Shirin Mohebbi

Std number: 9935597

HW #1

For solving 8 queens problem, i used Genetic algorithm, with below specification:

representation: permutation of 8 numbers

data structure: list

each list element show queens position.

element value: queens row position - element index: queens column position

search space: 8!

fitness: inverse sum of each queens of phenotype penalty

one queen penalty: number of queens she can check.

```
def fitness(pheno):
    fitnessInvers = 0
    for i in range(len(pheno)):
        for j in range(len(pheno)):
            if (abs(pheno[i] - pheno[j]) == abs(i - j) and i != j):
                fitnessInvers += 1
    return fitnessInvers
```

population: 100 fixed phenotype

initialization: 100 random permutation of 1-8

```
def initializePopulation(numOfqueens, numPopultion):
    population = []
    for i in range(numPopultion):
        population.append(random.sample(range(1, numOfqueens+1), numOfqueens))
    return population
```

mutation: swapping values of two randomly chosen positions

mutation probability 80%

```
def mutation(child):
    i = random.randint(0, 7)
    j = random.randint(0, 7)
    child[i], child[j] = child[j], child[i]
```

cross over: for each 2 parent, after permutation we'll have 2 child

select random crossover point, copy first parts of each parent to children, and for second parts of each child insert values from opposite parent.

```
def crossOver(parents):
  crossOverPoint = random.randint(1, 7)
  child = []
  for i in range (0,2):
    child.append( parents[i][:crossOverPoint] )
    oppositParentIndex = 1 - i
    for j in range(crossOverPoint, 8):
      if (parents[oppositParentIndex][j] not in child[i]):
        child[i].append(parents[oppositParentIndex][j])

    child[i] + parents[oppositParentIndex][:crossOverPoint]
    for k in range(crossOverPoint):
      el = parents[oppositParentIndex][:crossOverPoint][k]
      if (el not in child[i]):
        child[i].append(el)
  return (child)
```

parent selection: select 5 parents randomly, choose the two fittest to do cross over

```
def parentSelection(population):
  random5parents = []
  indexList = list(range(len(population)))
  for i in range(5):
    selectedIndex = random.choice(indexList)
    random5parents.append(population[selectedIndex])
    indexList.remove(selectedIndex)

  parentFitenss = []
  for i in range(5):
    parentFitenss.append((random5parents[i], fitness(random5parents[i])))
  parentFitenss.sort(key=lambda tup: tup[1])
  return (parentFitenss[0][0], parentFitenss[1][0])
```

survivor selection: sort whole population in order by increasing inverse fitness, select the top 100 for new generation

```
def survivorSelection(newPopulation, numPopultion):
  phenoFitness = []
  for i in range(len(newPopulation)):
    phenoFitness.append((newPopulation[i], fitness(newPopulation[i])))
  phenoFitness.sort(key=lambda tup: tup[1])
  populationSurvivor = []
  for item in phenoFitness[:numPopultion]:
    populationSurvivor.append(item[0])
  return populationSurvivor
```

termination condition: if reverse fitness of a phenotype is equal to 0(which is solution phenotype) or if we pass 1000 iteration(which we couldn't find any solution)

```
if (fitness(survivorPopulation[0]) == 0):
  print ("the solution is", survivorPopulation[0], "with", iterations, "iterations")
  return population
```

```
if (iterations >= 1000):
  print ("no solution found with 1000 iterations")
  return population
```