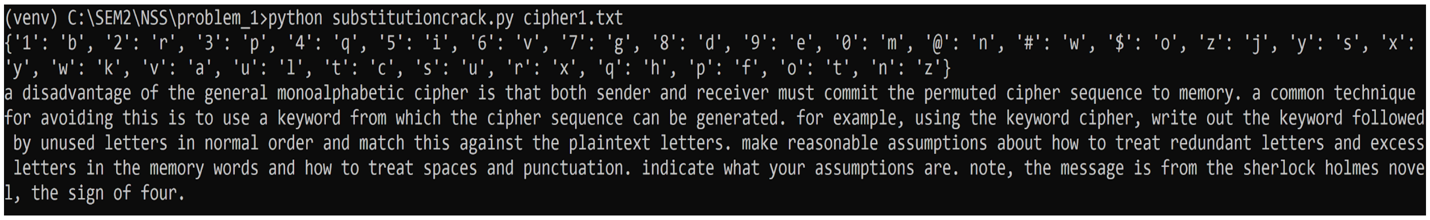
Problem 1:

I have used genetic algorithm to find correct substitution from cipher char set to real char

```
pop_size = 500       # total randomly initialize mapping
select_pop = 150     # top most fit mapping
select_random = 50   # randomly selected from unfit mapping
child_count = 300    # new child mapping after each generation
generation = 200     # total generation
alphabete_set = 26   # alphabet set length for input and output charset
crossover_rate = 30   # crossover
mutation_rate = 3     # mutation rate
```

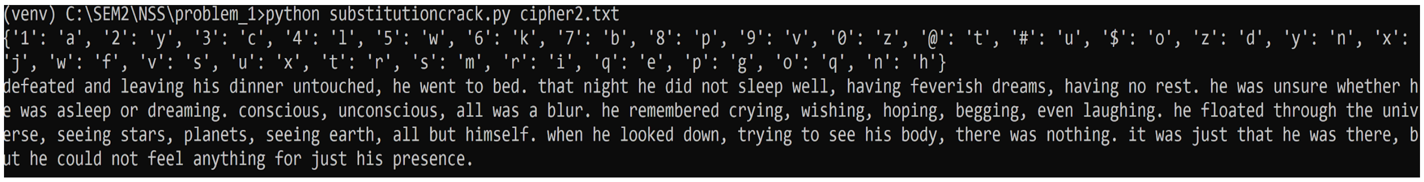
Fitness Calculation:
Step1: Select random population of mappings (defined constant above).
Step2: Calculate fitness of each mapping(matching with english language)
For fitness calculation I have used [1], Data is obtained from [2]
Step3: Arrange population by their fitness, select top half of population. Select 10% random from rest
Step4: Choose any two random parents and perform crossover (crossover rate).
Step5: Apply mutation (defined mutation rate)
Step6: Repeat for generations configured.

How to run.
python substitutioncrack.py cipher1.txt



```
(venv) C:\SEM2\NSS\problem_1>python substitutioncrack.py cipher1.txt
{'1': 'b', '2': 'r', '3': 'p', '4': 'q', '5': 'i', '6': 'v', '7': 'g', '8': 'd', '9': 'e', '0': 'm', '@': 'n', '#': 'w', '$': 'o', 'z': 'j', 'y': 's', 'x':
'y', 'w': 'k', 'v': 'a', 'u': 'l', 't': 'c', 's': 'u', 'r': 'x', 'q': 'h', 'p': 'f', 'o': 't', 'n': 'z'}
a disadvantage of the general monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. a common technique
for avoiding this is to use a keyword from which the cipher sequence can be generated. for example, using the keyword cipher, write out the keyword followed
 by unused letters in normal order and match this against the plaintext letters. make reasonable assumptions about how to treat redundant letters and excess
 letters in the memory words and how to treat spaces and punctuation. indicate what your assumptions are. note, the message is from the sherlock holmes nove
l, the sign of four.
```

python substitutioncrack.py cipher2.txt



```
(venv) C:\SEM2\NSS\problem_1>python substitutioncrack.py cipher2.txt
{'1': 'a', '2': 'y', '3': 'c', '4': 'l', '5': 'w', '6': 'k', '7': 'b', '8': 'p', '9': 'v', '0': 'z', '@': 't', '#': 'u', '$': 'o', 'z': 'd', 'y': 'n', 'x':
'j', 'w': 'f', 'v': 's', 'u': 'x', 't': 'r', 's': 'm', 'r': 'i', 'q': 'e', 'p': 'g', 'o': 'q', 'n': 'h'}
defeated and leaving his dinner untouched, he went to bed. that night he did not sleep well, having feverish dreams, having no rest. he was unsure whether h
e was asleep or dreaming. conscious, unconscious, all was a blur. he remembered crying, wishing, hoping, begging, even laughing. he floated through the univ
erse, seeing stars, planets, seeing earth, all but himself. when he looked down, trying to see his body, there was nothing. it was just that he was there, b
ut he could not feel anything for just his presence.
```

Note: Since we have used genetic algorithm
Sometimes multiple run of the program required to get correct mapping (or increase configuration)

References:
[1] https://people.cs.uct.ac.za/~jkenwood/JasonBrownbridge.pdf
[2]
http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/#a-python-implementation