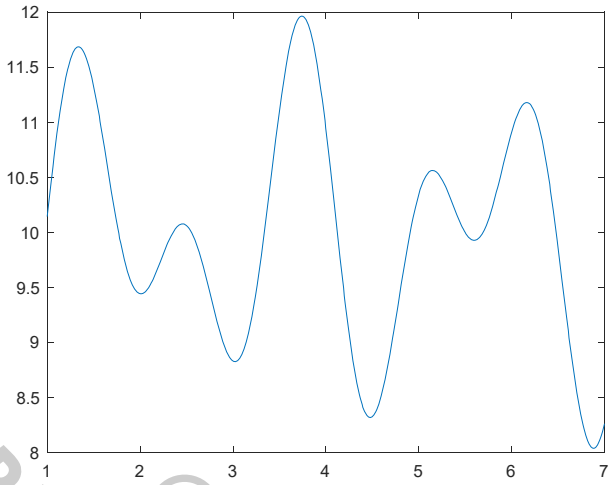

图 8 目标函数曲线

下面用算法流程图 9 来说明我在解决这个具体问题时，使用遗传算法的设计思想，首先是选择适合问题的编码方式，我这里选择了二进制编码，编码方式应能够有效地表示问题的解空间，使得个体之间的遗传操作能够产生有意义的结果。随机生成初始种群，确保个体的多样性和初始解的广泛覆盖。种群大小的选择需考虑到问题的复杂性和计算资源的限制。定义适应度函数，用于评估个体在问题空间中的优劣。在函数最大值求解中，适应度函数通常为目标函数的值。采用轮盘赌选择确保适应度较高的个体有更高的概率被选择。通过交叉操作，模拟遗传过程中的基因组合。引入变异操作，以增加搜索空间的多样性。对个体的某些基因进行随机变异，模拟生物基因的突变。变异有助于避免陷入局部最优解。将经过选择、交叉和变异操作得到的子代个体与原种群合并，形成新一代种群。新种群的生成是通过模拟自然选择和遗传机制实现的。重复进行选择、交叉、变异等操作，直至满足停止条件（达到最大迭代次数）。每一代都经过遗传操作，逐渐趋向于全局最优解。

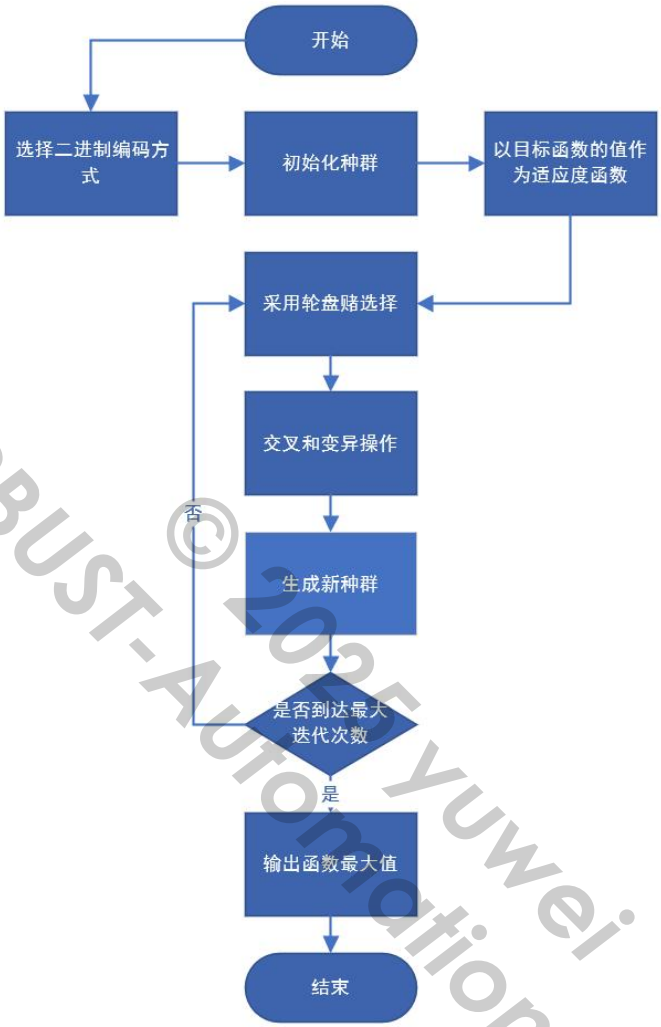

图 9 实验算法设计思想

四、实验总结（200-300 字）

在进行遗传算法求函数最大值的实验过程中，我遇到了一个关键问题：收敛速度较慢，算法难以在有限的迭代次数内达到较高的适应度值。收敛速度较慢可能由于初始种群的质量不佳或者遗传操作的选择方式不合适导致。这影响了算法的搜索效率和性能。为了解决这个问题，我采取了以个策略。首先，我调整了初始种群的生成策略，确保更好的覆盖搜索空间。经过调整，实验中的算法收敛速度得到了显著提升。

