Programming Assignment

For the partial fulfillment of the course

BITS F463 - CRYPTOGRAPHY

BITS Pilani, Pilani Campus

BITS Pilani

Department of Computer Science & Information Systems

Submitted By:

Rohit Lodha - 2015A7PS0040P

Index

Decrypted Plaintext

could a machine communicate with humans on an unlimited set of topics through fluent use of human language?
could a language using machine give the appearance of understanding sentences and coming up with ideas while
in truth being as devoid of thoughts and as empty inside as a nineteenth century adding machine or a
twentieth century word processor? how might we distinguish between a genuinely conscious and intelligent
mind and but a cleverly constructed but hollow language using facade? are understanding and reasoning
incompatible with materialistic, mechanistic view of living beings? could a machine ever be said to have
made its own decisions? could a machine have beliefs. could a machine make mistakes? could a machine believe
it made its own decisions? could a machine erroneously free will to itself? could a machine come up with
ideas that have not being programmed into it in advance. could creatively emerge from a set of fixed rules? are
we even the most creative among us but, passive slaves physics that govern our neurons?

Methodology

Convert into only alphabet strings (Remove non-alphabetic characters)

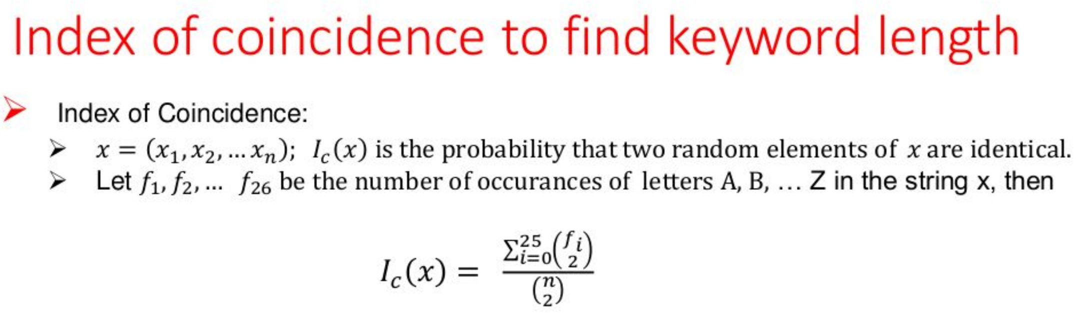
I started by converting the given cipher text into a string without any special characters.
This string will be used for finding the keylength and the key.
I considered all the characters whose ASCII value lies between 97 and 122 (inclusive).
Rest of the characters are discarded. String formed is given below

czuywudipniyephgdcaocltrpckpuamlnfhhrvhtltmvmyuarzoqtbictagnrzutazccrttfsrhzyczgnwaazororioflqtfrvtaarehlceozgcsiaxazdrzhpa
ciyrzntcposnhumeytlnqbhxarttpnpxmrvqioxiazogevzhtdrtmnpvretngkokpokiygnlxvdbodzfgailouzslnqtmvuczytnfbxvifgntnrmyvvgncpng
nlpiqjiygztwyqakocagpyebvktscrgnlzlcocdckitmfyochbpgzouzwpdvlnzvtaidhoxnnmrtareancemyecznfvcfcfgnoiamyctvmeytzbhuiajbftn
vfvdrxljcbgmkzhitpdonnywyrohlngalitkudiazzrknjelrrnhumeytlnqbhxiajrpafhhzvtonnoziukqorehigagrbrxillvlnzkzkcsaabmkqpbipwbyfzd
vtgmevgajkbaloaztwyqakegeeuyjivjtzhnoydiqkiesbphumpostoalwfcyjaxapacemugvpbrecvnfioflqtgrkuonpmndydqfzavefviltqgmlcubhvj
rripvrbndiqkiesbphumpostoalwfcyjaxapacemrxrznrhojtllrpejbfcbbotdeyywfcyjaxapacempumpucpckpvjelsgaukpnbeyoguyzvtvrzgetgd
mqoneovmceiqbaycrviltqirpagbpvtlkmprtxziwzgsptbyzzfrjrflrluimjkegeambvubytnrrtnzdrgmzntnmscgvadsvoyjtnbedpurmzkfzhltthpvza
uucnrnlfvf

Relevant Code - SC-1

To find the cipher:

I started by finding the Index of Coincidence ($I_c(x)$) of the ciphertext.



