6. Write a C program for ciphertext has been generated with an affine cipher. The most frequent letter of the ciphertext is “B,” and the second most frequent letter of the ciphertext is “U.”Break this code.

Code:

/\*\*

\* Program to break an affine cipher where:

\* - 'B' is the most frequent ciphertext letter (corresponding to plaintext 'E')

\* - 'U' is the second most frequent ciphertext letter (corresponding to plaintext 'T')

\*

\* Affine cipher formula: C = (a\*p + b) mod 26

\* Where:

\* - C is the ciphertext letter value (0-25)

\* - p is the plaintext letter value (0-25)

\* - a and b are the cipher keys

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

// Function to calculate GCD using Euclidean algorithm

int gcd(int a, int b) {

int temp;

while (b != 0) {

temp = b;

b = a % b;

a = temp;

}

return a;

}

// Function to calculate modular multiplicative inverse

int modInverse(int a, int m) {

a = a % m;

for (int x = 1; x < m; x++) {

if ((a \* x) % m == 1) {

return x;

}

}

return -1; // No inverse exists

}

// Function to solve for 'a' and 'b' using known letter mappings

int solveAffineParameters(int c1, int p1, int c2, int p2, int \*a, int \*b) {

// We have the system of equations:

// c1 = (a\*p1 + b) mod 26

// c2 = (a\*p2 + b) mod 26

// Subtract the equations to eliminate b

// c1 - c2 = a(p1 - p2) mod 26

int diff\_c = (c1 - c2) % 26;

if (diff\_c < 0) diff\_c += 26;

int diff\_p = (p1 - p2) % 26;

if (diff\_p < 0) diff\_p += 26;

// Find modular multiplicative inverse of (p1 - p2)

int diff\_p\_inv = modInverse(diff\_p, 26);

if (diff\_p\_inv == -1) {

return 0; // No solution exists

}

// Calculate 'a'

\*a = (diff\_c \* diff\_p\_inv) % 26;

// Calculate 'b' using the first equation

\*b = (c1 - (\*a \* p1) % 26) % 26;

if (\*b < 0) \*b += 26;

return 1; // Solution found

}

// Function to decrypt a character using the affine cipher

char decryptChar(int a, int b, char c) {

if (!isalpha(c)) {

return c; // Non-alphabetic characters remain unchanged

}

// Calculate a\_inverse

int a\_inverse = modInverse(a, 26);

if (a\_inverse == -1) {

printf("Error: 'a' value has no modular inverse. Decryption impossible.\n");

exit(1);

}

// Convert to 0-25 range

int c\_val = isupper(c) ? c - 'A' : c - 'a';

// Apply decryption formula: p = a^(-1) \* (c - b) mod 26

int p\_val = (a\_inverse \* (c\_val - b + 26)) % 26;

// Convert back to ASCII

return isupper(c) ? p\_val + 'A' : p\_val + 'a';

}

// Function to decrypt the entire ciphertext

void decrypt(int a, int b, const char \*ciphertext, char \*plaintext) {

int i;

for (i = 0; ciphertext[i] != '\0'; i++) {

plaintext[i] = decryptChar(a, b, ciphertext[i]);

}

plaintext[i] = '\0';

}

// Function to check if a key is valid (a must be coprime with 26)

int isValidKey(int a) {

return gcd(a, 26) == 1;

}

int main() {

int a, b;

char ciphertext[1000];

char plaintext[1000];

printf("=== Affine Cipher Breaker ===\n");

printf("This program breaks an affine cipher where:\n");

printf("- 'B' is the most frequent letter (corresponding to 'E')\n");

printf("- 'U' is the second most frequent letter (corresponding to 'T')\n\n");

// Known mappings:

// 'E' (4) maps to 'B' (1)

// 'T' (19) maps to 'U' (20)

int c1 = 1; // 'B' (0-based index)

int p1 = 4; // 'E' (0-based index)

int c2 = 20; // 'U' (0-based index)

int p2 = 19; // 'T' (0-based index)

// Solve for 'a' and 'b'

if (!solveAffineParameters(c1, p1, c2, p2, &a, &b)) {

printf("Failed to solve for parameters. Check your frequency analysis.\n");

return 1;

}

// Verify that 'a' is valid (coprime with 26)

if (!isValidKey(a)) {

printf("Error: Computed value of 'a' = %d is not coprime with 26.\n", a);

printf("This suggests an error in the frequency analysis.\n");

return 1;

}

printf("Computed keys: a = %d, b = %d\n\n", a, b);

printf("Enter the ciphertext to decrypt: ");

fgets(ciphertext, sizeof(ciphertext), stdin);

// Remove trailing newline if present

size\_t len = strlen(ciphertext);

if (len > 0 && ciphertext[len-1] == '\n') {

ciphertext[len-1] = '\0';

}

// Perform decryption

decrypt(a, b, ciphertext, plaintext);

printf("\nResults:\n");

printf("Ciphertext: %s\n", ciphertext);

printf("Plaintext: %s\n", plaintext);

// Double-check our mappings

printf("\nVerifying key mappings:\n");

printf("'E' -> '%c' (Expected 'B')\n", (a \* 4 + b) % 26 + 'A');

printf("'T' -> '%c' (Expected 'U')\n", (a \* 19 + b) % 26 + 'A');

return 0;

}

Output:

