ITC8240 Cryptography

Homework 1: Decrypt given ciphertext.

Urmas Pitsi, 4.oct.2020 (amended 18.oct 2020)

Following is a full report of my solution to Homework 1. The report contains full source code of the solution. It is a pdf output of a fully functional Jupyter Notebook.

I decrypted the given ciphertext using Vineger cipher with the key='PRINT'. I followed these steps to arrive to the solution:

- Analyzed ciphertext to decide what kind of cipher was used for encryption.

- Guessing the key length by using the index of coincidence measure.

- Guessing letter shifts by analyzing relative frequencies of letters in the ciphertext and comparing these with the frequencies in common English.

- As frequency patterns were pretty close to common English, I was pretty lucky with my guesses. Decrypted the ciphertext.

- Derived the key from ciphertext, plaintext and key length.

- Finally I derived the key automatically. Encoded the function(s) that automatically choose shifts that produce minimal distances compared to the letter frequecy distribution of common english.

Assigned ciphertext

In [3]:

```
ciphertext = 'IYMFXHZBHTIZWALPJZRVPCTRWQPXYTIFIAWPJDVOXUTLTRKMQNEFVORRFZGXOYIIXPTWZFDEI
AWXEBRKTJBVGVLVQXGCGVGVCWTBREWGBRVBUTIKPRLDCLVXGJIEXCFBZHIZDNMTUBBKTKZRTIACFMDIMIXCDIVG
APJLMWVQEKPKQBGPCIFLTJAZXCKWSMWVLNGVVZFHUSIGMAVIAWQPBUXXIAREUZVGXGVAGKPKPRKIYMLWXJKBOTI
IFHJELEXPJWAMDICATLRGORGVIYBOZVTMWRBJAPKQGFPBMFLTEARYDIBUXBKWQHSVXRGSJWAPWRBVMLZTYFPBMF
XCJMSHGFBUXGJBBWDRVQMWRBNEAFNGATFBUXGJKNGCFBVVTKPVLIFWROTEIDNXKMOKPMMFHAUQRKBRGCKTWMEMD
ICAKPKPRKIYIAATIWVVPCTLUJKXBBCKTRLHCGQBTKZLBCXBBLIVUGATFVPHBZVTMXUMNEASGUBBJMYYIYCFPTTW
HESZUNZXEMJBIYWHMRFVGKPUQPMXFVNVXIKHFHKIAVTZVJAXTPNGPIULTACWSPWFARFTDJRKHRZRUGRDRYAVMFT
IKWCLEVMQUTWWEXIYMRGTDGZTZVANFDMMVYIYMFHAUQRKHIMNEAPIEXQIIIXIYMAMWZAFNGVTLBHEBGATFCGVDD
MNGNFNGATDENGIVLRTRYEBNAUPNOTGZRYTIZRWIYIGTACAGTCUIAWUZOUMLYIGPTYIIXWVZRMWVVVLPTIFXXEEU
BRYBUXXEBRKPTBVHCFNZTCPQAWXMQQNPCTLKPKQBGPCLRVXJQBGBRSVGVGZBVTJARLDEMCKDTMFLEVZFHAUQRKE
IWQNRVANGDLBPHBVQAMTELRWQPVBHCVUBLIRZZBTJBERIFIIHXUBUBHGZBUAVUWNHKIFVDIBRSSZLFBCTMGATPK
NGILAHTACGZTZVZRMGVIGIWPAVVPCTLBBGWFLXSTRMWVGZTZVQGXRFVBFXTIYENZUCHHJQOETKPRRHYWBMSVARK
IVZFMWVVFMPELVGVRVQYXXPGBCXQFXPTPFHAUQRKHZVQBKZLHTACGETIZWATATWHKHVWSTRKQBGPWBRKPCTOXRR
CFXIYMPHHKWSKJEVVGVZAFNGVBBUTRBYXPJBNLWZOUTHKPRVDJBBYHKILBCX'
```

Import Python libraries that we are going to use

In [4]:

```
import math
from collections import Counter
from matplotlib import pyplot as plt
from string import ascii_uppercase
```

In [5]:

```
alphabet = ascii_uppercase #list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
alphabet_to_idx = {x:i for i,x in enumerate(alphabet)}
len(alphabet)
```

Out[5]:

26

Some helper functions

In [6]:

```
def plot_bar_chart(x_values, y_values, title=None, x_label=None, y_label=None):
    fig = plt.figure()
    ax = fig.add_axes([0,0,1,1])
    ax.bar(x_values, y_values)
    if title != None: ax.set_title(title)
    if x_label != None: ax.set_xlabel(x_label)
    if y_label != None: ax.set_ylabel(y_label)
    plt.show()

def letters_and_frequencies_from_counter_dict(frequencies_dict, letters_to_check):
    letters, freqs = list(zip(*[(letter, frequencies_dict[letter]) for letter in letter
s_to_check]))
    return letters, freqs
```

Start Cryptanalysis

1. Let's analyze the given ciphertext

In [7]:

```
frequencies_in_ciphertext = Counter(ciphertext)
```

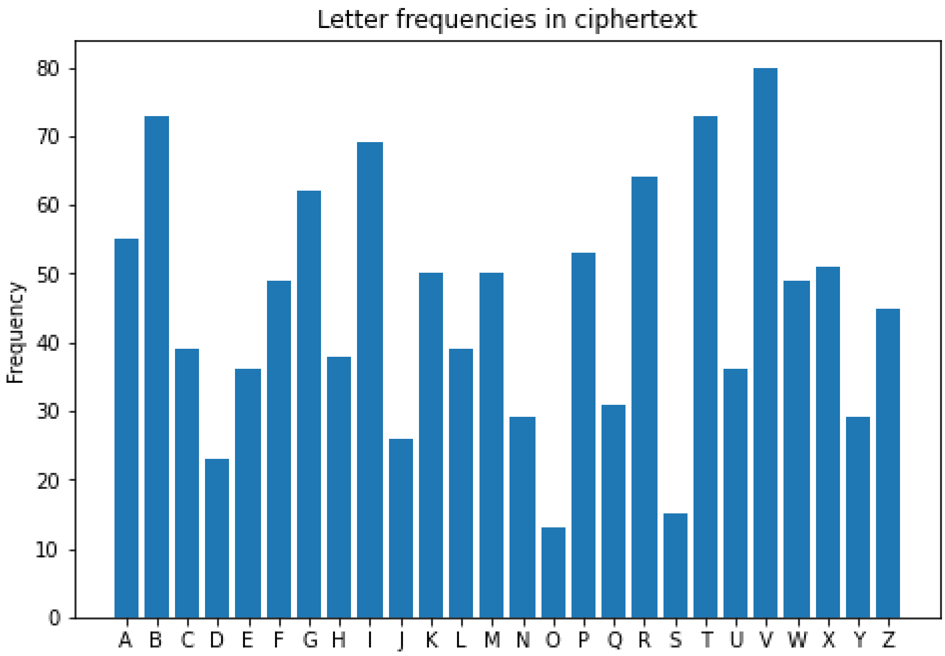
In [8]:

```
#letters, freqs = frequencies_in_ciphertext.keys(), frequencies_in_ciphertext.values()
letters, freqs = list(zip(*[(letter, frequencies_in_ciphertext[letter]) for letter in a
lphabet])) # Sorted alphabetically
assert len(ciphertext) == sum(freqs)
print('Length of ciphertext:' , sum(freqs))
```

```
Length of ciphertext: 1177
```