通过实验，我得出了遗传算法在函数最大值求解中是一种强大的优化工具。算法能够有效地在搜索空间中寻找最优解，对于复杂的非线性问题具有较强的适应性。在实验中，我深刻认识到参数的选择和算法操作的设计对算法性能的影响至关重要。仅仅依赖默认参数可能不足以获得令人满意的结果，需要结合具体问题进行调优。在进一步优化项目方面，我考虑引入自适应参数调整策略，根据算法运行过程中的动态信息动态调整参数，提高算法的自适应性。另外，可以尝试并行化处理，以加速算法在大规模搜索空间中的求解过程。

附录 3--实验 3 源代码：

```python
import matplotlib.pyplot as plt
import random
import numpy as np
import pandas as pd
'''
遗传算法的基本步骤：
①初始化种群（第一代动物）
②对个体编码（也就是根据表现型写出基因型）
③适应度函数计算（求最大值的话也就是代入函数里算）
④选择（优胜劣汰）
⑤随机选取交叉点交叉
⑥随机选取某个位置变异
'''
# 定义目标函数
def fun(x):
    return np.cos(5 * x) - np.sin(3 * x) + 10
# ①初始化种群
# def inital_population():
# ②二进制编码
def encode(population, scale=2 ** 16, bin_len=16):
    norm_data = (population - 1) / (7 - 1) * scale
    bin_data = np.array([np.binary_repr(x, width=bin_len) for x in norm_data.astype(int)])  # 转换成二进制编码
    return bin_data
def decode(population_genes, scale=2 ** 16):
    return np.array([(int(x, base=2) / scale * 6) + 1 for x in population_genes])
# 选择（轮盘赌的方式）
'''
1.计算出每个个体的适应度
2.计算出每个个体遗传到下一代群体的概率
3.计算出每个个体累计概率
4.随机产生一个【0,1】的数，看看落在哪里
5.重复选择 M 次，M 为个体数
'''
def select_cross(chroms, fitness, prob=0.6):
    probs = fitness / np.sum(fitness)
    probs_cum = np.cumsum(probs)  # 概率累加分布
    # 产生均匀分布的伪随机
    each_rand = np.random.uniform(size=len(fitness))  # 产生一个伪随机数
    # 根据随机的概率选择出新的基因编码
    newX = np.array([chroms[np.where(probs_cum > rand)[0][0]] for rand in each_rand])
    # 繁殖交叉
```

```
        pairs = np.random.permutation(int(len(newX) * prob // 2 * 2)).reshape(-1, 2)
        center = len(newX[0]) // 2
        for i, j in pairs:
            # 在中间位置交叉
            x, y = newX[i], newX[j]
            newX[i] = x[:center] + y[center:]
            newX[j] = y[:center] + x[center:]
        return newX
    # 变异
    def change(chroms):
        prob = 0.1  # 变异概率0.1
        clen = len(chroms[0])
        m = {'0': '1', '1': '0'}
        newchroms = []
        each_prob = np.random.uniform(size=len(chroms))  # 产生一个伪随机数
        for i, chrom in enumerate(chroms):
            if each_prob[i] < prob:
                pos = np.random.randint(clen)
                chrom = chrom[:pos] + m[chrom[pos]] + chrom[pos + 1:]
            newchroms.append(chrom)
        return np.array(newchroms)
    # 结果可视化
    def show_pictures(chroms1, chroms2, fitfun):
        Xs = np.linspace(1, 7, 100)
        fig, (axs1, axs2) = plt.subplots(1, 2, figsize=(14, 5))
        dechroms = decode(chroms1)
        fitness = fitfun(dechroms)
        for dec, fit in zip(dechroms, fitness):
            print("dec=%5.2f,fit=%.2f" % (dec, fit))
        axs1.plot(Xs, fitfun(Xs))
        axs1.plot(dechroms, fitness, 'o')
        dechroms2 = decode(chroms2)
        fitness2 = fitfun(dechroms)
        for dec, fit in zip(dechroms, fitness):
            print("dec=%5.2f,fit=%.2f" % (dec, fit))
        axs2.plot(Xs, fitfun(Xs))
        axs2.plot(dechroms, fitness, '*')
        plt.show()
    if __name__ == "__main__":
        # 产生均匀的X 数字序列
        x_data = np.linspace(1, 8, 100)
       # plt.plot(x_data, fun(x_data))
      #  plt.show()
        # ①初始化种群
```

```
    np.random.seed(0)
    population = np.random.uniform(1, 7, 20)  # 随机生成20个小数，范围是[1,7]，种群大小是20
    genes = encode(population)
    for i in range(20):
        fitness = fun(decode(genes))
        max_fit = fitness.max()
        fitness = fitness - fitness.min() + 0.0001
        newgenes = change(select_cross(genes, fitness))
        if i % 10 == 1:
            show_pictures(genes, newgenes, fun)
    genes = newgenes
    show_pictures(genes, newgenes, fun)
    print("max:", max_fit)
```

