Indi 2 – Pac109

!!! This is an INDIVIDUAL project. Submit your project report as one file (word or pdf) in Canvas by 5pm, May 5. !!!

====== Project Description ======

The project is to crack a cipher text, which was encrypted using the Vigènere Cipher. The cipher is described in the lecture. You can also find many tutorials and examples online. A subtle difference is that the cipher uses 27 characters (instead of 26) in clear text, cipher text and key. The encryption E is c=(m+k)%27. The decryption D is m=(27+c-k)%27.

The characters include the upper case 'A' to 'Z' and the space ' '. 'A' is equivalent to 0. 'B' is equivalent to 1. ... 'Z' is equivalent to 25. And, the space ' ' is equivalent to 26.

For example, if the clear text is "HELLO WORLD" and the key is "ABC", then the cipher text is "HFNLPBWPTLE".

The cipher text of the project is in cipher.txt. The cipher text is only ONE line of characters. Please follow the cracking approach in the lecture to find the key. The key length is smaller than 10.

You need to develop a program(s) in C/C++, Python, Java, Matlab, or R to automate the crypto-analysis. If you

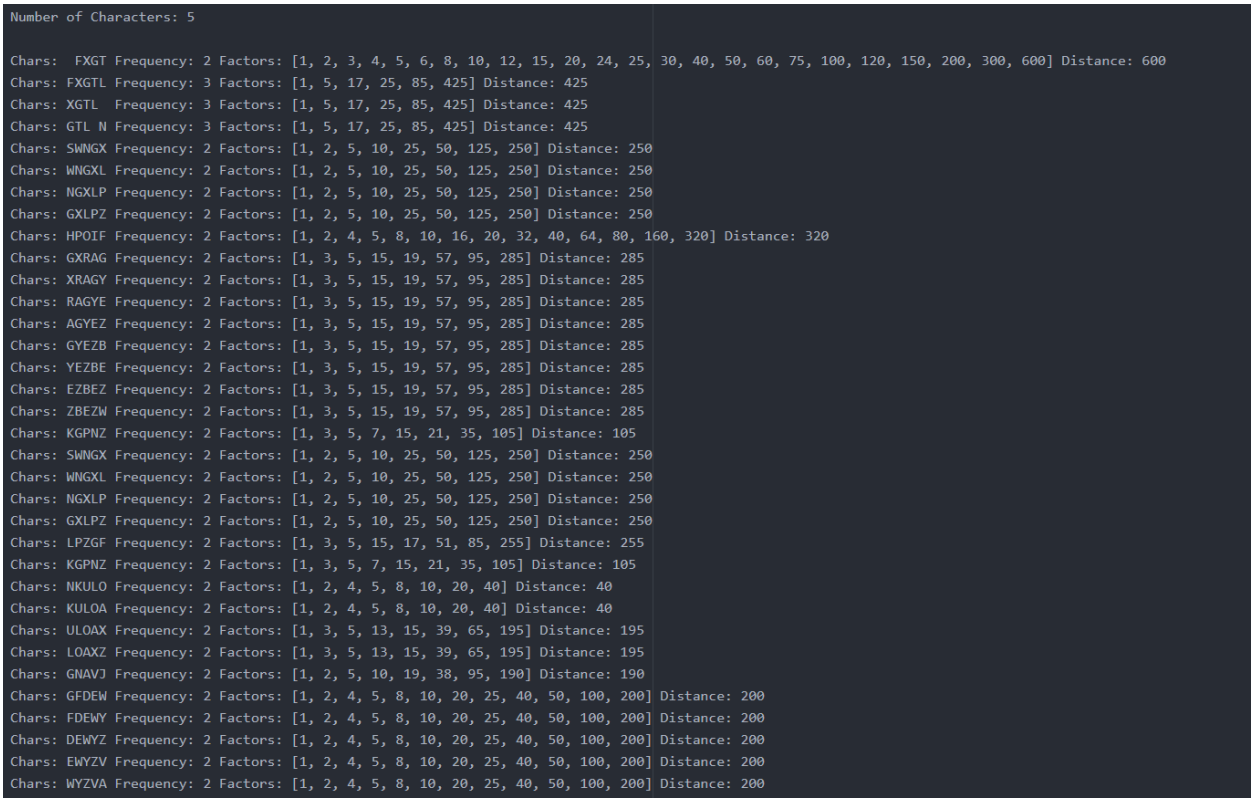
Manual analysis (such as using Excel) will not be counted towards your project grade.

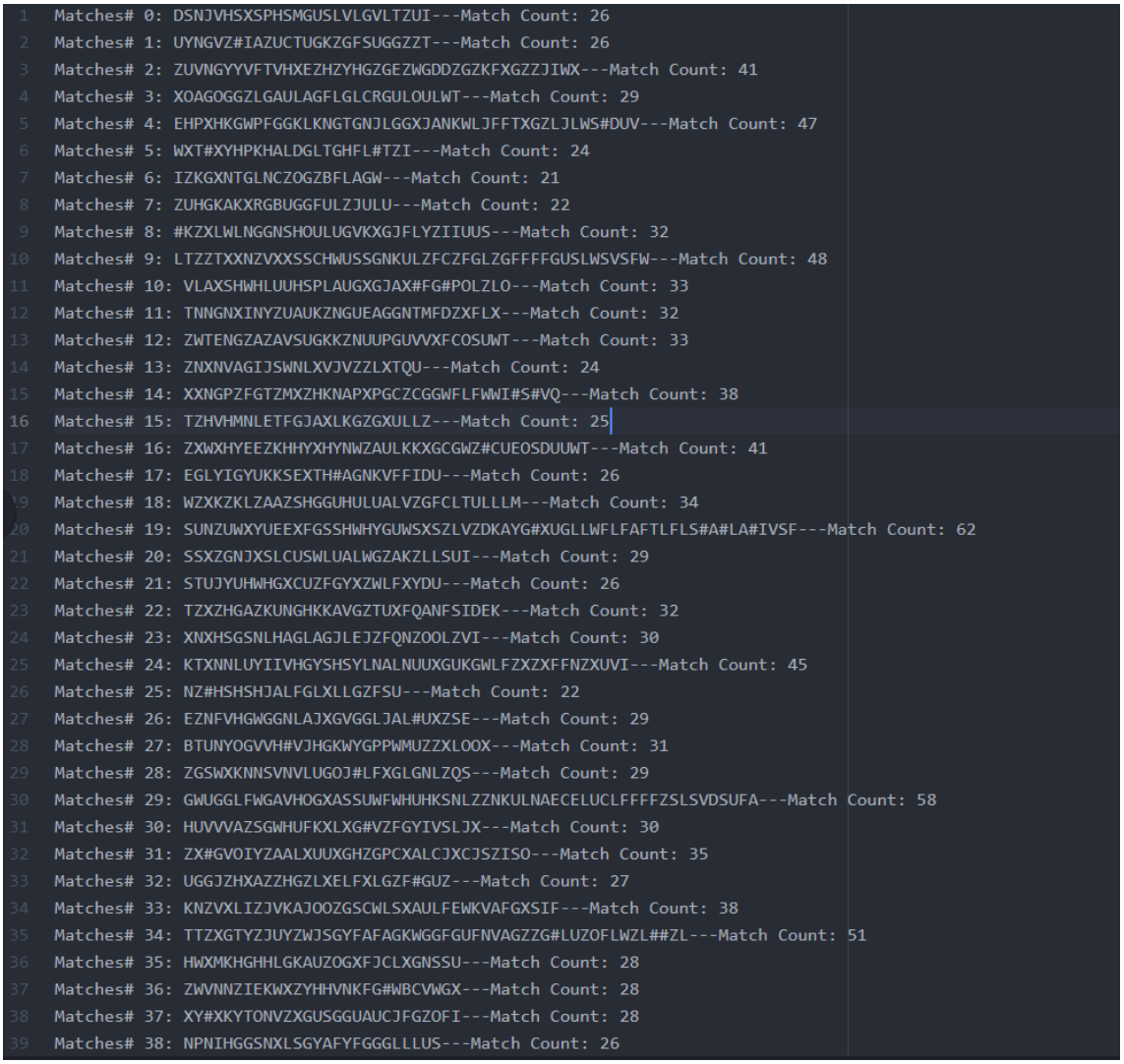
====== Report and Grading ======

The score of the project is 10 points.

Your report should include what you did and the results you obtained in the four steps below.

Step 1: You will get 2 points if you are able to do the analysis on repeating patterns in the cipher text and do factorizing to guess the possible size of keys. Show a screenshot of the repeating patterns when running your analysis program.





So I went about solving this problem two different ways. The first screenshot shows the algorithm the professor used in lecture where it finds the chars that repeat and it finds the distance between the repeats and then factorizes them. I was able to see that 5 was the most likely but 2,3,4,6 also seemed likely. To help decern which factor to use first I found another algorithm online that suggested to find the number of matches per line and then increments whenever you shift the cypher text by one and whenever that's done you find which lines give the highest number to find a pattern between them. That number also happened to be 5 so I used that as my first potential key size.

Step 2: After step 1, you will get 2 more points, if you are able to divide the cipher text into groups based on the guessed key size, do the 1-gram character analysis for all groups, and then guess the possible key characters for the groups. Show a screenshot of the cipher text groups when running your analysis program.

```
(base) PS C:\Users\phili\Desktop\pythonindi2> python analysis.py
EFHTATSWEETXTGXASLVG DLTXXGVLFTKKAEZ GEJSCHAQSJSKGVSALXMSXLAYUGKSCSLSKK SLYLWHADVLXDTSEXKAXVRFGEWLGAXDJJXGGGVFQHLSLGA LDVKXTGQFXCNAFLGAGTTS XXSLSEJJYDSX
SSULXDJUMELDXSTK

VG WZBIGHHSVUTYLWPYNPNOYZKAVHHYZAVHGGYZKIGYJGHZPGYHWIVHJZTGZLSGVHUWPTGLLHOVLGAVEAO GULWTGUY HGYZGLTUJVLLZSUGVAG OLGYVLPEAJZ BGVGLL SLTUCUTPLGGIOZLLLM PT
  LO SLS WPSJTU

 ZKDWEXKPPWN LWJNZUNZKIAEWJIPH WIJGOXEWWNEXAX XPPAHKHWGDUWEWXA WJKNZAPWINANNZPJWNAEXZCNAEPWKPXEWBNLCDCMWAQOYJXXAATEIJKZWNDWWXZPYJJDE LCAWLOWPLQAZOWWEWIA
DDOAEUXAWQKUQAE

KXMDNRYTNOZDF YIGKVGFXYFMGJVOFTGVYO RZNZKTIFTAXFNZRYKHFFYHZOYFUHDBGGGNYKKXKSUGGYKUIJKFUTTKLXORZGGF FTD ZFZFKZRJJXOTGFXGYKKZVHGFXFFFSF FXKRFUOGZYNFSJIZVT
GKZUIFQFIZTFXIY

LSLUZSZUZXBUXNXZXUNJVCUYHLUOIBZMNUHXABVIVUNWSZSCZCSMULMVNOUHCNUSUHXFHZXMUZUZGNFZULVPMCPNUAVCIABHMXNNIUCBMCNUCFVUZM NNZFZUGBLFHWIZQOCXNJURZNHGYUZZVICOIFN
HUULVWVWIVVMZBG
```

```
1  Line 1: ('A', 12) ('B', 0) ('C', 3)  ('D', 8) ('E', 9)  ('F', 6)  ('G', 15) ('H', 4)  ('I', 0) ('J', 7) ('K', 9)  ('L', 17) ('M', 2) ('N', 1)
2  Line 2: ('A', 6)  ('B', 2) ('C', 1)  ('D', 0) ('E', 2)  ('F', 0)  ('G', 22) ('H', 11) ('I', 4) ('J', 5) ('K', 2)  ('L', 20) ('M', 1) ('N', 2)
3  Line 3: ('A', 17) ('B', 1) ('C', 4)  ('D', 7) ('E', 13) ('F', 0)  ('G', 2)  ('H', 3)  ('I', 6) ('J', 9) ('K', 8)  ('L', 5)  ('M', 1) ('N', 12
4  Line 4: ('A', 1)  ('B', 1) ('C', 0)  ('D', 4) ('E', 0)  ('F', 26) ('G', 16) ('H', 4)  ('I', 7) ('J', 5) ('K', 15) ('L', 1)  ('M', 2) ('N', 6)
5  Line 5: ('A', 3)  ('B', 7) ('C', 11) ('D', 0) ('E', 0)  ('F', 6)  ('G', 4)  ('H', 9)  ('I', 8) ('J', 2) ('K', 0)  ('L', 7)  ('M', 9) ('N', 16
6
```

```
('N', 1)  ('O', 0) ('P', 0)  ('Q', 3) ('R', 1) ('S', 20) ('T', 11) ('U', 3)  ('V', 7)  ('W', 3)  ('X', 18) ('Y', 3)  ('Z', 1)  (' ', 5)
('N', 2)  ('O', 6) ('P', 8)  ('Q', 0) ('R', 0) ('S', 7)  ('T', 9)  ('U', 9)  ('V', 12) ('W', 6)  ('X', 0)  ('Y', 10) ('Z', 11) (' ', 11)
('N', 12) ('O', 5) ('P', 11) ('Q', 4) ('R', 0) ('S', 0)  ('T', 1)  ('U', 4)  ('V', 0)  ('W', 25) ('X', 12) ('Y', 2)  ('Z', 9)  (' ', 6)
('N', 6)  ('O', 7) ('P', 0)  ('Q', 1) ('R', 6) ('S', 3)  ('T', 10) ('U', 6)  ('V', 5)  ('W', 0)  ('X', 10) ('Y', 12) ('Z', 14) (' ', 5)
('N', 16) ('O', 4) ('P', 2)  ('Q', 1) ('R', 1) ('S', 6)  ('T', 0)  ('U', 24) ('V', 12) ('W', 4)  ('X', 9)  ('Y', 2)  ('Z', 19) (' ', 1)
```