In [9]:

```
plot_bar_chart(letters, freqs, title='Letter frequencies in ciphertext', y_label='Frequ
ency')
```



1.1. English letter frequencies

In [25]:

```
# http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html
common_letter_frequencies = {'A':8.12, 'B':1.49, 'C':2.71, 'D':4.32, 'E':12.02, 'F':2.3
, 'G':2.03, 'H':5.92,
                        'I':7.31, 'J':0.1, 'K':0.69, 'L':3.98, 'M':2.61, 'N':6.95, 'O':
7.68, 'P':1.82,
                        'Q':0.11, 'R':6.02, 'S':6.28, 'T':9.1, 'U':2.88, 'V':1.11, 'W':
2.09, 'X':0.17,
                        'Y':2.11, 'Z':0.07}
```

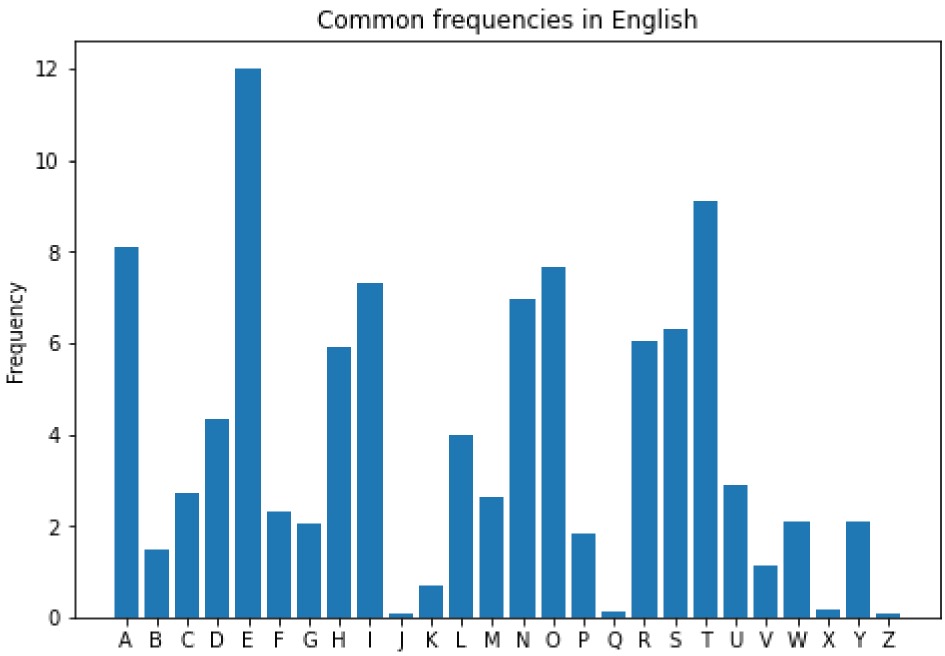
In [26]:

```
common_letters, common_freqs = common_letter_frequencies.keys(), common_letter_frequenc
ies.values()
print('Sum of frequencies:', round(sum(common_freqs), 1))
```

```
Sum of frequencies: 100.0
```

In [27]:

```
plot_bar_chart(common_letters, common_freqs, title='Common frequencies in English', y_l
abel='Frequency')
```



2. Figure out whether monoalapbetic or polyalpabetic cipher

2.1. Calculate Indices of Coincidence. IC = sum((f *(f - 1)) / (N* (N - 1)))

In [13]:

```
def index_of_coincidence(frequencies_dict, length_of_ciphertext):
    N_times_Nminus1 = length_of_ciphertext * (length_of_ciphertext - 1)
    return sum([v * (v - 1) / N_times_Nminus1 for _,v in frequencies_dict.items()])
```

In [14]:

```
index_of_coincidence(frequencies_in_ciphertext, len(ciphertext))
```

Out[14]:

```
0.043466324507713025
```

- IC value 0.04, which is quite low suggests that we might be dealing with polyalphabetic cipher. Let's try Vigenere cipher.

3. Vigenere Cipher

3.1. Find key length

- First we split ciphertext consecutively into pieces of length in some range, eg from key length=2 until key length=15

In [15]:

```
def compute_key_length(cipher, min_key_len=2, max_key_len=10, verbose=True):
    res = {}
    for key_length in range(min_key_len, max_key_len + 1):
        sub_cipher = cipher[::key_length]
        total = index_of_coincidence(Counter(sub_cipher), len(sub_cipher))
        res[key_length] = total
        if verbose:
            print(f'key length:{key_length}, Index of Coincidence:{round(total,3)}')
    return res
```