实验名称	实验四 专家系统	时间	2023 年 11 月 11 日
		地点	
同实验者		班组	

一、实验预习（准备）报告

1、实验目的

熟悉和掌握专家系统，理解专家系统流程。专家系统是人工智能的重要研究内容和组成部分之一。本实验设计一个简单的专家系统，加深学生对专家系统的组成结构和构造原理的理解并转化为具体的应用。

2、实验相关原理及内容

- 实验原理

1. 专家系统的核心原理：

专家系统的核心是领域知识。这些知识以规则、事实、关系或其他形式被存储在系统的知识库中。知识库是专家系统的灵魂，它包含了解决问题所需的领域专家知识。

知识库中的知识以规则的形式表示。规则是条件与结论的逻辑陈述，形如“如果条件，则结论”。例如，“如果动物有毛发且有脊椎，则是哺乳动物”。

推理引擎是专家系统的核心组件，负责执行规则，根据输入的事实和条件进行推理。推理引擎通过匹配规则，推导出结论，模拟人类专家的决策过程。

专家系统的知识获取是一个关键步骤。它涉及从领域专家获取知识，并将其转化为计算机可理解的形式，存储在知识库中。知识获取是构建专家系统的基础。

用户接口是专家系统与用户交互的桥梁。通过用户接口，用户可以输入问题、事实或条件，系统则输出推理结果。用户接口的友好性对于专家系统的成功应用至关重要。

推理方式包括前向推理和后向推理。前向推理从已知事实出发，通过匹配规则逐步得出结论。后向推理则从目标出发，通过逐步询问用户获取所需事实，最终得出结论。

为提高系统的可理解性，专家系统通常包含用户反馈机制。系统能够向用户解释推理结果，说明为何得出这样的结论。这有助于用户理解系统的工作原理。

在真实世界中，问题往往伴随不确定性。专家系统需要能够处理模糊、不完全或不确定的信息。因此，专家系统的设计需要考虑不确定性的建模和处理。

- 实验内容

算法基础设计思想：①知识库构建：构建一个动物分类的知识库，包含不同种类动物的属性及其对应值。例如，毛发有/无、产卵有/无、有脊椎/无脊椎等属性。②规则定义：根据知识库，定义一系列规则，以形成专家系统的推理规则。规则可通过条件-结论的形式表示，例如，“有毛发且有脊椎”可以判定为“哺乳动物”。③用户接口设计：设计用户接口，允许用户输入动物的属性信息。可以通过图形界面方式实现。④实现专家系统的推理引擎，负责根据用户

输入的属性值匹配规则并做出分类判断。可以采用基于规则的推理引擎，如基于前向推理或后向推理。⑤用户反馈：在系统中加入用户反馈机制，使系统能够向用户解释分类的依据。例如，当系统判断输入的属性值符合“哺乳动物”时，输出解释信息：“有毛发且有脊椎”。⑥测试与验证：设计一系列测试用例，验证专家系统对不同动物的分类准确性。通过不同的输入属性值，检查系统的输出是否符合预期。⑧系统优化:根据测试结果和用户反馈，对系统进行优化。可以调整规则，添加新的属性，提高系统的分类准确性和适应性。

