Jose Antonio García del Saz
Cristina Kasner Tourné

1.Sustitución Monoalfabeto

i)Método Afín

INTRODUCCION

Para la realización de este apartado se nos pedía elaborar una rutina en C llamada *afín* con la cual pudiéramos cifrar y descifrar mensajes mediante cifrado afín dada una clave (A,B). Además se debía utilizar la librería de C *GMP*, la cual sirve para realizar operaciones aritméticas con grandes números de forma eficiente.

El programa será invocado de la siguiente manera :

afin [-C(cifrar)|-D(descifrar] [-m M] [-a A] [-b B] [-i in] [-o out]

donde (A,B) es la clave, M es el tamaño del espacio del texto cifrado, *in* es el fichero de entrada y *out* el de salida.

DESARROLLO

La primera comprobación que se hace en el código es ver que los valores A,M sean ambos menores que M, como se puede ver a continuación:

```
if((mpz_cmp(a, m) > 0) || (mpz_cmp(b, m) > 0)) {
  printf("Error\n");
  return(0);
}
```

ALGORITMOS Y FUNCIONES

La mayoría de líneas de código de esta rutina se corresponden con la función ***euclidesExtendido***, la cual implementa el algoritmo de Euclides extendido. La función recibe los siguientes parámetros:

-a , b : Entrada de dos números enteros (en formato mpz_t)

-mcd: Entero en formato mpz_t donde se escribirá el mcd entre a y b.

-s , t : son dos valores enteros en formato mpz_t donde se escribirán los dos valores s,t tales que $mcd = a \cdot s + b \cdot t$

El pseudocódigo del algoritmo es el siguiente:

1. $r_0 \leftarrow a, r_1 \leftarrow b, s_0 \leftarrow 1, t_0 \leftarrow 0, s_1 \leftarrow 0, t_1 \leftarrow 1$
2. $i \leftarrow 1$
3. **Mientras** $r_i \neq 0$ **haga lo siguiente:**
    1. Divida r_{i-1} entre r_i para obtener el cociente q_i y el residuo r_{i+1}
    2. $s_{i+1} \leftarrow s_{i-1} - q_i s_i$
    3. $t_{i+1} \leftarrow t_{i-1} - q_i t_i$
    4. $i \leftarrow i + 1$
4. **El resultado es:** r_{i-1} es un máximo común divisor de a y b y se expresa $r_{i-1} = as_{i-1} + bt_{i-1}$

En nuestro caso, la entrada a,b se corresponderá con los valores A,M. Lo que necesitamos es que el mcd(A,M)=1, para que a función afín que se define sea inyectiva y se pueda llevar a cabo el descifrado. Además como mdc(A,M)=1, este algoritmo también deja ver quien es el inverso de A en Z_M :

$$mcd = a \cdot s + b \cdot t\,;\, 1 = A \cdot s + M \cdot t\,;$$

y como sabemos que para todo K natural $K \cdot M \equiv 0 \, mod \, M$ tenemos que $1 = A \cdot s$
por lo que s es el inverso de A en Z_M , valor necesario y por tanto útil para el descifrado.

CIFRAR Y DESCIFRAR
Utilizamos la función *f(c) = ax+b* para cifrar el texto y su inversa $f^{-1}(x) = (c-b)a^{-1}$ para descifrar.

TEXTO
Nuestra función acepta texto en mayúsculas o minúsculas y espacios. En el caso de poner espacio debemos especificar que el tamaño de nuestro lenguaje es 27.

i)Método Afín Modificado

Hemos implementado un método similar al afín que en vez de ir cifrando el texto letra por letra lo hace en bloques de dos letras.
Vamos a ver un ejemplo:

*DE = 3*26 + 4 = 82*

Esto es como entender el mensaje como pares de letras vistas como números de dos cifras en base 26. Por lo tanto ya no nos encontramos en Z/26 sino en Z/26x26.

Tenemos que tener cuidado ya nuestras claves también estarán en base 26x26 y debemos calcular su inverso multiplicativo teniendo en cuenta esto.

