مقدمه:

در این پروژه می‌خواهیم با استفاده از الگوریتم ژنتیک یک متن رمزنگاری شده را بر گردانیم
برای این کار ابتدا نیاز به یک لغت‌نامه داریم تا کلمات ای که درون رمز ما قرار دارند در آن وجود داشته باشد و بتوانیم آن‌ها را شناسایی کنیم
سپس باید با استفاده از الگوریتم ژنتیک کلمات متن رمز شده را جدا کرده و با توابع درست آن را به متن اصلی بر گردانیم

تمیز کردن کد:

ما ابتدا برای داشتن یک لغت‌نامه معتبر نیاز داریم تا آن را تمیز کنیم تا بتوانیم در کد از آن استفاده کنیم
برای تمیز کردن کد یک سری کار هایی باید روی متن اصلی انجام دهیم از جمله آن‌ها :

۱) ابتدا می‌دانیم کلمات رمز شده فقط از حروف اصلی الفبا هستند پس می‌توانیم مانند زیر تمام اعداد آن را حذف کینم مانند زیر:

```
def drop_numbers(self):
    return re.sub(r'\d+', '', self.text)
```

۲) سپس می‌دانیم باید تمام علایم نگارشی را از آن حذف کنیم مانند زیر:

```
def drop_punctuation(self):
    my_punctuation = string.punctuation.replace("'", "")
    self.text = self.text.translate(str.maketrans(my_punctuation, ' ' * len(my_punctuation)))
    self.text = self.text.replace("'", "")
    return self.text
```

۳)سپس برای کم کردن لغت‌نامه ها هم متن رمز شده هم لغت‌نامه می‌توانیم کلمه های ۱ ۲ ۳ حرفی را از آن‌ها حذف کنیم تا سرعت گشتن ما زیاد شود مانند زیر:

```
def remove_two_three_length_words(self):
    self.text = re.sub(r'\b\w{1,3}\b', '', self.text)
    return self.text
```

۴) برای اینکه بتوانیم در متن سرچ کنیم نیاز است تا لغات را جدا جدا کنیم برای این کار متن را tokenize می‌کنیم و سپس می‌توانیم تمام ایست واژه‌ها را نیز حذف کنیم زیرا این واژه‌ها نمی‌توانند تأثیری روی متد ما بگزارند

```
def drop_stop_words_and_tokenize(self):
    tokens = self.tokenize_text()
    return [i for i in tokens if i not in ENGLISH_STOP_WORDS]
```

۵) سپس می‌توانیم تمام واژه‌های تکراری را از متن حذف کنیم که این کار نیز می‌تواند سرعت گشتن ما را در لغت‌نامه ها زیاد کند:

```
self.text = list(dict.fromkeys(self.text))
```

این کار ها را برای هر دو لغت‌نامه یعنی هم کلمات رمز شده و هم کلمات دیکشنری خودمون انجام می‌دهیم تا به یک لیست از لغات برسیم و بتوانیم از آن‌ها در کد اصلی استفاده کنیم

الگوریتم ژنتیک:

برای پیاده‌سازی الگوریتم چند مرحله نیاز است تا انجام دهیم که آن‌ها را به ترتیب می گوییم:

۱) ابتدا نیاز است تا مفهوم کروموزوم را تعریف کنیم برای این کار در یک کلاس به اسم individual نوشته شده است که به معنی همین کروموزوم ماست
هر کروموزوم ما در این پروژه دارا ی یک عدد که معرف میزان تناسب آن (Fitness) آن و یک دیکشنری که معرف نگاشت حروف الفبا است

```
def __init__(self, mapped_alphabet, fitness=0):
    self.fitness = fitness
    self.mapped_alphabet = dict(zip(string.ascii_lowercase, mapped_alphabet))
```