3、实验方法及步骤设计

①确定实验目标：定义专家系统的目标，例如构建一个动物分类专家系统，根据属性值自动识别动物的种类。

②知识库构建：确定动物分类的知识库，包括不同种类动物的属性及其对应值，例如毛发有/无、产卵有/无等。

③规则定义：根据知识库，定义规则，形成专家系统的推理规则。每个规则都是一个条件-结论的逻辑陈述。

④用户接口设计：设计用户友好的接口，以便用户能够输入动物的属性信息。这可以是一个图形用户界面。

⑤推理引擎实现： 开发推理引擎，实现根据规则进行推理的功能。推理引擎应能够匹配用户输入的属性值，并根据规则得出动物的分类。

⑥用户反馈机制： 加入用户反馈机制，使系统能够向用户解释分类结果。用户输入的属性值不仅产生分类结果，还输出解释信息，说明系统判定的依据。

⑦测试用例设计： 设计一系列测试用例，覆盖各种可能的输入情况，以验证专家系统对不同动物的分类准确性。

⑧系统验证： 使用测试用例验证系统的分类准确性。检查系统的输出是否符合预期，特别是在包含复杂规则的情况下。

⑨优化系统：根据测试结果和用户反馈，对系统进行优化。可以调整规则、增加新的属性，提高系统的分类准确性和适应性。

⑩实验记录：记录实验过程中的关键步骤、参数设置、系统反馈等信息，以便事后分析和实验结果的复现。

⑩实验报告撰写和总结：撰写实验报告，包括实验的背景、目的、方法、步骤、程序源代码、实验结果及分析等内容。总结实验结果，分析算法的效率和可行性。如果存在改进的空间，提出可能的优化方案。

4、实验设备仪器选择及材料

一台笔记本电脑、pycharm 软件、A4 记录纸、截屏软件、键盘、鼠标等。

5、实验注意事项（包括实验过程、仪器设备、个人操作等方面）

（1）在开始实验前，确保对专家系统的设计目标有清晰的认识，明确定义实验的目标和预期结果。

（2）知识库是专家系统的核心，确保其中的属性、规则和事实准确无误。对知识库的设计要仔细考虑，确保能够涵盖多种情况。

（3）推理引擎的实现应当准确无误。测试不同规则的匹配和推理过程，确保系统能够正确地根据输入的属性值做出分类判断。

（4）避免知识库和规则设计的冗余和重复。合理使用规则的组织方式，确保系统的推理过程高效而准确。

二、实验过程及记录

（一）构造规则库

（1）题目事实（概念）

```
# 属性字典，包含 11 个属性
self.attributes = {
    "有毛发": tk.BooleanVar(),
    "会飞行": tk.BooleanVar(),
    "食肉": tk.BooleanVar(),
    "有鳞片": tk.BooleanVar(),
    "会游泳": tk.BooleanVar(),
    "夜行性": tk.BooleanVar(),
    "有角": tk.BooleanVar(),
    "有尾巴": tk.BooleanVar(),
    "驯养的": tk.BooleanVar(),
    "奔跑快": tk.BooleanVar(),
    "群居型": tk.BooleanVar(),
}
```

（2）题目产生式规则

```
# 规则字典，根据属性值确定动物类型
self.animal_rules = {
    (True, False, True, False, True, False, False, True, False, True, True): "狮子",
    (False, False, False, True, True,False,    False, False, False, True, False): "鱼",
    (True, False, False, True, True, True, False, True, False, False, True): "蛇",
    (False, True, False, False, True, False, False, True, False, False, True): "鹦鹉",
    # 添加更多规则...
}
```

三、实验结果分析

总体界面展示：

图 1 总体界面

接着对模型进行测试。通过选择相应的特征，系统会推理出对应的类别。如下图 2 所示，我勾选了“有鳞片”“会游泳”“奔跑快”这三个特征，得到的结果是“鱼”。测试结果如下：