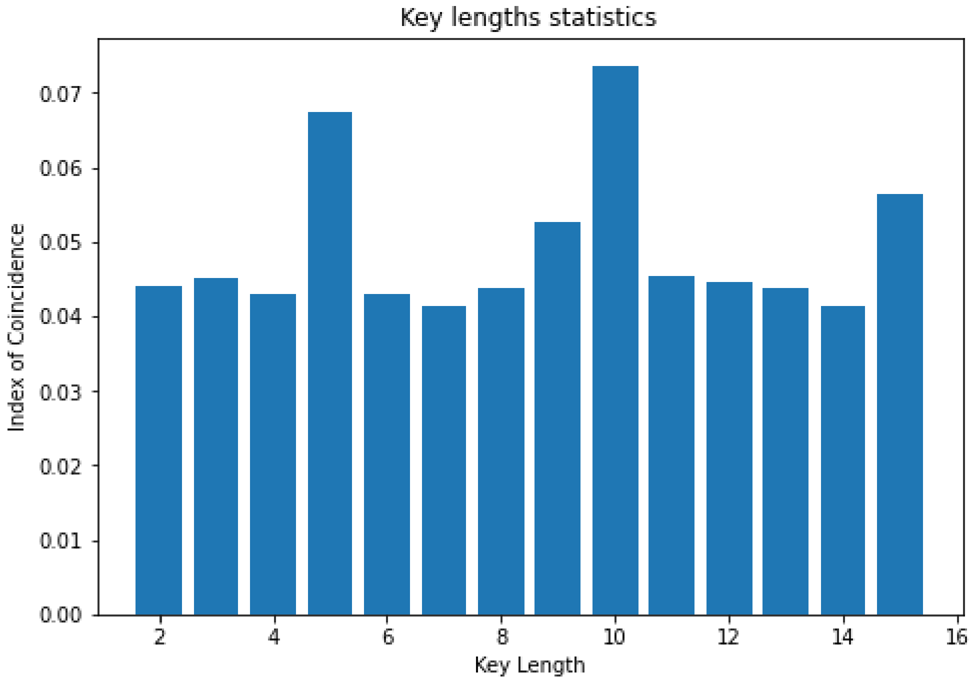
In [16]:

```
key_lengths_statistics = compute_key_length(ciphertext, min_key_len=2, max_key_len=15,
verbose=True)
```

```
key length:2, Index of Coincidence:0.044
key length:3, Index of Coincidence:0.045
key length:4, Index of Coincidence:0.043
key length:5, Index of Coincidence:0.067
key length:6, Index of Coincidence:0.043
key length:7, Index of Coincidence:0.041
key length:8, Index of Coincidence:0.044
key length:9, Index of Coincidence:0.053
key length:10, Index of Coincidence:0.074
key length:11, Index of Coincidence:0.045
key length:12, Index of Coincidence:0.045
key length:13, Index of Coincidence:0.044
key length:14, Index of Coincidence:0.041
key length:15, Index of Coincidence:0.056
```

In [17]:

```
plot_bar_chart(key_lengths_statistics.keys(),
               key_lengths_statistics.values(),
               title='Key lengths statistics',
               x_label='Key Length', y_label='Index of Coincidence')
```



- As we can see from the numbers and chart: the key length is probably 5, because we see higher values of IC occurring after 5 steps.

3.2. Let's try to find correct key using relative letter frequencies in common English

Our alphabet with indexes

In [18]:

```
[(i,a) for i,a in enumerate(alphabet)]
```

Out[18]:

```
[(0, 'A'),
 (1, 'B'),
 (2, 'C'),
 (3, 'D'),
 (4, 'E'),
 (5, 'F'),
 (6, 'G'),
 (7, 'H'),
 (8, 'I'),
 (9, 'J'),
 (10, 'K'),
 (11, 'L'),
 (12, 'M'),
 (13, 'N'),
 (14, 'O'),
 (15, 'P'),
 (16, 'Q'),
 (17, 'R'),
 (18, 'S'),
 (19, 'T'),
 (20, 'U'),
 (21, 'V'),
 (22, 'W'),
 (23, 'X'),
 (24, 'Y'),
 (25, 'Z')]
```

Calculate sub parts of the ciphertext with the step size 5, ie key length

In [19]:

```
key_length = 5
sub_cipher1 = ciphertext[::key_length]  # Starting from position 0 with step size 5.
sub_cipher2 = ciphertext[1::key_length] # Starting from position 1 with step size 5.
sub_cipher3 = ciphertext[2::key_length] # Starting from position 2 with step size 5.
sub_cipher4 = ciphertext[3::key_length] # Starting from position 3 with step size 5.
sub_cipher5 = ciphertext[4::key_length] # Starting from position 4 with step size 5.
```

Just in case let's see index of coincidence values for each slice. If everything is correct then all these numbers must be high.

In [20]:

```
for i in range(1, key_length + 1):
    sub_cipher = ciphertext[i::key_length]
    ic = index_of_coincidence(Counter(sub_cipher), len(sub_cipher))
    print(i, ic)
```

```
1 0.06473133790119004
2 0.069721767594108
3 0.06764866339334422
4 0.06408437897799601
5 0.06713947990543734
```

- Seems everything is good! Pretty high IC scores.

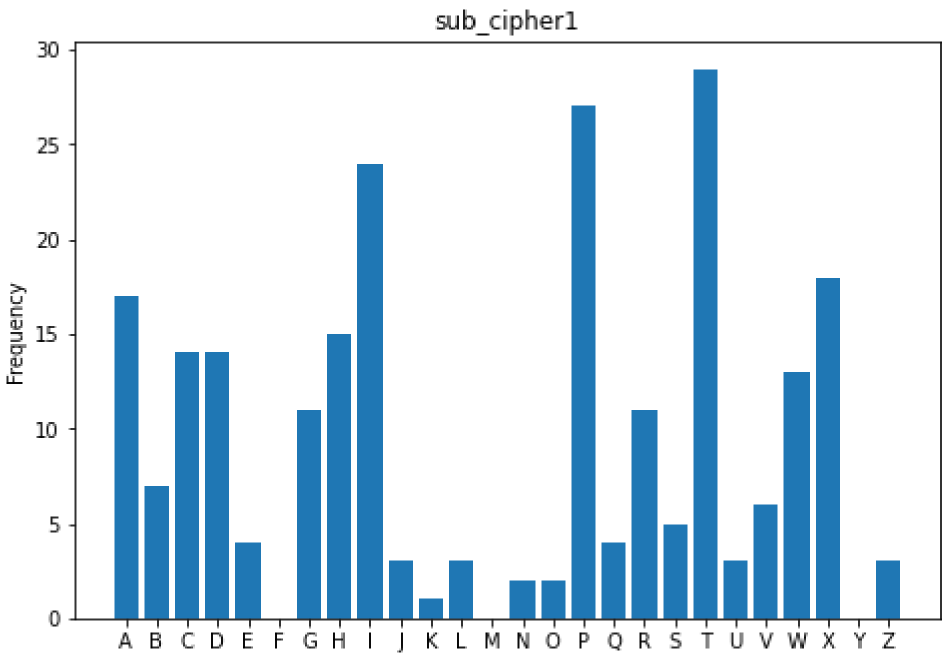
Now let's observe relative frequences of letters for each sub-part that we sliced

In [21]:

```
L, F = letters_and_frequencies_from_counter_dict(Counter(sub_cipher1), alphabet)
```

In [223]:

```
plot_bar_chart(L, F, title='sub_cipher1', y_label='Frequency')
```



