Tristan Erney

October 25th, 2021

Intro to Cryptology

Programming Assignment 1

1)



2)

a)



b)



3)

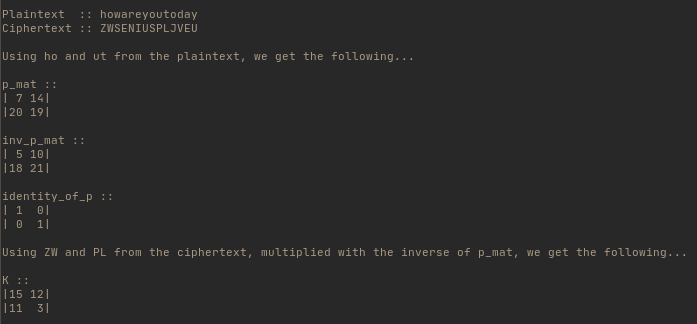
a)



b)



4)



FrequencyCount.cpp

#include <iostream>

#include <iomanip>

#include <stdlib.h>

#include <string.h>

struct CipherFreqTable {

char letter[26];

float freq[26];

};

int mod(int x, int y) {

if (x < 0) {

while (x < 0) x += y;

return x;

} else {

return x % y;

}

}

CipherFreqTable\* letter\_freq(const char\* p) {

CipherFreqTable\* t = new CipherFreqTable;

for (int i = 0; i < 26; i += 1) {

t->letter[i] = 'A' + i;

t->freq[i] = 0;

}

for (int i = 0; i < strlen(p); i += 1) {

int j = p[i] - 'A';

t->freq[j] += 1;

}

return t;

}

char\* shift\_decrypt(const char\* c, int k) {

int c\_size = (int)strlen(c);

char\* p = (char\*)malloc(sizeof(c) \* c\_size + 1);

p[c\_size] = 0;

for (size\_t i = 0; i < c\_size; i += 1) {

p[i] = mod((c[i] - 'A') - k, 26) + 'a';

}

return p;

}

int main() {

const char\* ciphertext = "LCLLEWLJAZLNNZMVYIYLHRMHZA";

// frequency analysis

int k = -1;

CipherFreqTable\* freq\_table = letter\_freq(ciphertext);

int score = 0;

char letter = ' ';

for (size\_t j = 0; j < 26; j += 1) {

if (score < freq\_table->freq[j]) {

score = freq\_table->freq[j];

letter = freq\_table->letter[j];

}

}

letter += 32;

std::cout << "Most Frequent Letter :: " << letter << "\n";

k = letter - 'e';

char\* plaintext = shift\_decrypt(ciphertext, k);

std::cout << "Key :: " << k << "\n";

std::cout << "Plaintext :: " << shift\_decrypt(ciphertext, k) << "\n";

return 0;

}

HillCipher.cpp

#include <iostream>

#include <iomanip>

#include <string.h>

#include <stdlib.h>

int mod(int x, int m) {

if (x < 0) {

while (x < 0) x += m;

return x;

} else {

return x % m;

}

}

int mult\_inv(int i) {

int k = 0;

int j = 0;

for (j = 0; j < 26; j += 1) {

k = (i \* j);

if (k % 26 == 1) break;

}

return j;

}

struct Mat22 {

int data[2][2];

};

Mat22\* mat22\_set(int a, int b, int c, int d) {

Mat22\* m = (Mat22\*)malloc(sizeof(Mat22));

m->data[0][0] = a;

m->data[0][1] = b;

m->data[1][0] = c;

m->data[1][1] = d;

return m;

}

void mat22\_print(Mat22\* m) {

std::cout << "|" << std::setw(2) << (m->data[0][0]) << " " << std::setw(2) << (m->data[0][1]) << "|";

std::cout << "\n";

std::cout << "|" << std::setw(2) << (m->data[1][0]) << " " << std::setw(2) << (m->data[1][1]) << "|";

std::cout << "\n";

