

**教学上机实验报告**

**课程名称：** 人工智能

**任课教师姓名：** 侯蓓蓓

**学生学号：**  312105010207

**学生姓名：**  刘晨阳

**学生专业班级：** 计算机2106

**2022 ～ 2023 学年 第 2 学期**

|  |
| --- |
| **河南理工大学**  **教学上机实验报告评价分值标准** |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 序号 | 评价指标 | 分值 | 评价等级及参考分值 | | | | | 评价分 | | 优 | 良 | 中 | 合格 | 差 | | 1 | 实验报告内容完整充实 | 10 | 10 | 8 | 7 | 6 | 3 |  | | 2 | 实验内容书写规范、字迹工整认真 | 10 | 10 | 8 | 7 | 6 | 3 |  | | 3 | 实验过程叙述详细、概念正确，语言表达准确，结构严谨，调理清楚，逻辑性强，自己努力完成，没有抄袭。 | 30 | 30 | 26 | 23 | 20 | 10 |  | | 4 | 对实验过程中存在的问题分析详细透彻、深刻、全面、规范、，结合实验内容，有自己的个人见解和想法，并能结合该实验提出相关问题，给出解决方法。 | 30 | 30 | 26 | 23 | 20 | 10 |  | | 5 | 实验结果、分析和结论正确无误 | 20 | 20 | 17 | 15 | 13 | 6 |  | | 总得分 | | | | | | | |  |   签名（签章）：  日期： 年 月 日 |
|  |

|  |
| --- |
| **河南理工大学教学上机实验报告** |
| 上机时间 2023 年 3 月 23 日 |
| **实验题目：**  **动物识别系统** |
| **实验目的和要求：**  **实验目的**  1.熟悉知识的表示方法  2.掌握产生式系统的运行机制  3.理解产生式系统推理的基本方法  **实验要求**  设计编程实现一个小型动物识别系统，能识别虎、金钱豹、斑马、长颈鹿、鸵鸟、企鹅、信天翁等七种动物的产生式系统。  要求给定初始条件，能识别出是哪种动物。比如已知初始事实存放在综合数据库中：暗斑点，长脖子，长腿，奶，蹄。运行后得该动物是：长颈鹿等 |
| **实验过程**  1. 应用所学知识，编写产生式系统实验程序，建立知识库，运行推理求解。  2. 产生式系统实验程序，构建规则库如下所示。系统分为两部分，第一部分是初始化  综合数据库，分为两类，一类是间接数据库，即不指向最终动物的信息。另一类是直接数据库，即指向最终动物的信息。使用的是二维列表加上一个一维列表存储 key 与 value。  规则库：  r1： IF 该动物有毛发 THEN 该动物是哺乳动物  r2： IF 该动物有奶 THEN 该动物是哺乳动物  r3： IF 该动物有羽毛 THEN 该动物是鸟  r4： IF 该动物会飞 AND 会下蛋 THEN 该动物是鸟  r5： IF 该动物吃肉 THEN 该动物是食肉动物  r6： IF 该动物有犬齿 AND 有爪 AND 眼盯前方  THEN 该动物是食肉动物  r7： IF 该动物是哺乳动物 AND 有蹄  THEN 该动物是有蹄类动物  r 8： IF 该动物是哺乳动物 AND 是反刍动物  THEN 该动物是有蹄类动物  r9： IF 该动物是哺乳动物 AND 是食肉动物 AND 是黄褐色  AND 身上有暗斑点 THEN 该动物是金钱豹  r10：IF 该动物是哺乳动物 AND 是食肉动物 AND 是黄褐色  AND 身上有黑色条纹 THEN 该动物是虎  r11： IF 该动物是有蹄类动物 AND 有长脖子 AND 有长腿  AND 身上有暗斑点 THEN 该动物是长颈鹿  r 12：IF 该动物有蹄类动物 AND 身上有黑色条纹  THEN 该动物是斑马  r13：IF 该动物是鸟 AND 有长脖子 AND 有长腿 AND 不会飞  AND 有黑白二色 THEN 该动物是鸵鸟  r14： IF 该动物是鸟 AND 会游泳 AND 不会飞  AND 有黑白二色 THEN 该动物是企鹅  r15： IF 该动物是鸟 AND 善飞 THEN 该动物是信天翁  def init(key1,key2,value1,value2,list):  print(" 构建规则库：")  print("构建间接数据库，输入0结束：")  while(1):  str = input("请输入条件，以空格隔开：")  if(str=='0'):  break  a = str.split()  b = input("请输入结果：")  l = len(a)  for i in range(0,l):  list.append(a[i])  key1.append(a)  value1.append(b)  print("构建直接数据库，输入0结束：")  while(1):  str = input("请输入条件，以空格隔开：")  if (str == '0'):  break  a = str.split()  b = input("请输入结果：")  l = len(a)  for i in range(0, l):  list.append(a[i])  key2.append(a)  value2.append(b)  print("初始化完毕！")  