EECS 475, Winter 2018
Introduction to Cryptography

Solutions written by:
David Guerra

Cryptography and Network Security
Principles and Practice (Seventh Edition)
Author - William Stallings

Electrical Engineering and Computer Science Department
University of Michigan

Contents

1 Homework 01

Problem 3.1

A generalization of the Caesar cipher, known as the affine Caesar cipher, has the following form: For each plaintext letter p, substitute the ciphertext letter C:

$$C = E([a, b], p) = (ap + b) \bmod 26$$

A basic requirement of any encryption algorithm is that it be one-to-one. That is, if $p \neq q$, then $E(k, p) \neq E(k, q)$. Otherwise, decryption is impossible, because more than one plaintext character maps into the same ciphertext character. The affine Caesar cipher is not one-to-one for all values of a. For example, for $a = 2$ and $b = 3$, then $E([a, b], 0) = E([a, b], 13) = 3$.

a. Are there any limitations on the value of b? Explain why or why not. [3 points]

> **Solution:** Addition by b represents a shift substitution. Therefore, impose $b \in \mathbb{Z}_{26}$ so each shift is unique.

b. Determine which values of a are not allowed. [3 points]

> **Solution:** The decryption algorithm is easy to construct.
>
> $$D([a, b], C) := a^{-1}(C - b) \pmod{26} = p$$
>
> For $a^{-1} \pmod{26}$ to exists, we require $\gcd(a, 26) = 1$. Hence, a cannot be even or 13.

Problem 3.8

A disadvantage of the general monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technnique for avoiding this is to use a keyword from which the cipher sequence can be generated. For example, using the keyword CRYPTO, write out the keyword followed by unused letters in normal order and match this against the plaintext letters:

```
plain:  a b c d e f g h i j k l m n o p q r s t u v w x y z
CIPHER: C R Y P T O A B D E F G H I J K L M N Q S U V W X Z
```

If it is felt that this process does not produce sufficient mixing, write the remaining letters on successive lines and then generate the sequence by reading down the columns:

```
C R Y P T O
A B D E F G
H I J K L M
N Q S U V W
X Z
```

This yields the sequence:

```
C A H N X R B I Q Z Y D J S P E K U T F L V O G M W
```

Such a system is used in the example in Section 3.2 (the one that begins "it was disclosed yesterday"). Determine the keyword. [3 points]

Solution: From the decrypt, we know the correspondence below:

```
plain:  a b c d e f g h i j k l m n o p q r s t u v w x y z
CIPHER: S A H V P B J W U ? ? X T D M Y ? E O Z I F Q ? G ?
```

The keyword must have a length of 6 so that `A` and `B` line up horizontally.

```
S P U T ? I ?                S P U T N I K
A B ? D E F G        =>      A B C D E F G
H J ? M O Q ?        =>      H J L M O Q R
V W X Y Z                    V W X Y Z
```

With little effort, we see the keyword is `SPUTNIK`.

Problem 3.9

When the PT-109 American patrol boat, under the command of Lieutenant John F. Kennedy, was sunk by a Japanese destroyer, a message was received at an Australian wireless station in Playfair code:

```
KXJEY UREBE ZWEHE WRYTU HEYFS
KREHE GOYFI WTTTU OLKSY CAJPO
BOTEI ZONTX BYBNT GONEY CUZWR
GDSON SXBOU YWRHE BAAHY USEDQ
```

The key used was *royal new zealand navy*. Decrypt the message. Translate `TT` into tt. [3points]

Solution: Begin by creating the 5×5 Playfair matrix from the known keyword.

R	O	Y	A	L
N	E	W	Z	D
V	B	C	F	G
H	I/J	K	M	P
Q	S	T	U	X

Then, we can break up the ciphertext into digrams and decrypt each by applying the algorithm in reverse.

```
CIPHER: KX JE YU RE BE ZW E H  EW RY TU  H E YF S K  RE  H E GO YF IW TT TU OL  K S YC AJ
plain:  pt bo at on eo we ni/j ne lo st i/jn ac ti/j on i/jn bl ac ke tt st ra i/jt tw om

CIPHER: P O  BO TE IZ ON TX BY BN TG ON EY CU ZW RG DS ON SX BO UY WR  H E BA AH YU  S E DQ
plain:  i/jl es sw me re su co ve xc re wo ft we lv ex re qu es ta ny i/jn fo rm at i/jo nx
```

When rearranging the cipher, we can ignore extra x's to get the plaintext:

```
pt boat one owe nine lost in action in blackett strait two miles
sw meresu cove crew of twelve request any information
```

Problem 3.11

a. Using this Playfair matrix:

M	F	H	I/J	K
U	N	O	P	Q
Z	V	W	X	Y
E	L	A	R	G
D	S	T	B	C

Encrypt this message:

Must see you over Cadogan West. Coming at once.

Note: This message is from the Sherlock Holmes story, The Adventure of the Bruce-Partingon plans. [3 points]

Solution: Split the plaintext into digrams and add pad with an 'x' to encrypt as shown below.

```
plain:  mu st se ey ou ov er ca do ga nw es tc om in ga to nc ex
CIPHER: UZ TB DL GZ PN NW LG TG TU ER VO LD BD UH FP ER HW QS RZ
```

b. Repeat part (a) using the keyword *largest*. [3 points]

Solution: Using the keyword *largest*, we construct the Playfair matrix below and encrypt.

L	A	R	G	E
S	T	B	C	D
F	H	I/J	K	M
N	O	P	Q	U
V	W	X	Y	Z

```
CIPHER: UZ TB DL GZ PN NW LG TG TU ER OV DL BD UH PF ER HW QS RZ
```

c. How do you account for the results of this problem? Can you generalize your conclusion? [3 points]

Solution:

The Playfair matrix can be thought of as a torus $\mathcal{T}$ by folding and gluing the vertical edges to each other and the same with the horizontal edges. The figure to the right shows the two Playfair matrices from parts **a** and **b**, outlined in red and blue. Notice that they both generate $\mathcal{T}$. It makes sense that our ciphertexts for parts **a** and **b** were the same. If we wanted different ciphertexts, we'd need to swap rows or columns of the Playfair matrix. This wouldn't be equivalent to any shift and thus providing a new ciphertext.

