DES

#include <iostream>

#include <string>

#include <stdio.h>

#include <math.h>

using namespace std;

void setB(int &i , int in,int loc){

int hl=0;

hl=1<<31-(loc-1);

if (in)

i=i|hl;

else

i=i&~hl;

}

int getB(int i ,int loc){

i=i>>31-(loc-1);

return i&1;

}

int getV(int in,int b,int e){

int sum=0;

int t=0;

for (int i=e;i>=b;i--){

sum+=getB(in,i)<<t;

t+=1;

}

return sum;

}

void pKEY(int &l,int &r){

int map[56]={57,49,41,33,25,17,9,1,58,50,42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,30,22,14,6,61,53,45,37,29,21,13,5,28,20,12,4};

int hl=l;

int hr=r;

for (int t=0;t<56;t++){

if (map[t]>32){

if (t>27){

setB(hr,getB(r,map[t]-32),t-27);

}

else{

setB(hl,getB(r,map[t]-32),t+1);

}

}

else{

if (t>27){

setB(hr,getB(l,map[t]),t-27);

}

else{

setB(hl,getB(l,map[t]),t+1);

}

}

}

r=hr;

l=hl;

r=r>>4;

r=r<<4;

l=l>>4;

l=l<<4;

}

void pSUBKEY(int &l,int &r){

int map[48]={14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,37,47,55,30,40,51,45,33,48,44,49,39,56,34,53,46,42,50,36,29,32

};

int hl=l;

int hr=r;

for (int t=0;t<48;t++){

if (map[t]>28){

if (t>23){

setB(hr,getB(r,map[t]-28),t-23);

}

else{

setB(hl,getB(r,map[t]-28),t+1);

}

}

else{

if (t>23){

setB(hr,getB(l,map[t]),t-23);

}

else{

setB(hl,getB(l,map[t]),t+1);

}

}

}

r=hr;

l=hl;

r=r>>8;

r=r<<8;

l=l>>8;

l=l<<8;

}

void pIP(int &l,int &r){

int map[64]={58,50,42,34,26,18,10,2,60,52,44,36,28,20,12,4,62,54,46,38,30,22,14,6,64,56,48,40,32,24,16,8,57,49,41,33,25,17,9,1,59,51,43,35,27,19,11,3,61,53,45,37,29,21,13,5,63,55,47,39,31,23,15,7};

int hl=l;

int hr=r;

for (int t=0;t<64;t++){

if (map[t]>32){

if (t>31){

setB(hr,getB(r,map[t]-32),t-31);

}

else{

setB(hl,getB(r,map[t]-32),t+1);

}

}

else{

if (t>31){

setB(hr,getB(l,map[t]),t-31);

}

else{

setB(hl,getB(l,map[t]),t+1);

}

}

}

r=hr;

l=hl;

}

void funcE(int &l,int &r){

int map[48]={32,1,2,3,4,5,4,5,6,7,8,9,8,9,10,11,12,13,12,13,14,15,16,17,16,17,18,19,20,21,20,21,22,23,24,25,24,25,26,27,28,29,28,29,30,31,32,1};

int hl=l;

int hr=r;

for (int t=0;t<48;t++){

if (t>23){

setB(hr,getB(r,map[t]),t-23);

}

else{

setB(hl,getB(r,map[t]),t+1);

}

}

r=hr;

l=hl;

r=r>>8;

r=r<<8;

l=l>>8;

l=l<<8;

}

int funcS(int num,int row,int column){\

int S[512]={14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,

0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,

4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,

15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13,

15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,

3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,

0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,

13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9,

10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,

13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,

13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,

1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12,

7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,

13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,

10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,

3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14,

2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,

14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,

4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,

11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3,

12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,

10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,

9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,

4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13,

4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,

13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,

1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,

6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12,

13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,

1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,

7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,

2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11};

return (S[(num-1)\*64+row\*16+column]);

return 0;

}

void finalP(int &l,int &r){

int map[64]={40,8,48,16,56,24,64,32,39,7,47,15,55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,49,17,57,25};

int hl=l;

int hr=r;

for (int t=0;t<64;t++){

if (map[t]>32){

if (t>31){

setB(hr,getB(r,map[t]-32),t-31);

