*# Md. Moynul Asik Moni  
# ID:19101189,lab assignment2(genetic algo,n queen)  
# CSE422,section: 05***import** numpy **as** np  
**import** random  
  
  
**def** fitness\_all(population, n):  
 *'''calculates the fitness score of each  
 of the individuals in the population  
  
 returns a 1D numpy array: index referring to  
 ith individual in population, and value referring  
 to the fitness score.'''* fittness\_fn = []  
 fittness\_fn\_final =[]  
 maxFitness = (n \* (n - 1)) / 2  
 population\_count =0  
 **while** (population\_count<len(population)):  
  
 child =[]  
 **for** q **in** range(0,len(population[population\_count]),1):  
 child.append(population[population\_count][q])  
 horizontal\_col = 0  
 diagonal\_col = 0  
  
 **for** i **in** range(0, n, 1):  
 counter = 0  
 **for** j **in** range(0, n, 1):  
 **if** child[j] == i:  
 counter = counter + 1  
 **if** counter == 2:  
 horizontal\_col = horizontal\_col + 1  
 **elif** counter > 2:  
 horizontal\_col = horizontal\_col +(counter \*(counter - 1))/2  
  
 temp\_list = [[-1 **for** i **in** range(n)] **for** j **in** range(n)]  
  
 counter\_child = 0;  
  
 **for** i **in** range(0, len(child), 1):  
 j = child[i]  
 temp\_list[j][i] = **'q'** a = np.array(temp\_list)  
 diags = [a[::-1, :].diagonal(i) **for** i **in** range(-a.shape[0] + 1, a.shape[1])]  
 diags.extend(a.diagonal(i) **for** i **in** range(a.shape[1] - 1, -a.shape[0], -1))  
 b = [n.tolist() **for** n **in** diags]  
  
  
  
 **for** i **in** range(0, len(b), 1):  
 counter = 0  
 **for** j **in** range(0, len(b[i]), 1):  
 **if** b[i][j] == **'q'**:  
 counter = counter + 1  
 **if** counter == 2:  
 diagonal\_col = diagonal\_col + 1  
 **elif** counter > 2:  
 diagonal\_col = diagonal\_col + (counter \* (counter - 1)) / 2  
  
 fittness\_fn.append(maxFitness-(diagonal\_col+horizontal\_col))  
 population\_count = population\_count + 1  
  
 summm = 0  
 **for** i **in** range(0,len(fittness\_fn),1):  
 summm+=fittness\_fn[i]  
 **for** i **in** range(0, len(fittness\_fn), 1):  
 fittness\_fn\_final.append(fittness\_fn[i]/summm)  
  
 **return** fittness\_fn\_final  
  
  
  
  
**def** fitness(child, n):  
 *'''calculates the fitness score of each  
 of the individuals in the population  
  
 returns a 1D numpy array: index referring to  
 ith individual in population, and value referring  
 to the fitness score.'''* maxFitness = (n \* (n - 1)) / 2  
 horizontal\_col = 0  
 diagonal\_col = 0  
  
  
  
 **for** i **in** range(0,n,1):  
 counter = 0  
 **for** j **in** range(0,n,1):  
 **if** child[j] == i:  
 counter = counter+1  
 **if** counter == 2:  
 horizontal\_col = horizontal\_col+1  
 **elif** counter>2:  
 horizontal\_col = horizontal\_col+(counter\*(counter-1))/2  
  
  
 temp\_list = [[-1 **for** i **in** range(n)] **for** j **in** range(n)]  
  
 counter\_child = 0;  
  
 **for** i **in** range(0,len(child),1):  
 j = child[i]  
 temp\_list [j][i] = **'q'** a = np.array(temp\_list)  
 diags = [a[::-1, :].diagonal(i) **for** i **in** range(-a.shape[0] + 1, a.shape[1])]  
 diags.extend(a.diagonal(i) **for** i **in** range(a.shape[1] - 1, -a.shape[0], -1))  
 b = [n.tolist() **for** n **in** diags]  
  
  
  