return 1  def check(key1,key2,value1,value2,list):  mp = {}  l = len(list)  for i in range(0,l):  mp[list[i]]=0  str = input("请输入动物主要特征(使用空格分开)：")  list1 = str.split()  l1 = len(list1)  for i in range(0,l1):  mp[list1[i]]=1  len1=len(key1)  vis = [0]\*1000  while(1):  v=0  for i in range(0,len1):  if(vis[i]):  continue  list1 = key1[i]  len11 = len(list1)  flag = 1  for j in range(0,len11):  if(mp[list1[j]]==0):  flag = 0  break  if(flag):  mp[value1[i]]=1  vis[i]=1  v=1  if(v==0):  break  len1=len(key2)  for i in range(0, len1):  list1 = key2[i]  len11 = len(list1)  flag = 1  for j in range(0, len11):  if (mp[list1[j]] == 0):  flag = 0  break  if (flag):  return value2[i]  s = '无法识别'  return s  def solve(key1,key2,value1,value2,list):  while(1):  flag = int(input("请输入接下来操作(1.添加间接数据库 2.添加直接数据库 3.查询动物 0.结束操作）："))  if(flag==1):  str = input("请输入条件，以空格隔开：")  a = str.split()  b = input("请输入结果：")  l = len(a)  for i in range(0, l):  list.append(a[i])  key1.append(a)  value1.append(b)  elif(flag==2):  str = input("请输入条件，以空格隔开：")  a = str.split()  b = input("请输入结果：")  l = len(a)  for i in range(0, l):  list.append(a[i])  key2.append(a)  value2.append(b)  elif(flag==3):  str = check(key1,key2,value1,value2,list)  print("该动物：",str)  elif(flag==0):  break  def main():  list = []  key1 = []  key2 = []  value1 = []  value2 = []  init(key1,key2,value1,value2,list)  solve(key1,key2,value1,value2,list)  main() |
| **实验结果：** |
| **实验分析：**  对于产生式系统实验程序，首先设置初始事实放在综合数据库中，然后建立规则库，构建间接数据库，首先根据一些比较简单的条件，对动物进行比较粗的分类，使得当事实不完全时也可以得到一个大致的分类结果，另一方面，也方便了增加物种的操作，通过只增加关于这个动物个性方面的知识，就可以添加到新的规则中。  程序通过一个二维列表来存储数据库的对应关系以及简介数据库之间的关系，同时利用python中的字典结构，建立了直接数据库与间接数据库的联系，利用字典键值对的关系即可一键查询结果 |
| **实验成绩：**  日期： 年 月 日 |
| **河南理工大学教学上机实验报告** |
| 上机时间 2023年 3 月 23日 |
| **实验题目：**  过河问题 |
| **实验目的和要求：**  **实验目的**  1.理解并掌握深度优先搜索算法  2.理解递归的思想  **实验内容**  　　有 N 个传教士和 N 个野人来到河边渡河，河岸有一条船，每次至多可供 k 人乘渡。问：传教士为了安全起见，应如何规划摆渡方案，使得任何时刻， 河两岸以及船上的野人数目总是不超过传教士的数目(否则不安全，传教士有可能被野人吃掉)。 即求解传教士和野人从左岸全部摆渡到右岸的过程中，任何时刻满足 M (传教土数) ≥ C 野人数)和 M+C≤k 的摆渡方案。 |
| **实验过程：**  假设以传教士和野人的数量N都为3，船一次最大的载人量K为2人。  初始状态：河左岸有3个野人、3个传教士；河右岸有0个野人和0个传教士；船停在左岸，船上有0个人。初始状态（3，3，1）  目标状态：河左岸有0个野人和0个传教士；河右岸有3个野人和3个传教士；船停在右岸，船上有0个人。目标状态（0，0，0）  将整个问题抽象成怎样从初始状态经一系列的中间状态从而达到目标状态，状态的改变是通过划船渡河来引发的。  根据要求，共得出以下5中可能的渡河方案：  　　（1）渡2传教士  　　（2）渡2野人  　　（3）渡1野人1传教士  　　（4）渡1传教士  　　（5）渡1野人  本程序使用类来定义状态结点，使用集合存储状态结点，使用递归（深度优先查询）的思想来寻找目标状态。  **传教士和食人者问题的全部可能状态**   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 状 态 | m, c, b | 状 态 | m, c, b | 状 态 | m, c, b | 状 态 | m, c, b | | S0 | 3,3,1 | S8 | 1,3,1 | S16 | 3,3,0 | S24 | 1,3,0 | | S1 | 3,2,1 | S9 | 1,2,1 | S17 | 3,2,0 | S25 | 1,2,0 | | S2 | 3,1,1 | S10 | 1,1,1 | S18 | 3,1,0 | S26 | 1,1,0 | | S3 | 3,0,1 | S11 | 1,0,1 | S19 | 3,0,0 | S27 | 1,0,0 | | S4 | 2,3,1 | S12 | 0,3,1 | S20 | 2,3,0 | S28 | 0,3,0 | | S5 | 2,2,1 | S13 | 0,2,1 | S21 | 2,2,0 | S29 | 0,2,0 | | S6 | 2,1,1 | S14 | 0,1,1 | S22 | 2,1,0 | S30 | 0,1,0 | | S7 | 2,0,1 | S15 | 0,0,1 | S23 | 2,0,0 | S31 | 0,0,0 |   划去不合法的状态以加快搜索求解的效率，划去后仅有16个合理状态。  