人工智能原理实验报告

学生姓名: 文华

学 号: 2017218007

专 业: 物联网工程

班 级: 2017 级 2 班

指导教师: 李磊、卜晨阳

完成日期: 2020 年 4 月 30 日

目 录

实验一 猴子摘香蕉问题的 Python 编程实现

1.1 实验目的

1) 熟悉谓词逻辑表示法；
2) 掌握人工智能谓词逻辑中的经典例子——猴子摘香蕉问题的编程实现。

1.2 实验内容

如图 1.1 所示，房间里有一只猴子（机器人），位于 a 处。在 c 处的上方的天花板上有一串香蕉，猴子想吃，但是摘不到。房间的 b 处还有一个箱子，如果猴子站到箱子上就可以摸到天花板。对于上述问题，可以通过谓词表示法来描述知识。要求通过 python 语言编程实现猴子摘香蕉问题的求解过程。

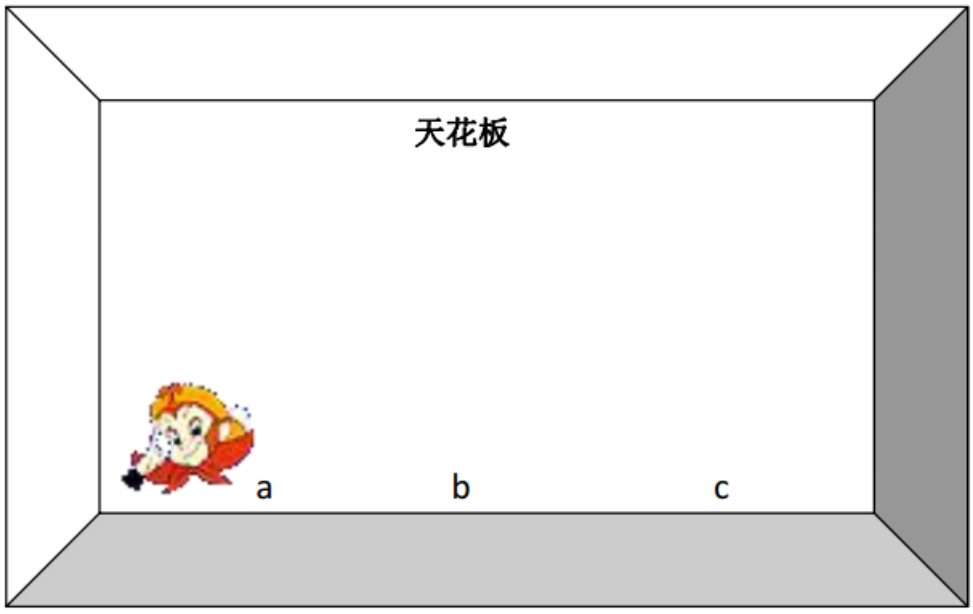

图 1.1 猴子摘桃示意图

1.3 实验环境

硬件：Dell G3 3579；

软件：

OS：Windows 10 Pro N for Workstations；

开发工具：PyCharm 2019.3.4 (Professional Edition)、微信小程序 onLab；

编程语言：Python 7.3.4。

1.4 源码实现

本次实验的 Python 实现代码如下所示。

```
'''
猴子摘香蕉问题的 Python 编程实现
'''
#全局变量 i
i = 0
def Monkey_go_box(x, y):
    global i
    i = i + 1
    print('step:', i, 'monkey 从', x, '走到' + y)

def Monkey_move_box(x, y):
    global i
    i = i + 1
    print('step:', i, 'monkey 把箱子从', x, '运到' + y)

def Monkey_on_box():
    global i
    i = i + 1
    print('step:', i, 'monkey 爬上箱子')

def Monkey_get_banana():
    global i
    i = i + 1
    print('step:', i, 'monkey 摘到香蕉')


import sys
```

```
if __name__ == '__main__':
    print('请用‘a’、‘b’、‘c’表示猴子香蕉箱子的位置')

    # 读取输入的运行参数
    codeIn = sys.stdin.read()
    # codeIn = 'bac'
    codeInList = codeIn.split()
    # 将运行参数赋值给 monkey、banana、box
    monkey = codeInList[0]
    banana = codeInList[1]
    box = codeInList[2]
    print('操作步骤如下：')
    if monkey != box:
        Monkey_go_box(monkey, box)
    if box != banana:
        Monkey_move_box(box, banana)
    Monkey_on_box()
    Monkey_get_banana()
```