Index of coincidence to find keyword length

- Index of Coincidence:
  - $x = (x_1, x_2, \dots x_n)$; $I_c(x)$ is the probability that two random elements of x are identical.
  - Let $f_1, f_2, \dots f_{26}$ be the number of occurances of letters A, B, … Z in the string x, then

$$I_c(x) = \frac{\sum_{i=0}^{25} \binom{f_i}{2}}{\binom{n}{2}}$$

First, I found the frequency(f_i) of every alphabet in the whole ciphertext. The above formula can be rewritten as

$$I_c = \frac{\sum_{i=0}^{25} f(f-1)}{N(N-1)}$$

f = frequency of ith letter, N= no. of letters in the whole cipher text

By the above formula, Ic comes out to be 0.0422565405. Since it is less than 0.065 which is the Ic of the mono-alphabetic cipher, I deduced it to be a **polyalphabetic cipher**. Hence, the cipher used is **Vigenere cipher.**

Relevant Code - SC- 2

To predict the key length:

Now I divided the whole text in n-grams, made a table of n columns (cosets), arranged those n-grams in the table and calculated the Ic of each column. If the average I_c of the ciphertext divided into n-grams is the highest and closer to
I_c(monoalphabetic) = I_c(English) = 0.065,
it is likely to be the correct length of the key.The following list the average I_c value when ciphertext is decomposed into n substrings. Here n varies from 1 to 40 assuming the key length to not exceed 40 characters.

N	I_c	N	I_c	N	I_c	N	I_c
1	0.0422565405	11	0.0410008097	21	0.0414217219	31	0.0395578138
2	0.0440403399	12	0.0447173783	22	0.0419764446	32	0.0448468661
3	0.0419374948	13	0.0424670622	23	0.0423169165	33	0.0401899335
4	0.0442951777	14	0.0438631613	24	0.0460473078	34	0.0423529412
5	0.0552765178	15	0.0555209864	25	0.0533088872	35	0.0570807453
6	0.0451492626	16	0.0452363194	26	0.0448460853	36	0.0465982287
7	0.0424264769	17	0.0415450886	27	0.0410045798	37	0.042899619
8	0.0454350268	18	0.0476410731	28	0.043090206	38	0.0425966101
9	0.0428367757	19	0.0431962647	29	0.041164147	39	0.0413161718
10	0.0649931601	20	0.0639278286	30	0.0699264125	40	0.0647186147

It can be seen that the local maximum Ic occurs at N=10,20,30,40. When the above values are subtracted from 0.065 and the absolute value is stored and then sorted, 10 comes out to be the closest. This means **10 is the probable key length** for the given ciphertext.
Below is the sorted list of closeness to 0.65:
10,40,20,30,35,15,5,25,18,36,24,8,16,6,32,26,12,4,2,14,19,28,37,9,38
It can also be seen that the first four closest N to 0.065 are 10,40,20,30.
All 4 have common gcd = 10. This is sufficient to prove that key length of the ciphertext can be assumed to be 10 without any rethinking.

Relevant Code - SC- 3

To find the key:

After finding the key length is exactly 10, I divided the encrypted text into 10 columns placing consecutive letters adjacent to each other. Every 10th element is kept in the same columns. Since the key length is same as no. of columns if we can guess the encrypt key letter for any letter in the ciphertext we can apply the same decrypt operation on the whole column (consisting of that letter in the ciphertext) using the same key letter.
The first 3 columns of the matrix[N X 10] is shown below.

c	z	u	y	w	u	d	i	p	n
i	y	e	p	h	g	d	c	a	o
c	l	t	r	p	c	k	p	u	a
….	….	….	….	….	….	….	….	….	….

Now, we calculate the frequency of a letter in every column. Since the key repeats itself after every 10 letter, it is most probable that the most frequent letter in English is encrypted to same cipher letter.
Given below are the frequency of top 3 letters in every column.