Y	*Z*	*V*	*W*	*X*	*Y*	*Z*	*V*	*W*
G	*E*	*L*	*A*	*R*	*G*	*E*	*L*	*A*
C	*D*	*S*	*T*	*B*	*C*	*D*	*S*	*T*
K	*M*	*F*	*H*	*I/J*	*K*	*M*	*F*	*H*
Q	*U*	*N*	*O*	*P*	*Q*	*U*	*N*	*O*
Y	*Z*	*V*	*W*	*X*	*Y*	*Z*	*V*	*W*
G	*E*	*L*	*A*	*R*	*G*	*E*	*L*	*A*
C	*D*	*S*	*T*	*B*	*C*	*D*	*S*	*T*
K	*M*	*F*	*H*	*I/J*	*K*	*M*	*F*	*H*

Problem 3.14

a. Encrypt the message "meet me at the usual place at ten rather than eight o clock" using the Hill cipher with the key $\begin{bmatrix} 7 & 3 \\ 2 & 5 \end{bmatrix}$. Show your calculations and the result. [3 points]

Solution: Since the Hill cipher matrix $K \in \mathbb{Z}^{2\times 2}$, we'll rewrite the plaintext into digrams and convert to numeric. We'll also pad the string with 'x' at the end to achieve an even length.

`plain: me et me at th eu su al pl ac ea tt en ra th er th an ei gh to cl oc kx`

This gives us the following list of vectors p_i for $1 \leq i \leq 24$:

$$(12,4),(4,19),(12,4),(0,19),(19,7),(4,20),(18,20),(0,11),(15,11),(0,2),(4,0),(19,19),$$
$$(4,13),(17,0),(19,7),(4,17),(19,7),(0,13),(4,8),(6,7),(19,14),(2,11),(14,2),(10,23).$$

To encrypt, let p_i be the i^{th} row of matrix P so that

$$\begin{aligned} PK &= \begin{bmatrix} 12 & 4 & 12 & 0 & 19 & \dots & 14 & 10 \\ 4 & 19 & 4 & 19 & 7 & \dots & 2 & 23 \end{bmatrix}^T \begin{bmatrix} 7 & 3 \\ 2 & 5 \end{bmatrix} \pmod{26} \\ &\equiv \begin{bmatrix} 14 & 14 & 14 & 12 & 17 & \dots & 24 & 12 \\ 4 & 3 & 4 & 17 & 14 & \dots & 0 & 15 \end{bmatrix}^T = C. \end{aligned}$$

We now convert each row c_i of C to alphabet characters to get the ciphertext below.

`CIPHER: OE OD OE MR QI KY WD XW EK CM PW CZ PZ RO AN SA EB FX KJ YA MP`

b. Show the calculations for the corresponding decryption of the ciphertext to recover the original plaintext. [3 points]

Solution: For decryption, compute $\det(K) = 29 \equiv 3 \pmod{26}$. Since $9 = 3^{-1} \pmod{26}$, we have

$$K^{-1} = 9\begin{bmatrix} 5 & -3 \\ -2 & 7 \end{bmatrix} \equiv \begin{bmatrix} 19 & 25 \\ 8 & 11 \end{bmatrix} \pmod{26}.$$

To obtain the plaintext, we simply take the product

$$\begin{aligned} CK^{-1} &= \begin{bmatrix} 14 & 14 & 14 & 12 & 17 & \dots & 24 & 12 \\ 4 & 3 & 4 & 17 & 14 & \dots & 0 & 15 \end{bmatrix}^T \begin{bmatrix} 19 & 25 \\ 8 & 11 \end{bmatrix} \pmod{26} \\ &\equiv \begin{bmatrix} 12 & 4 & 12 & 0 & 19 & \dots & 14 & 10 \\ 4 & 19 & 4 & 19 & 7 & \dots & 2 & 23 \end{bmatrix}^T = P. \end{aligned}$$

This recovers plaintext matrix P from part **a**.

Problem 3.18

b. Determine the inverse mod 26 of

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}. \qquad \text{[3 points]}$$

Solution: Denote matrix A. To compute A^{-1} mod 26, we first need the determinant.

$$\det(A) = 6(16 \cdot 15 - 17 \cdot 10) - 24(13 \cdot 15 - 20 \cdot 10) + (13 \cdot 17 - 20 \cdot 16) \equiv 25 \pmod{26}$$

We can then find the cofactors of $A^T = \begin{bmatrix} 6 & 13 & 20 \\ 24 & 16 & 17 \\ 1 & 10 & 15 \end{bmatrix}$ as shown below.

$$\begin{array}{lll} C_{1,1} = (-1)^2(16 \cdot 15 - 10 \cdot 17) & C_{1,2} = (-1)^3(24 \cdot 15 - 1 \cdot 17) & C_{1,3} = (-1)^4(24 \cdot 10 - 1 \cdot 16) \\ C_{2,1} = (-1)^3(13 \cdot 15 - 10 \cdot 20) & C_{2,2} = (-1)^4(6 \cdot 15 - 1 \cdot 20) & C_{2,3} = (-1)^5(6 \cdot 10 - 1 \cdot 13) \\ C_{3,1} = (-1)^4(13 \cdot 17 - \cdot 16 \cdot 20) & C_{3,2} = (-1)^5(6 \cdot 17 - 24 \cdot 20) & C_{3,3} = (-1)^6(6 \cdot 16 - 24 \cdot 13) \end{array}$$

This allows us to create the adjoint matrix

$$\operatorname{adj}(A) = \begin{bmatrix} 70 & -343 & 224 \\ 5 & 70 & -47 \\ -99 & 378 & -216 \end{bmatrix}.$$