هر کروموزوم یک تابع دارد که آن برای پیدا کردن تناسب آن کروموزوم است
پیدا کردن تناسب به این‌گونه است که ابتدا باید با نگاشتی که آن کروموزوم دارد متن را دیکود کنیم و سپس با استفاده از لغات بدست آمده یک تناسب برای آن کروموزوم بدست می‌آوریم

تناسبی که برای این پروژه استفاده شده است تعداد کلمات کامل درست از متن رمز شده است به این معنا که ابتدا متن را با نگاشت بر می گردانیم سپس به ازای هر لغت درست در آن یک واحد به تناسب آن اضافه می‌کنیم
تناسب را برای هر کروموزوم به شکل زیر بدست می اوریم:

```
def calculate_fitness(self, global_dictionary, encoded_dictionary):
    self.fitness = 0
    decoded_text = self.decode_string_list(encoded_dictionary)
    self.__find_fitness(decoded_text, global_dictionary)
    return self.fitness
```

۲) سپس باید قسمت کراس اور را پیاده‌سازی کنیم. این قسمت به این‌گونه است که بخشی از جمعیت حاضر را به عنوان والد در نظر می‌گیریم و از آن‌ها هر کدام ۲ بچه درست می‌کنیم که بخشی از ژن آن‌ها از والد اول و بخشی دیگر از والد دوم است و این کار را در هر نسل برای تعدادی از والد ها که از بهترین آن جمعیت انتخاب می‌شود درست می کنیم. تیکه اسفاده شده از ژن ها در هر نسل به صورت شانسی و از ادقام والدها بدست می‌آید
در این پروژه برای این قسمت به این شکل عمل شده است که برای هر حرف یک سکه می‌اندازیم و اگر خط امد(احتمال ۵۰ درصد) آن حرف از والد اول را برای بچه اول و حرف والد دوم را برای بچه ی دوم می گذازیم و این کار را برای تمامی حروف ادامه می‌دهیم و به این صورت ۲ بچه از دو والد انتخاب شده بدست می‌آید
این کار را برای هر نسل به یک احتمال به اسم cross over rate انجام می‌دهیم در این صورت بد ها نیز شانس کراس شدن را دارند

۳) قسمت بعد برای میوتیشن یا جهش است. در هر نسل بچه‌های تولید شده را جهش می‌دهیم در این صورت می‌توانیم از سو گیری کوروموزوم ها جلوگیری کنیم و نمی‌گذاریم که الگوریتم در یک مینیموم یا ماکسیموم محلی گیر کند.
در این پروژه میوتیشن را به این‌گونه انجام می‌دهیم که هر سری یک احتمال رندوم می‌دهیم و اگر آن احتمال بالا تر از mutation rate شد جای دو ژن از کروموزوم را به صورت رندوم جا به جا می‌کنیم

۴) در قسمت آخر نیز دوباره تناسب تمام کروموزوم های نسل جدید را بدست می‌آوریم و آن را مرتب می‌کنیم و آن‌ها را برای اجرای دوباره الگوریتم به نسل بعد می دهیم

قسمت‌های مختلف را در ادامه دوباره توضیح می‌دهیم

اگر جمعیت را در هر نسل اضافه کنیم باعث می‌شود بتوانیم فضای بیشتری از فضای کل (فضای حالت) بدست اوریم ولی به همین اندازه چون جمعیت بیشتر شده است باعث می‌شود سرعن اگوریتم برای ایجاد نسل های جدید تر بیشتر شود پس نمی‌توان این جمعیت را خیلی افزایش داد ولی این کار دقت رسیدن به جواب را افزایش می‌دهد

mutation برای این است که از سو گیرکروموزوم ها جلو گیری شود با این کار الگوریتم کم‌تر در مینیموم و ماکسیموم محلی خود گیر می‌کند بدون آن ممکن است کروموزوم ها یک سو گیری انجام دهند و بعد از مدتی دیگر با ایجاد نسل جدید تغییر نکنند به این دلیل نیاز به mutation داریم.

