人工智能第一次实验报告

编程语言：python

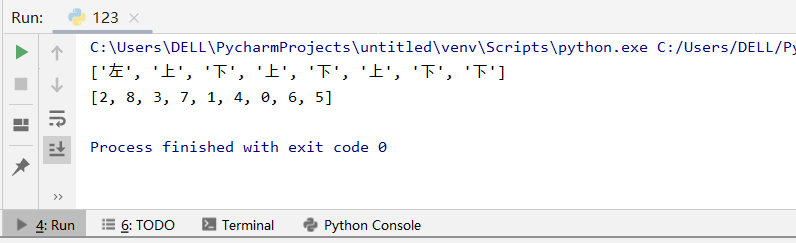
运行环境：pycharm2019.2

1. 八数码算法：

遗传算法

源程序：

**import** math   
**import** random  
  
C = 540 *# 最大适应度*LEN = 140 *# 基因长度*maxGene = []  
maxi = 0 *# 最大值最初出现的进化代数*findAim = **False**POPMAX = 32 *# 种群数量*P\_XOVER = 0.8 *# 交叉概率*P\_MUTATION = 0.15 *# 变异概率*MAXGENERATIONS = 1000 *# 总的进化代数./1*goal = [2, 8, 3, 7, 1, 4, 0, 6, 5] *# 目标状态*initiate = [2, 8, 3, 1, 0, 4, 7, 6, 5] *# 初始状态  
# initiate = [1,0,2,3,4,5,6,7,8]  
# goal=[0,1,2,3,4,5,6,7,8]*pop = [] *# 种群所有对象***class** Gene:  
 **def** \_\_init\_\_(self, gene):  
 self.gene = gene *# 基因，数组* self.fitness = 0  
 self.rf = 0 *# 选择的概率* self.cf = 0 *# 累积的概率  
  
  
# 随机初始化基因组***def** initGenes():  
 count = 0  
 maxFit = 100 *# 随机生成的基因适应度的最大值* **while** (count < POPMAX):  
 tmp = []  
 **for** j **in** range(LEN):  
 pow = round(random.random() \* 3) *# 随机生成0上，1下, 2左, 3右* tmp.append(pow)  
  
 pop.append(Gene(tmp))  
 count += 1  
  
  
*# 上下左右操作***def** move(current, dire):  
 space = 0 *# 空格位置* block = 0 *# 移动的格子的位置* **for** i **in** range(len(current)):  
 **if** (current[i] == 0):  
 space = i  
 block = space  
 **break  
  
 if** (dire == 0):  
 **if** (space - 3 >= 0):  
 block = space - 3  
  
 **elif** (dire == 1 **and** (space + 3 < 9)):  
 block = space + 3  
 **elif** (dire == 2):  
 **if** (space % 3 > 0):  
 block = space - 1  
  
 **elif** (dire == 3):  
 **if** (space % 3 < 2):  
 block = space + 1  
  
 current[space], current[block] = current[block], current[space]  
 **if** (space == block):  
 **return False  
 else**:  
 **return True***# 计算适应度***def** fitness(current):  
 f = 0  
 **for** i **in** range(len(current)):  
 **if** (current[i] == goal[i]):  
 f += 100 - current[i] \* 10  
 **return** f  
  
  
**def** envaluateFitness(maxi): *# max参数只是用来记录进化代数* totalFitness = 0  
 **for** i **in** range(POPMAX):  
 s0 = initiate[:] *# 每一步移动后的状态* pop[i].fitness = 0  
 **for** j **in** range(LEN):  
 *# 每移动一次后计算一次适应度函数，若为540说明已找到解* move(s0, pop[i].gene[j])  
 pop[i].fitness = fitness(s0)  
 **if** (pop[i].fitness == C):  
 **global** findAim  
 findAim = **True  
 global** maxGene  
 maxGene = pop[i].gene[0:j + 1]  
 **return** totalFitness  
  