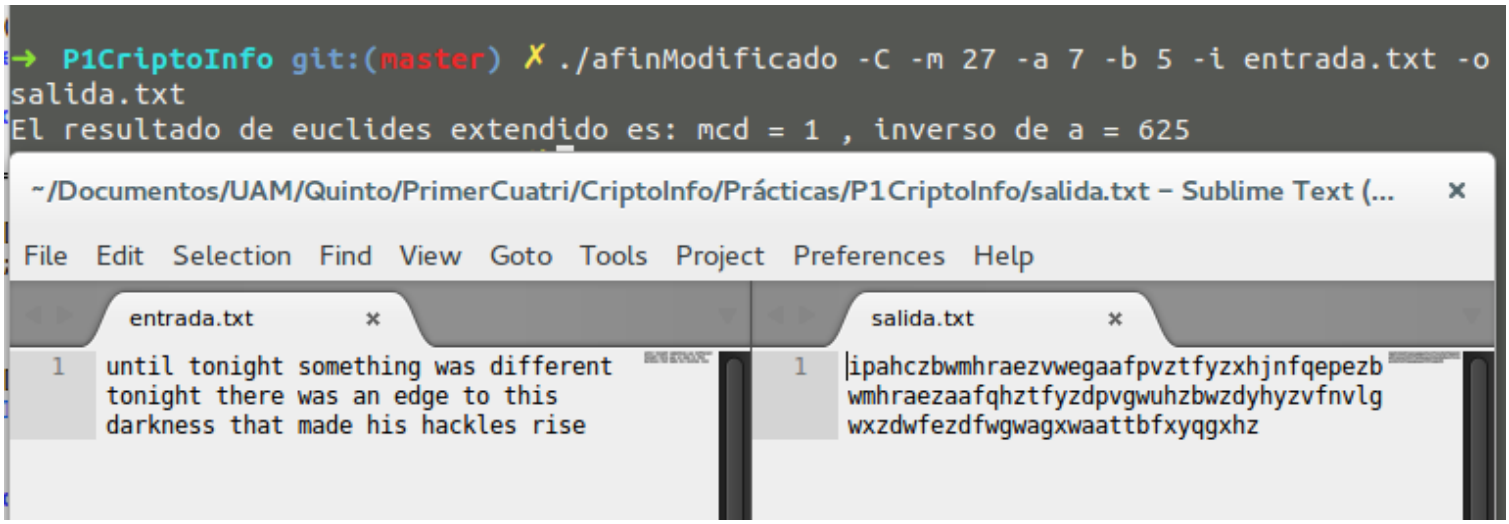
Aplicamos la función de afinModificado, por ejemplo para a=7, b=5.

*F(82) = 82*7 +5 = 579 = 22*26 +7 = WH*

Para desencriptar utilizamos la función inversa:

$F^{-1}(c)=(x-b)*a^{-1}$

Adjuntamos una imagen de un texto plano y el correspondiente texto cifrado tras aplicar este método.



Para calcular la robusted de este método vemos que a pertenece a las unidades de Z/m^2 y b a Z/m^2 por lo tanto la robusted será:

$$|Z^*/m^2| \cdot |Z/m^2| = \varphi(m^2) \cdot m^2$$

2.Sustitución Polialfabeto

i)Método de Hill

Para este apartado hemos creado una librería , matrix, en la que hemos implementado todas las funciones necesarias para trabajar con matrices en modulo m.

```
matrix *initMatrix(int size);
void transpuesta (matrix*src,matrix*res);
void inversa(matrix* src,matrix *res,int mod);
void setValue(matrix *m,const int fila,const int col, const double value);
double getValue(matrix *m,const int fila,const int col);
void readMatrix(FILE *in, matrix *m);
void printMatrix(matrix *m);
void copyMatrix(matrix* src, matrix *dest);
double detMatrix(matrix *m);
void eraseMatrix(matrix *m);
void exchangeRows(matrix *m, int f1, int f2);
void cofactores(matrix *src, matrix *des);
void matrixMod(matrix *m,int mod);
int inversoZm(int a, int m);
int mod(int a ,int b);
```

El método está implementado tal y como hemos visto en clase.

ii)Método de Vigenere

Para implementar este método hemos utilizado las funciones estudiadas en clase. No hemos utilizado la librería GMP.

Hemos implementado la función mod, que calcula n mod m de forma que para cifrar y descifrar solo tenemos que hacer lo siguiente:

```
cifrado = mod(plano + clave , m);
```

```
descifrado = mod(cifrado - clave , m);
```

TEXTO

Nuestra función acepta texto en mayúsculas o minúsculas y espacios. En el caso de poner espacio debemos especificar que el tamaño de nuestro lenguaje es 27.
Los saltos de linea los tomará como espacios.

ii)Criptoanálisis

ÍNDICE DE COINCIDENCIA

Para implementar el índice de coincidencia hemos creado los ficheros funciones.c y funciones.h.
En dichos ficheros se encuentran las siguientes funciones:

```
double calculaFrecuencia(char letra, char *cadena, int sizeCadena);
double calculaIndiceC(double *frecs, int size, int sizeFrecs);
double indiceCdeColumna(FILE *f,int columna , int ngrama);
```

calculaFrecuencia: calcula la frecuencia del carácter **letra** en el array **cadena.**
calculaIndiceC: calcula el IC dado una array de frecuencias con la fórmula vista en clase.
indiceCdeColumna: calcula el IC de la columna **columna** del texto que hay en el fichero **f** dividido en bloques de tamaño **ngrama**.