Lastly, notice that $25 \cdot 25 \equiv 1 \pmod{26}$ so we take the product

$$(\det(A))^{-1} \cdot \operatorname{adj}(A) = 25 \cdot \operatorname{adj}(A) \equiv \begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \pmod{26} = A^{-1}.$$

Problem 3.19

Using the Vigenère cipher, encrypt the word "explanation" using the word "leg". [3 points]

Solution: Begin by repeating the keyword over the plaintext as shown below.

```
key:    legleglegle
plain:  explanation
```

We can then perform the required shifts by taking the sum mod 26.

key	11	4	6	11	4	6	11	4	6	11	4
plain	4	23	15	11	0	13	0	19	8	14	13
CIPHER	15	1	21	22	4	19	11	23	14	25	17

Encryption is completed by writing the numerical values into an alphabetic ciphertext.

```
CIPHER: PBVWETLXOZR
```

Problem A

A precursor to the ADFGVX cipher was the ADFGX cipher which used a table such as this:

	A	D	F	G	X
A	b	t	a	l	p
D	d	h	o	z	k
F	q	f	v	s	n
G	g	i/j	c	u	x
X	m	r	e	w	y

Encrypt the phrase ”neither do they spin” below. Use the grid on the left below to write the two-letter substitutions row-wise. Then rearrange the columns so that the column headers are in alphabetical order in the grid on the right.

M	**E**	**R**	**I**	**T**

Write the ciphertext by reading out the grid on the right column-wise. [3 points]

Solution: Using the provided ADFGX table, fill out the two tables below.

M	**E**	**R**	**I**	**T**
F	X	X	F	G
D	A	D	D	D
X	F	X	D	D
A	D	F	A	D
D	D	X	F	X
X	F	G	A	X
G	D	F	X	

$\Rightarrow$

E	**I**	**M**	**R**	**T**
X	F	F	X	G
A	D	D	D	D
F	D	X	X	D
D	A	A	F	D
D	F	D	X	X
F	A	X	G	X
D	X	G	F	

Thus, we have the encryption of the phrase ”neither do they spin” below.

```
CIPHER: XAFDDFDFDDAFAXFDXADXGXDXFXGFGDDDXX
```

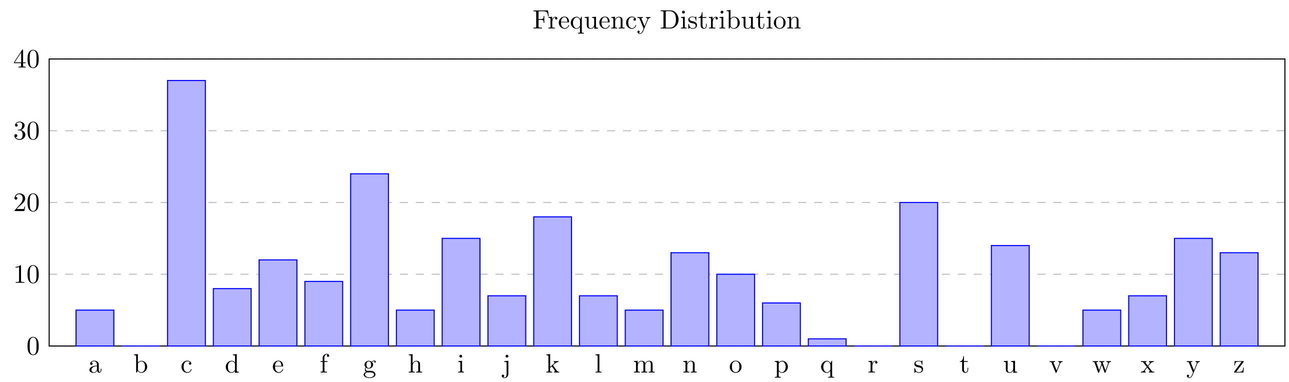
Problem B

Decrypt the following permutation substitution cipher. [3 points]

```
emglosudcgdncuswysfhnsfcykdpumlwgyicoxysipjckqpkugkmgolicgincgacksnisacykzsckxecjckshy
sxcgoidpkzcnkshicgiwygkkgkgoldsilkgoiusigledspwzugfzccndgyysfuszcnxeojncgyeoweupxezgac
gnfglknsacigoiyckxcjuciuzcfzccndgyysfeuekuzcsocfzccnciaczejncshfzejzegmxcyhcjumgkucy
```

Solution: Begin by analyzing the frequency distribution using C.

```c
while (fread(&buffer, sizeof(char), 1, in) == 1)
{
    if (isalpha(buffer) == 0) { continue; }
    freq[buffer-97]++;
}

for (int i = 0; i < ALPHABET; i++)
    { printf("%c %d\n", i+97, freq[i]); }
```



The frequency distribution is very close to that of a monoalphabetic cipher. We can begin to take guesses for certain letters based on the distribution. Notice, there is an 11-letter word repeated twice with a 5-letter prefix.

```
CIPHER: EMGLOSUDCGDNCUSWYSFHNSFCYKDPUMLWGYICOXYSIPJCKQPKUGKMGOLICGINCGACKSNI
plain:          e   e     t   te           e       e            e   e  e

CIPHER: SACYKZSCKXECJCKSHYSXCGOIDPKZCNKSHICGIWYGKKGKGOLDSILKGOIUSIGLEDSPWZUG
plain:    e  h e   e e      e      he     e                              h

CIPHER: FZCCNDGYYSF  USZCNXEOJNCGYEOWEUPXEZGACGNFGLKNSACIGOIYCKXCJUCIUZC
plain:  thee      t    he      e          h  e  t      e     e  e  e  he

CIPHER: FZCCNDGYYSF  EUEKUZCSOC  FZCCN  CIACZEJNCSHFZEJZEGMXCYHCJUMGKUCY
plain:  thee      t       he  e  thee   e  eh   e  th  h    e  e      e
```