 **if** (pop[i].fitness == 0):  
 pop[i].fitness = 1  
  
 totalFitness += pop[i].fitness  
  
 **return** totalFitness  
  
  
*# 适应度更高的基因有更高的概率往下遗传***def** selectBetter(totalFitness): *# 轮盘赌选择* lastCf = 0  
 newPop = [**None for** i **in** range(POPMAX)]  
 **global** pop  
 **for** i **in** range(POPMAX): *# 计算个体选择概率和累积概率* pop[i].rf = pop[i].fitness / totalFitness  
 pop[i].cf = lastCf + pop[i].rf  
 lastCf = pop[i].cf  
  
 **for** i **in** range(POPMAX): *# 轮盘赌式选择* p = random.random()  
 **if** (p < pop[0].cf):  
 newPop[i] = Gene(pop[0].gene)  
 *# newPop.append(Gene(pop[0].gene))* **else**:  
 **for** j **in** range(POPMAX - 1):  
 **if** (p >= pop[j].cf **and** p < pop[j + 1].cf):  
 newPop[i] = Gene(pop[j + 1].gene)  
 *# newPop.append(Gene(pop[j+1].gene))* **break  
  
 if** (len(newPop) == 0):  
 *# console.log(pop)* print(pop)  
 **import** copy  
 pop = copy.deepcopy(newPop)  
  
  
**def** exChgOver(first, second): *# 基因交换函数* ecc = round(random.random() \* LEN)  
 **for** i **in** range(int(ecc)):  
 index = math.floor(random.random() \* LEN)  
 pop[first].gene[index], pop[second].gene[index] = pop[second].gene[index], pop[first].gene[index]  
  
  
*# 交叉***def** crossover():  
 first = -1  
 **for** i **in** range(POPMAX):  
 p = random.random()  
 **if** (p < P\_XOVER):  
 **if** (first < 0):  
 first = i  
 **else**: *# 选择了两个随机个体，进行基因交换* exChgOver(first, i)  
 first = -1  
  
  
**def** reverseGene(index): *# 变异操作函数* mcc = round(random.random() \* LEN)  
 **for** i **in** range(int(mcc)):  
 gi = math.floor(random.random() \* LEN)  
 pop[index].gene[gi] = 3 - pop[index].gene[gi]  
  
  
*# 变异***def** mutation():  
 **for** i **in** range(POPMAX):  
 p = random.random()  
 **if** (p < P\_MUTATION): *# 只有当随机数小于变异概率才进行变异操作* reverseGene(i)  
  
  
initGenes()  
f = envaluateFitness(0)  
**for** i **in** range(MAXGENERATIONS):  
 selectBetter(f)  
 crossover()  
 mutation()  
 f = envaluateFitness(i)  
 **if** (findAim):  
 **break***# console.log(maxGene);***def** transform(gene):  
 s0 = initiate[:]  
 options = []  
 **for** i **in** range(len(gene)):  
 **if** (move(s0, gene[i])):  
 **if** (gene[i] == 0):  
 options.append(**'上'**)  
 **elif** (gene[i] == 1):  
 options.append(**'下'**)  
 **elif** (gene[i] == 2):  
 options.append(**'左'**)  
 **elif** (gene[i] == 3):  
 options.append(**'右'**)  
  
 print(options)  
 print(s0)  
  
  
transform(maxGene)  
  
*# let m = maxGene  
# let s0 = initiate.slice(0)  
# for (let i = 0; i< m.length; i++) {  
# move(s0, m[i])  
# }  
# console.log(s0)*

运行结果：

结果的第一个列表显示初始状态想要移动到目标状态空格该想那个方向走，第二个列表显示最终的目标节点。

具体实现想法：

首先 遗传算法的基本过程：1.初始化 进化代数初始为0，最大为T，随机生成M个个体作为初始群体P（0），2.计算群体p(t)的各个个体的生硬程度。3.选择运算：将选择算子应用于群里，旨在把优化的个体直接遗传到下一代，或通过配对交叉生成新的个体遗传到下一代。