برای ایجاد نسل جدید و رسیدن به جواب cross over ما را زود تر به جواب می‌رساند ولی بدون mutation ممکن است الگوریتم گیر کند و نتواند هیچ وقت به جواب برسد

با وجود mutation باز ممکن است کروموزوم ها جهت گیری داشته باشند برای جلو گیری از آن می‌توانیم mutation rate را زیاد و یا cross over rate را کم کنیم تا اجازه دهیم الگوریتم بتواند از مقادیر محلی خود خارج شود و کروموزوم ها جهت گیری نکنند

در این پروژه الگوریتم به صورت زیر پیاده‌سازی شده است که داریم:

برای فیتنس داریم:

```
def decode_string_list(self, encoded_dictionary):
    new_text = []
    for word in encoded_dictionary:
        new_word = ""
        for letter in word:
            if letter.isupper():
                letter = self.mapped_alphabet[letter.lower()].upper()
            if letter.islower():
                letter = self.mapped_alphabet[letter]
            new_word += letter
        new_text.append(new_word)
    return new_text

def __find_fitness(self, decoded_dictionary, global_dictionary):
    fitness = 0
    for decoded_word in decoded_dictionary:
        for global_word in global_dictionary:
            if decoded_word == global_word:
                fitness += 1
    self.fitness = fitness
    return self.fitness

def calculate_fitness(self, global_dictionary, encoded_dictionary):
    self.fitness = 0
    decoded_text = self.decode_string_list(encoded_dictionary)
    self.__find_fitness(decoded_text, global_dictionary)
    return self.fitness
```

و برای کراس اور داریم:

```
@staticmethod
def __crossover(individual1_passed, individual2_passed):
    child1 = dict.fromkeys(string.ascii_lowercase, 0)
    child2 = dict.fromkeys(string.ascii_lowercase, 0)
    used_alphabet_child1 = []
    used_alphabet_child2 = []

    for letter in string.ascii_lowercase:
        if random.uniform(0, 1) < 0.5:
            temp = individual1_passed
            individual1_passed = individual2_passed
            individual2_passed = temp
        if individual1_passed.mapped_alphabet[letter] not in child1.values():
            child1[letter] = individual1_passed.mapped_alphabet[letter]
            used_alphabet_child1.append(individual1_passed.mapped_alphabet[letter])
        else:
            child1[letter] = 'None'
        if individual2_passed.mapped_alphabet[letter] not in child2.values():
            child2[letter] = individual2_passed.mapped_alphabet[letter]
            used_alphabet_child2.append(individual2_passed.mapped_alphabet[letter])
        else:
            child2[letter] = 'None'
```

```
unused_alphabet_child1 = list(set(string.ascii_lowercase) - set(used_alphabet_child1))
unused_alphabet_child2 = list(set(string.ascii_lowercase) - set(used_alphabet_child2))
# i = 0
# j = 0
for alphabet in string.ascii_lowercase:
    if child1[alphabet] == 'None':
        random_letter = random.randint(0, len(unused_alphabet_child1) - 1)
        child1[alphabet] = unused_alphabet_child1[random_letter]
        unused_alphabet_child1.remove(unused_alphabet_child1[random_letter])
        # i += 1
    if child2[alphabet] == 'None':
        random_letter = random.randint(0, len(unused_alphabet_child2)-1)
        child2[alphabet] = unused_alphabet_child2[random_letter]
        unused_alphabet_child2.remove(unused_alphabet_child2[random_letter])
        # j += 1
return Individual(list(child1.values())), Individual(list(child2.values()))
```

که در آن برای هر حرف یک سکه انداخته و اگر خط بیاید حرف از والد اول به بچه اول و اگر شیر بیاید از والد دوم به بچه اول می رود

و برای میوتیشن داریم:

```
def mutation(self, individual):
    if random.uniform(0, 1) < self.mutation_rate:
        for letter in range(4):
            random_letter_1 = random.randint(0, 13)
            random_letter_2 = random.randint(13, 25)
            temp_letter = individual.mapped_alphabet[string.ascii_lowercase[random_letter_1]]
            individual.mapped_alphabet[string.ascii_lowercase[random_letter_1]] = individual.mapped
            individual.mapped_alphabet[string.ascii_lowercase[random_letter_2]] = temp_letter
    return individual
```

که در آن هر سری ۴ حرف به صورت رندوم جای آن‌ها با ۴ حرف دیگر عوض می‌شود

و برای الگوریتم داریم:

```
evolve(self):
my_text_object_global = TextProcessing(text="", text_file_address="Attachment/global_text.txt")
my_text_object_encoded = TextProcessing(text="", text_file_address="Attachment/encoded_text.txt")
global_text = my_text_object_global.clean_text()
encoded_text = my_text_object_encoded.clean_text()
print(global_text)
print(encoded_text)
generation = Population(self.population_size, True)
# print(my_answer.decode_string_list(encoded_text))
max_fitness = generation.find_max_fitness(encoded_text, global_text)
# max_fitness = generation.find_max_fitness(encoded_text)
print(max_fitness)
for j in range(self.number_of_generations):
    if generation.get_individuals()[0].fitness != max_fitness:

        print("Generation number: ", j)
        for chromosome in generation.get_individuals():
            chromosome.calculate_fitness(global_text, encoded_text)

        generation.get_individuals().sort(key=lambda x: x.fitness, reverse=True)
```

مه ابتدا متن ها را از فایل خوانده و یک جمعیت اولیه می‌سازیم

سپس بیشترین مقدار فیتنس را بدست می‌آوریم که آن شرط توقف الگوریتم است

سپس هر سری در هر نسل مقادیر تناسب آن جمعیت را بدست می‌آوریم و نسل را نسبت به آن مرتب می‌کنیم

```
        new_generation = generation.get_individuals()[:int(self.population_size * 0.1)]
        for i in range(int(self.population_size * 0.7)):
            if random.uniform(0, 1) < self.cross_chance:
                while True:
                    parent1, parent2 = random.choices(new_generation[:int(self.population_size * 0.8)]
                    # parent1, parent2 = self.tournament(new_generation)
                    if parent1 != parent2:
                        break
                child1, child2 = self.__crossover(parent1, parent2)
                # child1 = self.__crossover(parent1, parent2)
                child1 = self.mutation(child1)
                child2 = self.mutation(child2)
                new_generation.append(child1)
                new_generation.append(child2)

        # for i in range(len(new_generation[:int(self.population_size * 0.4)])):
        #     new_generation[i] = self.mutation(new_generation[i])

        generation.save_individuals(new_generation)
        print("Report Best Fitness: ", generation.get_individuals()[0].fitness)
        print("Population number is:", len(generation.get_individuals()))
    else:
        break
```

سپس ۲۰ درصد اول جمعیت را مستقیم به نسل بعد منتقل می‌کنیم
به احتمال cross over rate از ۸۰ درصد بهتر جمعیت ۲ والد انتخاب کرده آن‌ها را کراس می‌کنیم و بچه‌ها را جهش می‌دهیم

این کار را انقدر انجام می‌دهیم تا به نتیجه برسیم

سپس متن را با بالاترین فیتنس دیکود می‌کنیم

کد را به صورت زیر اجرا می کنیم:

```
class Decoder:
    def __init__(self, encoded_text):
        self.encoded_text = encoded_text

    def decode(self):
        ga = GeneticAlgorithm(50, 500, 0.4, 2, self.encoded_text)
        decoded_text = ga.evolve()
        return decoded_text
```

جواب الگوریتم به صورت زیر می شود:

```
o', 'r', 's', 'f', 'w', 'm', 'b', 't', 'i', 'k', 'g', '
'h', 'k', 'n', 'v', 'e', 'l', 'p', 'd', 'j', 'c', 'u', 'y',
''q', 'a', 'x
```

می‌شود که به ترتیب نگاشت حروف الفبا است