1.5 实验结果

本次实验的结果如下所示，图 1.2 与图 1.3 均是在小程序的运行结果。

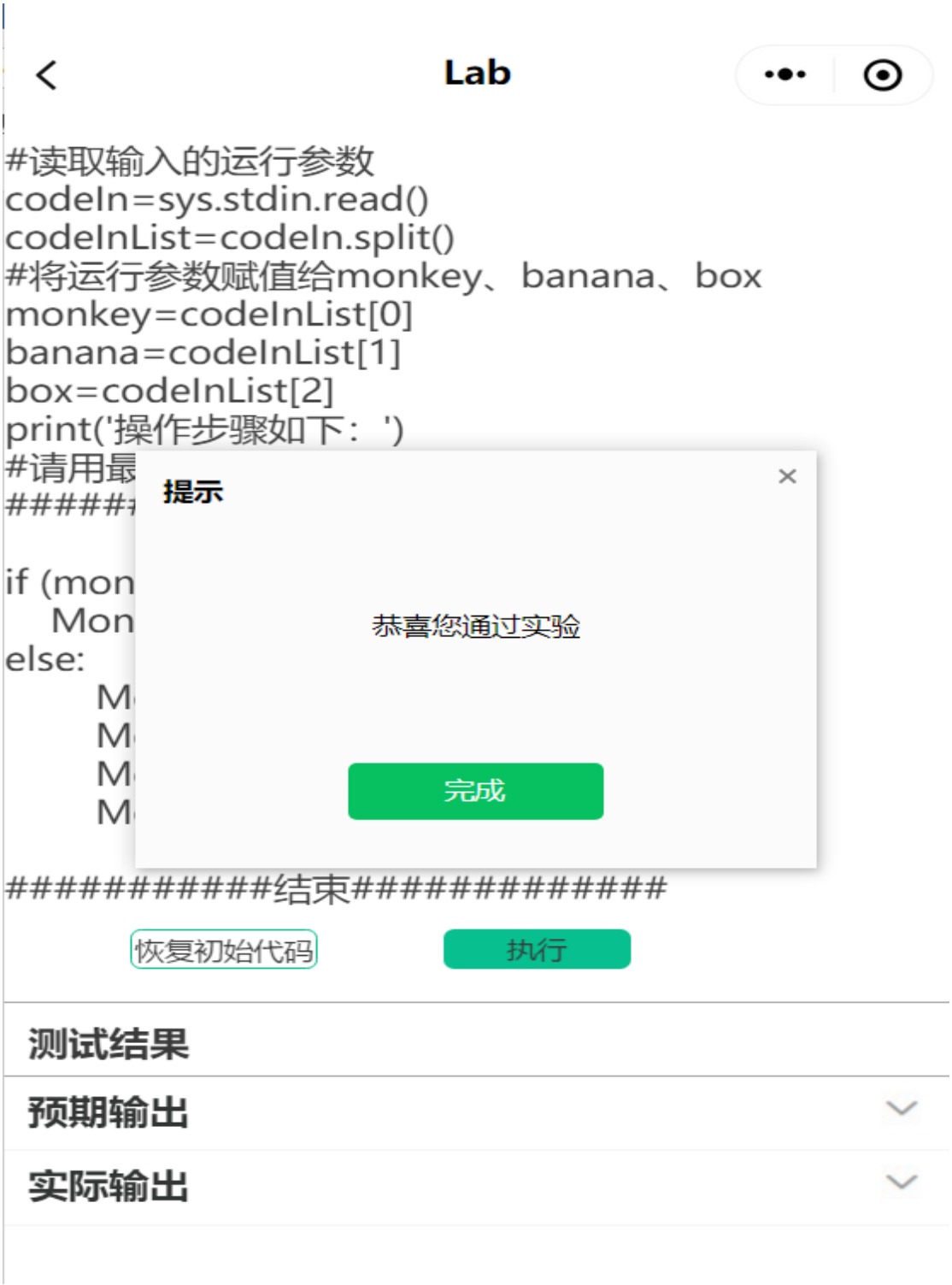

图 1.2 实验 1 在小程序平台的运行情况 1

图 1.3 实验 1 在小程序平台的运行情况 2

1.6 心得体会

通过本次实验，我加深了对谓词逻辑的理解，想进一步认识其内涵。以前粗略学习的 Python 在做实验时派上了用武之地，总算没有白学，甚是欣慰。这个小程序的编辑功能很弱，但编译较理想。

实验三 搜索算法求解 8 数码问题

2.1 实验目的

1) 熟悉人工智系统中的问题求解过程；
2) 熟悉状态空间中的盲目搜索策略；
3) 掌握盲目搜索算法，重点是宽度优先搜索和深度优先搜索算法。

2.2 实验内容

用 python 语言编程，采用宽度优先搜索和深度优先搜索的方法，求解 8 码问题。

采用宽度优先算法，运行程序，要求输入初始状态

假定输入状态如下：

2 8 3

1 6 4

7 0 5

目标状态为：

2 1 6

4 0 8

7 5 3

每次选扩展结点时，从数组的最后一个生成的节点开始找，找出一个没有扩展的节点，这样也需要对节点添加一个是否被扩展过的标志。

2.3 实验环境

硬件：Dell G3 3579；

软件：

OS：Windows 10 Pro N for Workstations；

开发工具：PyCharm 2019.3.4 (Professional Edition)、微信小程序 onLab；

编程语言：Python 7.3.4。

2.4 源码实现

本次实验的 Python 实现代码如下所示。

```
import numpy as np

Depth = 1;
class State:
    def __init__(self, state, directionFlag=None, parent=None,depth = 1):
        self.state = state
        # state is a ndarray with a shape(3,3) to storage the state
        self.direction = ['up', 'down', 'right', 'left']
        if directionFlag:
            self.direction.remove(directionFlag)
        # record the possible directions to generate the sub-states
        self.parent = parent
        self.depth = 1

    def showInfo(self):
        for i in range(3):
            for j in range(3):
                print(self.state[i, j], end='  ')
            print("\n")
        print('->')
        return

    def getEmptyPos(self):
        postion = np.where(self.state == self.symbol)
        return postion

    def generateSubStates(self):  # 产生子节点
        if not self.direction:
            return []
        subStates = []
        boarder = len(self.state) - 1
        # the maximum of the x,y
```

```
        row, col = self.getEmptyPos()
        if 'left' in self.direction and col > 0:  # 向左移动
            s = self.state.copy()
            # 标志位 symbol=0 向左移动，产生新的状态节点，加入到 subStates 中

            temp = s.copy()
            s[row, col] = s[row, col - 1]
            s[row, col - 1] = temp[row, col]
            news = State(s, directionFlag='right', parent=self)
            subStates.append(news)

        if 'up' in self.direction and row > 0:
            s = self.state.copy()
            # 标志位 symbol=0 向上移动，产生新的状态节点，加入到 subStates 中

            temp = s.copy()
            s[row, col] = s[row - 1, col]
            s[row - 1, col] = temp[row, col]
            news = State(s, directionFlag='down', parent=self)
            subStates.append(news)

        if 'down' in self.direction and row < boarder:  # it can move to down place
            s = self.state.copy()
            # 标志位 symbol=0 向下移动，产生新的状态节点，加入到 subStates 中

            temp = s.copy()
            s[row, col] = s[row + 1, col]
            s[row + 1, col] = temp[row, col]
            news = State(s, directionFlag='up', parent=self)
            subStates.append(news)

        if self.direction.count('right') and col < boarder:  # it can move to right place
            s = self.state.copy()
            # 标志位 symbol=0 向右移动，产生新的状态节点，加入到 subStates 中

            temp = s.copy()
            s[row, col] = s[row, col + 1]
            s[row, col + 1] = temp[row, col]
            news = State(s, directionFlag='left', parent=self)
            subStates.append(news)

        # end1
        return subStates

    def BFS(self):
        # generate a empty openTable
        openTable = []  # 存放状态的地方
        path = [] # 存放答案的地方
        # append the origin state to the openTable
        openTable.append(self)  # 将初始状态加入
        steps = 1  # 步骤
        seen = []
        while len(openTable) > 0 :
            n = openTable.pop(0)  # pop() 函数用于移除列表中的一个元素（默认最后一个元素），并且返回该元素的值。
            subStates = n.generateSubStates()
            seen.append(n);
            steps = steps + 1;  # 每展开一个节点，步骤+1
```

```
            # 查看子状态中有没有最终状态，如果有则输出之前的父状态到 path
中，输出 step+1

            for subState in subStates :
                if (subState.state == State.answer).all():
                    subState = subState.parent
                    #通过父节点不断追溯寻找父节点，直到找到最初结点
                    while(subState.state != s1.state).any() :
                        path.append(subState);
                        subState = subState.parent;
                    path.reverse();
                    return path,steps


            # 将子状态添加到 openTable 中

                if subState not in seen:
                    openTable.append(subState);

        else:
            return None, None


if __name__ == '__main__':
    State.symbol = 0
    State.answer = np.array([[1, 2, 3], [8, 0, 4], [7, 6, 5]])
    s1 = State(np.array([[2, 8, 3], [1, 6, 4], [7, 0, 5]]))
    path, steps = s1.BFS()
    if path:  # if find the solution
        for node in path:
            # print the path from the origin to final state
            node.showInfo()
        print(State.answer)
        print("Total steps is %d" % steps)
```