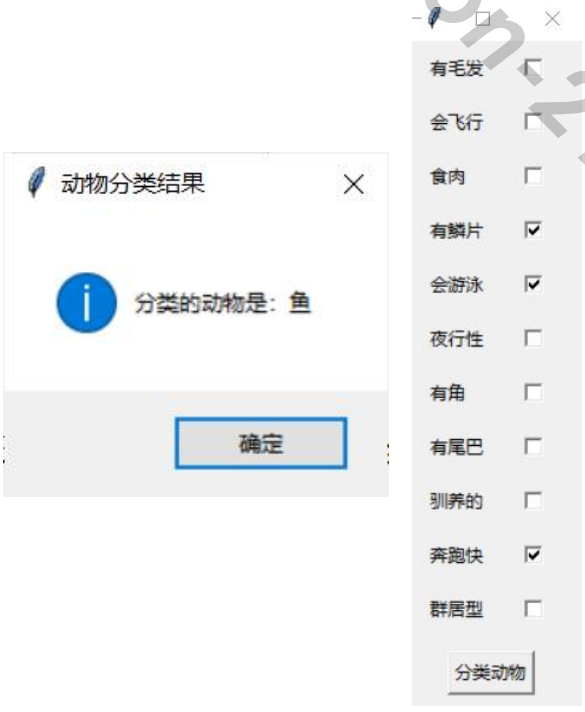

图 2 3　总体界面和分类结果

六、实验总结（200-300 字）

通过本次的专家系统实验让我知道了所谓的专家系统，就是一种智能的电脑程序，里面包含了很多专业人士的知识与经验，可以将人工智能与电脑技术结合在一起，通过对系统的知识与经验进行分析，从而模拟出专家们的决策过程，从而解决一些复杂的问题。

在实验的过程当中，推理机运用这些事实，依次与知识库中的规则的前提匹配，若某规则的前提全被事实满足，则规则可以得到运用，而且规则的结论部分作为新的事实存储，可以用更新过的事实再与其它规则的前提匹配，直到不再有可匹配的规则为止。

本次实验让我受益良多，课本上的知识能够运用到了实践当中，希望以后能有更多类似的机会，来锻炼我们各方面的能力。

附录 4-实验 4 源代码：

```
import tkinter as tk
from tkinter import messagebox

class AnimalExpertSystemGUI:
    def __init__(self, master):
        self.master = master
        self.master.title("动物专家系统")

        # 属性字典，包含 11 个属性
        self.attributes = {
            "有毛发": tk.BooleanVar(),
            "会飞行": tk.BooleanVar(),
            "食肉": tk.BooleanVar(),
            "有鳞片": tk.BooleanVar(),
            "会游泳": tk.BooleanVar(),
            "夜行性": tk.BooleanVar(),
            "有角": tk.BooleanVar(),
            "有尾巴": tk.BooleanVar(),
            "驯养的": tk.BooleanVar(),
            "奔跑快": tk.BooleanVar(),
            "群居型": tk.BooleanVar(),
        }

        # 规则字典，根据属性值确定动物类型
        self.animal_rules = {
            (True, False, True, False, True, False, False, True, False, True, True): "狮子",
            (False, False, False, True, True,False,  False, False, False, True, False): "鱼",
            (True, False, False, True, True, True, False, True, False, False, True): "蛇",
            (False, True, False, False, True, False, False, True, False, False, True): "鹦鹉",
            # 添加更多规则...
        }

        self.create_gui()

    def create_gui(self):
        # 为每个属性创建标签和复选框
        row_num = 0
        for attribute, var in self.attributes.items():
            label = tk.Label(self.master, text=attribute)
            label.grid(row=row_num, column=0, sticky=tk.W, padx=10, pady=5)

            checkbox = tk.Checkbutton(self.master, variable=var, onvalue=True, offvalue=False)
            checkbox.grid(row=row_num, column=1, padx=10, pady=5)
```

```
44.
45.             row_num += 1
46.
47.         # 添加一个按钮用于分类动物
48.         classify_button = tk.Button(self.master, text="分类动物", command=self.classify_animal)
49.         classify_button.grid(row=row_num, column=0, columnspan=2, pady=10)
50.
51.     def classify_animal(self):
52.         # 根据选择的属性对动物进行分类
53.         input_values = tuple(var.get() for var in self.attributes.values())
54.
55.         animal_type = self.get_animal_type(input_values)
56.
57.         # 在消息框中显示结果
58.         messagebox.showinfo("动物分类结果", f"分类的动物是：{animal_type}")
59.
60.     def get_animal_type(self, input_values):
61.         # 根据输入的属性值确定动物类型的逻辑
62.         # 从规则字典中查找匹配的规则
63.         for rule, animal_type in self.animal_rules.items():
64.             if input_values == rule:
65.                 return animal_type
66.
67.         return "未知动物类型"
68.
69.
70. def main():
71.     root = tk.Tk()
72.     app = AnimalExpertSystemGUI(root)
73.     root.mainloop()
74.
75. if __name__ == "__main__":
76.     main()
```