4.交叉运算：将起核心作用的交叉算子作用于群体 5.变异：将变异算子作用于群体6.终止：若t=T 最大适应度的输出为最优解。

个人理解：达尔文学说为适者生存不适者讨厌，适应度高的而进化，这应该就是遗传算法中的选择运算，孟德尔遗传定律说明生物的演变又基因 排列组合而成，对应的遗传算法里的交叉运算和变异运算。也就是说通过适应度函数选择，通过交叉和变异实现优化。所以说遗传算法十一中随机的算法，可能会产生近似最优解，但不一定一定是最优解。这也就是为什么程序 多次运行可能会出现不一样的结果，多次后甚至报错。具体到八数码问题上，编码0 1 2 3分别代表上下左右，选择算法参考网上所给轮盘赌选择法，适应度越高越留下，初代种群是随机产生，所有随机数是通过rand()产生。

2.

实验主题：

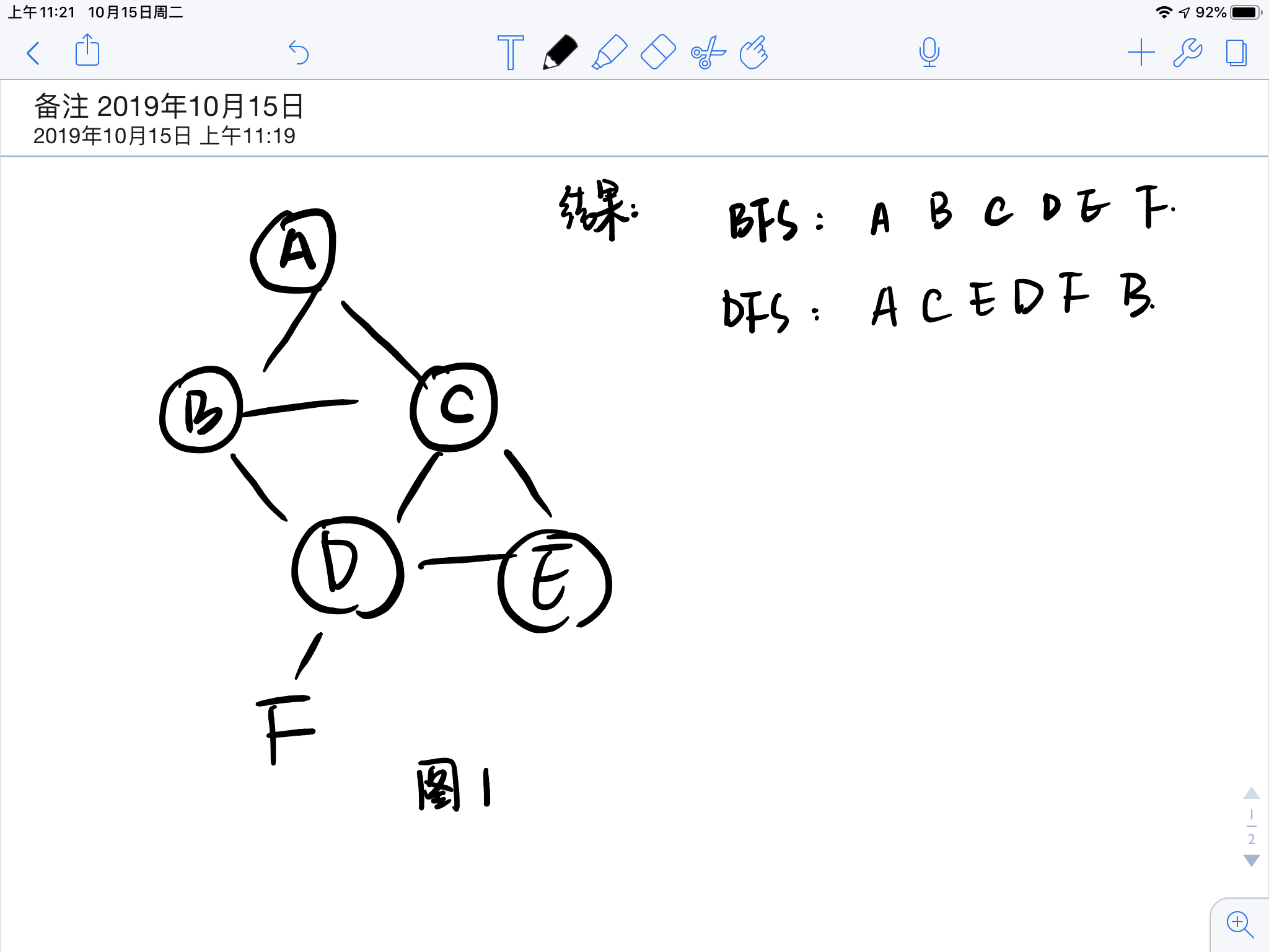
深度优先搜索DFS

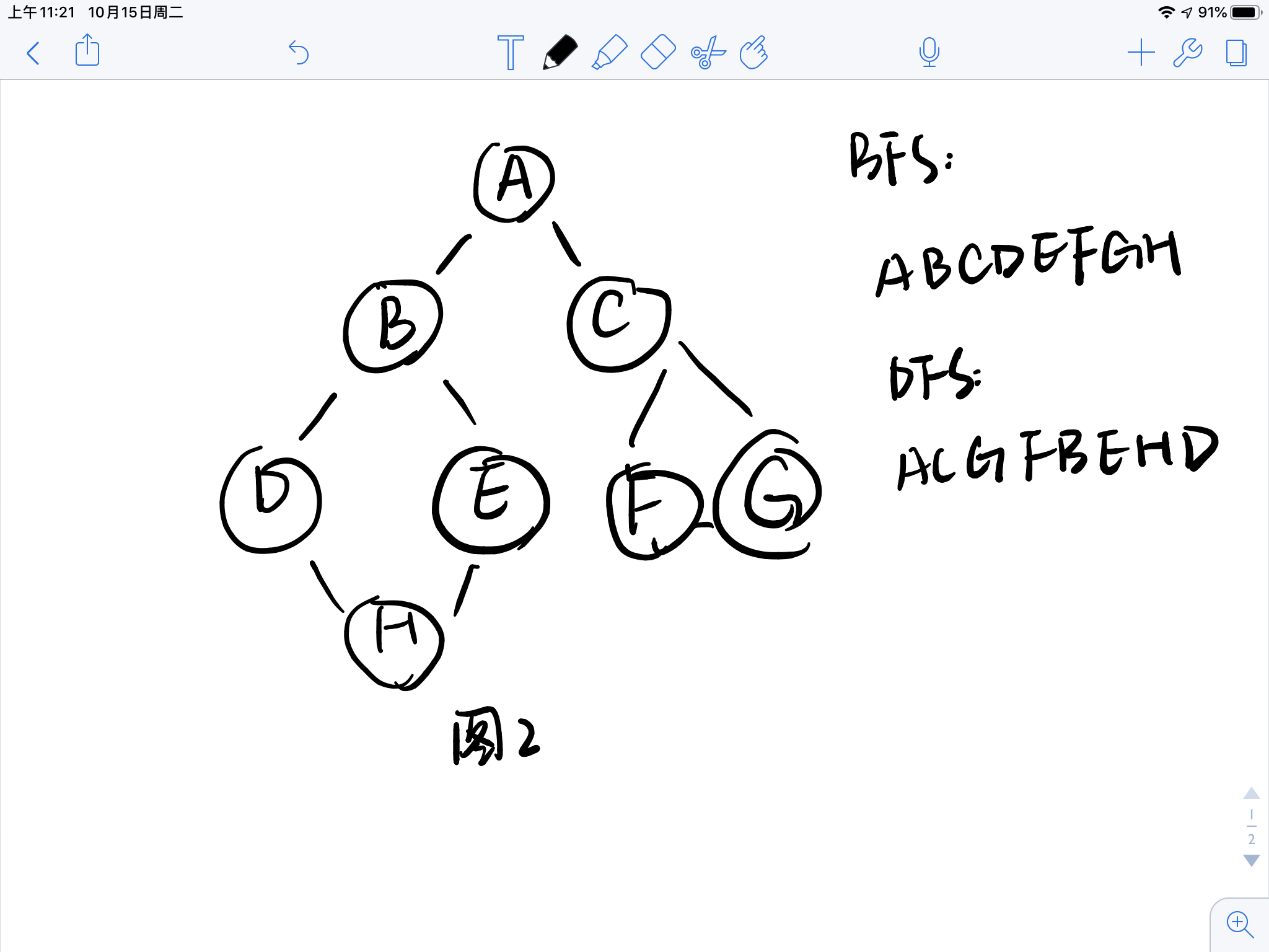
广度优先搜索BFS

图搜索：

两次不同的图：

*# 定义图结构  
'''图1*

 *graph = {  
 "A": ["B","C"],  
 "B": ["A", "C", "D"],  
 "C": ["A", "B", "D","E"],  
 "D": ["B", "C", "E","F"],  
 "E": ["C", "D"],  
 "F": ["D"],  
}  
'''*

graph = {  
