# -\*- coding: utf-8 -\*-

"""Genetic\_Algorithm\_Skeleton\_Code\_Student.ipynb

Automatically generated by Colaboratory.

Original file is located at

    https://colab.research.google.com/drive/1S1iEGF4FooL0NM-vseVOozTBnlQgRG47

## CSE422 lab: Genetic Algorithm

#### Genetic Algorithm Pseudo code:

\*\*function\*\* GENETIC-ALGORITHM( population, FITNESS-FN) \*\*returns\*\* an individual

> \*\*inputs:\*\* population- a set of individuals/chromosomes; FITNESS-FN- a function that measures the fitness of an individual

>\*\*repeat\*\*

new\_population $\leftarrow$ empty set

>>\*\*for\*\* $i=1$ \*\*to\*\* size ($ population$) \*\*do\*\*

$$

\begin{array}{l}

x \leftarrow \text { RANDOM-SELECTION }(\text { population, FITNESS-FN }) \\

y \leftarrow \text { RANDOM-SELECTION }(\text { population, FITNESS-FN }) \\

child  \leftarrow \operatorname{CROSSOVER}(x, y)

\end{array}

$$

>>>\*\*if\*\* (some\_random\_number < mutation\_threshold) \*\*then\*\* child$\leftarrow$ MUTATE ( child )

>>>add child to new\_population

>>population $\leftarrow$ new\_population

>\*\*until\*\* some individual is fit enough, or enough time has elapsed

>\*\*return\*\* the best individual in population, according to FITNESS-FN

### Skeleton Code:

### Importing libraries

"""

import numpy as np

import math

from numpy.random.mtrand import random\_integers

"""### Fitness function"""

def fitness(population, values):

    '''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.'''

    fitness\_result = []

    fitness\_summation = []

    values=np.array(values)

    for chromosome in population:

        x=chromosome\*values

        #print(chromosome,values,x)

        summation = np.sum(x)

        fitness\_summation.append(summation)

    for a in range(len(fitness\_summation)):

        if fitness\_summation==0:

            fitness\_result=[0 for a in range(len(summation))]

            fitness\_result[a]=1.0

            return fitness\_result

        else:

            x=1/np.absolute(fitness\_summation[a])

            fitness\_result.append(x)

    #print(np.sum(chromosome))

    fitness\_result=np.array(fitness\_result)

    multiplier = 1/np.sum(fitness\_result)

    for x in range(len(fitness\_result)):

        fitness\_result[x] \*= multiplier

        if np.isinf(fitness\_result[x]):

            fitness\_result=[0 for a in range(len(summation))]

            fitness\_result[x]=1.0

            return fitness\_result

        if np.isinf(fitness\_result[x]):

            fitness\_result[x]=0

    print(fitness\_result)

    return fitness\_result

"""### Random Selection function

This built-in function might help to create the weighted random selection:

`numpy.random.choice(a, size, replace, p)`

`p` are the weights of the individuals- value between 0 and 1; refers to the probability of each individual being selected.

`a` is the array

`size` how many samples to return

`replace = True`

"""

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'''

    #a = [0,1,2,3,4]

    a = [a for a in range(len(population))]

    #size = 1

    #p = [.31, .29, 0.26, 0.14]

    #print(fit)

    #print(population)

    selected = np.random.choice(a,len(population),True,fit)

    selected = [population[a] for a in selected]

    #print(selected)

    return selected

"""### Crossover function

\*\*function\*\* CROSSOVER $(x, y)$ \*\*returns\*\* an individual

>\*\*inputs\*\*: $x, y$  which are the parent individuals

>$n \leftarrow \mathrm{LENGTH}(x) ; c \leftarrow$ random number from 1 to $n$

>\*\*return\*\* APPEND(SUBSTRING $(x, 1, c),$ SUBSTRING $(y, c+1, n))$

"""

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'''

    crossover\_point = np.random.randint(2,len(x)-2)

    temp\_1=np.concatenate((x[0:crossover\_point],y[crossover\_point:]))

    #temp\_2=np.concatenate((y[0:crossover\_point],x[crossover\_point:]))

    return temp\_1

"""###Mutation function"""

def mutate(child):

    '''take input: a child

        mutates a random

        gene into another random gene

        returns: mutated child'''

    random\_pos = np.random.randint(0,len(child))

    if child[random\_pos]==1:

        child[random\_pos]=0

    else:

        child[random\_pos]=1

    return child

"""### Genetic Algorithm Function"""

def GA(population, values, mutation\_threshold = 0.3):

    '''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'''

    count=0

    while count<1000:

        new\_population=[]

        fitness\_calculated=fitness(population,values)

        for a in range(len(fitness\_calculated)):

            if fitness\_calculated[a] == 1:

                return population[a]

        random\_selected=select(population,fitness\_calculated)

        for a in range(0,len(random\_selected),2):

            if a<len(random\_selected)-1:

                child = crossover(random\_selected[a],random\_selected[a+1])

            else:

                child = random\_selected[a]

            if np.random.normal()<mutation\_threshold:

                child = mutate(child)

            new\_population.append(child)

        population=new\_population

        count+=1

    #z=crossover(y[0],y[1])

    #print(y[0],y[1])

    #print(z)

    return -1

"""Running the Genetic Algorithm function"""

file\_1=open("input\_1.txt","r")

count=int(file\_1.readline().rstrip())

values=[]

for a in range(count):

    x=file\_1.readline().rstrip()

    x=x.split()

    y=int(x[1])

    if x[0]=="l":

        y\*=-1

    values.append(y)

#print(values)

'''for 8 queen problem, n = 8'''

n = len(values)

'''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 = 0.3

'''creating the population with random integers between 0 to 7 inclusive

   for n = 8 queen problem'''

population = np.random.randint(0, 2, (start\_population, n))

#print(population)

'''calling the GA function'''

answer = GA(population, values, mutation\_threshold)

print("".join(str(a) for a in list(answer)))