from pickle import FALSE, TRUE

import random

#################

# taking input

#################

# taking number of transections

transection\_amount\_input = -1

transection\_amount\_input\_accepted = FALSE

while (transection\_amount\_input < 1 or transection\_amount\_input > 10\*\*2) and transection\_amount\_input\_accepted == FALSE:

transection\_amount\_input = int(input())

if transection\_amount\_input >= 1 and transection\_amount\_input <= 10\*\*2:

transection\_amount\_input\_accepted = TRUE

else:

print('invalid number. N( 1 ≤ N ≤ 10^2 )')

value\_accepted = FALSE

# taking transection values

transections\_input = []

for i in range(transection\_amount\_input):

value\_accepted = FALSE

while value\_accepted == FALSE:

transection = input().replace(' ', '')

if transection[0] == 'l':

value = 0-int(transection[1:])

transections\_input.append(value)

value\_accepted = TRUE

elif transection[0] == 'd':

value = 0+int(transection[1:])

transections\_input.append(value)

value\_accepted = TRUE

else:

print('wrong input, give again')

value\_accepted = FALSE

#####################################################

# if all inputs are fine then start Genetic Algorithm

#####################################################

if(value\_accepted == TRUE):

transection\_amount = transection\_amount\_input

transections = transections\_input

# fitness\_function

def fitness\_function(transections, total\_genome, genome\_length, population):

# counting sum for each genome

for i in range(total\_genome):

fitness = 0

for j in range(genome\_length):

if population[i][0][j] == '1':

#print(count, '+', transections[j], '= ')

fitness = fitness+int(transections[j])

# store the sum of that genome as fitness

population[i][1] = fitness

ans\_found = FALSE

run\_time = 0

while ans\_found == FALSE and run\_time < 10000:

################################

# generate initial population

################################

# will create 'total\_genome' number of genomes

total\_genome = 60

population = [[]]

for i in range(total\_genome):

genome = ''

genome\_checker = False

# if genome isn't appropriate then make again

while(genome\_checker == False):

# create genome where length = transection\_amount

for j in range(transection\_amount):

genome = genome+(str(random.randint(0, 1)))

# check if it's acceptable add in population

# at least two 1 is mandatory

if(genome.count('1') < 2):

genome = ''

genome\_checker = False

else:

population[i].append(genome)

genome\_checker = True

if (i < total\_genome-1):

population.append([])

#######################

# Fitness calculation

#######################

for i in range(total\_genome):

population[i].append(int(0))

fitness\_function(transections, total\_genome,

transection\_amount, population)

#######################

# Parent selection

#######################

# sort population by fitness

population.sort(key=lambda s: s[1])

# collecting 1/2 genomes as parent from sorted population

parents = [[]]

end\_collection = int(total\_genome/2)

parents = population[0:end\_collection]

#######################

# Crossover

#######################

child\_population = [[]]

# swap two random positions

row\_index = 0

for i in range(int(len(parents)/2)):

for j in range(2):

# select a random position

swaping\_index = random.randint(0, transection\_amount-1)

# store the character of that position

temp = str(parents[row\_index][0][swaping\_index])

# swaping the character with it's next row

parents[row\_index][0] = parents[row\_index][0][:swaping\_index] + \

str(parents[row\_index+1][0][swaping\_index]) + \

parents[row\_index][0][swaping\_index+1:]

# swaping the next parent's position which belongs to the next row

parents[row\_index+1][0] = parents[row\_index+1][0][:swaping\_index] + \

str(temp)+parents[row\_index+1][0][swaping\_index+1:]

row\_index = row\_index+2

child\_population = parents

#######################

# Mutation

#######################

for i in range(len(child\_population)):

# select a random position

muted\_index = random.randint(0, transection\_amount-1)

child\_population[i][0] = child\_population[i][0][:muted\_index] + \

str(random.randint(0, 1)) + \

child\_population[i][0][muted\_index+1:]

###########################################

# Fitness calculation of new population

###########################################

fitness\_function(transections, len(child\_population),

transection\_amount, child\_population)

###########################################

# Check if ans exist in new population

###########################################

iterator = 0

while ans\_found == FALSE and iterator < int(len(child\_population)):

if child\_population[iterator][1] == 0:

if(child\_population[iterator][0].count('1') >= 2):

#print('ANS: ', child\_population[iterator])

print(child\_population[iterator][0])

ans\_found = TRUE

iterator = iterator+1

run\_time = run\_time+1

if ans\_found == FALSE:

#print('Total run time = ',run\_time)

#print('ANS NOT FOUND: -1')

print('-1')

else:

print('wrong input given')