}

Mat22\* mat22\_mul(Mat22\* m, Mat22\* n) {

Mat22\* o = (Mat22\*)malloc(sizeof(Mat22));

o->data[0][0] = mod((m->data[0][0] \* n->data[0][0]) + (m->data[0][1] \* n->data[1][0]), 26);

o->data[0][1] = mod((m->data[0][0] \* n->data[0][1]) + (m->data[0][1] \* n->data[1][1]), 26);

o->data[1][0] = mod((m->data[1][0] \* n->data[0][0]) + (m->data[1][1] \* n->data[1][0]), 26);

o->data[1][1] = mod((m->data[1][0] \* n->data[0][1]) + (m->data[1][1] \* n->data[1][1]), 26);

return o;

}

Mat22\* mat22\_adj(Mat22\* m) {

Mat22\* adj = (Mat22\*)malloc(sizeof(Mat22));

adj->data[0][0] = m->data[1][1];

adj->data[0][1] = m->data[0][1] \* -1;

adj->data[1][0] = m->data[1][0] \* -1;

adj->data[1][1] = m->data[0][0];

return adj;

}

// inv = (1/(ad - bc))[a1 a2]

// [b1 b2]

Mat22\* mat22\_inv(Mat22\* m) {

Mat22\* inv = mat22\_adj(m);

int denom = mod((m->data[0][0] \* m->data[1][1]) - (m->data[0][1] \* m->data[1][0]), 26);

int inverse = mult\_inv(denom);

inv->data[0][0] = mod(inv->data[0][0] \* inverse, 26);

inv->data[0][1] = mod(inv->data[0][1] \* inverse, 26);

inv->data[1][0] = mod(inv->data[1][0] \* inverse, 26);

inv->data[1][1] = mod(inv->data[1][1] \* inverse, 26);

return inv;

}

// C = E(P, K) = P \* K (mod 26)

char\* hillcipher\_encrypt(char\* P, Mat22\* K) {

int p\_size = (int)strlen(P);

char\* C = (char\*)malloc(sizeof(char) \* p\_size + 1);

C[p\_size] = 0;

// encode C = P\*K

int i = 0;

int j = 0;

for (; i < p\_size; i += 4) {

Mat22\* p\_mat = mat22\_set(P[i + 0] - 'a',

(i + 1 < p\_size) ? P[i + 1] - 'a' : 0,

(i + 2 < p\_size) ? P[i + 2] - 'a' : 0,

(i + 3 < p\_size) ? P[i + 3] - 'a' : 0);

Mat22\* c\_mat = mat22\_mul(p\_mat, K);

if (i < p\_size) C[j++] = c\_mat->data[0][0] + 'A';

if (i + 1 < p\_size) C[j++] = c\_mat->data[0][1] + 'A';

if (i + 2 < p\_size) C[j++] = c\_mat->data[1][0] + 'A';

if (i + 3 < p\_size) C[j++] = c\_mat->data[1][1] + 'A';

free(p\_mat);

free(c\_mat);

}

return C;

}

char\* hillcipher\_decrypt(char\* C, Mat22\* inv\_K) {

int c\_size = (int)strlen(C);

char\* P = (char\*)malloc(sizeof(char)\* c\_size + 1);

P[c\_size] = 0;

// add padding to C

int pad = 0;

while ((c\_size + pad) % 4 != 0) {

pad += 1;

C = (char\*)realloc(C, c\_size + pad);

C[c\_size + pad - 1] = ' ';

}

// decode P = C\*K^(-1)

int i = 0;

int j = 0;

for (; i < c\_size; i += 4) {

Mat22\* c\_mat = mat22\_set((i + 0 < c\_size) ? C[i + 0] - 'A' : 0,

(i + 1 < c\_size) ? C[i + 1] - 'A' : 0,

(i + 2 < c\_size) ? C[i + 2] - 'A' : 0,

(i + 3 < c\_size) ? C[i + 3] - 'A' : 0);

Mat22\* p\_mat = mat22\_mul(c\_mat, inv\_K);

if (i + 0 < c\_size) P[j++] = p\_mat->data[0][0] + 'a';

if (i + 1 < c\_size) P[j++] = p\_mat->data[0][1] + 'a';

if (i + 2 < c\_size) P[j++] = p\_mat->data[1][0] + 'a';

if (i + 3 < c\_size) P[j++] = p\_mat->data[1][1] + 'a';

free(c\_mat);

free(p\_mat);

}

return P;

}

int main() {

{

Mat22\* K = mat22\_set(9, 4, 5, 7);

char P[] = "meetmeattheusualplaceattenratherthaneightoclock";

char\* C = hillcipher\_encrypt(P, K);

std::cout << "Plaintext :: " << P << "\n";

std::cout << "Ciphertext :: " << C << "\n\n";

free(K);

free(C);

}

std::cout << "--------------------------------------------------\n\n";

{

Mat22\* K = mat22\_set(9, 13, 2, 3);

char\* C = (char\*)malloc(sizeof(char) \* 6); // this needs to be heap allocated so we can manipulate it in decrypt

memcpy(C, "YIFZMA", 6);

Mat22\* inv\_K = mat22\_inv(K);

Mat22\* identity = mat22\_mul(K, inv\_K);

char\* P = hillcipher\_decrypt(C, inv\_K);

std::cout << "Ciphertext :: " << C << "\n";

std::cout << "Plaintext :: " << P << "\n\n";

free(inv\_K);

free(K);

free(P);

}

std::cout << "--------------------------------------------------\n\n";

{

// known plaintext attack

char P[] = "howareyoutoday";

char C[] = "ZWSENIUSPLJVEU";

std::cout << "Plaintext :: " << P << "\n";

std::cout << "Ciphertext :: " << C << "\n\n";

// load p\_mat and see if it is invertible

Mat22\* p\_mat = mat22\_set(P[0] - 'a', P[1] - 'a', P[8] - 'a', P[9] - 'a');

Mat22\* inv\_p\_mat = mat22\_inv(p\_mat);

Mat22\* identity\_of\_p = mat22\_mul(p\_mat, inv\_p\_mat);

std::cout << "Using " << P[0] << P[1] << " and " << P[8] << P[9] << " from the plaintext, we get the following...\n\n";

std::cout << "p\_mat :: \n";

mat22\_print(p\_mat);

std::cout << "\n";

std::cout << "inv\_p\_mat :: \n";

mat22\_print(inv\_p\_mat);

std::cout << "\n";

std::cout << "identity\_of\_p :: \n";

mat22\_print(identity\_of\_p);

std::cout << "\n";

// since inv\_p is invertible, we will use inv\_p\_mat \* c\_mat to get K

// build a c\_mat

Mat22\* c\_mat = mat22\_set(C[0] - 'A', C[1] - 'A', C[8] - 'A', C[9] - 'A');

Mat22\* K = mat22\_mul(inv\_p\_mat, c\_mat);

std::cout << "Using " << C[0] << C[1] << " and " << C[8] << C[9] << " from the ciphertext, multiplied with the inverse of p\_mat, we get the following...\n\n";

std::cout << "K :: \n";

mat22\_print(K);

std::cout << "\n";

}

return 0;

}

TranspositionCipher.cpp

#include <iostream>

#include <vector>

#include <stdlib.h>

#include <string.h>

// expand string by adding 'x''s to the end if needed

char\* expand\_string(char\* str, int desired\_size) {

int str\_size = (int)strlen(str);

char\* s = (char\*)malloc(sizeof(char) \* desired\_size + 1);

s[desired\_size] = 0;

for (int i = 0; i < desired\_size; i += 1) {

s[i] = (i < str\_size) ? str[i] : /\* (('x' - 'a' + i) % 26) + 'a' + 1 \*/ 'x';

}

return s;

}

/\*

Row Transposition Encryption:

encrypt by building a table of (k\_size x (c\_size / k\_size))

and rebuild a new ciphertext string by reading the table

by column.