import operator  \_\_metaclass\_\_ = type    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："))  # 每船乘坐人数  child = [] # child用来存所有的拓展节点  open\_list = [] # open表  closed\_list = [] # closed表      class State:  def \_\_init\_\_(self, m, c, b):  self.m = m #左岸传教士数量  self.c = c #左岸野人数量  self.b = b # b = 1: 船在左岸；b = 0: 船在右岸  self.g = 0  self.f = 0 #f = g+h  self.father = None  self.node = [m, c, b]    init = State(M, C, 1) # 初始节点  goal = State(0, 0, 0) # 目标    #0 ≤ m ≤ 3,0 ≤ c ≤ 3, b ∈ {0,1}, 左岸m > c(m 不为 0 时), 右岸3-m > 3-c(m 不为 3 时)  def safe(s):  if s.m > M or s.m < 0 or s.c > C or s.c < 0 or (s.m != 0 and s.m < s.c) or (s.m != M and M - s.m < C - s.c):  return False  else:  return True      # 启发函数  def h(s):  return s.m + s.c - K \* s.b  # return M - s.m + C - s.c    def equal(a, b):  if a.node == b.node:  return 1,b  else:  return 0,b    # 判断当前状态与父状态是否一致  def back(new, s):  if s.father is None:  return False  #判断当前状态与祖先状态是否一致  c=b=s.father  while(1):  a,c=equal(new, b)  if a:  return True  b=c.father  if b is None:  return False  # 将open\_list以f值进行排序  def open\_sort(l):  the\_key = operator.attrgetter('f') # 指定属性排序的key  l.sort(key=the\_key)      # 扩展节点时在open表和closed表中找原来是否存在相同mcb属性的节点  def in\_list(new, l):  for item in l:  if new.node == item.node:  return True, item  return False, None      def A\_star(s):  A=[]  global open\_list, closed\_list  open\_list = [s]  closed\_list = []  #print(len(open\_list))  # print （'closed list:'） # 选择打印open表或closed表变化过程  #print(s.node)  #a=1  while(1): # open表非空  #get = open\_list[0] # 取出open表第一个元素get  for i in open\_list:  if i.node == goal.node: # 判断是否为目标节点  A.append(i)  open\_list.remove(i)  if not(open\_list):  break  get=open\_list[0]  open\_list.remove(get) # 将get从open表移出  closed\_list.append(get) # 将get加入closed表    # 以下得到一个get的新子节点new并考虑是否放入openlist  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 get.b == 1: # 当前船在左岸，下一状态统计船在右岸的情况  new = State(get.m - i, get.c - j, 0)  child.append(new)  #print(1)  else: # 当前船在右岸，下一状态统计船在左岸的情况  new = State(get.m + i, get.c + j, 1)  child.append(new)  #print(2)  #优先级：not>and>ture。如果状态不安全或者要拓展的节点与当前节点的父节点状态一致。  