Step 3: After step 2, you will get 2 more points, if you are able to guess at least two correct key characters and crack the corresponding 2 groups of the cipher text. Show a screenshot of the key characters when running your analysis program.

I noticed a pattern repeating throughout the different 1 gram lines that being:

Line 1: ('S', 20) ('T', 11) ('U', 3) ('V', 7) ('W', 3) ('X', 18) – HMLLLH

Line 2: ('G', 22) ('H', 11) ('I', 4) ('J', 5) ('K', 2) ('L', 20) – HMLLLH

LINE 3: ('W', 25) ('X', 12) ('Y', 2) ('Z', 9) (' ', 6) ('A', 17) – HMLLLH

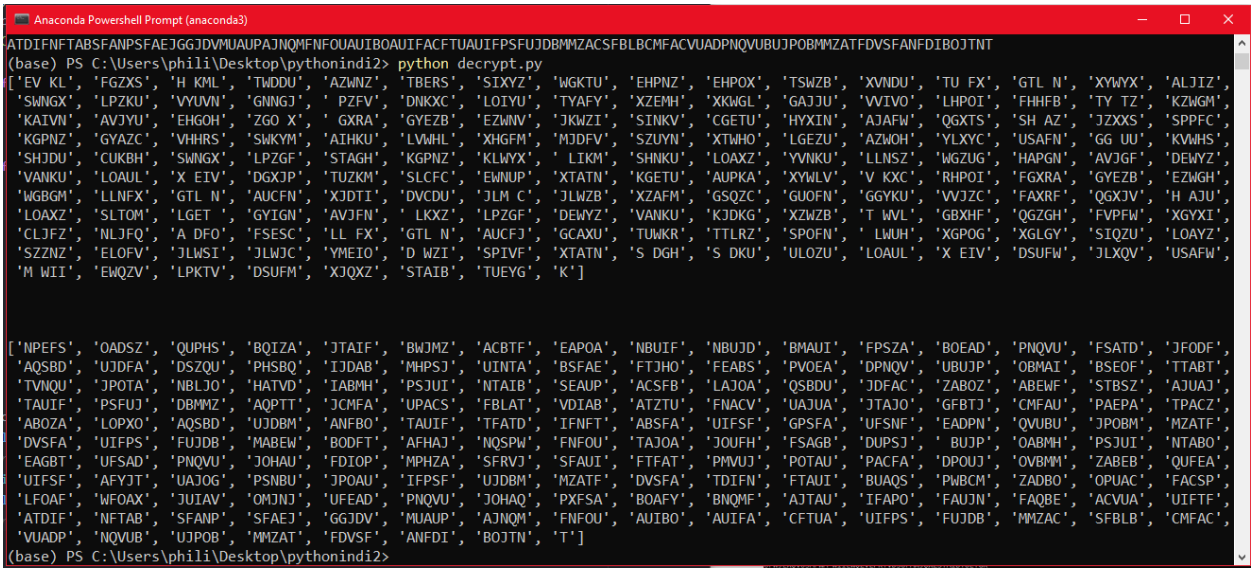
LINE 4: ('F', 26) ('G', 16) ('H', 4) ('I', 7) ('J', 5) ('K', 15) – HMLLLH

LINE 5: ('U', 24) ('V', 12) ('W', 4) ('X', 9) ('Y', 2) ('Z', 19) – HMLLLH

Based on this information I decided to use the start of each patten as my key for the decryption.