}

else{

setB(hl,getB(r,map[t]-32),t+1);

}

}

else{

if (t>31){

setB(hr,getB(l,map[t]),t-31);

}

else{

setB(hl,getB(l,map[t]),t+1);

}

}

}

r=hr;

l=hl;

}

int lastP(int s){

int map[32]={16,7,20,21,29,12,28,17,1,15,23,26,5,18,31,10,2,8,24,14,32,27,3,9,19,13,30,6,22,11,4,25};

int ss=0;

for (int t=0;t<32;t++){

setB(ss,getB(s,map[t]),t+1);

}

return ss;

}

int funcF(int r,int kl,int kr){

int er=r,el=0;

funcE(el,er);

kl=kl^el;

kr=kr^er;

int b=0;

int s=0;

for (int i=8;i>0;i--){

if (i>4){

b=getV(kr,i\*6-24-5,i\*6-24);

}else {

b=getV(kl,i\*6-5,i\*6);

}

int column=getV(b,28,31);

int row=getV(getV(b,32,32)|(getV(b,27,27)<<1),30,32);

//cout<<row<<"||"<<column<<endl;

s+=funcS(i,row,column)<<(8-i)\*4;

//cout<<s<<endl;

}

return lastP(s);

}

int shiftL(int inp,int size,int rep){

for (int i=0;i<rep;i++){

int bit=0;

bit=getB(inp,1);

bit=bit<<32-size;

inp=inp<<1;

inp=inp|bit;

}

return inp;

}

string cInt(long long int number)

{

char t[8];

for (int i=0;i<8;i++){

t[i]=((char)(number>>i\*8));

}

string s(t);

return s;

}

void Cipher(long long int msg,long long int key){

int f=0;

int keyr=(int)(key);

int keyl=(int)(key>>32);

cout<<"Key Left Int:= "<<keyl<<" Key Right Int:= "<<keyr<<endl;

int c0=0,d0=0;

pKEY(keyl,keyr);

c0=keyl;

d0=keyr;

int kl=0,kr=0;

int r0=(int)(msg);

int l0=(int)(msg>>32);

cout<<"Msg Left Int:= "<<l0<<" Msg Right Int:= "<<r0<<endl;

pIP(l0,r0);

for (int i=1;i<17;i++){

int t=2;

if (i==1||i==2||i==9||i==16){

t=1;

}

c0=shiftL(c0,28,t);

d0=shiftL(d0,28,t);

kl=c0;

kr=d0;

pSUBKEY(kl,kr);

int pl0=l0;

l0=r0;

r0=pl0^funcF(r0,kl,kr);

}

int rt=0;

rt=r0;

r0=l0;

l0=rt;

finalP(l0,r0);

long long int final=l0;

final=final<<32;

final+=r0;

cout<<"[ "<<"Cipher int:#### "<<hex<<final<<" ####\nChipher Left INT:= "<<l0<<" Right INT:= "<<r0 <<endl;

}

int main (){

long long int keya[8]={56,98,121,116,101,107,101,121};

long long int msga[8]={109,101,115,115,97,103,101,46};

long long int msg=0;//=7882833662174520622;

long long int key=0;//=4062943354666313081;

for (int i=0;i<8;i++){

msg+=(msga[i]<<((7-i)\*8));

}

for (int i=0;i<8;i++){

key+=(keya[i]<<((7-i)\*8));

}

cout<<"[ Your Msg: "<<msg<<" ]"<<endl;

cout<<"[ Your Key: "<<key<<" ]"<<endl;

// Msg 6D6573736167652E //

// Key 38627974656B6579 //

//Cipher hex 7CF45E129445D451 //

//cipher int 9003924984615523409 //

Cipher(msg,key);

return 0;

}

Multiplicative inverse

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

///returns the multiplicative inverse of the given number

///in modulo 26

int mulInverse(int x)

{

int testers[]= {1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25};

int numEle = 12;

int i = 0;

for(i = 0; i < numEle; i++) {

if ((testers[i] \* x) % 26 == 1) {

return testers[i];