meetmeaftertogaparty

key = {3, 5, 1, 6, 2, 4}

TBL:

m e e t m e

a f t e r t

o g a p a r

t y x y z w

-----------

3 5 1 6 2 4

etaxmrazmaotetrwefgytepy

\*/

char\* row\_transposition\_encrypt(char\* \_P, int\* K, int k\_size) {

int p\_size = strlen(\_P);

int pad = ((p\_size % k\_size) != 0) ? k\_size - (p\_size % k\_size) : 0;

int c\_size = p\_size + pad;

char\* P = expand\_string(\_P, c\_size);

char\* C = (char\*)malloc(c\_size + 1);

C[c\_size] = 0;

char\*\* TBL = (char\*\*)malloc(sizeof(char\*) \* (c\_size / k\_size));

for (int i = 0; i < (c\_size / k\_size); i += 1) {

char\* p\_ptr = P + (i \* k\_size);

TBL[i] = (char\*)malloc(sizeof(char) \* k\_size);

for (int j = 0; j < k\_size; j += 1) {

TBL[i][j] = p\_ptr[j];

}

}

int k = 0;

for (int i = 0; i < k\_size; i += 1) {

// get keys in order of 0 to k\_size - 1

int c = 0;

for (int j = 0; j < k\_size; j += 1) {

if (K[j] == i + 1) {

c = j;

break;

}

}

for (int j = 0; j < (c\_size / k\_size); j += 1) {

C[k++] = TBL[j][c];

}

}

return C;

}

/\*

Row Transposition Decrypt:

decrypt by building a table of (k\_size x (c\_size / k\_size))

by reading in the table by each (c\_size / k\_size) sized chunk

ordered by the key, which is then read into a new plaintext

string by reading the table left-right top-bottom.

key = {3, 5, 1, 6, 2, 4}

#3 #5 #1 #6 #2 #4

amrt moep eaeg rtfy tzty xawe

e t a x m r

a z m a o t

e t r w e f

g y t e p y

etaxmrazmaotetrwefgytepy

\*/

char\* row\_transposition\_decrypt(char\* C, int\* K, int k\_size) {

size\_t c\_size = strlen(C);

size\_t p\_size = c\_size;

char\* P = (char\*)malloc(p\_size + 1);

P[p\_size] = 0;

// create table

char\*\* TBL = (char\*\*)malloc(sizeof(char\*) \* (c\_size / k\_size));

for (int i = 0; i < (c\_size / k\_size); i += 1) {

char\* c\_ptr = C + (i \* k\_size);

TBL[i] = (char\*)malloc(sizeof(char) \* k\_size);

}

// read data into table

for (int i = 0; i < k\_size; i += 1) {

char\* c\_ptr = C + (i \* (c\_size / k\_size));

for (int j = 0; j < (c\_size / k\_size); j += 1) {

TBL[j][K[i] - 1] = c\_ptr[j];

}

}

int k = 0;

for (int i = 0; i < (c\_size / k\_size); i += 1) {

for (int j = 0; j < k\_size; j += 1) {

P[k++] = TBL[i][j];

}

}

return P;

}

int main() {

{

int key[] = {4, 3, 1, 2, 5, 6, 7};

char P[] = "attackpostponeduntiltwoam";

char\* C1 = row\_transposition\_encrypt(P, key, 7);

char\* C = row\_transposition\_encrypt(C1, key, 7);

std::cout << "Key :: ";

for (int i = 0; i < 7; i += 1) {

std::cout << key[i] << " ";

}

std::cout << "\n";

std::cout << "Plaintext :: " << P << "\n";

std::cout << "Ciphertext :: " << C << "\n\n";

}

{

int key[] = {3, 5, 1, 6, 2, 4};

char C[] = "amrtmoepeaegrtfytztyxawe";

char\* P = row\_transposition\_decrypt(row\_transposition\_decrypt(C, key, 6), key, 6);

std::cout << "Key :: ";

for (int i = 0; i < 6; i += 1) {

std::cout << key[i] << " ";

}

std::cout << "\n";

std::cout << "Ciphertext :: " << C << "\n";

std::cout << "Plaintext :: " << P << "\n\n";

}

return 0;

}