Con estas funciones ya es fácil calcular el IC total, llamamos a indiceCdeColumna con para cada columna de nuestro texto (que hemos dividido en n-gramas), y luego se hace una media de todos los Ics.

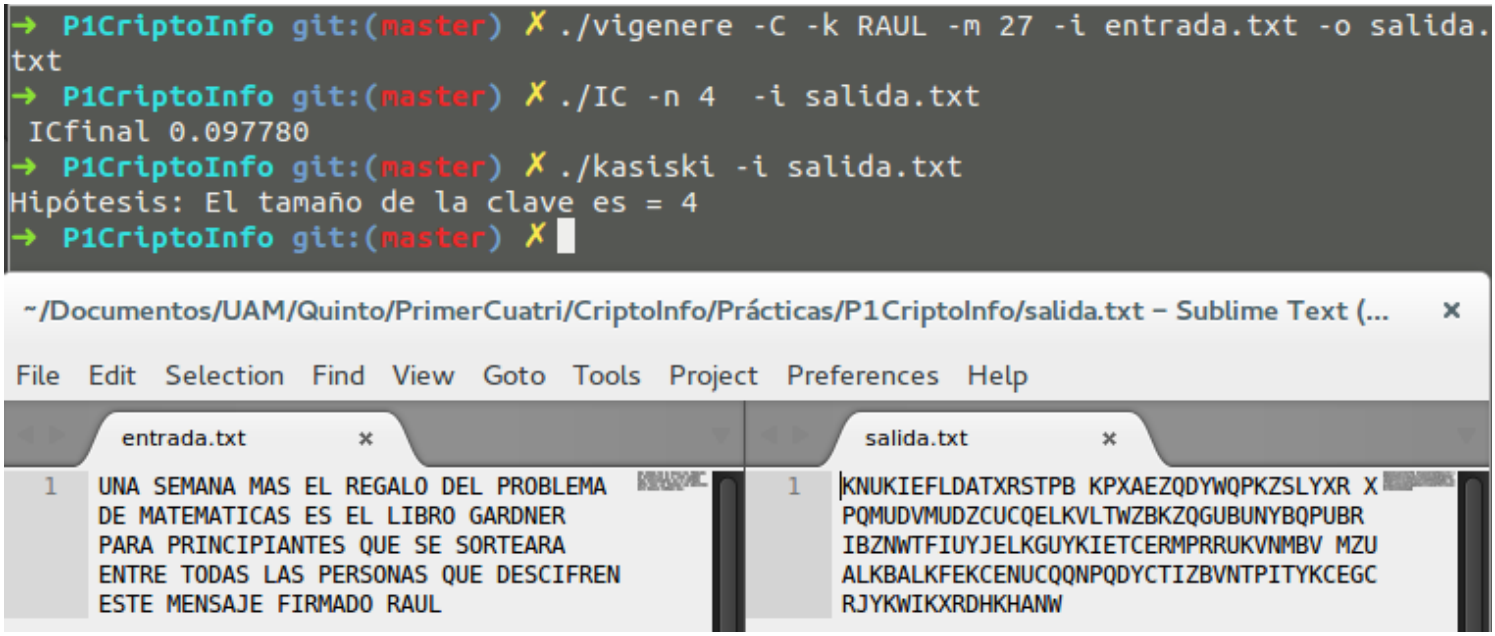
KASISKI

Para implementar Kasiski también hemos creado funciones en los ficheros funciones.c y funciones.h

```
int cmpfunc (const void * a, const void * b);
int mcd(int a,int b);
int mcd_numeros_recusiva(int *numeros,int cont);
int mcd_numeros(int *numeros, int longitud);
```

Para llamar a la función ejecutamos ./kasiski -i [file.in]
El método de kasiski va cogiendo el texto en bloques de 3 y va comparando dichos bloques.
Cuando dos bloques son iguales guarda la distancia entre ellos.
El mcd de todas las distancias será el tamaño de clave esperado.