if not safe(new) or back(new, get): # 状态非法或new折返了  child.pop()  #如果要拓展的节点满足以上情况，将它的父亲设为当前节点，计算f，并对open\_list排序  else:  new.father = get  new.g = get.g + 1 #与起点的距离  new.f = get.g + h(get) # f = g + h  open\_list.append(new)  #print(len(open\_list))  open\_sort(open\_list)  # 打印open表或closed表  #for o in open\_list:  # for o in closed\_list:  #print(o)  #print(o.node)  # print(o.father)  #print(a)  return(A)    # 递归打印路径  def printPath(f):  if f is None:  return  printPath(f.father)  #注意print()语句放在递归调用前和递归调用后的区别。放在后实现了倒叙输出  print(f.node )      if \_\_name\_\_ == '\_\_main\_\_':  print ('有%d个传教士，%d个野人，船容量:%d' % (M, C, K))  final = A\_star(init)  print("有{}种方案".format(len(final)))  if final:  for i in(final):  print ('有解，解为：')  printPath(i)  else:  print ('无解！') |
| **实验结果：** |
| **实验分析：**  函数初始状态将包含有状态信息的结构体节点传入对应的处理函数中，然后判断该传入节点状态是否为要求的目标状态，如果是则遍历这个结构体数组，反之则继续判断该传入状态的人数是否合理，如果不合理则返回递归调用处，如果合理，则继续判断传教士和野人的限制条件，如果不合理则顺序执行之后代码，反之则进行渡河方案的变换。 |
| **实验成绩：**  日期： 年 月 日 |
| **河南理工大学教学上机实验报告** |
| 上机时间 2023年 3月 30日 |
| **实验题目：**  **A\*算法** |
| **实验目的和要求：**  **实验目的：**  **1.熟悉启发式搜索的定义、估价函数和算法过程**  **2.利用A\*算法求解N数码难题，理解求解流程和搜索顺序**  **3.熟练掌握numpy库的相关函数** |
| **实验过程：**  原理：  A\*算法是一种启发式图搜索算法，其特点在于对估价函数的定义上。对于一般的启发式图搜索，总是选择估价函数f值最小的节点作为扩展节点。因此，f是根据需要找到一条最小代价路径的观点来估算节点的，所以，可考虑每个节点n的估价函数值为两个分量：从起始节点到节点n的实际代价以及从节点n到达目标节点的估价代价。  八数码问题的求解 八数码问题是在3×3的九宫格棋盘上，摆有8个刻有1～8数码的将牌。棋盘中有一个空格，允许紧邻空格的某一将牌可以移到空格中，这样通过平移将牌可以将某一将牌布局变换为另一布局。针对给定的一种初始布局或结构（目标状态），问如何移动将牌，实现从初始状态到目标状态的转变。  以8数码问题为例实现A\*算法的求解程序。  估价函数f(n) = g(n) + h(n)  g(n)=d(n)——结点n在搜索树中的深度  h(n)可选择h1(n)——结点n中“不在位”的数码个数 或 h2(n) =p(n)= 将牌“不在位”的距离和（哈密尔顿距离或曼哈顿距离）  **1 2 3**  **8 4**  **7 6 5**  **2 8 3**  **1 6 4 7 5**   1. **7 6 5**   **2 8 3**  **1 6 4 7 5**   1. **7 6 5**   import numpy as np  import operator  from time import time  import os  direction = [(-1, 0), (1, 0), (0, -1), (0, 1)]  steps = [] #保存路径  openlist = [] # open表  closed = [] # closed表  class State:  def \_\_init\_\_(self, m):  self.node = m  self.f = 0  self.g = 0  self.h = 0  self.father = None  # 找到列表中对应元素的下标  def isContained(num\_list, num):  for i in range(len(num\_list)):  if (num.node == num\_list[i].node).all():  return i  return -1  # 逆序数：所有排列队列在当前位置之前比当前位置上的元素大的计数总和("0"不参与计数)，用于判断奇数列，还是偶数列  def getStatus(numList):  res = 0  for i in range(1, len(numList)):  if numList[i] == 0:  continue  for j in range(i):  if numList[j] > numList[i]:  res += 1  return res  # 判断是否存在解  def existSolution(a\_list, b\_list):  a\_f = getStatus(a\_list)  b\_f = getStatus(b\_list)  print(f"状态1逆序数:{a\_f}", f"状态2逆序数:{b\_f}")  if (a\_f % 2 == b\_f % 2): ## 逆序数相同，有解  return True  