 **"A"**: [**"B"**,**"C"**],  
 **"B"**: [**"D"**,**"E"**],  
 **"C"**: [**"F"**,**"G"**],  
 **"D"**: [**"H"**],  
 **"E"**:[**"H"**],  
 **"F"**: [**"G"**],  
 **"G"**:[**"F"**],  
 **"H"**:[**"E"**],  
}  
**def** BFS(graph,vertex):  
 *# 使用列表作为队列* queue = []  
 *# 将首个节点添加到队列中* queue.append(vertex)  
 *# 使用集合来存放已访问过的节点* looked = set()  
 *# 将首个节点添加到集合中表示已访问* looked.add(vertex)  
 *# 当队列不为空时进行遍历* **while**(len(queue)>0):  
 *# 从队列头部取出一个节点并查询该节点的相邻节点* temp = queue.pop(0)  
 nodes = graph[temp]  
 *# 遍历该节点的所有相邻节点* **for** w **in** nodes:  
 *# 判断节点是否存在于已访问集合中,即是否已被访问过* **if** w **not in** looked:  
 *# 若未被访问,则添加到队列中,同时添加到已访问集合中,表示已被访问* queue.append(w)  
 looked.add(w)  
 print(temp,end=**' '**)  
  
**def** DFS(graph,vertex):  
 *# 使用列表作为栈* stack = []  
 *# 将首个元素添加到队列中* stack.append(vertex)  
 *# 使用集合来存放已访问过的节点* looked = set()  
 *# 将首个节点添加到集合中表示已访问* looked.add(vertex)  
 *# 当队列不为空时进行遍历* **while** len(stack)>0:  
 *# 从栈尾取出一个节点并查询该节点的相邻节点* temp = stack.pop()  
 nodes = graph[temp]  
 *# 遍历该节点的所有相邻节点* **for** w **in** nodes:  
 *# 判断节点是否存在于已访问集合中,即是否已被访问过* **if** w **not in** looked:  
 *# 若未被访问,则添加到栈中,同时添加到已访问集合中,表示已被访问* stack.append(w)  
 looked.add(w)  
 print(temp,end=**' '**)  
*# 由于无向图无根节点，则需要手动传入首个节点，此处以"A"为例*print(**"BFS"**,end=**" "**)  
BFS(graph,**"A"**)  
print(**""**)  
print(**"DFS"**,end=**" "**)  
DFS(graph,**"A"**)