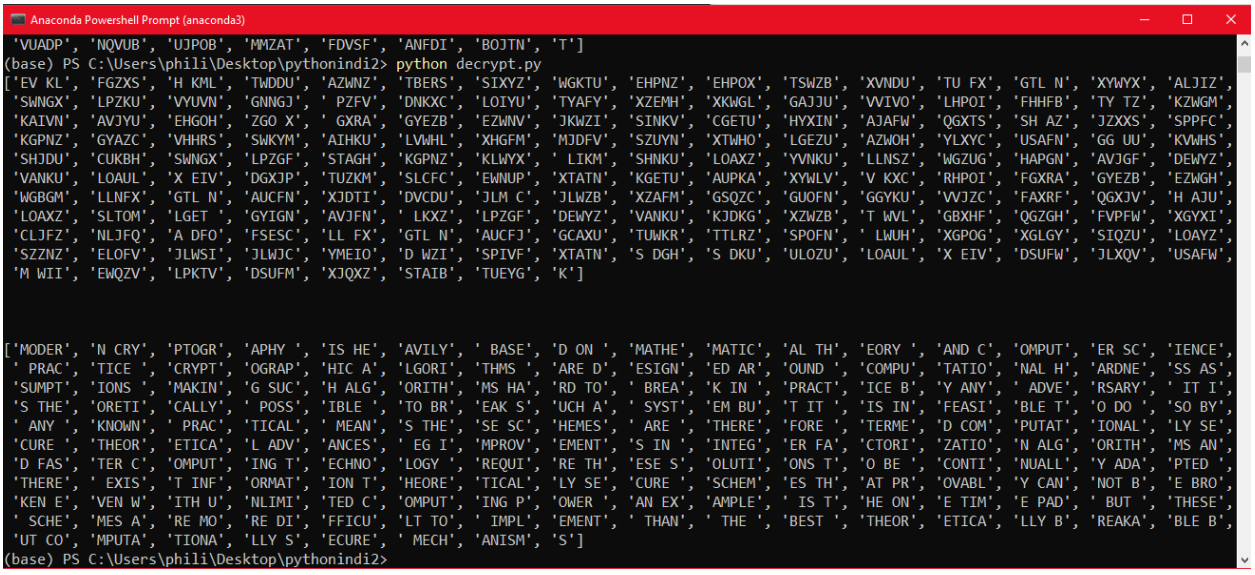
```
key = "SGWFU"
```

And we get garbage:



I was still sure that the patten I had found was the key based on the lecture slides so I shifted everything over to the right one and got `key = "THXGV"` as the key.

Which gave the results:



SUCCESS!

Step 4: After step 3, you will get 2 more points, if you are able to find the correct key and the clear text. Show a screenshot of the clear text when running your analysis program.

Step 5: Attach the source code of your program at the end of your report to get 2 more points.

NOTE

My code is in python and is weird about indentations and since copy and paste is really bad about keeping the indentations where they need to be, I am including the source code as separate files in my submission.

```
import os

from collections import Counter


class analysis():


    cypher = "EV KLFGZXSH KMLTWDDUAZWNZTBERSSIXYZWGKTUEHPNZEHPOXTSWZBXVNDUTU FXGTL
NXYWYXALJIZSWNGXLPZKUVYUVNGNNGJ
PZFVDNKXCLOIYUTYAFYXZEMHXKWGLGAJJUVVIVOLHPOIFHHFBTY TZKZWGMKAIVNAVJYUEHGOHZGO X
GXRAGYEZBEZWNVJKWZISINKVCGETUHYXINAJAFWQGXTSSH
AZJZXXSSPPFCKGPNZGYAZCVHHRSSWKYMAIHKULVWHLXHGFMMJDFVSZUYNXTWHOLGEZUAZWOHYLXY
CUSAFNGG UUKVWHSSHJDUCUKBHSWNGXLPZGFSTAGHKGPNZKLWYX
LIKMSHNKULOAXZYVNKULLNSZWGZUGHAPGNAVJGFDEWYZVANKULOAULX
EIVDGXJPTUZKMSLCFCEWNUPXTATNKGETUAUPKAXYWLVV
KXCRHPOIFGXRAGYEZBEZWGHWGBGMLLNFXGTL NAUCFNXJDTIDVCDUJLM
CJLWZBXZAFMGSQZCGUOFNGGYKUVVJZCFAXRFQGXJVH AJULOAXZSLTOMLGET GYIGNAVJFN
LKXZLPZGFDEWYZVANKUKJDKGXZWZBT WVLGBXHFQGZGHFVPFWXGYXICLJFZNLJFQA DFOFSESCLL
FXGTL NAUCFJGCAXUTUWKRTTLRZSPOFN
LWUHXGPOGXGLGYSIQZULOAYZSZZNZELOFVJLWSIJLWJCYMEIOD WZISPIVFXTATNS DGHS
DKUULOZULOAULX EIVDSUFWJLXQVUSAFWM WIIEWQZVLPKTVDSUFMXJQXZSTAIBTUEYGK"

    text = "V KLFGZXSH KMLTWDDUAZWNZTBERSSIXYZWGKTUEHPNZEHPOXTSWZBXVNDUTU FXGTL
NXYWYXALJIZSWNGXLPZKUVYUVNGNNGJ
PZFVDNKXCLOIYUTYAFYXZEMHXKWGLGAJJUVVIVOLHPOIFHHFBTY TZKZWGMKAIVNAVJYUEHGOHZGO X
```