The prefix `thee` suggests that `F` probably does not correspond to `t`. Consider the word `wheel` instead and use this to begin making some assertions and decrypt as shown below.

```
i may not be able to grow flowers but my garden produces just as many dead leave sold
over shoes pieces of rope and bushels of dead grass as any bodys and today i bought a
wheelbarrow to help in clearing it up i have always loved and respected the wheelbarrow
it is the one wheeled vehicle of which i am perfect master
```

Problem C

Decrypt the following Vigenère ciphertext using the Index of Coincidence method. [3 points]

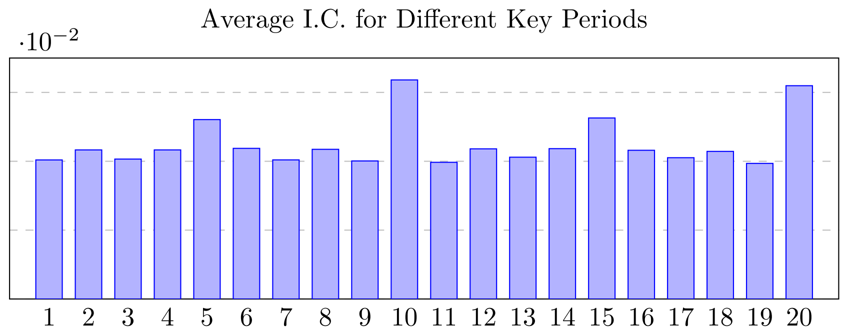
```
hlfiusvsaqfpfpwaryxewdudwbrvxvrthapcjlhrlalbkwfeecmlpfvuyimxqpiovczogidthjgdhrlifyxkwhuigkull
qqqhltvyzckbseelxrpikavbrxmysirovgipszwxpiwyauszeuiqhglxesombmhuepeyplvxoethjysytwfydbxndutim
vhtzjzzazwiigktqzqpsfpeijyuklfrgnpfeckohuevmwnoiihvogamphbdseiymsogvasyfzvthckoueifegzoqbvrmh
yysqembrvxvnecpirnmihwttwtmaooirmyoqbzylszwqembrvxvnqcklalsxpkthswzhnmewtfvxohsbrlmycwfrchjkt
htifrhhyxpmxvrorbrlthdcdghxvjlllnsiekckfhbzoyifijeuklsfpxgnlfigkuegiapljgvlphwlpnpcquagyuayop
oadhjrpneihvtkivfcomgnsmvtyajwfnlivnywfxzrthtwppbvhzeyvtxxbtwoekeypfvahrpimsrckbcghxwycybboad
fiysiaqqnlecpyizswagiitafbavntlskqnembvavgtfhqqhuizlytgbbctemxtdyxigfohrfdczibghbwndgvaiosmmy
fnbncepbwyzfxvroaepbtneiduiesxzjeqqnlyptflfavpamsysllegu ifwjwzrxcahbwxhiolwdubiywsqiyrthxmpme
qdghxvjtmkwhuigkxflmzwfigknyneqgvfmljjvrbyaepmylfjwggaeprphfvhuebvipaomsfofiytgbwfbtaiwnbbzwf
hoiwjhbifyymljdujmtreemsrmqwknrwwysylksnnpmysgbbvrrxrthcpgchrbrxffxzqvtrskebbuoahtxyzypjsytxh
wzoklplwaewgypigvnwmfycptsfbrgtcuizsrflgtxgbzqrsnvwzoklgvtpmysbbzghryvnrbqibqlxjyebbaheexxxeu
hmmbupeypltifqimwjinomardhaseitvwftaiglnqmflwaiwpneihaoupjxiimwfwtwmpxygknvxwfyxzwcyewfdmlbmn
rsplnnbxnsjhhywdjomjvonwbplbwigoywnrbqwtyaghqzihihghxgwzqaacswtxjcaxhsesmljcy
```

Solution: We begin by testing keyword periods and analyzing their corresponding Index of Coincidences.

```c
double* getAverageIOCs(char *in_cipher)
{
    double *avgs = malloc(TRIALS*sizeof(double));
    int freq[ALPHA_SIZE] = {0};
    double ioc = 0;

    for (int k = 1; k <= TRIALS; k++)
    {
        for (int gap = 0; gap < k; gap++)
        {
            setFrequency(in_cipher, gap, k, CIPHER_SIZE, freq);
            ioc = getIOC(freq);
            avgs[k-1] += ioc;
        }
        avgs[k-1] /= k;
    }

    return avgs;
}

void setFrequency(char *data, int start, int jump, int end, int *freq)
{
    memset(freq, 0, ALPHA_SIZE*sizeof(int));   // set all vals to 0
    for (int i = start; i < end; i += jump)
        { freq[data[i]-97]++; }
}

double getIOC(int *freq)
{
    double out = 0;   // Index of Coincidence to return
    int length = 0;

    for (int i = 0; i < ALPHA_SIZE; i++) { length += freq[i]; }
    for (int i = 0; i < ALPHA_SIZE; i++)
    {
        out += 1.0*freq[i]*(freq[i] - 1)/(length*(length - 1));
    }

    return out;
}
```

The maximum of `avgs` yields 10 but it's clear from the results below that 20 is also a possible period.