2.5 实验结果

本次实验的结果如下所示，图 2.1 与图 2.2 均是在小程序的运行结果，图 2.3 为本地的运行结果。

图 2.1 实验 2 在小程序平台的运行情况 1

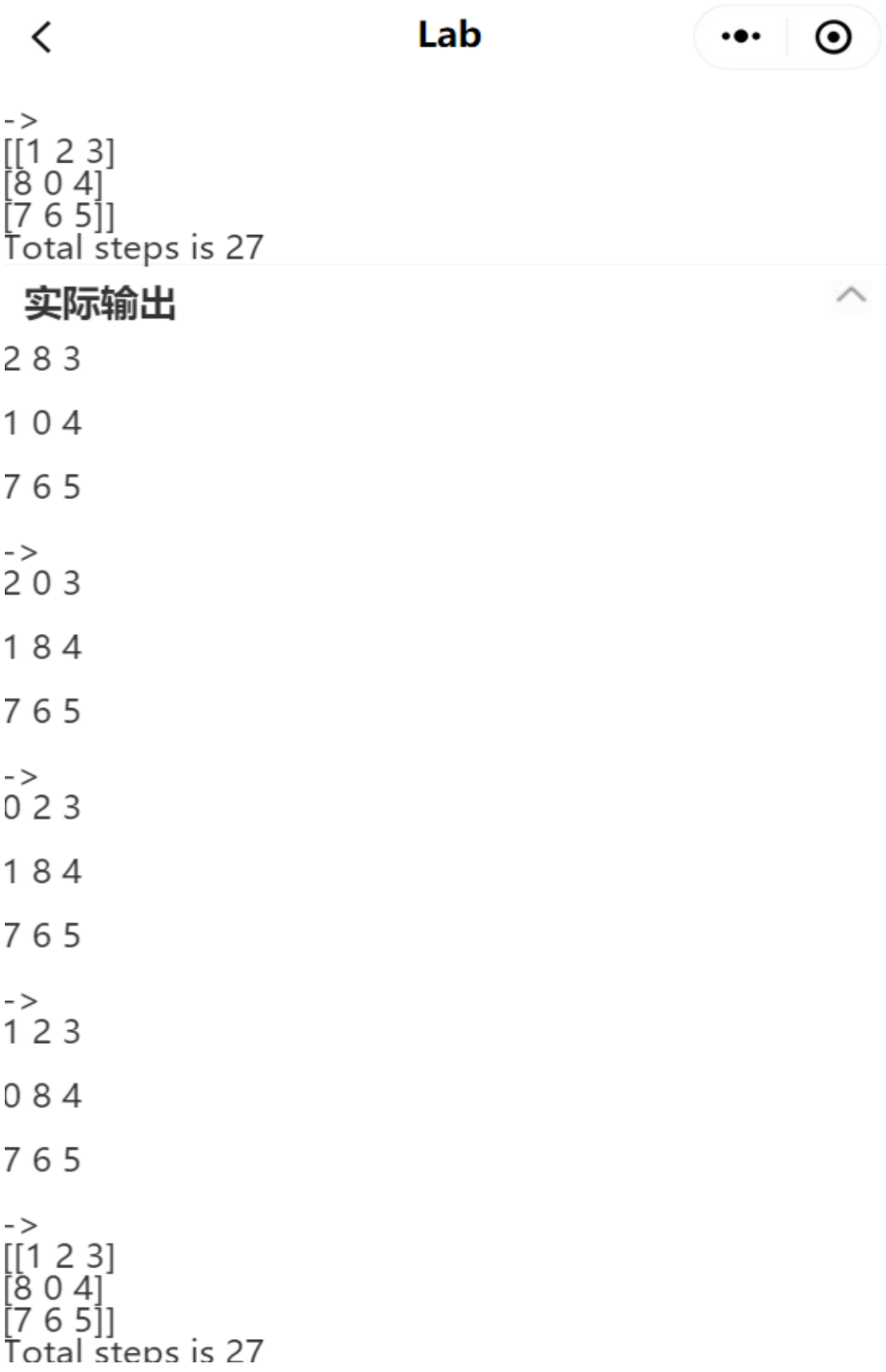

图 2.2 实验 2 在小程序平台的运行情况 2

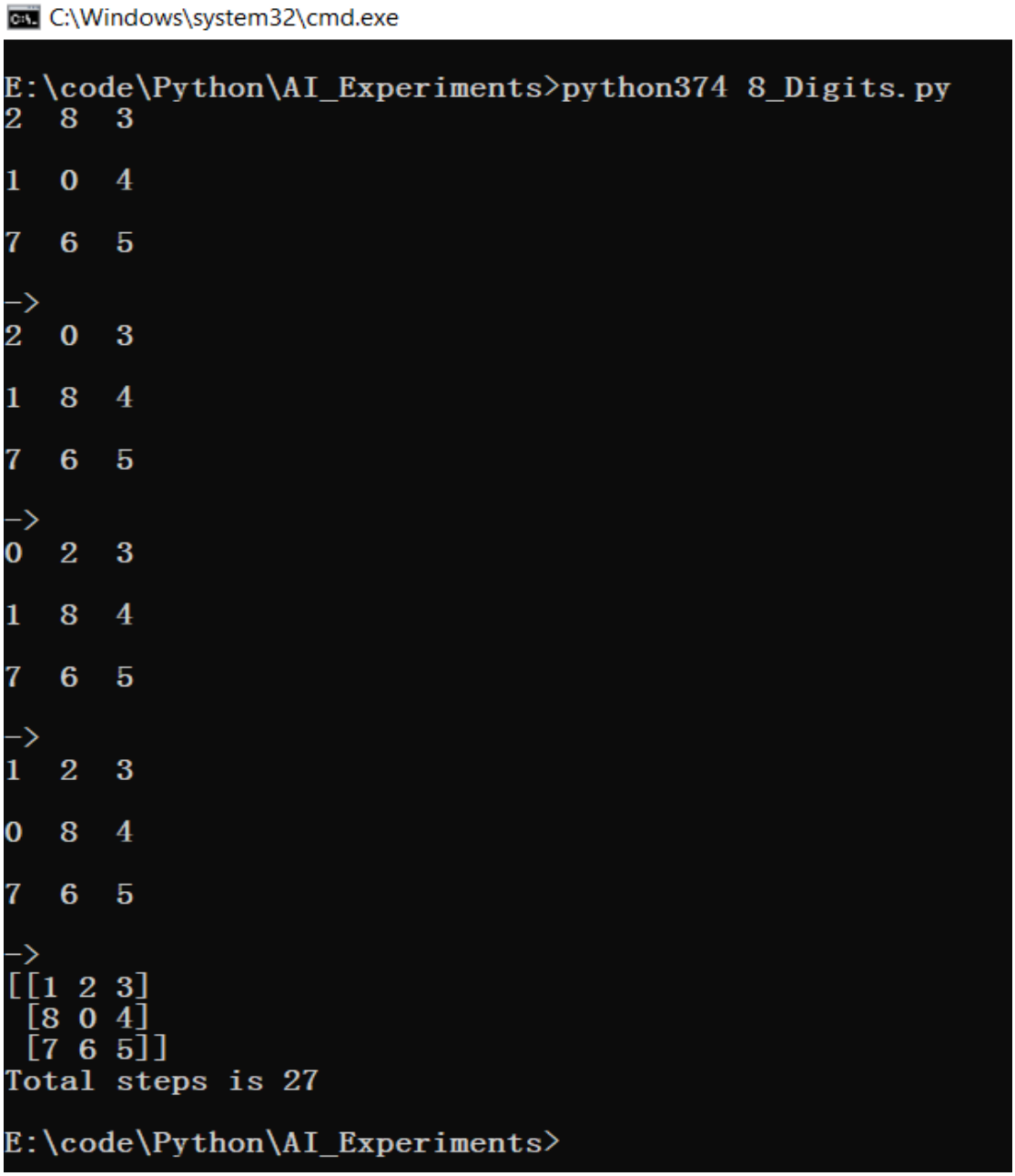

图 2.3 实验 2 在本地的运行情况

2.6 心得体会

通过本次实验，增强了原先对深搜和宽搜的理解，经过编码最终得到了问题的结果，加深了我对算法的感性认识，提高动手能力与解决实际问题的能力。看来搜索真的很重要，好几门课都提到了。

实验四 子句集消解实验

3.1 实验目的

1) 熟悉子句集化简的九个步骤；
2) 理解消解规则，能把任意谓词公式转换成为子句集。

3.2 实验内容

在谓词逻辑中，任何一个谓词公式都可以通过应用等价关系以及推理规则化成相应的子句集。其化简的一般步骤为：

（1） 消去蕴涵符号

使用等价公式 $P\rightarrow Q \Leftrightarrow \neg P\vee Q$，消去谓词公式中所有的蕴含连接词。

例如公式

$(\forall x)((\forall y)P(x, y) \rightarrow \neg (\forall y)(Q(x,y)\rightarrow R(x, y)))$