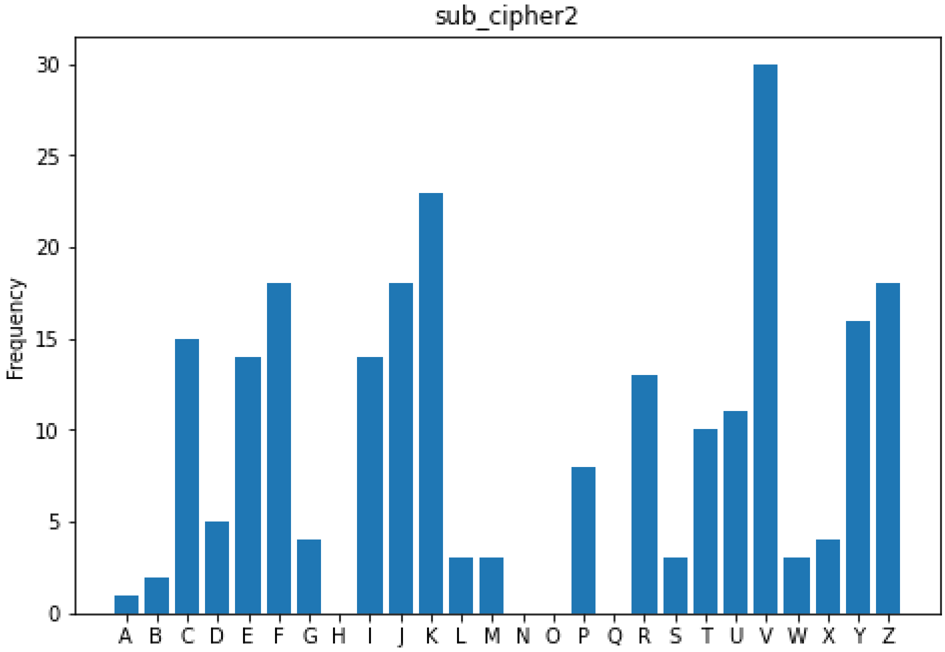
Based on frequencies we can make a guess that P corresponds to A, T->E: shift=-15 => 11 mod 26

In [224]:

```
L, F = letters_and_frequencies_from_counter_dict(Counter(sub_cipher2), alphabet)
```

In [225]:

```
plot_bar_chart(L, F, title='sub_cipher2', y_label='Frequency')
```



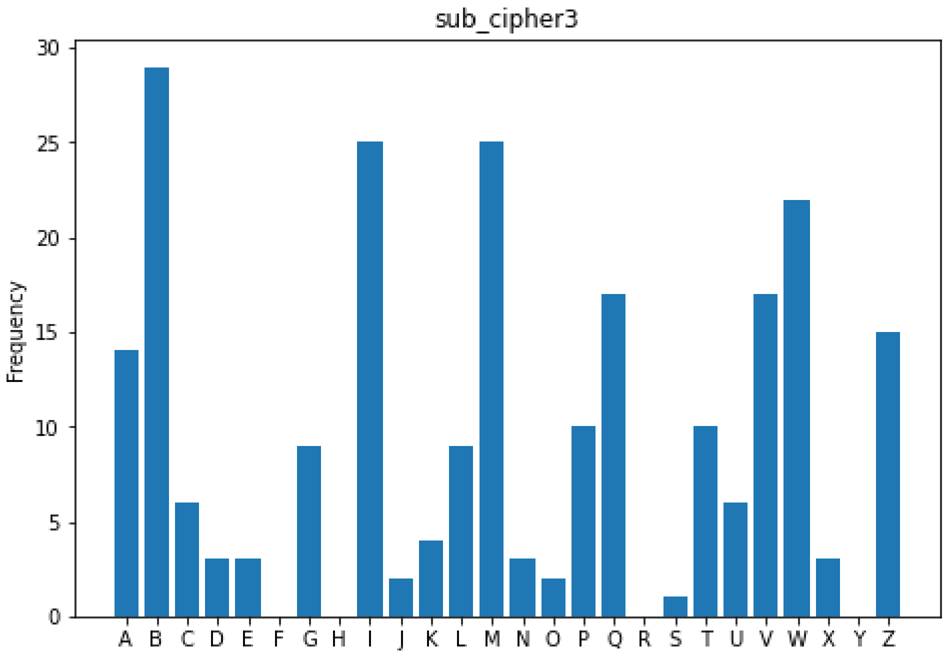
Based on frequencies we can make a guess that R corresponds to A, V->E: shift=-17 => 9 mod 26

In [226]:

```
L, F = letters_and_frequencies_from_counter_dict(Counter(sub_cipher3), alphabet)
```

In [227]:

```
plot_bar_chart(L, F, title='sub_cipher3', y_label='Frequency')
```



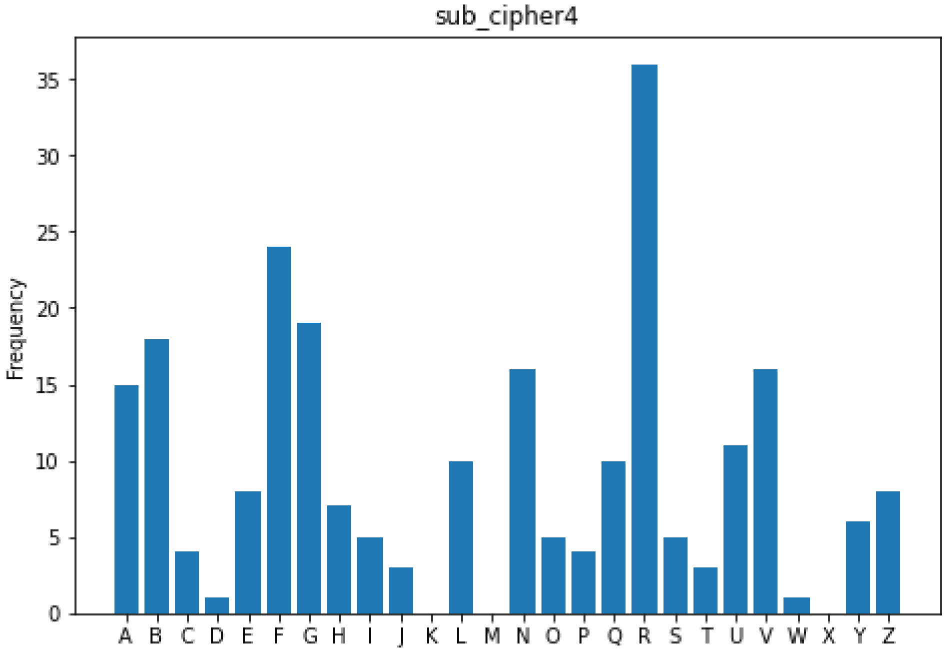
Based on frequencies we can make a guess that I corresponds to A, M->E: shift=-8 => 18 mod 26

In [228]:

```
L, F = letters_and_frequencies_from_counter_dict(Counter(sub_cipher4), alphabet)
```

In [229]:

```
plot_bar_chart(L, F, title='sub_cipher4', y_label='Frequency')
```