Column	1	2	3	4	5	6	7	8	9	10
Letter 1	**i:9**	**p:16**	**n:13**	**r:13**	**b:9**	**c:11**	**z:10**	**m:12**	**r:12**	**j:10**
Letter 2	**e:9**	**l:10**	**a:10**	**a:10**	**t:8**	**y:10**	**v:10**	**i:10**	**q:8**	**k:10**
Letter 3	**t:9**	**z:8**	**e:9**	**g:8**	**n:7**	**h:10**	**r:8**	**c:8**	**f:7**	**o:8**

Since p occurs 16 times in column 2, which is more frequent than any other letter in that column or any other letter in any of the column, we can guess that e is mapped to p. “e” is the most frequent letter in the English language. So guessing “p” to be mapped to “e” is a good guess.

decrypt("p") = "e"
"p" = ("e" + K) % 26
Where K is the key of column 2.
K =("p" -"e") %26
K = (112 - 101)%26 + 97 [Using Ascii Value to calculate K, 97 is the ascii value of "a"]
K = 108
K = "l" [Ascii value of "l" is 108]

Thus key of column 2 is "l".
We can now decrypt every letter in column 2 using key "l" following the below formula:

decrypt(X) = (((ord(X) - ord("l"))%26)+97)

- **ord(X) gives the ascii value of X.**
- **97 is the ascii value of "a"**

Hence, the matrix[N * 10] will look like this now. (Only first 3 rows are shown)

c	o	u	y	w	u	d	i	p	n
i	n	e	p	h	g	d	c	a	o
c	a	t	r	p	c	k	p	u	a
....	....	....	....	....	....	....	....	....	....

There are many single letters in the ciphertext. The only valid single letter word in English is "a" or "I". "a" is mostly used as an article whereas "I" is used in the first person.
So we can guess all single letter word as "a" or "I". Let's start by marking all single letter word and their corresponding column.

[Relevant Code - SC- 4](#)

czuyw u dipniye phgdcaocltr pckp uamlnf hh rv htltmvmyu arz oq tbicta gnrzuta zccrtt fsr hz yczgn waazoror? ioflq t frvtaare hlceo zgcsiax azdr zhp aciyrzntcp os nhumeytlnqbhx arttpnpxm rvq ioxiaz og evzh tdrtm npvre tn gkokp okiyg nl xvdbod zf gailouzs lnq tm vuczy tnfbxv if g ntnrmyvvgn cpngnlp iqjiyg ztwyqak oc a gpyebvkts crgnlzl cocd ckitmfyoc? hbp gzouz wp dvlnzvtaidh oxnnmrt a reancemye cznfvcfcf gno iamyctvmeyt zbhu iaj bft n vfvdrxlj cbgmkzhitpd onn ywyroh lngalitk udiaz zrknje? lrr nhumeytlnqbhx iaj rpafhhzvt onnoziukqore higa grbrxillvlnzk, zkcsaabmkqp bipw by fzdvtg mevgaj? kbalo a ztwyqak egee uy jivj tz hnoy diqk ies bph umpostoal? wfcyj a xapacem ugvp brecvnf. ioflq t grkuonp mndy dqfzavef? viltq g mlcubhv jrripvr bn diqk ies bph umpostoal? wfcyj a xapacem rxrznrhojtl lrpe jbfc bb otdeyy? wfcyj a xapacem pump uc pckp vjels gauk pnbe yog uyzvt vrzgetgdmq oneo vm ce iqbaycr. viltq irpagbpvtl kmprtx ziwz g spt by zzfrj rflrl? uim jk egea mbv ubyt nrrtnzdr gmznt nm scg, vadsvoy jtnbed purmzkf zhlt thpvza uuc nrnlfvf?

u	6
t	5
g	10
a	3
a	1
n	4
a	3
a	1
t	5
g	10
a	1
a	1
g	10

Let us guess that “u” in first line of the ciphertext is mapped to “a”. So,

decrypt(“u”) = “a”
“u” = (“a” + K) % 26
Where K is the key of column 2.
K =(“u” -”a”) %26
K = (117 -97)%26 + 97 [Using Ascii Value to calculate K, 97 is the ascii value of “a”]
K = 117
K = “u” [Ascii value of “u” is 117]

Thus key of column 6 is guessed to be “u”
Similarly for other one letter word, assuming them to be mapped to “a” (since for them to be “I” the text has to be written in the first person which is less likely) keys of their corresponding column is guessed to be that letter itself.(because any cipher letter mapped to “a” is by key which is same as the cipher letter)

ord(C) = (ord(K) + ord(“a”))%26
C is cipher letter
K is key

Since “a” represent 0th letter with respect to 26 letters. Thus K = C.
Hence, Key of 5th column is “t”, 10th column is “g” ,4th column is “n”, 1st and 3rd column is “a”.