消去蕴含后等价变化为：

$(\forall x)(\neg (\forall y)P(x, y) \vee \neg(\forall y)(\neg Q(x,y) \vee R(x, y)))$

（2）缩小否定符号的作用范围（辖域）

反复使用以下推理规则，将每个否定符号“$\neg$”移到紧靠谓词的位置，即使得每个否定符号只作用于一个谓词上。

双重否定律

$\neg(\neg P)\Leftrightarrow P$

德摩根定律

$\neg(P\wedge Q) \Leftrightarrow \neg P\vee \neg Q$

$\neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q$

量词转换律

$\neg (\forall x)P(x) \Leftrightarrow (\exists x) \neg P(x)$

$\neg (\exists x)P(x) \Leftrightarrow (\forall x) \neg P(x)$

（3）变量命名标准化

在一个量词的辖域内，把谓词公式中受该量词约束的变元全部用另外一个没有出现过的任意变元代替，使不同量词约束的变元有不同的名字。

例如，

$(\forall x)((\exists y) \neg P(x, y) \vee (\exists y)(Q(x, y) \wedge \neg R(x, y)))$

上式经变换后为

$(\forall x)\{(\exists y) \neg P(x, y) \vee (\exists z)[Q(x, z) \wedge \neg R(x, z)]\}$

(4)消去存在量词(Skolem 化)

引入 Skolem 函数，消去存在量词，分为两种情况。

①形如$(x) [(y)P(x,y)]$，存在量词在全称量词的辖域内存在量词 y 可能依赖于 x，令这种依赖关系为 y=f(x)，由 f(x)把每个 x 值映射到存在的那个 y，f(x)叫做 Skolem 函数。