return False  # 启发函数  def h(s):  a = 0  for i in range(len(s.node)):  for j in range(len(s.node[i])):  if s.node[i][j] != goal.node[i][j]:  a = a + 1  return a  # 对节点列表按照估价函数的值的规则排序  def list\_sort(l):  cmp = operator.attrgetter('f')  l.sort(key=cmp)  counter = 0  counter\_all = 1  # A\*算法  def A\_star(s):  global openlist, closed ,counter ,counter\_all  openlist = [s]  while (openlist):  get = openlist.pop(0)  counter += 1  closed.append(get)  if (get.node == goal.node).all():  return get  # 判断此时状态的空格位置(a,b)  for a in range(len(get.node)):  for b in range(len(get.node[a])):  if get.node[a][b] == 0:  break  if get.node[a][b] == 0:  break  row, col = len(get.node), len(get.node[0])  for dir in direction:  i = a + dir[0]  j = b + dir[1]  if 0 <= i < row and 0 <= j < col:  counter\_all += 1  c = get.node.copy()  c[a][b] = c[i][j]  c[i][j] = 0  if (get.father !=None and (c == get.father.node).all()):  continue  new = State(c)  new.father = get  new.g = get.g + 1  if (isContained(closed, new) == -1):  position = isContained(openlist, new)  if position != -1:  if new.g < openlist[position].g:  openlist.pop(position)  new.h = h(new)  new.f = new.g + new.h  openlist.append(new)  else:  new.h = h(new)  new.f = new.g + new.h  openlist.append(new)  list\_sort(openlist)  # 递归打印路径  def getpath(f):  global steps  if f is None:  return  getpath(f.father)  steps.append(f.node)  O, A, B = None, None, None  with open("input.txt", "a", encoding='utf-8') as f:  f.close()  with open("input.txt", mode="r", encoding='utf-8') as f:  O = int(f.readline().strip())  A = [int(i) for i in f.readline().strip().split(",")]  B = [int(i) for i in f.readline().strip().split(",")]  print(f"矩阵的规模:{O}",f"初始状态{A}", f"目标状态{B}")  if existSolution(A, B):  z = 0  M = np.zeros((O, O))  N = np.zeros((O, O))  for i in range(O):  for j in range(O):  M[i][j] = A[z]  N[i][j] = B[z]  z = z + 1  init = State(M)  goal = State(N)  time1 = time()  final = A\_star(init)  if final:  print("有解，解为：")  getpath(final)  for i, item in enumerate(steps):  print(f"\*\*\*\*\*第{i}步\*\*\*\*\*")  print(item)  print("总共考察了%d个节点"%counter)  print(f"该树已生成的节点数为{counter\_all}")  print(f"总用时为{time()-time1}s")  else:  print("无解")  else:  print("该八数码问题没有解，请重新选择！") |
| **实验结果：** |
| **实验分析：**  首先对程序输入初始状态与目标状态，并根据二者的逆序数是否相等，判断八数码问题是否有解，若有则初始open表和close表，开始核心算法的循环执行。若子状态在close表中，说明该状态我们已经考察过了，跳过该子状态。若子状态不在close表中，我们根据子状态是否在open中，进行不同的操作 |
| **实验成绩：**  日期： 年 月 日 |
| **河南理工大学教学上机实验报告** |
| 上机时间 2023年 3 月 30日 |
| **实验题目：**  **遗传算法应用** |