1	2	3	4	5	6	7	8	9	10
a	l	a	n	t	u				g

Accordingly we change the letter in each of the columns where key is known

c	o	u	l	d	a	d	i	p	h
i	n	e	c	o	m	d	c	a	i
c	a	t	e	w	i	k	p	u	u
….	….	….	….	….	….	….	….	….	….

The first two words - "could a" makes sense in English. So we are so far correct in our methodology and guessing. The third word is currently "diphine" where the marked red letters have not been decrypted. Also, the fourth letter is "comdcaicate". Valid 7 letter words that can end in "hine" are "archine", "errhine", "machine" and "reshine". But the only Valid 11 letter words that can end in "icate" and starts with "com" is "communicate".
"Comdcaicate" -> "communicate"
Assume "d" is mapped to "m", "c" to "u" and "a" to "n".
So the key in column 7 (d->m) is :
K = (ord("d") - ord("m"))%26 = -9%26
K = 17
K = chr(17+97) = "r"
Similarly key for column 8(c->u) is "i"
And key for column 9(a->n) is "n".
Thus the key found is -> **"alanturing"** - Father of theoretical computer science and artificial intelligence.
It can also be seen in the initial table of top 3 frequency of letters in each column that apart from column 2, top letter frequency of column 1,4,8, 10 also matched with "e"(Most frequent English letter) resulting in the same key "alanturing". Thus we were right from the beginning itself.

1	2	3	4	5	6	7	8	9	10
a	l	a	n	t	u	r	i	n	g

Accordingly we change the letter in each of the columns where key is known

c	o	u	l	d	a	m	a	c	h
i	n	e	c	o	m	m	u	n	i

c	a	t	e	w	i	t	h	h	u
….	….	….	….	….	….	….	….	….	….

Thus this is the plaintext with key “alanturing”

could a machine communicate with humans on an unlimited set of topics through fluent use of human language? could a language using machine give the appearance of understanding sentences and coming up with ideas while in truth being as devoid of thoughts and as empty inside as a nineteenth century adding machine or a twentieth century word processor? how might we distinguish between a genuinely conscious and intelligent mind and but a cleverly constructed but hollow language using facade? are understanding and reasoning incompatible with materialistic, mechanistic view of living beings? could a machine ever be said to have made its own decisions? could a machine have beliefs. could a machine make mistakes? could a machine believe it made its own decisions? could a machine erroneously free will to itself? could a machine come up with ideas that have not being programmed into it in advance. could creatively emerge from a set of fixed rules? are we even the most creative among us but, passive slaves physics that govern our neurons?

Relevant Code - SC- 5

Source Code

SC-1

```
"""@parameters
cip is the original cipher text"""
def toalpha(cip):
   print("Removing non-alphabetic charcters..\n")
   result = ""
   for i in range(len(cip)):
         if (ord(cip[i])>=97 and ord(cip[i])<=122):
               result=result+cip[i]
   return result
```

SC-2

```
"""@parameters
cip is the string of all alphabetic characters"""
def findCipherType(cip):
   print("Finding Cipher Type..\n")
   size = len(cip)  # length of ciphertext
   freq = dict()              # Dictionary containing frequency of letters
   for i in cip:
         if i in freq:
               freq[i]+=1
         else :
               freq[i]=1
   sum=0
   for i in freq:
         sum+=(freq[i]*(freq[i]-1))
   sum = sum/(size*(size-1))
   print(" Ic = " + str(sum))
   diff = abs(sum - 0.065)
   print ("Difference from mono-alphabetic cipher = " + str(diff))
   if (diff< 0.001):
         print("Cipher used is Mono-Alphabetic Cipher")
         return ("mono")
   else :
         print("Cipher used is Poly-Alphabetic Cipher")
         return ("poly")
```