GXRAGYEZBEZWNVJKWZISINKVCGETUHYXINAJAFWQGXTSSH
AZJZXXSSPPFCKGPNZGYAZCVHHRSSWKYMAIHKULVWHLXHGFMMJDFVSZUYNXTWHOLGEZUAZWOHYLXY
CUSAFNGG UUKVWHSSHJDUCUKBHSWNGXLPZGFSTAGHKGPNZKLWYX
LIKMSHNKULOAXZYVNKULLNSZWGZUGHAPGNAVJGFDEWYZVANKULOAULX
EIVDGXJPTUZKMSLCFCEWNUPXTATNKGETUAUPKAXYWLVV
KXCRHPOIFGXRAGYEZBEZWGHWGBGMLLNFXGTL NAUCFNXJDTIDVCDUJLM
CJLWZBXZAFMGSQZCGUOFNGGYKUVVJZCFAXRFQGXJVH AJULOAXZSLTOMLGET GYIGNAVJFN
LKXZLPZGFDEWYZVANKUKJDKGXZWZBT WVLGBXHFQGZGHFVPFWXGYXICLJFZNLJFQA DFOFSESCLL
FXGTL NAUCFJGCAXUTUWKRTTLRZSPOFN
LWUHXGPOGXGLGYSIQZULOAYZSZZNZELOFVJLWSIJLWJCYMEIOD WZISPIVFXTATNS DGHS
DKUULOZULOAULX EIVDSUFWJLXQVUSAFWM WIIEWQZVLPKTVDSUFMXJQXZSTAIBTUEYGK"