| **实验目的和要求：**  **实验目的：**  1.熟悉和掌握遗传算法的原理、流程和编码策略  2.理解求解TSP问题的流程并测试主要参数对结果的影响  **实验原理：**  旅行商问题，即TSP问题（Traveling Salesman Problem）是数学领域中著名问题之一。假设有一个旅行商人要拜访n个城市，他必须选择所要走的路径，路经的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。TSP问题是一个组合优化问题。该问题可以被证明具有NPC计算复杂性。因此，任何能使该问题的求解得以简化的方法，都将受到高度的评价和关注。  遗传算法的基本思想正是基于模仿生物界遗传学的遗传过程。它把问题的参数用基因代表，把问题的解用染色体代表（在计算机里用二进制码表示），从而得到一个由具有不同染色体的个体组成的群体。这个群体在问题特定的环境里生存竞争，适者有最好的机会生存和产生后代。后代随机化地继承了父代的最好特征，并也在生存环境的控制支配下继续这一过程。群体的染色体都将逐渐适应环境，不断进化，最后收敛到一族最适应环境的类似个体，即得到问题最优的解。要求利用遗传算法求解TSP问题的最短路径。 |
| **实验过程：**  实验原理：  旅行商问题，即TSP问题（Traveling Salesman Problem）是数学领域中著名问题之一。假设有一个旅行商人要拜访n个城市，他必须选择所要走的路径，路经的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。TSP问题是一个组合优化问题。该问题可以被证明具有NPC计算复杂性。因此，任何能使该问题的求解得以简化的方法，都将受到高度的评价和关注。  遗传算法的基本思想正是基于模仿生物界遗传学的遗传过程。它把问题的参数用基因代表，把问题的解用染色体代表（在计算机里用二进制码表示），从而得到一个由具有不同染色体的个体组成的群体。这个群体在问题特定的环境里生存竞争，适者有最好的机会生存和产生后代。后代随机化地继承了父代的最好特征，并也在生存环境的控制支配下继续这一过程。群体的染色体都将逐渐适应环境，不断进化，最后收敛到一族最适应环境的类似个体，即得到问题最优的解。要求利用遗传算法求解TSP问题的最短路径。  import random  import math  x = [0, 1.1, 3.5, 3, 7, 8, 4, 4.5, 9, 2]  y = [1.1, 3, 2, 4, 5.1, 8, 4, 4.5, 9, 2]  path = []  ans = 1000000000000  def eu(a,b):  dis = math.sqrt((x[a] - x[b]) \* (x[a] - x[b]) + (y[a] - y[b]) \* (y[a] - y[b]))  return dis  def distance(l, n):  global ans, path  dis = 0  for i in range(0, n-1):  dis += eu(l[i], l[i+1])  dis += eu(l[n-1], l[0])  if(dis < ans):  path = l  ans = dis  return dis  def init(n, size, popluation):  l1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  for i in range(0, size):  popluation.append([])  random.shuffle(l1)  for j in range(0, n):  popluation[i].append(l1[j])  def fit(fitness, population, n, size):  for i in range(0, size):  fitness.append(distance(population[i], n))  def choose\_jinbiaosai(fitness, population, new\_pop, size):  for i in range (0, size):  j = 0  maxn = 0  for k in range(0, int(size / 4)):  a = random.randint(0, size - 1)  if(fitness[a] > maxn):  maxn = fitness[a]  j = a  new\_pop.append(population[j])  def choose\_lunpan(population, new\_pop, fitness, size):  F = 0  roulette = []  p = 0  for i in range(0, size):  F=F+fitness[i]  for i in range(0, size):  p = p + fitness[i] / F  roulette.append(p)  for i in range(0, size):  p = random.uniform(0, 1)  for j in range(0, size):  if(roulette[j] >= p):  new\_pop.append(population[j])  def inheritance\_solve(population, size, inheritance, n):  for i in range(0,int(size / 2)):  p = random.uniform(0,1)  if(p < inheritance):  u = random.randint(0, n-1)  v = random.randint(0, n-1)  if(u == v):  