Period	Avg I.C.
1	0.0403894
2	0.0433027
⋮	⋮
7	0.0404037
8	0.0434730
9	0.0400959
10	0.0636256
⋮	⋮
20	0.0619492



We choose a period of 10 and try the 26 monoalphabetic ciphers to every 10th letter of the ciphertext to obtain the Chi-Square statistics. To do so, we need the most current frequency distribution according to Peter Norvig in 2012 (`http://norvig.com/mayzner.html`).

```
for (int k = 0; k < key_size; k++)
{
    for (char ltr = 'a'; ltr <= 'z'; ltr++)
    {
        setCaesarShift(cipher, shift, k, key_size, ltr);
        setFrequency(shift, 0, 1, shift_sz, freq);
        chi_vals[ltr-97] = getChiSq(eng_dist, shift_sz, freq);
    }

    keyword[k] = getChiMin(chi_vals);
}
```

```
void setCaesarShift(char *data, char *shifted, int start, int key, char c)
{
    for (int i = start, j = 0; i < CIPHER_SIZE; i += key, j++)
    {
        if (data[i] >= c)
            { shifted[j] = data[i] - c + 'a'; }
        else
            { shifted[j] = data[i] - c + 26 + 'a'; }
    }
}

double getChiSq(double *edist, int size, int *cfreq)
{
    double expect = 0;
    double out    = 0;

    for (int i = 0; i < ALPHA_SIZE; i++)
    {
        expect = size * edist[i];
        out += pow(cfreq[i] - expect, 2) / expect;
    }

    return out;
}
```

Notice that `setFrequency` was defined in the previous page. The values of `shift` and `chi_vals` are shown below.

Key	Caesar Shift	Chi-Sq
a	`hfwrlmqdfuyryzzsyynjtjfwmstgynwrznkwlkxljkj...sgwjmfdnwbnqwjj`	1139.65
b	`gevqklpcetxqxyyrxxmisievlrsfxmvqymjvkjwkiji...rfvilecmvampvii`	3627.85
c	`fdupjkobdswpwxxqwwlhrhdukqrewlupxliujivjhih...qeuhkdbluzlouhh`	876.493
d	`ectoijnacrvovwwpvvkgqgctjpqdvktowkhtihuighg...pdtgjcaktykntgg`	4924.34
e	`dbsnhimzbqunuvvouujfpfbsiopcujsnvjgshgthfgf...ocsfibzjsxjmsff`	826.637
f	`carmghlyaptmtuunttieoearhnobtirmuifrgfsgefe...nbrehayirwilree`	109.124
⋮	⋮	⋮
z	`igxsmnregvzszaatzzokukgxntuhzoxsaolxmlymklk...thxkngeoxcorxkk`	4619.63

This suggest that the letter `f` was used to encode every 10th plaintext letter. Hence, it is likely the first letter of the keyword. The full Chi-Square analysis yields the keyword `fluxionate`.

```c
char* getDecrypt(char *data, char *kword, int size)
{
    char *ptext = malloc(CIPHER_SIZE*sizeof(char));
    int j = 0;

    for (int i = 0; i < CIPHER_SIZE; i++)
    {
        j = i % size;
        if (data[i] >= kword[j])
        { ptext[i] = data[i] - kword[j] + 'a'; }
        else
        { ptext[i] = data[i] - kword[j] + ALPHA_SIZE + 'a'; }
    }

    return ptext;
}
```

The full decrypt is shown below with spacing and punctuation applied.

```
Call me Ishmael. Some years ago never mind how long precisely having little or no
money in my purse, and nothing particular to interest me on shore, I thought I
would sail about a little and see the watery part of the world. It is a way I have
of driving off the spleen and regulating the circulation. Whenever I find myself
growing grim about the mouth; whenever it is a damp, drizzly November in my soul;
whenever I find myself involuntarily pausing before coffin warehouses, and bringing
up the rear of every funeral I meet; and especially whenever my hypos get such an
upper hand of me, that it requires a strong moral principle to prevent me from
deliberately stepping into the street, and methodically knocking peoples hats off
then, I account it high time to get to sea as soon as I can. This is my substitute
for pistol and ball. With a philosophical flourish Cato throws himself upon his
sword; I quietly take to the ship. There is nothing surprising in this. If they
but knew it, almost all men in their degree, some time or other, cherish very
nearly the same feelings towards the ocean with me. There now is your insular city
of the Manhattoes, belted round by wharves as Indian isles by coral reefs commerce
surrounds it with her surf. Right and left, the streets take you waterward. Its
extreme downtown is the battery, where that noble mole is washed by waves, and
cooled by breezes, which a few hours previous were out of sight of land. Look at
the crowds of water gazers there.
```

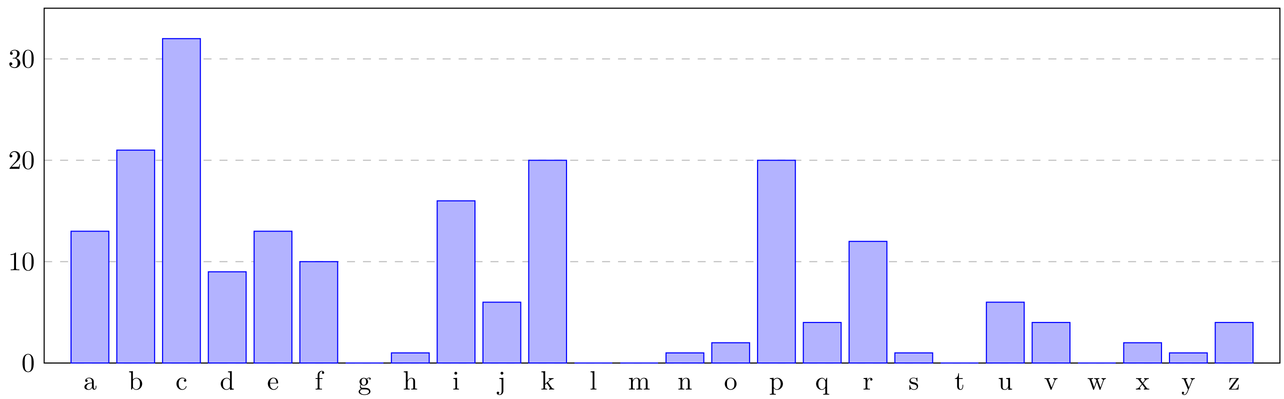
Problem D

Decrypt the following affine cipher. [3 points]

```
kqerejebcppcjcrkieacuzbkrvpkrbcibqcarbjcvfcupkriofkpacuzqepbkrxpei
ieabdkpbcpfcdccafieabdkpbcpfeqpkazbkrhaibkapcciburccdkdccjcidfuixp
afferbiczdfkabicbbenefcupjcvkabpcydccdpkbcocperkivkscpicbrkijpkabi
```

Solution: Using scripts from previous problems, it's simple to generate our frequency distribution.