cmpText = "EV KLFGZXSH KMLTWDDUAZWNZTBERSSIXYZWGKTUEHPNZEHPOXTSWZBXVNDUTU
FXGTL NXYWYXALJIZSWNGXLPZKUVYUVNGNNGJ
PZFVDNKXCLOIYUTYAFYXZEMHXKWGLGAJJUVVIVOLHPOIFHHFBTY TZKZWGMKAIVNAVJYUEHGOHZGO X
GXRAGYEZBEZWNVJKWZISINKVCGETUHYXINAJAFWQGXTSSH
AZJZXXSSPPFCKGPNZGYAZCVHHRSSWKYMAIHKULVWHLXHGFMMJDFVSZUYNXTWHOLGEZUAZWOHYLXY
CUSAFNGG UUKVWHSSHJDUCUKBHSWNGXLPZGFSTAGHKGPNZKLWYX
LIKMSHNKULOAXZYVNKULLNSZWGZUGHAPGNAVJGFDEWYZVANKULOAULX
EIVDGXJPTUZKMSLCFCEWNUPXTATNKGETUAUPKAXYWLVV
KXCRHPOIFGXRAGYEZBEZWGHWGBGMLLNFXGTL NAUCFNXJDTIDVCDUJLM
CJLWZBXZAFMGSQZCGUOFNGGYKUVVJZCFAXRFQGXJVH AJULOAXZSLTOMLGET GYIGNAVJFN
LKXZLPZGFDEWYZVANKUKJDKGXZWZBT WVLGBXHFQGZGHFVPFWXGYXICLJFZNLJFQA DFOFSESCLL
FXGTL NAUCFJGCAXUTUWKRTTLRZSPOFN
LWUHXGPOGXGLGYSIQZULOAYZSZZNZELOFVJLWSIJLWJCYMEIOD WZISPIVFXTATNS DGHS
DKUULOZULOAULX EIVDSUFWJLXQVUSAFWM WIIEWQZVLPKTVDSUFMXJQXZSTAIBTUEYG"

```
length = len(text)
firstFive = ""
secondFive = ""
thirdFive = ""
forthFive = ""
fifthFive = ""

firstDict = dict()
secondDict = dict()
thirdDict = dict()
forthDict = dict()
```

```
fifthDict = dict()


def find_matching_chars_(self):
    filename = 'matches.txt'
    with open(filename, 'w') as file_object:
        i = 0
        while ((self.length-i) != 0):
            count = 0
            file_object.write("Matches# " + str(i) + ": ")
            for j in range(self.length-i):
                if (self.text[j] == self.cmpText[j]):
                    if (self.text[j] == " "):
                        file_object.write("#")
                        count += 1
                    else:
                        file_object.write(self.text[j])
                        count += 1
            file_object.write("---Match Count: " + str(count))
            file_object.write("\n")
            self.text = self.text[1:]
            self.cmpText = self.cmpText[:-1:]
            i = 1 + i


def print_Gram_lines_(self):
    filename = 'oneGramLines.txt'
    with open(filename, 'w') as file_object:
        self.firstFive = self.cypher[0::5]
        self.firstDict = Counter(self.firstFive)
        file_object.write("Line 1: ")
```

```
for i in sorted (self.firstDict.keys()):
    l = ((i, self.firstDict[i]))
    line = str(l) + " "
    file_object.write(line)
file_object.write("\n")

print(self.firstFive + "\n")

self.secondFive = self.cypher[1::5]
self.secondDict = Counter(self.secondFive)
file_object.write("Line 2: ")
for i in sorted (self.secondDict.keys()):
    l = ((i, self.secondDict[i]))
    line = str(l) + " "
    file_object.write(line)
file_object.write("\n")

print(self.secondFive + "\n")

self.thirdFive = self.cypher[2::5]
self.thirdDict = Counter(self.thirdFive)
file_object.write("Line 3: ")
for i in sorted (self.thirdDict.keys()):
    l = ((i, self.thirdDict[i]))
    line = str(l) + " "
    file_object.write(line)
file_object.write("\n")

print(self.thirdFive + "\n")
```

```
        self.forthFive = self.cypher[3::5]
        self.forthDict = Counter(self.forthFive)
        file_object.write("Line 4: ")
        for i in sorted (self.forthDict.keys()):
            l = ((i, self.forthDict[i]))
            line = str(l) + " "
            file_object.write(line)
        file_object.write("\n")

        print(self.forthFive + "\n")

        self.fifthFive = self.cypher[4::5]
        self.fifthDict = Counter(self.fifthFive)
        file_object.write("Line 5: ")
        for i in sorted (self.fifthDict.keys()):
            l = ((i, self.fifthDict[i]))
            line = str(l) + " "
            file_object.write(line)
        file_object.write("\n")

        print(self.fifthFive + "\n")

analysis = analysis()
analysis.find_matching_chars_()
analysis.print_Gram_lines_()
```

```
cypher = "EV KLFGZXSH KMLTWDDUAZWNZTBERSSIXYZWGKTUEHPNZEHPOXTSWZBXVNDUTU FXGTL
NXYWYXALJIZSWNGXLPZKUVYUVNGNNGJ
PZFVDNKXCLOIYUTYAFYXZEMHXKWGLGAJJUVVIVOLHPOIFHHFBTY TZKZWGMKAIVNAVJYUEHGOHZGO X
GXRAGYEZBEZWNVJKWZISINKVCGETUHYXINAJAFWQGXTSSH
```