  
 **for** i **in** range(0,len(b),1):  
 counter = 0  
 **for** j **in** range(0,len(b[i]),1):  
 **if** b[i][j] == **'q'**:  
 counter = counter+1  
 **if** counter == 2:  
 diagonal\_col = diagonal\_col+1  
 **elif** counter>2:  
 diagonal\_col = diagonal\_col+(counter\*(counter-1))/2  
  
  
 **return** int(maxFitness-(diagonal\_col+horizontal\_col))  
  
  
  
  
**def** select(population, fit):  
 *''' take input: population and fit  
 fit contains fitness values of each of the individuals  
 in the population  
  
 return: one individual randomly giving  
 more weight to ones which have high fitness score'''* x = np.random.choice(len(population), 1, **True**, fit)  
  
 **return** population[x[0]]  
  
  
**def** crossover(x, y):  
 *'''take input: 2 parents - x, y.  
 Generate a random crossover point.  
 Append first half of x with second  
 half of y to create the child  
  
 returns: a child chromosome'''* n = len(x)  
 i = random.randint(0,n-1)  
 z = []  
 **for** k **in** range(0,i,1):  
 z.append(x[k])  
 **for** k **in** range(0,n-i,1):  
 z.append(y[k])  
  
 **return** z  
  
  
**def** mutate(child):  
 *'''take input: a child  
 mutates a random  
 gene into another random gene  
  
 returns: mutated child'''* n = len(child)  
 c = random.randint(0, n - 1)  
 m = random.randint(0, n - 1)  
 child[c] = m  
  
  
 **return** child  
  
  
**def** GA(population, n, mutation\_threshold):  
 *'''implement the pseudocode here by  
 calling the necessary functions- Fitness,  
 Selection, Crossover and Mutation  
  
 print: the max fitness value and the  
 chromosome that generated it which is ultimately  
 the solution board'''* mAXXXFIT = -1;  
  
 last\_fit = 0  
 last\_child =[]  
 maxFit = ((n \* (n - 1)) / 2)  
 imax = 100000  
  
 print(**"The dimension given for the test is:"**,n,**"X"**,n)  
 print(**"The population size given is:"**,len(population))  
 print(**"Maximum fitness for the given population is:"**,int(maxFit))  
 print(**"Maximum generations (maximum iteration):"**,imax)  
 print(**"Execution start>>>>>>>>>>>>>>>>>>>>>>"**)  
  
  
 counter = imax  
 **while** counter > 0:  
 new\_population = []  
 fitness\_fn = fitness\_all(population, n)  
 **for** i **in** range(len(population)):  
  
 x = select(population,fitness\_fn)  
 y = select(population,fitness\_fn)  
 child = crossover(x,y)  
  
 **if** random.uniform(0,1) < mutation\_threshold:  
 child = mutate(child)  
  
 last\_fit = fitness(child,n)  
 **if** mAXXXFIT == -1:  
 mAXXXFIT = last\_fit  
 last\_child = child  
 **if** last\_fit > mAXXXFIT:  
 mAXXXFIT = last\_fit  
 last\_child = child  
 **if** last\_fit == int(maxFit):  
 print(**"Got it...."**)  
 print(**"Desired Result (child):"**,child,**"after"**,imax-counter,**"generations."**)  
 **return** child  
 new\_population.append(child)  
 population = new\_population  
 counter-=1  
 print(**"Execution finished........."**)  
 print(**"No solution found after "**,imax,**" generations."**)  
 print(**"Nearest Result:"**,last\_child,**"where maximum fitness is"**,mAXXXFIT,**"out of"**,int(maxFit))  
 **return** last\_child  
  
  
  
**'''for 8 queen problem, n = 8'''**n = 8  
  
**'''start\_population denotes how many individuals/chromosomes are there  
 in the initial population n = 8'''**start\_population = 10  
  
**'''if you want you can set mutation\_threshold to a higher value,  
 to increase the chances of mutation'''**mutation\_threshold = 0.9  
  
**'''creating the population with random integers between 0 to 7 inclusive  
 for n = 8 queen problem'''**population = np.random.randint(0, n, (start\_population, n))  
  
**'''calling the GA function'''**GA(population, n, mutation\_threshold)