引入 Skolem 函数 f(x)后，上面的公式变为：

$(x) [P(x, f(x)]$

(5)化为前束范式

将公式中的所有约束量词移到公式最前面，形成前束公式，形式如下：

前束式=（约束量词前缀）（母式）

在移动时不能改变其相对顺序。

由于第(3)步已对变元进行了标准化，每个量词都有自己的变元，这就消除了任何由变元引起冲突的可能，即：移动不改变量词辖域。

(6)消去全称量词

由于母式中的全部变元均受全称量词的约束，并且全称量词的次序已无关紧要，因此可以省掉全称量词。剩下的母式，默认所有变元受全称量词约束。

(7)化为合取范式

反复运用谓词公式的结合律和分配律，将公式化为合取范式。

(8)将公式用子句集合表示

取出合取范式中的每个子句；

对子句变量换名，使任意两个子句中不出现相同的变量名，便于推理中的置换、合一。

3.3 实验环境

硬件：Dell G3 3579；

软件：

OS：Windows 10 Pro N for Workstations；

开发工具：PyCharm 2019.3.4 (Professional Edition)、微信小程序 onLab；

编程语言：Python 7.3.4。

3.4 源码实现

本次实验的 Python 实现代码如下所示。

```
'''
字句集消解实验
'''
'''
->:>
析取：%
合取：^
全称：@
存在：#
'''
# 1.消去>蕴涵项 a>b 变为~a%b
def del_inlclue(orign):

    ind = 0
    flag=0
    orignStack=[]
    right_bracket = 0
    while (ind < len(orign)):
        orignStack.append(orign[ind])
```

```
        if ((ind+1<len(orign)) and (orign[ind+1]=='>')):
            flag=1
            if orign[ind].isalpha():#是字母
                orignStack.pop()
                orignStack.append('~')
                orignStack.append(orign[ind])
                orignStack.append('%')
                ind=ind+1
            if orign[ind]==')':
                right_bracket=right_bracket+1
                tempStack = []
                while(right_bracket!=-1):
                    tempStack.append(orignStack[-1])
                    if orignStack[-1]=='(':
                        right_bracket=right_bracket-1
                    orignStack.pop()
                right_bracket = right_bracket + 1
                tempStack.pop()#去掉'('
                orignStack.append('(~')
                tempStack.reverse()
                for i in tempStack:
                    orignStack.append(i)
                orignStack.append('%')
                ind=ind+1
        ind=ind+1
    if flag==1:
        a=''
        for i in orignStack:
            a=a+i
        return a
    else:
        return orign


#2.处理否定连接词
def dec_neg_rand(orign):

    #处理~(@x)p(x) 变为(#x)~p(x)#########################################
    ind = 0
    flag = 0
    orignStack = []
    left_bracket = 0
    while (ind < len(orign)):
        orignStack.append(orign[ind])
        if orign[ind]=='~':
            if orign[ind+1]=='(':
                if orign[ind+2]=='@' or orign[ind+2]=='#':
                    flag=1
                    ind=ind+1
                    orignStack.pop()#去掉前面的~
                    orignStack.append(orign[ind])
                    if orign[ind+1]=='@':
                        orignStack.append('#')
                    else:
                        orignStack.append('@')
                    orignStack.append(orign[ind+2])#'x'
                    orignStack.append(orign[ind+3])#')'
                    orignStack.append('~')
```

```
                ind=ind+3
        ind=ind+1
    if flag==1:
        a=''
        for i in orignStack:
            a=a+i
        orign2=a
    else:
        orign2=orign
    #print('orign2',orign2)


    # 处理~(p%q) 变为(~p^~q)########################################
    ind = 0
    flag = 0
    flag2 = 0  # 判断是否进入 while left_bracket>=1:循环， 若进入， 出来后 ind 再减 1
    orignStack = []
    left_bracket = 0
    while (ind < len(orign2)):
        orignStack.append(orign2[ind])
        if orign2[ind] == '~':
            if orign2[ind + 1] == '(':
                orignStack.pop()

                ind=ind+2#此时为 p
                left_bracket=left_bracket+1
                orignStack.append('(~')
                while left_bracket>=1:
                    flag2=1
                    orignStack.append(orign2[ind])
                    if orign2[ind]=='(':
                        left_bracket=left_bracket+1
                    if orign2[ind]==')':
                        left_bracket=left_bracket-1
                    if left_bracket==1 and orign2[ind+1]=='%' and orign2[ind+2]!='@' and orign2[ind+2]!='#':
                        flag=1
                        orignStack.append('^~')
                        ind=ind+1
                    if left_bracket == 1 and orign2[ind + 1] == '^' and orign2[ind + 2] != '@' and orign2[ind + 2] != '#':
                        flag = 1
                        orignStack.append('%~')
                        ind = ind + 1
                    ind=ind+1
        if flag2==1:
            ind=ind-1
            flag2=0
        ind=ind+1
    if flag==1:
        a=''
        for i in orignStack:
            a=a+i
        orign3=a
    else:
        orign3=orign2
    #print('orign3',orign3)
```

```
# 处理~~p 变为 p########################################
ind = 0
flag = 0
bflag = 0
orignStack = []
while (ind < len(orign3)):
    orignStack.append(orign3[ind])
    if orign3[ind] == '~':
        if orign3[ind + 1] == '~':
            flag = 1
            orignStack.pop()
            ind = ind + 1
    ind = ind + 1

if flag == 1:
    a = ''
    for i in orignStack:
        a = a + i
    orign4 = a
else:
    orign4 = orign3
# print('orign4', orign4)
# 处理~(~p) 变为 p########################################
ind = 0
flag = 0
bflag=0
orignStack = []
while (ind < len(orign4)):
    orignStack.append(orign4[ind])
    if orign4[ind] == '~':
        if orign4[ind + 1] == '(':
            left_bracket = 1
            if orign4[ind+2]=='~':
                flag=1
                orignStack.pop()
                ind=ind+2
                while left_bracket>=1:
                    orignStack.append(orign4[ind+1])
                    if orign4[ind+1]=='(':
                        left_bracket=left_bracket+1
                    if orign4[ind+1]==')':
                        left_bracket=left_bracket-1
                    if orign4[ind+1]=='%' or orign4[ind+1]=='^':
                        bflag=1
                    ind=ind+1

                orignStack.pop()

    ind = ind + 1

if flag==1 and bflag==0:
    a=''
    for i in orignStack:
        a=a+i
    orign5=a
else:
```

```
        orign5=orign4
    #print('orign5',orign5)
    return orign5


#3.命题变量标准化，使后面 y=w
def standard_var(orign):#对变量标准化,简化,不考虑多层嵌套

    flag = 1
    desOri=[]
    des=['w','k','j']
    j=0
    orignStack = []
    left_bracket=0
    ind = 0
    while flag!=0:
        flag=0
        while (ind < len(orign)):
            orignStack.append(orign[ind])
            if orign[ind] == '@' or orign[ind]=='#':
                x=orign[ind+1]#保存 x
                if orign[ind+1] in desOri:
                    orignStack.append(des[j])
                    desOri.append(des[j])
                    j=j+1
                    orignStack.append(')')
                    ind=ind+3
                    if ind<len(orign):
                        if orign[ind].isalpha():#(@x)p(x,y)这种情况
                            orignStack.append(orign[ind])#p
                            ind = ind + 1
                            if orign[ind]=='(':
                                left_bracket = left_bracket + 1
                                orignStack.append(orign[ind])
                                ind=ind+1
                                while left_bracket>0:
                                    if orign[ind]== ')':
                                        left_bracket = left_bracket - 1
                                    if orign[ind]== '(':
                                        left_bracket=left_bracket+1
                                    if orign[ind]== x:
                                        flag=1
                                        orignStack.append(des[j-1])
                                    else:
                                        orignStack.append(orign[ind])
                                    ind=ind+1
                                ind=ind-1

                    if ind<len(orign):
                        if orign[ind] == '(' :
                            left_bracket = left_bracket + 1
                            orignStack.append(orign[ind])
                            ind = ind + 1
                            while left_bracket > 0:
                                if orign[ind] == ')':
                                    left_bracket = left_bracket - 1
                                if orign[ind] == '(':
                                    left_bracket = left_bracket + 1
```

```
                        if orign[ind] == x:
                            flag = 1
                            orignStack.append(des[j - 1])
                        else:
                            orignStack.append(orign[ind])
                        ind = ind + 1
                    ind=ind-1

            else:
                desOri.append(orign[ind+1])
        ind=ind+1

    a=''
    for i in orignStack:
        a=a+i
    orign2=a
    return orign2

#4.消去存在量词（skolem 化）
def del_exists(orign):
    ind = 0
    flag = 1
    orignStack = []
    x=''
    y=''
    # 第 1 种情况：前面有全称量词 (@x)((#y)p(x,y))变为（@x）p(x,f(x))
    while flag!=0: #为了嵌套的情况出现
        flag=0
        while (ind < len(orign)):
            orignStack.append(orign[ind])

            if orign[ind] == '(' and orign[ind+1] == '@' and orign[ind+4]=='(' :
                x=orign[ind+2]
                orignStack.append(orign[ind+1:ind+5])
                ind=ind+5#指向
                while orign[ind]!='#':
                    orignStack.append(orign[ind])
                    ind=ind+1
                orignStack.pop()
                y=orign[ind+1]#为 y
                ind=ind+2#指向 p
                flag=1

            ind = ind + 1

        if flag==1:
            orignStack2=[]
            for i in orignStack:
                if i==y:
                    orignStack2.append('g(')
                    orignStack2.append(x)
                    orignStack2.append(')')
                else:
                    orignStack2.append(i)
    a = ''
    for i in orignStack2:
        a = a + i
    orign2 = a
```

```
    ind = 0
    flag = 1
    orignStack = []
    # 第 2 种情况：前面没有全称量词 (#y)p(x,y)变为 p(x,A)
    while flag != 0:  # 为了嵌套的情况出现
        flag = 0
        while (ind < len(orign2)):
            orignStack.append(orign2[ind])
            if orign2[ind] == '#' :
                y=orign2[ind+1]
                orignStack.pop()
                orignStack.pop()
                ind=ind+2#指向')'
                flag=1
            ind = ind + 1
        if flag==1:
            orignStack2 = []
            for i in orignStack:
                if i == y:
                    orignStack2.append('A')
                else:
                    orignStack2.append(i)

    a = ''
    for i in orignStack2:
        a = a + i
    orign2 = a
    return orign2


#5.前束化
def convert_to_front(orign):#化为前束形
    ind = 0
    orignStack = []
    tempStack=[]#存放全称量词
    while (ind < len(orign)):
        orignStack.append(orign[ind])
        if orign[ind]=='(' and orign[ind+1]=='@':
            orignStack.pop()
            tempStack.append(orign[ind:ind+4])
            ind=ind+3

        ind = ind + 1

    orignStack=tempStack+orignStack
    a=''
    for i in orignStack:
        a = a + i
    orign2 = a
    return orign2

def del_all(orign):
    ind = 0
    orignStack = []
    ############开始 1##############
    while (ind < len(orign)):
        orignStack.append(orign[ind])
        if (orign[ind] == '@'):
```

```
            orignStack.pop()
            orignStack.pop()
            ind = ind + 2
        ind = ind + 1
    a = ''
    for i in orignStack:
        a = a + i
    orign2 = a
    return orign2

import sys

if __name__ == '__main__':
    codeIn = sys.stdin.read()
    orign=codeIn
    # orign = '(@x)(p(x)>(((@y)(p(y)>p(f(x,y)))^~(@y)(Q(x,y)>p(y)))))'
    print('orign:',orign)
    a=del_inlclue(orign)
    print('1.去除蕴含后:',a)
    a=dec_neg_rand(a)
    print('2.处理否定连接词后:')
    print(a)
    a=standard_var(a)
    print('3.变量命名标准化后:')
    print(a)
    a=del_exists(a)
    print('4.消去存在量词后:')
    print(a)
    a=convert_to_front(a)
    print('5.前束化后:')
    print(a)
    a=del_all(a)
    print('6.消去全称量词后:')
    print(a)

#选做(请私下完成)
# #定义化为合取范式、将公式转化为子句集合表示、更换变量名称的函数#
# 调用函数完成子句集的分解
```