AZJZXXSSPPFCKGPNZGYAZCVHHRSSWKYMAIHKULVWHLXHGFMMJDFVSZUYNXTWHOLGEZUAZWOHYLXY
CUSAFNGG UUKVWHSSHJDUCUKBHSWNGXLPZGFSTAGHKGPNZKLWYX
LIKMSHNKULOAXZYVNKULLNSZWGZUGHAPGNAVJGFDEWYZVANKULOAULX
EIVDGXJPTUZKMSLCFCEWNUPXTATNKGETUAUPKAXYWLVV
KXCRHPOIFGXRAGYEZBEZWGHWGBGMLLNFXGTL NAUCFNXJDTIDVCDUJLM
CJLWZBXZAFMGSQZCGUOFNGGYKUVVJZCFAXRFQGXJVH AJULOAXZSLTOMLGET GYIGNAVJFN
LKXZLPZGFDEWYZVANKUKJDKGXZWZBT WVLGBXHFQGZGHFVPFWXGYXICLJFZNLJFQA DFOFSESCLL
FXGTL NAUCFJGCAXUTUWKRTTLRZSPOFN
LWUHXGPOGXGLGYSIQZULOAYZSZZNZELOFVJLWSIJLWJCYMEIOD WZISPIVFXTATNS DGHS
DKUULOZULOAULX EIVDSUFWJLXQVUSAFWM WIIEWQZVLPKTVDSUFMXJQXZSTAIBTUEYGK"

```
numChars = 2

frequency = 0

it = 1

maxChars = 10


def get_factors(x, temp):
    for i in range(1, x + 1):
        if x % i == 0:
            temp.append(i)


filename = 'repeats.txt'
with open(filename, 'w') as file_object:
    while it < maxChars:
        file_object.write("\nNumber of Characters: " + str(numChars) + "\n\n")
        for i in range(len(cypher)):
            distance = []
            for j in range(len(cypher)):
                if (cypher[i:i+numChars] == cypher[j:j+numChars]):
                    frequency = frequency + 1
                    distance.append(abs(j-i))
            if frequency > 1:
                temp = []
```

```
            tempDistance = 0
            for t in range(len(distance)):
                if abs(distance[t]) != 0:
                    tempDistance = abs(distance[t])
                    break
            get_factors(tempDistance,temp)
            file_object.write("Chars: " + cypher[i:i+numChars] + " Frequency: " + str(frequency) + " Factors:
" + str(temp) + " Distance: " + str(tempDistance) + "\n")
        frequency = 0
        if ((numChars + i) >= len(cypher)):
            break
    numChars +=1
    it +=1
```

```
from itertools import starmap, cycle


alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ "
letter_to_index = dict(zip(alphabet, range(len(alphabet))))
index_to_letter = dict(zip(range(len(alphabet)), alphabet))
key = "THXGV"
cypher = "EV KLFGZXSH KMLTWDDUAZWNZTBERSSIXYZWGKTUEHPNZEHPOXTSWZBXVNDUTU FXGTL
NXYWYXALJIZSWNGXLPZKUVYUVNGNNGJ
PZFVDNKXCLOIYUTYAFYXZEMHXKWGLGAJJUVVIVOLHPOIFHHFBTY TZKZWGMKAIVNAVJYUEHGOHZGO X
GXRAGYEZBEZWNVJKWZISINKVCGETUHYXINAJAFWQGXTSSH
AZJZXXSSPPFCKGPNZGYAZCVHHRSSWKYMAIHKULVWHLXHGFMMJDFVSZUYNXTWHOLGEZUAZWOHYLXY
CUSAFNGG UUKVWHSSHJDUCUKBHSWNGXLPZGFSTAGHKGPNZKLWYX
LIKMSHNKULOAXZYVNKULLNSZWGZUGHAPGNAVJGFDEWYZVANKULOAULX
EIVDGXJPTUZKMSLCFCEWNUPXTATNKGETUAUPKAXYWLVV
KXCRHPOIFGXRAGYEZBEZWGHWGBGMLLNFXGTL NAUCFNXJDTIDVCDUJLM
CJLWZBXZAFMGSQZCGUOFNGGYKUVVJZCFAXRFQGXJVH AJULOAXZSLTOMLGET GYIGNAVJFN
LKXZLPZGFDEWYZVANKUKJDKGXZWZBT WVLGBXHFQGZGHFVPFWXGYXICLJFZNLJFQA DFOFSESCLL
FXGTL NAUCFJGCAXUTUWKRTTLRZSPOFN
LWUHXGPOGXGLGYSIQZULOAYZSZZNZELOFVJLWSIJLWJCYMEIOD WZISPIVFXTATNS DGHS
DKUULOZULOAULX EIVDSUFWJLXQVUSAFWM WIIEWQZVLPKTVDSUFMXJQXZSTAIBTUEYGK"
```

```
def split_by_(x, text):
    out = [(text[i:i+x]) for i in range(0, len(text), x)]
    print(out)


def decrypt(input, key):
    decrypted = ""
    split_input = [input[i:i + len(key)] for i in range(0, len(input), len(key))]


    for each_split in split_input:
        i = 0
        for letter in each_split:
            number = ((letter_to_index[letter] - letter_to_index[key[i]]) % len(alphabet))
            decrypted += index_to_letter[number]
            i += 1
    return decrypted


decrypted = decrypt(cypher, key)
#split_by_(5, cypher)
print(cypher)
print("\n\n")
#split_by_(5, decrypted)
print(decrypted)
```