}

}

return -1;

}

///returns the determinant of a 3 X 3 matrix

int det(int mat[][3])

{

return

mat[0][0]\*(mat[1][1] \* mat[2][2] - mat[1][2]\*mat[2][1])

- mat[0][1]\*(mat[1][0] \* mat[2][2] - mat[1][2]\*mat[2][0])

+ mat[0][2]\*(mat[1][0] \* mat[2][1] - mat[1][1]\*mat[2][0]);

}

///calculates the transpose of a 3 X 3 matrix

void transpose(int \*\*mat)

{

int i, j;

for(i = 0; i < 3; i++) {

for(j = i + 1; j < 3; j++) {

mat[i][j] ^= mat[j][i];

mat[j][i] ^= mat[i][j];

mat[i][j] ^= mat[j][i];

}

}

}

///calculates adjoint of a matrix

int\*\* adjoint(int mat[][3])

{

int \*\*res;

int i = 0;

res = (int \*\*)malloc(3 \* sizeof(int \*));

for(i = 0; i < 3; i++) {

res[i] = (int \*)malloc(3 \* sizeof(int));

}

res[0][0] = mat[1][1] \* mat[2][2] - mat[1][2]\*mat[2][1];

res[0][1] = -(mat[1][0] \* mat[2][2] - mat[1][2]\*mat[2][0]);

res[0][2] = mat[1][0] \* mat[2][1] - mat[1][1]\*mat[2][0];

res[1][0] = -(mat[0][1] \* mat[2][2] - mat[0][2]\*mat[2][1]);

res[1][1] = (mat[0][0] \* mat[2][2] - mat[0][2]\*mat[2][0]);

res[1][2] = -(mat[0][0] \* mat[2][1] - mat[0][1]\*mat[2][0]);

res[2][0] = mat[0][1] \* mat[1][2] - mat[0][2]\*mat[1][1];

res[2][1] = -(mat[0][0] \* mat[1][2] - mat[0][2]\*mat[1][0]);

res[2][2] = mat[0][0] \* mat[1][1] - mat[0][1]\*mat[1][0];

transpose(res); //adjoint is the transpose of cofactor

return res;

}

///calculates each\_element % N

void matrixMod(int \*\*mat, int N)

{

int i, j;

for(i = 0; i < 3; i++) {

for(j = 0; j < 3; j++) {

mat[i][j] = mat[i][j] - floor((double)(mat[i][j]) / N) \* N; //mat[i][j] % 26

}

}

}

///returns the inverse key matrix

int\*\* inverseKeyMatrix(int K[][3])

{

int detK, detKInv, \*\*adjK;

int i, j;

//calculating inverse key matrix for hill cipher

//step 1 : calculate determinant

detK = det(K);

//step 2 : calculate multiplicative inverse of the matrix

detKInv = mulInverse(detK % 26);

//step 3 : calculate the adoint matrix

adjK = adjoint(K);

//step 4 : multiply each element of the adjoint by the multiplicative

//inverse of the determinant

for(i = 0; i < 3; i++) {

for(j = 0; j < 3; j++) {

adjK[i][j] = adjK[i][j] \* detKInv;

}

}

//step 5 : take mod

matrixMod(adjK ,26);

return adjK;

}

///this function takes a string and returns an integer array that maps

char\* encrypt(int K[][3], char \*msg)

{

int charVec[3], i, j, k;

int l = strlen(msg);

int \*encI = malloc(l \* sizeof(int)); //the integer representation of the cipher text

char \*cipherText = malloc(l + 1);

for(i = 0; i < l; i += 3) {

charVec[0] = msg[i] - 97;

charVec[1] = msg[i + 1] - 97;

charVec[2] = msg[i + 2] - 97;

for(j = 0; j < 3; j++) {

encI[i + j] = 0;

for(k = 0; k < 3; k++) {

encI [i + j] = encI[i + j] + K[j][k] \* charVec[k];

}

encI[i + j] %= 26;

cipherText[i + j] = encI[i + j] + 97;

}

}

cipherText[l] = '\0';

return cipherText;