3.5 实验结果

本次实验的结果如下所示，图 3.1 与图 3.2 均是在小程序的运行结果，图 3.3 为本地的运行结果。

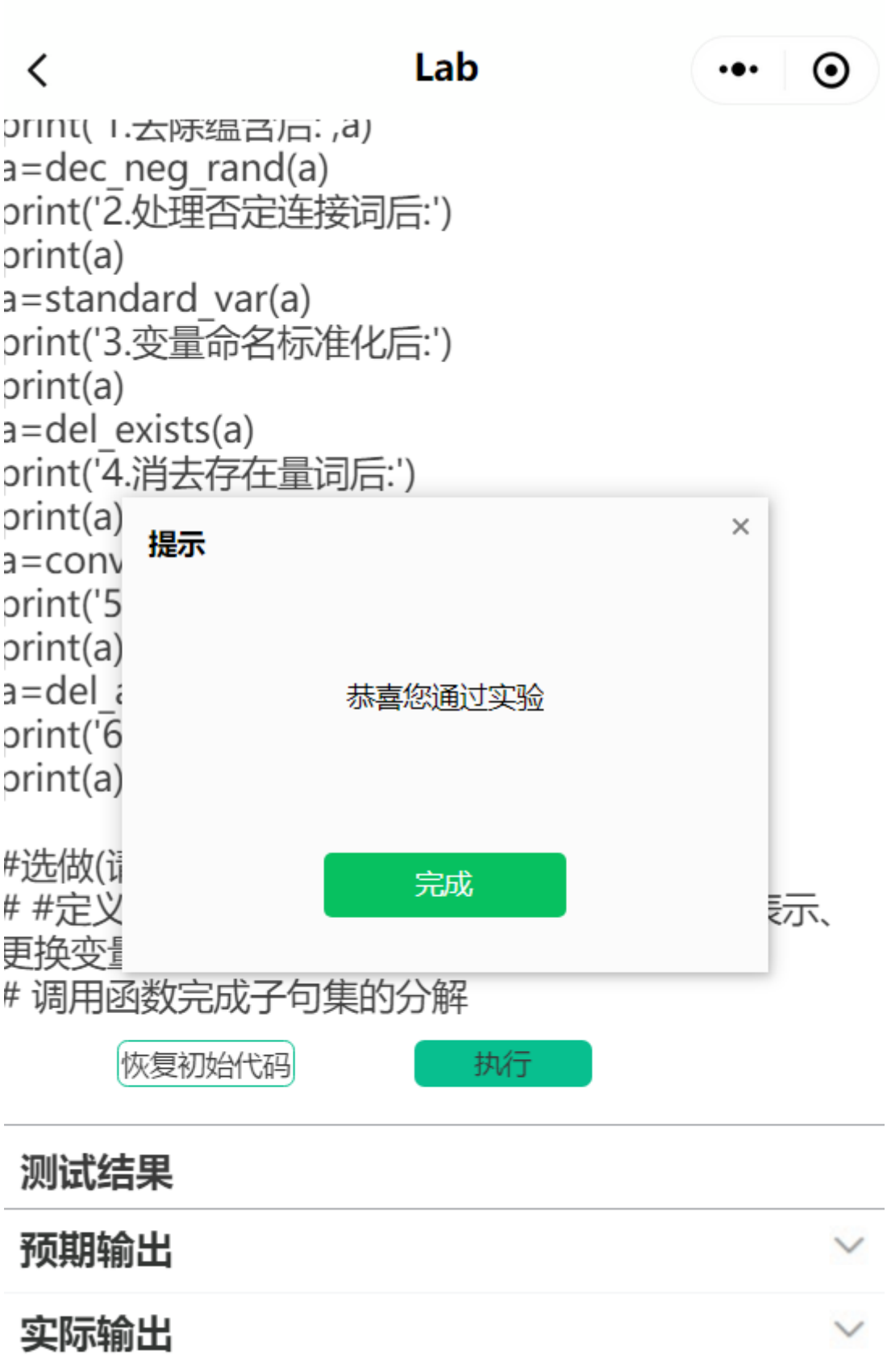

图 3.1 实验 3 在小程序平台的运行情况 1

图 3.2 实验 3 在小程序平台的运行情况 2

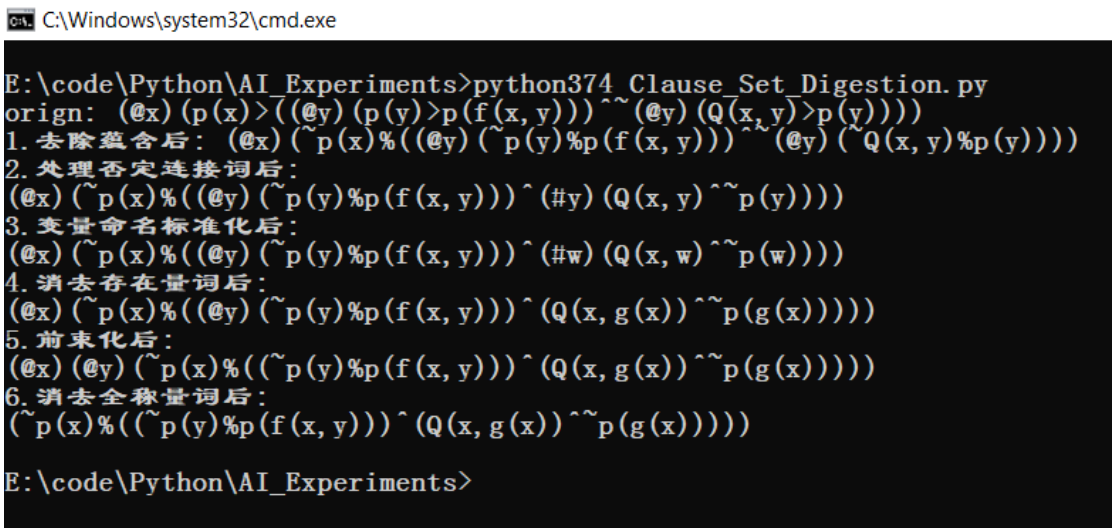

图 3.3 实验 3 在本地的运行情况

3.6 心得体会

通过本次实验，我比较全面地理解消解规则的基础上实现了子句集化解的“九步法”，并在实验编码中得到了验证与强化。子句消解太有意思啦！

实验六 蚁群算法在 TSP 问题中的实现

4.1 实验目的

1) 理解蚁群算法的原理以及优缺点；
2) 能够利用蚁群算法解决实际问题。

4.2 实验内容

旅行商问题（TSP）：一个商人去 n 个城市销货，所有城市走一遍再回到起点，使得所走的路程最短。要求使用 Python 语言利用蚁群算法解决 TSP 问题。

4.3 实验环境

硬件：Dell G3 3579；

软件：

OS：Windows 10 Pro N for Workstations；

开发工具：PyCharm 2019.3.4 (Professional Edition)、微信小程序 onLab；

编程语言：Python 7.3.4。

4.4 源码实现

本次实验的 Python 实现代码如下所示。

```
import random
import copy
import sys
# 参数
'''
ALPHA:信息启发因子，值越大，则蚂蚁选择之前走过的路径可能性就越大
    ，值越小，则蚁群搜索范围就会减少，容易陷入局部最优
BETA:Beta 值越大，蚁群越就容易选择局部较短路径，这时算法收敛速度会
   加快，但是随机性不高，容易得到局部的相对最优
'''
(ALPHA, BETA, RHO, Q) = (1.0, 2.0, 0.7, 100.0)
# 城市数，蚁群
(city_num, ant_num) = (50, 50)
distance_x = [
   178, 272, 176, 171, 650, 499, 267, 703, 408, 437, 491, 74, 532,
   416, 626, 42, 271, 359, 163, 508, 229, 576, 147, 560, 35, 714,
   757, 517, 64, 314, 675, 690, 391, 628, 87, 240, 705, 699, 258,
   428, 614, 36, 360, 482, 666, 597, 209, 201, 492, 294]
distance_y = [
   170, 395, 198, 151, 242, 556, 57, 401, 305, 421, 267, 105, 525,
   381, 244, 330, 395, 169, 141, 380, 153, 442, 528, 329, 232, 48,
   498, 265, 343, 120, 165, 50, 433, 63, 491, 275, 348, 222, 288,
   490, 213, 524, 244, 114, 104, 552, 70, 425, 227, 331]
# 城市距离和信息素
distance_graph = [[0.0 for col in range(city_num)] for raw in range(city_num)]
pheromone_graph = [[1.0 for col in range(city_num)] for raw in range(city_num)]


# ---------- 蚂蚁 ----------
class Ant(object):

    # 初始化
    def __init__(self, ID):

        self.ID = ID  # ID
        self.__clean_data()  # 随机初始化出生点

    # 初始数据
    def __clean_data(self):

        self.path = []  # 当前蚂蚁的路径
        self.total_distance = 0.0  # 当前路径的总距离
        self.move_count = 0  # 移动次数
        self.current_city = -1  # 当前停留的城市
        self.open_table_city = [True for i in range(city_num)]  # 探索城市的状态

        city_index = random.randint(0, city_num - 1)  # 随机初始出生点
        self.current_city = city_index
        self.path.append(city_index)
        self.open_table_city[city_index] = False
```

```
        self.move_count = 1

    # 选择下一个城市
    def __choice_next_city(self):

        next_city = -1
        select_citys_prob = [0.0 for i in range(city_num)] # 存储去下个城市的概率
        total_prob = 0.0

        # 获取去下一个城市的概率
        for i in range(city_num):
            if self.open_table_city[i]:
                try:
                    # 计算概率：与信息素浓度成正比，与距离成反比
                    select_citys_prob[i] = pow(pheromone_graph[self.current_city][i],
ALPHA) * pow(
                        (1.0 / distance_graph[self.current_city][i]), BETA)
                    total_prob += select_citys_prob[i]
                except ZeroDivisionError as e:
                    print('Ant ID: {ID}, current city: {current}, target city:
{target}'.format(ID=self.ID,
                                                                    current=self.current_city,
                                                                    target=i))
                    sys.exit(1)

        # 轮盘选择城市
        if total_prob > 0.0:
            # 产生一个随机概率,0.0-total_prob
            temp_prob = random.uniform(0.0, total_prob)
            for i in range(city_num):
                if self.open_table_city[i]:
                    # 轮次相减
                    temp_prob -= select_citys_prob[i]
                    if temp_prob < 0.0:
                        next_city = i
                        break

        # 未从概率产生，顺序选择一个未访问城市
        # if next_city == -1:
        #     for i in range(city_num):