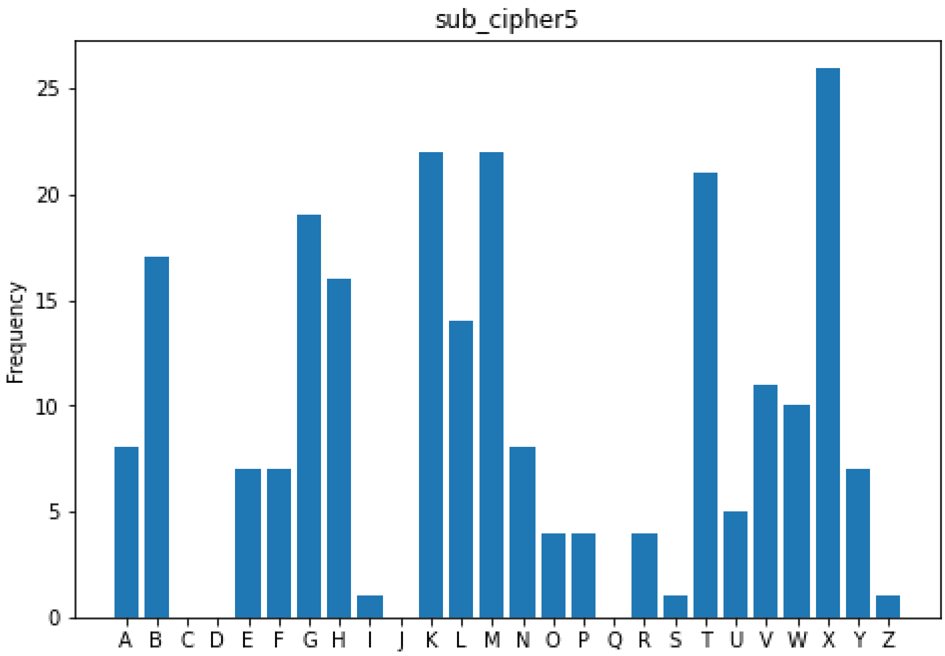
Based on frequencies we can make a guess that N corresponds to A, R->E: shift=-13 => 13 mod 26

In [230]:

```
L, F = letters_and_frequencies_from_counter_dict(Counter(sub_cipher5), alphabet)
```

In [231]:

```
plot_bar_chart(L, F, title='sub_cipher5', y_label='Frequency')
```



Based on frequencies we can make a guess that T corresponds to A, X->E: shift=-19 => 7 mod 26

- So we can try the key with following shift sizes: 11,9,18,13,7

Let's try our guessed shifts

In [265]:

```
ciphertext, len(ciphertext)
```

Out[265]:

```
('IYMFXHZBHTIZWALPJZRVPCTRWQPXYTIFIAWPJDVOXUTLTRKMQNEFVORRFZGXOYIIXPTWZFDE
IAWXEBRKTJBVGVLVQXGCGVGVCWTBREWGBRVBUTIKPRLDCLVXGJIEXCFBZHIZDNMTUBBKTKZRTI
ACFMDIMIXCDIVGAPJLMWVQEKPKQBGPCIFLTJAZXCKWSMWVLNGVVZFHUSIGMAVIAWQPBUXXIARE
UZVGXGVAGKPKPRKIYMLWXJKBOTIIFHJELEXPJWAMDICATLRGORGVIYBOZVTMWRBJAPKQGFPBMF
LTEARYDIBUXBKWQHSVXRGSJWAPWRBVMLZTYFPBMFXCJMSHGFBUXGJBBWDRVQMWRBNEAFNGATFB
UXGJKNGCFBVVTKPVLIFWROTEIDNXKMOKPMMFHAUQRKBRGCKTWMEMDICAKPKPRKIYIAATIWVVPC
TLUJKXBBCKTRLHCGQBTKZLBCXBBLIVUGATFVPHBZVTMXUMNEASGUBBJMYYIYCFPTTWHESZUNZX
EMJBIYWHMRFVGKPUQPMXFVNVXIKHFHKIAVTZVJAXTPNGPIULTACWSPWFARFTDJRKHRZRUGRDRY
AVMFTIKWCLEVMQUTWWEXIYMRGTDGZTZVANFDMMVYIYMFHAUQRKHIMNEAPIEXQIIIXIYMAMWZAF
NGVTLBHEBGATFCGVDDMNGNFNGATDENGIVLRTRYEBNAUPNOTGZRYTIZRWIYIGTACAGTCUIAWUZO
UMLYIGPTYIIXWVZRMWVVVLPTIFXXEEUBRYBUXXEBRKPTBVHCFNZTCPQAWXMQQNPCTLKPKQBGPC
LRVXJQBGBRSVGVGZBVTJARLDEMCKDTMFLEVZFHAUQRKEIWQNRVANGDLBPHBVQAMTELRWQPVBHC
VUBLIRZZBTJBERIFIIHXUBUBHGZBUAVUWNHKIFVDIBRSSZLFBCTMGATPKNGILAHTACGZTZVZRM
GVIGIWPAVVPCTLBBGWFLXSTRMWVGZTZVQGXRFVBFXTIYENZUCHHJQOETKPRRHYWBMSVARKIVZF
MWVVFMPELVGVRVQYXXPGBCXQFXPTPFHAUQRKHZVQBKZLHTACGETIZWATATWHKHVWSTRKQBGPWB
RKPCTOXRRCFXIYMPHHKWSKJEVVGVZAFNGVBBUTRBYXPJBNLWZOUTHKPRVDJBBYHKILBCX',
 1177)
```

In [269]:

```
letter_shifts = [11,9,18,13,7] * (len(ciphertext) // key_length + 1)
```

- We shift each letter in ciphertext by [11,9,18,13,7], looping these values until the end of ciphertext

In [270]:

```
plaintext = ''.join([alphabet[(alphabet_to_idx[c] + letter_shifts[i]) % len(alphabet)]
for i,c in enumerate(list(ciphertext))])
plaintext, len(plaintext)
```

Out[270]:

```
('THESESITUATIONSASRECALLEDBYPLATOANDASVIVIDLYACTEDUPONBYCORTEZHAVEACOMMON
ANDINTERESTINGUNDERLYINGLOGICNOTICETHATTHESOLDIERSARENOTMOTIVATEDTORETREAT
JUSTOREVENMAINLYBYTHEIRRATIONALASSESSMENTOFTHEDANGERSOFBATTLEANDBYTHEIRSEL
FINTERESTRATHERTHEYDISCOVERASOUNDREASONTORUNAWAYBYREALIZINGTHATWHATITMAKES
SENSEFORTHEMTODODEPENDSONWHATITWILLMAKESENSEFOROTHERSTODOANDTHATALLOFTHEOT
HERSCANNOTICETHISTOOEVENAQUITEBRAVESOLDIERMAYPREFERTORUNRATHERTHANHEROICAL
LYBUTPOINTLESSLYDIETRYINGTOSTEMTHEONCOMINGTIDEALLBYHIMSELFTHUSWECOULDIMAGI
NEWITHOUTCONTRADICTIONACIRCUMSTANCEINWHICHANARMYALLOFWHOSEMEMBERSAREBRAVEF
LEESATTOPSPEEDBEFORETHEENEMYMAKESAMOVEIFTHESOLDIERSREALLYAREBRAVETHENTHISS
URELYISNTTHEOUTCOMEANYOFTHEMWANTEDEACHWOULDHAVEPREFERREDTHATALLSTANDANDFIG
HTWHATWEHAVEHERETHENISACASEINWHICHTHEINTERACTIONOFMANYINDIVIDUALLYRATIONAL
DECISIONMAKINGPROCESSESONEPROCESSPERSOLDIERPRODUCESANOUTCOMEINTENDEDBYNOON
EMOSTARMIESTRYTOAVOIDTHISPROBLEMJUSTASCORTEZDIDSINCETHEYCANTUSUALLYMAKERET
REATPHYSICALLYIMPOSSIBLETHEYMAKEITECONOMICALLYIMPOSSIBLETHEYSHOOTDESERTERS
THENSTANDINGANDFIGHTINGISEACHSOLDIERSINDIVIDUALLYRATIONALCOURSEOFACTIONAFT
ERALLBECAUSETHECOSTOFRUNNINGISSURETOBEATLEASTASHIGHASTHECOSTOFSTAYING',
 1177)
```

- Pretty amazing! Seems to decrypt perfectly! No we have to convert the shift values into alphabet letters to obtain key string.

So, what is the encryption key?

- We shifted letters in ciphertext = 'IYMFX...' by (11,9,18,13,7,...) and got plaintext = 'THESE...'

- Now we have to recover the key string that corresponds to those shifts

In [285]:

```
cipher_idx = [alphabet_to_idx[c] for c in ciphertext[:key_length]] # Lookup letter inde
xes for cipher text
plain_idx = [alphabet_to_idx[c] for c in plaintext[:key_length]]   # Lookup letter inde
xes for plain text
diff_mod26 = [diff % len(alphabet) for diff in [c - p for c,p in zip(cipher_idx, plain_
idx)]] # Calculate difference mod 26
key = ''.join([alphabet[d] for d in diff_mod26])
print(f'Decryption key = {key}')
```

```
Decryption key = PRINT
```

Results

In [287]:

```
plaintext
```

Out[287]:

```
'THESESITUATIONSASRECALLEDBYPLATOANDASVIVIDLYACTEDUPONBYCORTEZHAVEACOMMONA
NDINTERESTINGUNDERLYINGLOGICNOTICETHATTHESOLDIERSARENOTMOTIVATEDTORETREATJ
USTOREVENMAINLYBYTHEIRRATIONALASSESSMENTOFTHEDANGERSOFBATTLEANDBYTHEIRSELF
INTERESTRATHERTHEYDISCOVERASOUNDREASONTORUNAWAYBYREALIZINGTHATWHATITMAKESS
ENSEFORTHEMTODODEPENDSONWHATITWILLMAKESENSEFOROTHERSTODOANDTHATALLOFTHEOTH
ERSCANNOTICETHISTOOEVENAQUITEBRAVESOLDIERMAYPREFERTORUNRATHERTHANHEROICALL
YBUTPOINTLESSLYDIETRYINGTOSTEMTHEONCOMINGTIDEALLBYHIMSELFTHUSWECOULDIMAGIN
EWITHOUTCONTRADICTIONACIRCUMSTANCEINWHICHANARMYALLOFWHOSEMEMBERSAREBRAVEFL
EESATTOPSPEEDBEFORETHEENEMYMAKESAMOVEIFTHESOLDIERSREALLYAREBRAVETHENTHISSU
RELYISNTTHEOUTCOMEANYOFTHEMWANTEDEACHWOULDHAVEPREFERREDTHATALLSTANDANDFIGH
TWHATWEHAVEHERETHENISACASEINWHICHTHEINTERACTIONOFMANYINDIVIDUALLYRATIONALD
ECISIONMAKINGPROCESSESONEPROCESSPERSOLDIERPRODUCESANOUTCOMEINTENDEDBYNOONE
MOSTARMIESTRYTOAVOIDTHISPROBLEMJUSTASCORTEZDIDSINCETHEYCANTUSUALLYMAKERETR
EATPHYSICALLYIMPOSSIBLETHEYMAKEITECONOMICALLYIMPOSSIBLETHEYSHOOTDESERTERST
HENSTANDINGANDFIGHTINGISEACHSOLDIERSINDIVIDUALLYRATIONALCOURSEOFACTIONAFTE
RALLBECAUSETHECOSTOFRUNNINGISSURETOBEATLEASTASHIGHASTHECOSTOFSTAYING'
```

Decryption key = PRINT

Given ciphertext:

In [291]:

```
ciphertext
```

Out[291]:

```
'IYMFXHZBHTIZWALPJZRVPCTRWQPXYTIFIAWPJDVOXUTLTRKMQNEFVORRFZGXOYIIXPTWZFDEI
AWXEBRKTJBVGVLVQXGCGVGVCWTBREWGBRVBUTIKPRLDCLVXGJIEXCFBZHIZDNMTUBBKTKZRTIA
CFMDIMIXCDIVGAPJLMWVQEKPKQBGPCIFLTJAZXCKWSMWVLNGVVZFHUSIGMAVIAWQPBUXXIAREU
ZVGXGVAGKPKPRKIYMLWXJKBOTIIFHJELEXPJWAMDICATLRGORGVIYBOZVTMWRBJAPKQGFPBMFL
TEARYDIBUXBKWQHSVXRGSJWAPWRBVMLZTYFPBMFXCJMSHGFBUXGJBBWDRVQMWRBNEAFNGATFBU
XGJKNGCFBVVTKPVLIFWROTEIDNXKMOKPMMFHAUQRKBRGCKTWMEMDICAKPKPRKIYIAATIWVVPCT
LUJKXBBCKTRLHCGQBTKZLBCXBBLIVUGATFVPHBZVTMXUMNEASGUBBJMYYIYCFPTTWHESZUNZXE
MJBIYWHMRFVGKPUQPMXFVNVXIKHFHKIAVTZVJAXTPNGPIULTACWSPWFARFTDJRKHRZRUGRDRYA
VMFTIKWCLEVMQUTWWEXIYMRGTDGZTZVANFDMMVYIYMFHAUQRKHIMNEAPIEXQIIIXIYMAMWZAFN
GVTLBHEBGATFCGVDDMNGNFNGATDENGIVLRTRYEBNAUPNOTGZRYTIZRWIYIGTACAGTCUIAWUZOU
MLYIGPTYIIXWVZRMWVVVLPTIFXXEEUBRYBUXXEBRKPTBVHCFNZTCPQAWXMQQNPCTLKPKQBGPCL
RVXJQBGBRSVGVGZBVTJARLDEMCKDTMFLEVZFHAUQRKEIWQNRVANGDLBPHBVQAMTELRWQPVBHCV
UBLIRZZBTJBERIFIIHXUBUBHGZBUAVUWNHKIFVDIBRSSZLFBCTMGATPKNGILAHTACGZTZVZRMG
VIGIWPAVVPCTLBBGWFLXSTRMWVGZTZVQGXRFVBFXTIYENZUCHHJQOETKPRRHYWBMSVARKIVZFM
WVVFMPELVGVRVQYXXPGBCXQFXPTPFHAUQRKHZVQBKZLHTACGETIZWATATWHKHVWSTRKQBGPWBR
KPCTOXRRCFXIYMPHHKWSKJEVVGVZAFNGVBBUTRBYXPJBNLWZOUTHKPRVDJBBYHKILBCX'
```