Frequency Distribution



From the frequency, it's likely that `C` and `B` are ciphers for `e` and `t` respectively. Assume this is true and solve the system below.

$$\begin{bmatrix} 4 & 1 \\ 19 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \equiv \begin{bmatrix} 2 \\ 1 \end{bmatrix} \pmod{26} \quad \Rightarrow \quad \begin{matrix} a = 19 \\ b = 4 \end{matrix}$$

Attempt decrypting by implementing $D([19, 4], C) := 11(C - 4) \equiv p \pmod{26}$ in the script below.

```c
while (fread(&buffer, sizeof(char), 1, in) == 1)
{
    if (isalpha(buffer) == 0) { continue; }

    dummy = (buffer - 97) - 4;
    if (dummy < 0) { dummy += 26; }
    dummy = ((dummy * 11) % 26) + 97;
    printf("%c", dummy);
}

fclose(in);
```

By introducing spacing to the plaintext generated, we see that our guess worked since this yields the Canadian anthem!

```
o canada terre de nos aieux ton front est ceint de fleurons glorieux car ton bras
sait porter lepee il sait porter la croix ton histoire est une epopee des plus
brillants exploits et ta valeur de foi trempee protegera nos foyers et nos droits
```

2 Homework 02

Problem 4.2

Consider a Feistel cipher composed of sixteen rounds with a block length of 128 bits and a key length of 128 bits. Suppose that, for a given k, the key scheduling algorithm determines values for the first eight round keys, $k_1, k_2, \dots k_8$, and then sets

$$k_9 = k_8, k_{10} = k_7, k_{11} = k_6, \dots, k_{16} = k_1$$

Suppose you have a ciphertext c. Explain how, with access to an encryption oracle, you can decrypt c and determine m using just a single oracle query. This shows that such a cipher is vulnerable to a chosen plaintext attack. (An encryption oracle can be thought of as a device that, when given a plaintext, returns the corresponding ciphertext. The internal details of the device are not known to you and you cannot break open the device. You can only gain information from the oracle by making queries to it and observing its responses.)

Solution: If we are given a subkey sequence $k_1 k_2 \dots k_8 k_8 \dots k_2 k_1$, the Feistel encryption and decryption algorithms become identical. This is due to the symmetry of the subkey sequence. Thus, with access to an encryption oracle and knowledge of a ciphertext $\mathbf{c}$, we simply query $E(\mathbf{c}) = D(\mathbf{c}) = \mathbf{m}$.

Problem 4.5

For any block cipher, the fact that it is a nonlinear function is crucial to its security. To see this, suppose that we have a linear block cipher EL that encrypts 256-bit blocks of plaintext into 256-bit blocks of ciphertext. Let $EL(k, m)$ denote the encryption of a 256-bit message m under a key k (the actual bit length of k is irrelevant). Thus,

$$EL(k, [m_1 \oplus m_2]) = EL(k, m_1) \oplus EL(k, m_2) \text{ for all 128-bit patterns } m_1, m_2.$$

Describe how, with 256 chosen ciphertexts, an adversary can decrypt any ciphertext without knowledge of the secret key k. (A "chosen ciphertext" means that an adversary has the ability to choose a ciphertext and then obtain its decryption. Here, you have 256 plaintext/ciphertext pairs to work with and you have the ability to choose the value of the ciphertexts.)

Solution: Consider the set of ciphertexts $\mathcal{C}$. Let $c_i \in \{0,1\}^{128}$ with $1 \leq i \leq 128$ be the chosen ciphertexts. Each c_i has one in the i^{th} position, zeros elsewhere and a corresponding plaintext m_i. So

$$EL(k, \mathbf{m}) = EL\left(k, \bigoplus_{i=1}^{n} m_i\right) \overset{linearity}{=\!=\!=} \bigoplus_{i=1}^{n} EL(k, m_i) = \bigoplus_{i=1}^{n} c_i = \mathbf{c}$$

where $1 \leq n \leq 128$ describes encryption. More importantly, $\bigoplus_{i=1}^{n} m_i$ corresponds to $\mathbf{c}$ and the adversary can easily compute $\mathbf{m}$. Note that $EL(k, \mathbf{0}) = \mathbf{0}$.

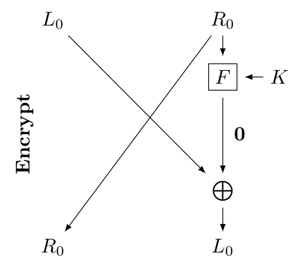
Problem 4.6

Suppose the DES F function mapped every 32-bit input R, regardless of the value of the input K, to **a** and **b**.

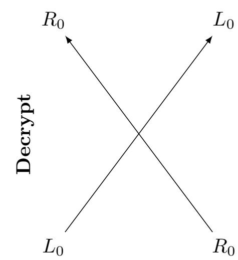
1. What function would DES then compute?
2. What would the decryption look like?

a. 32-bit string of zero

Solution: For message $\mathbf{M}$, we have $\mathrm{IP}(\mathbf{M}) = L_0 \| R_0$ and observe one round of DES encryption below [left].



Round	L_i	R_i
IP	L_0	R_0
1	R_0	L_0
2	L_0	R_0
16	L_0	R_0



The DES function described is $F(R) = \mathbf{0}$. It's clear from the table above [middle] that $\{L, R\}_0 = \{L, R\}_2 = \{L, R\}_{16}$. To finish encrypting, compute $\mathrm{IP}^{-1}(R_0 \| L_0) = \mathbf{C}$.

