```
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')
```