SC-3

```
"""@parameters
b and c are two integers"""
def gcd(b,c):
   return c if (b==0) else gcd(c%b,b)
"""@parameters
cip is the string of all alphabetic characters"""
```

```
def findProbableKeyLength(cip):
    print("Finding Key Length..\n")
    size = len(cip)  # length of ciphertext
    Ic = list()        # List of Ic, index i contains Ic for (i+1)-grams / (i+1)-substring
    """ Looping for divding cipher text in n-substrings """
    for i in range(1,41):
            substring=list() # List of List, index i contains (i+1)th substrings
            l1=list()            # List of Ic of individual substrings
            for j in range(0,i):
                    substring.append([])  #Initialise the list of list to empty lists
            for j in range(0,size):
                    substring[j%i].append(cip[j])   # Assign letter to their corresponding substring list
            for j in range(0,len(substring)):
                    freq=dict()       # store the frequency of each letter
                    for k in substring[j]:
                            if k in freq:
                                    freq[k] +=1
                            else :
                                    freq[k] =1
                    sum = 0
                    for k in freq:
                            sum += freq[k]*(freq[k]-1)
                    sum = sum/(len(substring[j])*(len(substring[j])-1))
                    l1.append(sum)
            sum=0
            for j in l1:
                    sum += j
            sum = sum/len(l1)
            Ic.append(sum)
    # Print the Ic values with corresponding N
    for i in range(0,len(Ic)):
            print(str(i+1) + "   :  " + str(Ic[i]))
    freq2=dict()              #
    for i in range(0,len(Ic)):
            Ic[i] = abs(Ic[i]-0.065)
            freq[i+1] = Ic[i]
    absdiff = sorted(freq.items(), key = lambda x: x[1])  #sorted list of absolute difference with 0.065
    print("Closest N to 0.065 = " + str(absdiff[0]))
    absdiff = absdiff[:3]     # Take top 3 closest key lengths and find their gcd, gcd is the key length
    key = gcd(absdiff[0][0],absdiff[1][0])
    key = gcd(key,absdiff[2][0])
    print("Key Length = " + str(key))
    return key
```

<u>SC-4</u>

```
"""@parameters
Cip is the original cipher text
keylength is the length of the key"""
```

```
def printSingleLetterWord(cip,keylength):
   print("Finding single letter word and their corresponding column")
   size = len(cip)
   cip = cip.split(" ")
   count = 0
   char=[]     # single letter word
   column=[]    # corresponding column
   for i in cip:
         if (len(i)==1):
               char.append(i)
               count+=1
               column.append((count)%keylength if (count)%keylength else keylength)
         else :
               if (122>=ord(i[len(i)-1])>=97):
                     count+=len(i)
               else :
                     count+=(len(i)-1)
   for i in range(0,len(char)):
         print(str(char[i]) + " : " + str(column[i]))
   return char,column
```

SC-5

```
"""@parameters
cip is string of all alphabetic characters
cip2 is the original cipher text
keylength is the length of the key"""
def convertToPlain(cip,cip2,keylength,key):
   print("Decrypting using key - " + str(key))
   size = len(cip)
   size2= len(cip2)
   columnword=list() # columns words
   for i in range(0,keylength):
         columnword.append([])
   for i in range(0,size):
         columnword[i%keylength].append(cip[i])
   # print the table
   for i in range(0,len(columnword[0])):
         for j in range(0,keylength):
               try:
                     print(columnword[j][i],end=" ")
               except:
                     pass
         print("\n")
   occ=list()  # list having occurence no.
   for i in range(0,keylength):
         d=dict()    # temp dictionary containing frequency of letters
         for i in columnword[i]:
            if i in d:
                d[i] = d[i]+1
            else :
                d[i] = 1
        occ.append(d)
tup = list()     # 3 most occured letter. in every column
for i in range(0,keylength):
        tup.append(sorted(occ[i].items(), key=lambda x: x[1])[-3:])
# print 3 most occured letter. in every column
for i in range(0,keylength):
        print(tup[i])

kount=0
while(kount!=keylength):
        ch2 = ord(key[kount])-ord("a")
        final =""
        count = kount
        for i in range(0,size2) :
            if (count==0 and (ord(cip2[i])>=97 and ord(cip2[i])<=122)):
                con = ord(cip2[i])-ch2
                if con>=97 and con <=122:
                    final = final + chr(con)
                elif (con<97):
                    con = 122 - (96-con)
                    final = final + chr(con)
                else :
                    con = 97 + (con-123)
                    final = final + chr(con)
                count = 9
            elif (ord(cip2[i])>=97 and ord(cip2[i])<=122):
                final = final + cip2[i]
                count =count -1
            else :
                final = final +cip2[i]
        kount = kount +1
        cip2 = final
return (final)
```