        #         if self.open_table_city[i]:
        #             next_city = i
        #             break

        if (next_city == -1):
            next_city = random.randint(0, city_num - 1)
            while ((self.open_table_city[next_city]) == False):  # if==False,说明已经遍
历过了
                next_city = random.randint(0, city_num - 1)

        # 返回下一个城市序号
        return next_city

    # 计算路径总距离
    def __cal_total_distance(self):

        temp_distance = 0.0
```

```
        for i in range(1, city_num):
            start, end = self.path[i], self.path[i - 1]
            temp_distance += distance_graph[start][end]

        # 回路
        end = self.path[0]
        temp_distance += distance_graph[start][end]
        self.total_distance = temp_distance

    # 移动操作
    def __move(self, next_city):

        self.path.append(next_city)
        self.open_table_city[next_city] = False
        self.total_distance += distance_graph[self.current_city][next_city]
        self.current_city = next_city
        self.move_count += 1

    # 搜索路径
    def search_path(self):

        # 初始化数据
        self.__clean_data()

        # 搜素路径，遍历完所有城市为止
        while self.move_count < city_num:
            # 移动到下一个城市
            next_city = self.__choice_next_city()
            self.__move(next_city)

        # 计算路径总长度
        self.__cal_total_distance()

# ---------- TSP 问题 ----------
class TSP(object):
    def __init__(self):
        self.new()
        # 计算城市之间的距离
        for i in range(city_num):
            for j in range(city_num):
                temp_distance = pow((distance_x[i] - distance_x[j]), 2) +
pow((distance_y[i] - distance_y[j]), 2)
                temp_distance = pow(temp_distance, 0.5)
                distance_graph[i][j] = float(int(temp_distance + 0.5))

    # 初始化
    def new(self):
        # 初始城市之间的距离和信息素
        for i in range(city_num):
            for j in range(city_num):
                pheromone_graph[i][j] = 1.0

        self.ants = [Ant(ID) for ID in range(ant_num)]  # 初始蚁群
        self.best_ant = Ant(-1)  # 初始最优解
        self.best_ant.total_distance = 1 << 31  # 初始最大距离
        self.iter = 1  # 初始化迭代次数

    # 开始搜索
```

```
    def search_path(self, evt=None):



        while (self.iter<=100):
            # 遍历每一只蚂蚁
            for ant in self.ants:
                # 搜索一条路径
                ant.search_path()
                # 与当前最优蚂蚁比较
                if ant.total_distance < self.best_ant.total_distance:
                    # 更新最优解
                    self.best_ant = copy.deepcopy(ant)
            # 更新信息素
            self.__update_pheromone_gragh()
            # print(u"迭代次数：", self.iter, u"最佳路径总距离：",
int(self.best_ant.total_distance))

            self.iter += 1
            if self.iter == 100 and int(self.best_ant.total_distance) <= 4000:
                # if int(self.best_ant.total_distance) <= 4000:
                    print('success')

    # 更新信息素
    def __update_pheromone_gragh(self):

        # 获取每只蚂蚁在其路径上留下的信息素
        temp_pheromone = [[0.0 for col in range(city_num)] for raw in range(city_num)]
        for ant in self.ants:
            for i in range(1, city_num):
                start, end = ant.path[i - 1], ant.path[i]
                # 在路径上的每两个相邻城市间留下信息素，与路径总距离反比
                temp_pheromone[start][end] += Q / ant.total_distance
                temp_pheromone[end][start] = temp_pheromone[start][end]

        # 更新所有城市之间的信息素，旧信息素衰减加上新迭代信息素
        for i in range(city_num):
            for j in range(city_num):
                pheromone_graph[i][j] = pheromone_graph[i][j] * (1 - RHO) + RHO *
temp_pheromone[i][j]

# ---------- 程序的入口处 ----------
if __name__ == '__main__':
    TSP().search_path()
```