Let's try to get results with automatic deciphering

First we define some helper functions that help us to do automatically what we did above using visual inspection.
Instead of visually trying to detect the shift size we calculate the differencies for each shift with common english frequencies.
We choose the shift that produces minimal difference as the most probable shift for that particular letter in the key.
As the key length is 5 letters, we do it 5 times. Finding minimal shift for each letter in the key.

In [42]:

```
def shift_text(input_text, shift_size=1):
    if shift_size == 0:
        return input_text
    else:
        n = len(alphabet)
        return ''.join(alphabet[(alphabet_to_idx[s] + shift_size) % n] for s in input_t
ext)
```

In [43]:

```
def normalize_frequencies(frequencies_dict, upper_bound=100.0):
    total = sum(frequencies_dict.values())
    return {k: v / total * upper_bound for k,v in frequencies_dict.items()}
```

In [44]:

```
def frequency_distribution_distance(frequencies_dict1, frequencies_dict2):
    keys = frequencies_dict1.keys()
    return sum([abs(frequencies_dict1[k] - frequencies_dict2[k]) for k in keys if k in
frequencies_dict2])
```

In [45]:

```
def text_rotations(input_text):
    ''' Return dictionary of shift_size: shifted_text'''
    return {i: shift_text(input_text, shift_size=i) for i in range(0, len(alphabet))}
```

In [48]:

```
def all_rotated_distances(input_text, comparison_frequencies_dict):
    rotations = text_rotations(input_text)
    res = {}
    for shift_size, shifted_text in rotations.items():
        res[shift_size] = frequency_distribution_distance(normalize_frequencies(Counter
(shifted_text), upper_bound=100.0), common_letter_frequencies)
    return res
```

In [57]:

```
def min_value_element_from_dict(input_dict):
    return sorted((v,k) for k,v in input_dict.items())[0][1]
```

In [72]:

```
def decrypt_vigenere(cipher_text, shift_sizes):
    shifts = shift_sizes * (len(cipher_text) // len(shift_sizes) + 1)
    return ''.join([alphabet[(alphabet_to_idx[s] + shifts[i]) % len(alphabet)] for i,s
in enumerate(cipher_text)])
```

In [85]:

```
def translate_to_key(cipher_text, shift_sizes):
    decrypted_text = decrypt_vigenere(cipher_text[:len(shift_sizes)], shift_sizes)
    return ''.join([alphabet[(alphabet_to_idx[cipher_text[i]] - alphabet_to_idx[s]) % l
en(alphabet)] for i,s in enumerate(decrypted_text)])
```

These are our minimal shifts for each letter in the key: [11, 9, 18, 13, 7]

In [74]:

```
min_distance_shift_sizes = [min_value_element_from_dict(all_rotated_distances(txt, comm
on_letter_frequencies))
                            for txt in [sub_cipher1, sub_cipher2, sub_cipher3, sub_ciph
er4, sub_cipher5]]
min_distance_shift_sizes
```

Out[74]:

```
[11, 9, 18, 13, 7]
```

Let's decrypt the ciphertext using suggested key (in shifts format)

In [75]:

```
plaintext = decrypt_vigenere(ciphertext, min_distance_shift_sizes)
plaintext
```

Out[75]:

```
'THESESITUATIONSASRECALLEDBYPLATOANDASVIVIDLYACTEDUPONBYCORTEZHAVEACOMMONA
NDINTERESTINGUNDERLYINGLOGICNOTICETHATTHESOLDIERSARENOTMOTIVATEDTORETREATJ
USTOREVENMAINLYBYTHEIRRATIONALASSESSMENTOFTHEDANGERSOFBATTLEANDBYTHEIRSELF
INTERESTRATHERTHEYDISCOVERASOUNDREASONTORUNAWAYBYREALIZINGTHATWHATITMAKESS
ENSEFORTHEMTODODEPENDSONWHATITWILLMAKESENSEFOROTHERSTODOANDTHATALLOFTHEOTH
ERSCANNOTICETHISTOOEVENAQUITEBRAVESOLDIERMAYPREFERTORUNRATHERTHANHEROICALL
YBUTPOINTLESSLYDIETRYINGTOSTEMTHEONCOMINGTIDEALLBYHIMSELFTHUSWECOULDIMAGIN
EWITHOUTCONTRADICTIONACIRCUMSTANCEINWHICHANARMYALLOFWHOSEMEMBERSAREBRAVEFL
EESATTOPSPEEDBEFORETHEENEMYMAKESAMOVEIFTHESOLDIERSREALLYAREBRAVETHENTHISSU
RELYISNTTHEOUTCOMEANYOFTHEMWANTEDEACHWOULDHAVEPREFERREDTHATALLSTANDANDFIGH
TWHATWEHAVEHERETHENISACASEINWHICHTHEINTERACTIONOFMANYINDIVIDUALLYRATIONALD
ECISIONMAKINGPROCESSESONEPROCESSPERSOLDIERPRODUCESANOUTCOMEINTENDEDBYNOONE
MOSTARMIESTRYTOAVOIDTHISPROBLEMJUSTASCORTEZDIDSINCETHEYCANTUSUALLYMAKERETR
EATPHYSICALLYIMPOSSIBLETHEYMAKEITECONOMICALLYIMPOSSIBLETHEYSHOOTDESERTERST
HENSTANDINGANDFIGHTINGISEACHSOLDIERSINDIVIDUALLYRATIONALCOURSEOFACTIONAFTE
RALLBECAUSETHECOSTOFRUNNINGISSURETOBEATLEASTASHIGHASTHECOSTOFSTAYING'
```

Result seems to be understandable english text. So let's translate shifts into plaintext key string

In [86]:

```
translate_to_key(ciphertext, min_distance_shift_sizes)
```

Out[86]:

```
'PRINT'
```

Results of automatic deciphering

Key = 'PRINT'
Using automatic deciphering we found key 'PRINT' which is exactly the same as we found previously by visual inspection.