}

///takes the inverse key matrix and the cipher text and returns the plain text

char\* decrypt(int \*\*iK, char \*cipher)

{

int charVec[3], i, j, k;

int l = strlen(cipher); //the null

int \*decI = malloc(l \* sizeof(int));

char \*plainText = malloc(l + 1);

for(i = 0; i < l; i += 3) {

charVec[0] = cipher[i] - 97;

charVec[1] = cipher[i + 1] - 97;

charVec[2] = cipher[i + 2] - 97;

for(j = 0; j < 3; j++) {

decI[i + j] = 0;

for(k = 0; k < 3; k++) {

decI[i + j] = decI[i + j] + iK[j][k] \* charVec[k];

}

decI[i + j] %= 26;

plainText[i + j] = decI[i + j] + 97;

}

}

plainText[l] = '\0';

return plainText;

}

int main()

{

int K[][3] = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}}, \*\*invK;

int i, l, j;

char msg[1000], choice;

while(1) {

printf("\nDo you want to enter a key matrix or use the default?(e : enter/ d : default/ x : exit)\n> ");

scanf("%c", &choice);

getchar(); //eat the newline

if (choice == 'x') {

return 0;

} else if (choice == 'e') {

for (i = 0; i < 3; i++) {

printf("\nEnter %d row : ", i + 1);

scanf("%d%d%d",&K[i][0], &K[i][1], &K[i][2]);

}

break;

} else if(choice == 'd') {

break;

}

else {

printf("\nSorry I don't understand that");

}

}

invK = inverseKeyMatrix(K);

printf("\n\t\tKey Matrix\n\t");

for(i = 0; i < 3; i++) {

for(j = 0; j < 3; j++) {

printf("%6d", K[i][j]);

}

printf("\n\t");

}

printf("\n\n\t\tInverse Key Matrix\n\t");

for(i = 0; i < 3; i++) {

for(j = 0; j < 3; j++) {

printf("%6d", invK[i][j]);

}

printf("\n\t");

}

printf("\n");

printf("Enter your message : ");

scanf("%s",msg);

l = strlen(msg);

if (l % 3 != 0) { //make the length / 3

for(i = 0; i < (3 - l%3); i++) {

msg[l + i] = 'x';

}

l = l + 3 - l % 3;

msg[l] = '\0';

}

char \*enc = encrypt(K, msg);

char \*dec = decrypt(invK, enc);

printf("\nencryption : %s\n\ndecryption : %s\n", enc, dec);

return 0;

}

from collections import Counter

import operator

alphamap = {'a': 0, 'c': 2, 'b': 1, 'e': 4, 'd': 3, 'g': 6, 'f': 5, 'i': 8, 'h': 7, 'k': 10, 'j': 9, 'm': 12, 'l': 11, 'o': 14, 'n': 13, 'q': 16, 'p': 15, 's': 18, 'r': 17, 'u': 20, 't': 19, 'w': 22, 'v': 21, 'y': 24, 'x': 23, 'z': 25}

reversemap = {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'}

def count\_freq(string):

return dict(Counter(string))

def max\_freq\_key(dict\_key):

return max(dict\_key.iteritems(), key=operator.itemgetter(1))[0]

def determine\_key(c, m):

return (alphamap[c] - alphamap[m] + 26)%26

def return\_message(c, k):

return ''.join([reversemap[(alphamap[i] - k+26)%26] for i in c])

if \_\_name\_\_=="\_\_main\_\_":

print "Enter encrypted text"

raw\_cipher = raw\_input().strip()

freq\_letter = count\_freq(raw\_cipher)

max\_freq\_letter\_order = ['e', 't', 'a', 'o', 'i', 'n', 's', 'r', 'h', 'd', 'l', 'u', 'c', 'm', 'f', 'y', 'w', 'g']

len\_max\_freq\_list = len(max\_freq\_letter\_order)

i = 0

cl = max\_freq\_key(freq\_letter)

while i<len\_max\_freq\_list:

ml = max\_freq\_letter\_order[i]

possible\_key = determine\_key(cl, ml)

print "possible key", possible\_key

print "message decrypted", return\_message(raw\_cipher, possible\_key)

i+=1