For decryption, perform at 32-bit swap on $\mathrm{IP}(\mathbf{C})$ which yields $L_0 \| R_0$. One round of decryption, shown above [right], is simply a swap. As with encryption, 16 rounds yields a circular result. To finish, compute $\mathrm{IP}^{-1}\big(L_0 \| R_0\big) = \mathbf{M}$.

b. R

Solution: The tables below, show encryption on the left and decryption on the right.

Round	L_i	R_i	Simplified
IP	L_0	R_0	
1	R_0	$L_0 \oplus R_0$	R_1
2	R_1	$R_0 \oplus R_1$	L_0
3	L_0	$R_1 \oplus L_0$	R_0
16	R_0	$L_0 \oplus R_0$	R_1

Round	L_i	Simplified	R_i
16	R_0		R_1
15	$R_0 \oplus R_1$	L_0	R_0
14	$L_0 \oplus R_0$	R_{14}	L_0
13	$R_{14} \oplus L_0$	R_0	R_{14}
12	$R_0 \oplus R_{14}$	L_0	R_0

For encryption, have $\mathrm{IP}(\mathbf{M}) = L_0 \| R_0$ with $F(X) = X$ i.e. the identify function. Notice,

$$R_{i+1} = L_i \oplus F(R_i) = L_i \oplus R_i.$$

Then $\{L, R\}_0 = \{L, R\}_3 = \{L, R\}_{15}$. To finish encrypting, compute $\mathrm{IP}^{-1}(R_1 \| R_0) = \mathbf{C}$.

To decrypt, certainly $F = F^{-1}$. Begin with $\mathrm{IP}(\mathbf{C}) = R_1 \| R_0$ followed by 32-bit swap which yiels $R_0 \| R_1$. In this case,

$$L_{i-1} = R_{i-1} \oplus R_i.$$

Then $\{L, R\}_{15} = \{L, R\}_{12} = \{L, R\}_0$. To finish decrypting, compute $\mathrm{IP}^{-1}\big(L_0 \| R_0\big) = \mathbf{M}$.

Problem 4.11

This problem provides a numerical example of encryption using a one-round version of DES. We start with the same bit pattern for the key K and the plaintext, namely:

Hexadecimal notation:	0 1 2 3 4 5 6 7 8 9 A B C D E F
Binary Notation:	0000 0001 0010 0011 0100 0101 0110 0111
	1000 1001 1010 1011 1100 1101 1110 1111

a. Derive K_1, the first-round subkey.

Solution: The code below computes K_1 according to Figure 4.5 in the text.

```
#define MASK_LOW_28  0xfffffff   // mask low bits
#define MASK_LOW_32 0xffffffff   // mask low bits
int main()
{
    uint64_t K0 = 0x0123456789abcdef;          // 64-S_of_Bits long

    // PART A
    // pc1 is 56-S_of_Bits long
    uint64_t PC1 = applyTransform("permuted_choice_one", K0, 64);

    uint64_t C0 = PC1 >> 28;            //  low 28-S_of_Bits of pc1
    uint64_t D0 = PC1 & MASK_LOW_28;    // high 28-S_of_Bits of pc1
    uint64_t C1 = leftShift(C0);
    uint64_t D1 = leftShift(D0);

    uint64_t concat = (C1 << 28) ^ D1;   // 56-S_of_Bits long

    // k1 is 48-S_of_Bits long
    uint64_t K1 = applyTransform("permuted_choice_two", concat, 56);
    printf("%llx\n", K1);
```

The output of our code is the 48-bit key

$$\mathbf{K_1} = \texttt{0x0b02679b49a5} = \texttt{000010 110000 001001 100111 100110 110100 100110 100101}$$

b. Derive L_0, R_0.

Solution: Since $\mathbf{M} = \mathbf{K}$, have:

```
uint64_t IP = applyTransform("initial_permutation", K0, 64);
uint32_t L0 = IP >> 32;
uint32_t R0 = IP & MASK_LOW_32;
printf("%x   %x\n", L0, R0);
```

$$\mathbf{L_0} = \texttt{0xcc00ccff} = \texttt{11001100 00000000 11001100 11111111}$$
$$\mathbf{R_0} = \texttt{0xf0aaf0aa} = \texttt{11110000 10101010 11110000 10101010}$$

c. Expand R_0 to get $E[R_0]$, where $E[\cdot]$ is the expansion function of Table S.1.

Solution: The result of `applyTransform("expansion", r_zero, 32)` is

$$\mathbf{E[R_0]} = \texttt{0x7a15557a1555} = \texttt{01111010 00010101 01010101 01111010 00010101 01010101}$$

d. Calculate $A = E[R_0] \oplus K_1$.

Solution: $\mathbf{A} = \texttt{0x711732e15cf0} = \texttt{01110001 00010111 00110010 11100001 01011100 11110000}$

e. Group the 48-bit result of (d) into sets of 6 bits and evaluate the corresponding S-box substitutions.

Solution: Regroup as follows.

$$\begin{array}{llll} \mathbf{B_1} = \texttt{011100} & \mathbf{B_2} = \texttt{010001} & \mathbf{B_3} = \texttt{011100} & \mathbf{B_4} = \texttt{110010} \\ \mathbf{B_5} = \texttt{111000} & \mathbf{B_6} = \texttt{010101} & \mathbf{B_7} = \texttt{110011} & \mathbf{B_8} = \texttt{110000} \end{array}$$

Then apply S-box substitutions accordingly.

```c
uint8_t S_of_Bi[8] = {0};    // elements are 4-bits long
uint8_t mask = 0x3f;         // mask 6 low bits
char box_name[6];

for (int j = 8; j > 0; j--)
{
    snprintf(box_name, 6, "sbox%d", j);
    S_of_Bi[j-1] = applySBox((A & mask), box_name);
    printf("%x ", S_of_Bi[j-1]);
    A = A >> 6;
}
printf("\n");
```

This gives us

$$\begin{array}{llll} \mathbf{S(B_1)} = \texttt{0000} & \mathbf{S(B_2)} = \texttt{1100} & \mathbf{S(B_3)} = \texttt{0010} & \mathbf{S(B_4)} = \texttt{0001} \\ \mathbf{S(B_5)} = \texttt{0110} & \mathbf{S(B_6)} = \texttt{1101} & \mathbf{S(B_7)} = \texttt{0101} & \mathbf{S(B_8)} = \texttt{0000}. \end{array}$$

f. Concatenate the results of (e) to get a 32-bit result, B.