Resulting plaintext =

In [1]:

```
plaintext
```

Out[1]:

```
'THESESITUATIONSASRECALLEDBYPLATOANDASVIVIDLYACTEDUPONBYCORTEZHAVEACOMMONA
NDINTERESTINGUNDERLYINGLOGICNOTICETHATTHESOLDIERSARENOTMOTIVATEDTORETREATJ
USTOREVENMAINLYBYTHEIRRATIONALASSESSMENTOFTHEDANGERSOFBATTLEANDBYTHEIRSELF
INTERESTRATHERTHEYDISCOVERASOUNDREASONTORUNAWAYBYREALIZINGTHATWHATITMAKESS
ENSEFORTHEMTODODEPENDSONWHATITWILLMAKESENSEFOROTHERSTODOANDTHATALLOFTHEOTH
ERSCANNOTICETHISTOOEVENAQUITEBRAVESOLDIERMAYPREFERTORUNRATHERTHANHEROICALL
YBUTPOINTLESSLYDIETRYINGTOSTEMTHEONCOMINGTIDEALLBYHIMSELFTHUSWECOULDIMAGIN
EWITHOUTCONTRADICTIONACIRCUMSTANCEINWHICHANARMYALLOFWHOSEMEMBERSAREBRAVEFL
EESATTOPSPEEDBEFORETHEENEMYMAKESAMOVEIFTHESOLDIERSREALLYAREBRAVETHENTHISSU
RELYISNTTHEOUTCOMEANYOFTHEMWANTEDEACHWOULDHAVEPREFERREDTHATALLSTANDANDFIGH
TWHATWEHAVEHERETHENISACASEINWHICHTHEINTERACTIONOFMANYINDIVIDUALLYRATIONALD
ECISIONMAKINGPROCESSESONEPROCESSPERSOLDIERPRODUCESANOUTCOMEINTENDEDBYNOONE
MOSTARMIESTRYTOAVOIDTHISPROBLEMJUSTASCORTEZDIDSINCETHEYCANTUSUALLYMAKERETR
EATPHYSICALLYIMPOSSIBLETHEYMAKEITECONOMICALLYIMPOSSIBLETHEYSHOOTDESERTERST
HENSTANDINGANDFIGHTINGISEACHSOLDIERSINDIVIDUALLYRATIONALCOURSEOFACTIONAFTE
RALLBECAUSETHECOSTOFRUNNINGISSURETOBEATLEASTASHIGHASTHECOSTOFSTAYING'
```

External resources

- [1] https://gist.github.com/dssstr/aedbb5e9f2185f366c6d6b50fad3e4a4 (https://gist.github.com/dssstr/aedbb5e9f2185f366c6d6b50fad3e4a4)

- [2] https://github.com/ferreirafabio/vigenere-py (https://github.com/ferreirafabio/vigenere-py)

- [3] http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html (http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html)

In [292]:

```python
# Source: https://gist.github.com/dssstr/aedbb5e9f2185f366c6d6b50fad3e4a4
def encrypt(plaintext, key):
    key_length = len(key)
    key_as_int = [ord(i) for i in key]
    plaintext_int = [ord(i) for i in plaintext]
    ciphertext = ''
    for i in range(len(plaintext_int)):
        value = (plaintext_int[i] + key_as_int[i % key_length]) % 26
        ciphertext += chr(value + 65)
    return ciphertext

def decrypt(ciphertext, key):
    key_length = len(key)
    key_as_int = [ord(i) for i in key]
    ciphertext_int = [ord(i) for i in ciphertext]
    plaintext = ''
    for i in range(len(ciphertext_int)):
        value = (ciphertext_int[i] - key_as_int[i % key_length]) % 26
        plaintext += chr(value + 65)
    return plaintext
```

Verify correctness of encryption-decryption

In [78]:

```python
txt1, key1 = 'ALDKQORJDSDCNVHAPFQORIYVBCXYRNVBGDLMVSACMCMD', 'GOODKEY'
assert txt1 == decrypt(encrypt(txt1, key1), key1)
print('Encryption-Decryption round trip is correct!')
```

```
Encryption-Decryption round trip is correct!
```

In [5]:

```python
key = 'PRINT'
```

In [6]:

```python
decrypt(ciphertext, key)
```

Out[6]:

```
'THESESITUATIONSASRECALLEDBYPLATOANDASVIVIDLYACTEDUPONBYCORTEZHAVEACOMMONA
NDINTERESTINGUNDERLYINGLOGICNOTICETHATTHESOLDIERSARENOTMOTIVATEDTORETREATJ
USTOREVENMAINLYBYTHEIRRATIONALASSESSMENTOFTHEDANGERSOFBATTLEANDBYTHEIRSELF
INTERESTRATHERTHEYDISCOVERASOUNDREASONTORUNAWAYBYREALIZINGTHATWHATITMAKESS
ENSEFORTHEMTODODEPENDSONWHATITWILLMAKESENSEFOROTHERSTODOANDTHATALLOFTHEOTH
ERSCANNOTICETHISTOOEVENAQUITEBRAVESOLDIERMAYPREFERTORUNRATHERTHANHEROICALL
YBUTPOINTLESSLYDIETRYINGTOSTEMTHEONCOMINGTIDEALLBYHIMSELFTHUSWECOULDIMAGIN
EWITHOUTCONTRADICTIONACIRCUMSTANCEINWHICHANARMYALLOFWHOSEMEMBERSAREBRAVEFL
EESATTOPSPEEDBEFORETHEENEMYMAKESAMOVEIFTHESOLDIERSREALLYAREBRAVETHENTHISSU
RELYISNTTHEOUTCOMEANYOFTHEMWANTEDEACHWOULDHAVEPREFERREDTHATALLSTANDANDFIGH
TWHATWEHAVEHERETHENISACASEINWHICHTHEINTERACTIONOFMANYINDIVIDUALLYRATIONALD
ECISIONMAKINGPROCESSESONEPROCESSPERSOLDIERPRODUCESANOUTCOMEINTENDEDBYNOONE
MOSTARMIESTRYTOAVOIDTHISPROBLEMJUSTASCORTEZDIDSINCETHEYCANTUSUALLYMAKERETR
EATPHYSICALLYIMPOSSIBLETHEYMAKEITECONOMICALLYIMPOSSIBLETHEYSHOOTDESERTERST
HENSTANDINGANDFIGHTINGISEACHSOLDIERSINDIVIDUALLYRATIONALCOURSEOFACTIONAFTE
RALLBECAUSETHECOSTOFRUNNINGISSURETOBEATLEASTASHIGHASTHECOSTOFSTAYING'
```