Aqui vemos un ejemplo de uso de IC y kasiski sobre un texto cifrado con la clave -RAUL



Tras aplicar Kasiski a varios textos con distinta clave vemos que para textos muy largos con claves cortas Kasiski suele fallar ya que coinciden muchas cadenas y no todas provienen del mismo texto plano cifrado con la misma parte de la clave, por lo que el mcd da 1.

Para textos largos podríamos analizar n-gramas más grandes.

3.Método de transposición

Este ejercicio lo hemos hecho después del ejercicio 4 ya que, al leer el enunciado de la práctica pensamos que si al producto de criptosistemas permutación le pasamos como segundo vector un 1, estaríamos realizando un método de transposición.

4.Producto de criptosistemas permutación

INTRODUCCION

Para este apartado se nos pedía realizar un producto de criptosistemas permutación, el cual recibe un texto plano y lo devuelve cifrado. Este cifrado consiste más bien en una desordenación de los caracteres que es única para cada clave, ya que las claves son en sí los vectores permutación que multiplicaremos por la matriz de caracteres para desordenarla. La rutina será invocada de la siguiente manera (ejemplo):

permutacion –C –k1 (2,3,1) –k2 (2,3,4,1) –i input.txt –o output.txt

con lo que tendríamos una clave de dimensión 3 y una clave de dimensión 4, siendo las matrices con las que vamos a trabajar de dimensión 3x4. Es **importante** que las claves se den en el formato especificado y no en otro (paréntesis incluidos).

DESARROLLO

Lo primero que hace nuestro programa es leer las claves. Una vez hallado el tamaño de cada una y leídos los valores que contienen, se procede a reservar memoria para las dos matrices P y C (de dimensiónes MxN). Después independientemente de si ciframos o desciframos lo que va haciendo es leer el fichero de entrada carácter a carácter e ir rellenando una matriz MxN con los mismos. Cuando la matriz está llena, es pasada a la función correspondiente (cifrar o descifrar según convenga) y el resultado se manda al fichero de salida en forma de caracteres otra vez. Cuando se acaba el fichero, o bien el programa termina, o si queda alguna matriz a medio llenar la rellena con espacios y entonces termina.

ALGORITMOS Y FUNCIONES

Para este programa hemos implementado una librería para manejar matrices M*N de enteros en C. Los prototipos de las funciones están definidos en *matrixmn.h* y el código de las mismas en *matrixmn.c*. Esta librería nos da soporte para crear (reservar memoria e inicializar a 0 los valores), cambiar valor de una fila y una columna, obtener valor de una fila y una columna, intercambiar dos filas, intercambiar dos columnas, imprimir la matriz por pantalla, etc.

La estructura de datos *matrixmn* queda definida en la cabecera de la siguiente manera:

```
typedef struct{
    int **data;
    int rows;
    int columns;
}matrixmn;
```

Con este tipo de datos podemos realizar las siguientes operaciones básicas:

```
    matrixmn *initMatrixMN(int rows, int cols);
int getRowsMN(matrixmn *m);
int getColumnsMN(matrixmn *m);
void setValueMN(matrixmn *m,int row, int col, int value);
int getValueMN(matrixmn *m,int row, int col);
void zerosMN(matrixmn * m);
void exchangeRowsMN(matrixmn *m, int f1, int f2);
void exchangeColumnsMN(matrixmn *m, int f1, int f2);
void setRowMN(matrixmn *m,int f,int *row);
void setColumnMN(matrixmn *m,int c,int *col);
void printMatrixMN(matrixmn *m);
void copyMatrix(matrixmn* src, matrixmn *dest);
void cifra(matrixmn* P, matrixmn *C,int *k1,int *k2);
void descifra(matrixmn* C, matrixmn *P,int *k1,int *k2);
void toFileMN(FILE *f,matrixmn *m);
```

Además, en esta librería están las funciones que se encargan del cifrado y el descifrado en sí (llamadas *cifra* y *descifra*). La función *cifra* recibe una matriz de con caracteres del texto plano (ordenados) y las dos claves (como arrays de enteros) para devolver una matriz de la misma dimensión y con los mismos elementos, pero ya cifrados (desordenados en este caso). Para ello primero permuta las filas de la matriz origen y luego las columnas, según el orden que le indiquen las claves. La función *descifra* hace lo mismo pero de forma contraria, empezando por las columnas y acabando con las filas para convertir la matriz C cifrada en otra P descifrada.

Jose Antonio García del Saz
Cristina Kasner Tourné