continue  if(u > v):  a = u  u =v  v = a  mp1 ={}  mp2 ={}  for k in range(0 , n):  mp1[k] = -1  mp2[k] = -1  l1 = []  l2 = []  for j in range(u, v + 1):  mp2[population[2 \* i][j]] = population[2 \* i + 1][j]  mp1[population[2 \* i + 1][j]] = population[2 \* i][j]  for j in range(0, u):  if(mp1[population[2 \* i][j]] == -1):  l1.append(population[2 \* i][j])  else:  a = mp1[population[2 \* i][j]]  while(mp1[a] != -1):  a = mp1[a]  l1.append(a)  if (mp2[population[2 \* i + 1][j]] == -1):  l2.append(population[2 \* i + 1][j])  else:  a = mp2[population[2 \* i + 1][j]]  while(mp2[a] != -1):  a = mp2[a]  l2.append(a)  for j in range(u, v + 1):  l2.append(population[2 \* i][j])  l1.append(population[2 \* i + 1][j])  for j in range(v + 1, n):  if (mp1[population[2 \* i][j]] == -1):  l1.append(population[2 \* i][j])  else:  a = mp1[population[2 \* i][j]]  while (mp1[a] != -1):  a = mp1[a]  l1.append(a)  if (mp2[population[2 \* i + 1][j]] == -1):  l2.append(population[2 \* i + 1][j])  else:  a = mp2[population[2 \* i + 1][j]]  while (mp2[a] != -1):  a = mp2[a]  l2.append(a)  population[i] = l1  population[i + 1] = l2  def variation\_solve(population, size, variation, n):  for i in range(0, size):  p = random.uniform(0, 1)  if(p < variation):  u = random.randint(0, n - 1)  v = random.randint(0, n - 1)  if (u == v):  continue  else:  a = population[i][u]  population[i][u] = population[i][v]  population[i][v] = a  def main():  cnt = 100  n = 10  size = 100  inheritance = 0.85  variation = 0.15  population = []  print("城市坐标：")  for i in range(0, n):  print("[",x[i],",",y[i],"]")  init(n, size, population)  while(cnt):  cnt = cnt - 1  new\_pop = []  fitness = []  fit(fitness, population, n, size)  choose\_lunpan(population, new\_pop, fitness, size)  population = new\_pop  inheritance\_solve(population, size, inheritance, n)  variation\_solve(population, size, variation, n)  print("最好适应度：",ans)  print("最好的解：",end = ":" )  for i in range(0,n):  print(path[i],end = " ")  print()  if \_\_name\_\_ == '\_\_main\_\_':  main() |
| **实验结果：** |
| **实验分析：**  不同城市序列下的TSP算法所得到的结果不同，他们的收敛速度也不同，然而总体运行时间相似，其他条件相同，城市的数量不同  对于更多的城市数量，在大概2k次的迭代次数中达不到收敛，故应适当增加迭代次数，而随着城市数量的增加，程序的运行时间也相应更长一些  其他条件相同，群体规模不同的时候，增加种群数量，总体程序运行时间会增加，算法会收敛的更快，但搜索出来的最短路径却更长，搜索性能下降  其他条件相同，交叉概率不同时，减小交叉概率，算法收敛的速度会大大降低，同时搜索出来的路径也会更长，故较小的交叉概率搜索能力下降，但程序运行时间要更短一些  其他条件相同，变异概率不同时，增大变异概率，搜索出来的最短路劲变小，程序总体运行时间更快一点，但算法收敛速度较慢。  根据该算法的自身因素，城市规模越大迭代数越大，搜索最优解时间越长，效率越低，所以可以对该原始群体进行合理初始化，在提高初始群体质量的同时，不影响到群体种类的多样性，以此来优化算法效率和性能。 |
| **实验成绩：**  日期： 年 月 日 |