Solution: Using the code below, we get: $\mathbf{B} = \texttt{0x0c216d50} = \texttt{00001100001000010110110101010000}$.

```c
uint32_t B_concat = S_of_Bi[0];

for (int j = 1; j < 8; j++)
{ B_concat = (B_concat << 4) ^ S_of_Bi[j]; }
printf("%x\n", B_concat);
```

g. Apply the permutation to get $P(B)$.

Solution: The result of `applyTransform("permutation", B, 32)` is

$$\mathbf{P(B)} = \texttt{0x921c209c} = \texttt{10010010000111000010000010011100}$$

h. Calculate $R1 = P(B) \oplus L_0$.

Solution: $\mathbf{R_1} = \mathbf{P(B)} \oplus \mathbf{L_0} = \texttt{0x5e1cec63} = \texttt{01011110000111001110110001100011}$.

i. Write down the ciphertext.

Solution: Perform a 32-bit swap on $\mathbf{L_1} \| \mathbf{R_1}$ and apply the inverse permutation as shown below.

```
uint64_t    swap = (R1 << 32) ^ R0;
uint64_t cipher = applyTransform("ip_inverse", swap, 64);
printf("%llx\n", cipher);
```

Thus, the ciphertext is

```
0000 0001 0110 0011 0101 0100 0111 0110 1101 1000 1010 1111 1100 1101 1010 1110
                            0x 0 1 6 3 5 4 7 6 D 8 A F C D A E
```

Solution: Finally, we made use of the three functions defined below.

```
uint64_t applyTransform(char *name, uint64_t x, int x_size)
{
    FILE *fp = fopen(name, "r");
    int buffer = 0;
    uint8_t      nth_bit = 0;    // big endian
    uint64_t transformed = 0;

    while( fscanf(fp, "%d", &buffer) == 1 )
    {
        nth_bit = (x >> (x_size - buffer)) & 1;
        transformed = (transformed << 1) ^ nth_bit;
    }

    return transformed;
}

uint64_t leftShift(uint64_t x)
{
    uint8_t high_bit = (x >> (28 - 1)) & 1;
    uint64_t shifted = ((x << 1) & MASK_LOW_28) ^ high_bit;
    return shifted;
}

uint8_t applySBox(uint8_t partition, char *name)
{
    FILE *fp = fopen(name, "r");
    int buffer = 0;

    // 0x20 masks highest bit of 6-bit partition arg
    uint8_t row = ((partition & 0x20) >> 4) ^ (partition & 1);
    // 0x1e masks middle 4 bits
    uint8_t col = (partition & 0x1e) >> 1;
    int location = 16*row + col + 1;

    for (int i = 0; i < location; i++)
    { fscanf(fp, "%d", &buffer); }
```

Problem 4.14

a. Let X' be the bitwise complement of X. Prove that if the complement of the plaintext block is taken and the complement of an encryption key is taken, then the result of DES encryption with these values is the complement of the original ciphertext. That is,

$$\begin{array}{rcl} \text{If} \quad Y & = & E(K, X) \\ \text{Then} \quad Y' & = & E(K', X') \end{array}$$

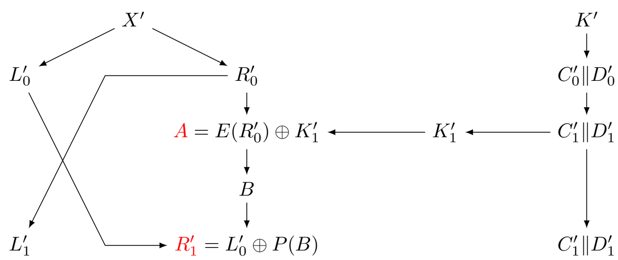
Solution: To begin, notice that

$$X = L_0 \| X_0 \quad \Rightarrow \quad X' = L_0' \| R_0'.$$

On the other hand, let P be a bitwise permutation such that $P(b_i) = b_j$ where b_i, b_j are the i^{th} and j^{th} bits respectively. It follows that $P(b_i') = b_j'$. Then

$$PC1(K) = C_0 \| D_0 \quad \Rightarrow \quad PC1(K') = C_0' \| D_0'.$$

Likewise, applying a left circular shift and $PC2$ yield the expected results. See diagram below for reference.



Now we address the more involved portion. At the first XOR, we have $E(R_0') \oplus K_1'$. Below [left], we show that $A = E(R) \oplus K = E(R') \oplus K'$.

E	K	$E \oplus K$	E'	K'	$E' \oplus K'$
0	0	0	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

L	P	$L \oplus P$	$(L \oplus P)'$	L'	$L' \oplus P$
0	0	0	1	1	1
0	1	1	0	1	0
1	0	1	0	0	0
1	1	0	1	0	1

Applying the S-boxes then yields the same result B and for the second XOR, we see above [right] that $L' \oplus P = (L \oplus P)' = R'$.

It follows that Round 16 would yield L_{16}' and R_{16}'. Finally, since IP^{-1} is a permutation as well

$$IP^{-1}(R_{16}' \| L_{16}') = (IP^{-1}(R_{16} \| L_{16}))' = Y'$$

Thus, proving if $Y = E(K, X)$, then $Y' = E(K', X')$.

b. It has been said that a brute-force attack on DES requires searching a key space of 2^{56} keys. Does the result of part (a) change that?

Solution: In any set of 2^n bit keys, half of them are complements of the other half. This makes it computationally inexpensive to find Y' if a brute-force attack has already been performed to find Y. Hence, the true key space is actually $2^{56}/2 = 2^{55}$, which is still large.