4.5 实验结果

本次实验的结果如下所示，图 4.1 为本地的运行结果，图 4.2 与图 4.3 均是在小程序的运行结果。

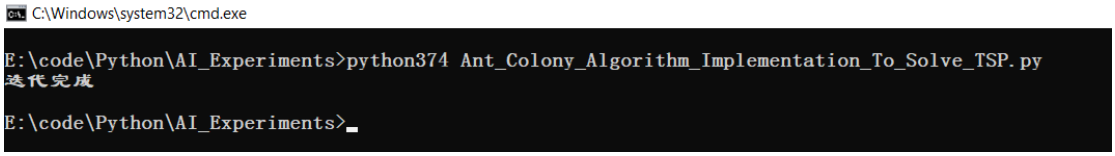

图 4.1 实验 4 在本地的运行情况

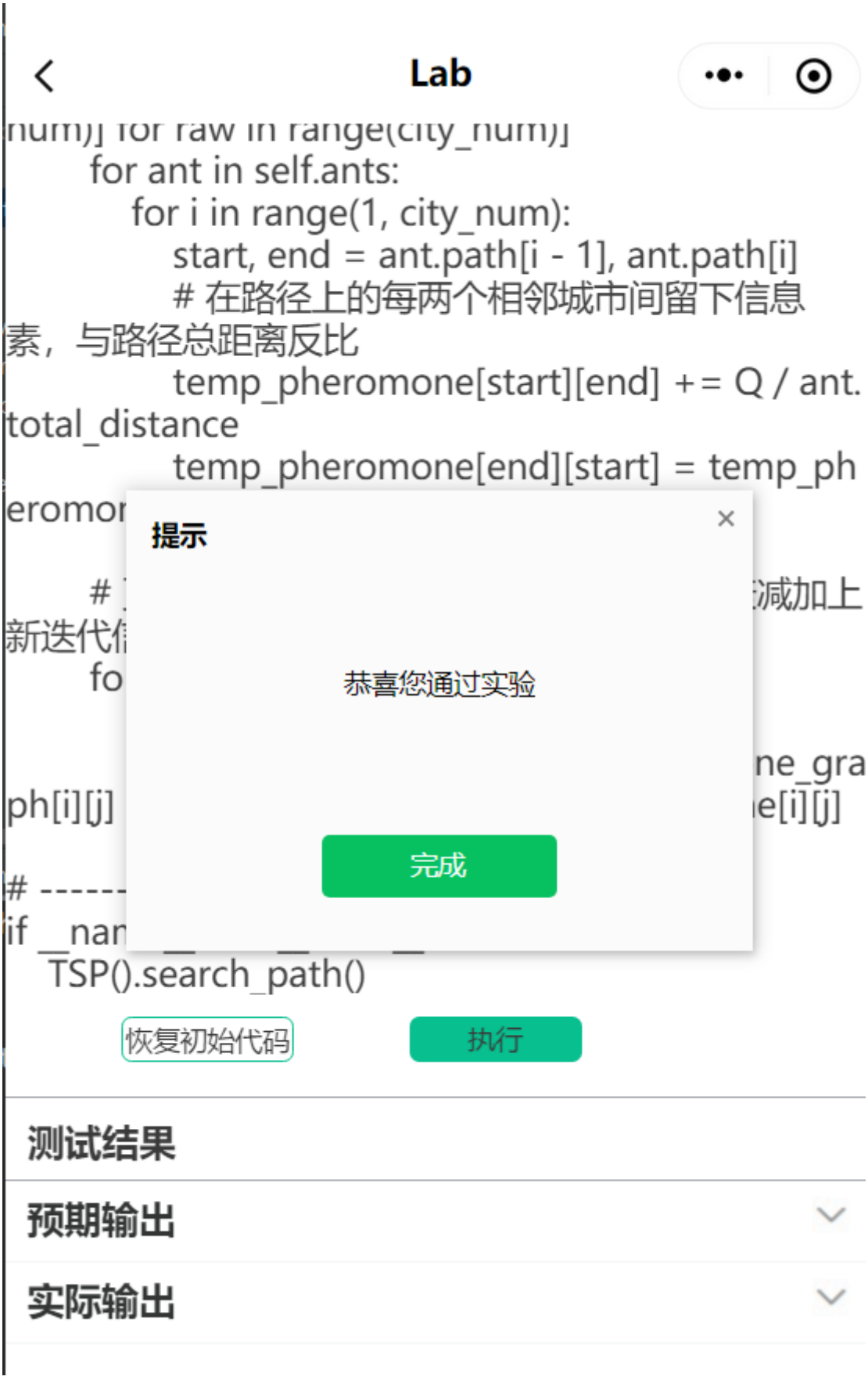

图 4.2 实验 4 在小程序平台的运行情况 1

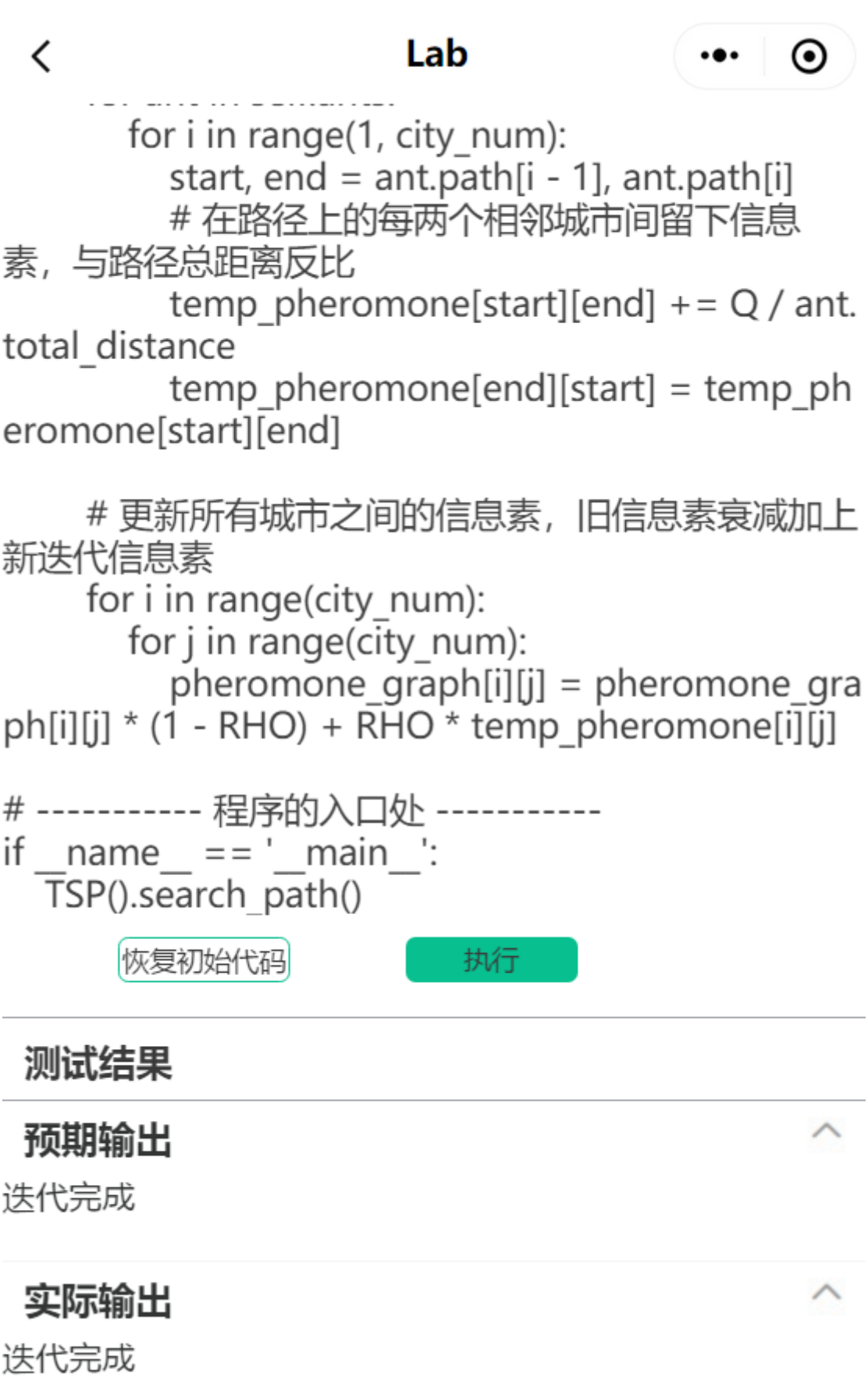

图 4.3 实验 4 在小程序平台的运行情况 2

4.6 心得体会

通过本次实验，使我加深了对蚁群算法的理解与感性认识，更激发了我进一步了解群体智能的兴趣。在离散数学实验中曾经使用图论中的最短路算法求解过 TSP 问题，这次实验使用蚁群算法，殊途同归、各有千秋，喵啊！