Encriptación de mensajes

Otoño 2022

Contenidos:

1. Enunciado de la práctica

La Criptografía es la ciencia que trata sobre encriptar y desencriptar información. Desde la edad antigua se han encriptado mensajes diplomáticos, militares o administrativos para conseguir comunicaciones seguras, a salvo de espías.

En la actualidad, la Criptografía desempeña un papel de gran importancia dentro de la Informática. Su uso es prácticamente cotidiano, ya que elementos tan familiares como los passwords, los números de tarjetas, etc., se guardan encriptados e incluso hay programas que permiten cifrar y descifrar los mensajes de correo electrónico y, en general, cualquier documento en soporte digital.

Un mensaje puede encriptarse por *permutación* o por *sustitución* de sus caracteres, siguiendo unas reglas predefinidas y quizá introduciendo algún elemento adicional (claves, números aleatorios, ...). Obviamente, si se desea descifrar un mensaje ya encriptado, resulta esencial que las reglas de cifrado sean reversibles.

En esta práctica proponemos dos métodos de encriptación, uno de cada tipo. Para cada caso solo mostramos el algoritmo de encriptación, el de desencriptación es fácilmente deducible.

1.1. Sustitución polialfabética

Consideremos un alfabeto A y un texto T escrito en dicho alfabeto. El método que proponemos para encriptar T emplea además una clave C, que será otro texto (probablemente más pequeño), compuesto por caracteres del mismo alfabeto.

Podemos describir el algoritmo de encriptación mediante un ejemplo. Sea A el intervalo de las letras minúsculas de la tabla ASCII más el carácter blanco `''` (en este orden), el texto T a encriptar `"dont worry if there is a hell below we are all going to go"` y la clave C `"curtis mayfield"`.

En primer lugar consideramos una matriz con todos los desplazamientos posibles de los caracteres del alfabeto, módulo el tamaño del mismo, manteniendo el orden:

```
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
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  a
c d e f g h i j k l m n o p q r s t u v w x y z  a b
d e f g h i j k l m n o p q r s t u v w x y z  a b c
e f g h i j k l m n o p q r s t u v w x y z  a b c d
f g h i j k l m n o p q r s t u v w x y z  a b c d e
g h i j k l m n o p q r s t u v w x y z  a b c d e f
h i j k l m n o p q r s t u v w x y z  a b c d e f g
i j k l m n o p q r s t u v w x y z  a b c d e f g h
j k l m n o p q r s t u v w x y z  a b c d e f g h i
k l m n o p q r s t u v w x y z  a b c d e f g h i j
l m n o p q r s t u v w x y z  a b c d e f g h i j k
m n o p q r s t u v w x y z  a b c d e f g h i j k l
n o p q r s t u v w x y z  a b c d e f g h i j k l m
o p q r s t u v w x y z  a b c d e f g h i j k l m n
p q r s t u v w x y z  a b c d e f g h i j k l m n o
q r s t u v w x y z  a b c d e f g h i j k l m n o p
```

```
r s t u v w x y z   a b c d e f g h i j k l m n o p q
s t u v w x y z   a b c d e f g h i j k l m n o p q r
t u v w x y z   a b c d e f g h i j k l m n o p q r s
u v w x y z   a b c d e f g h i j k l m n o p q r s t
v w x y z   a b c d e f g h i j k l m n o p q r s t u
w x y z   a b c d e f g h i j k l m n o p q r s t u v
x y z   a b c d e f g h i j k l m n o p q r s t u v w
y z   a b c d e f g h i j k l m n o p q r s t u v w x
z   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
  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
```

A continuación, escribimos la clave debajo del texto, repitiéndola tantas veces como sea necesario:

```
dont worry if there is a hell below we are all going to go
curtis mayfieldcurtis mayfieldcurtis mayfieldcurtis mayfie
```

Por último, para codificar cada carácter x del texto tomamos el carácter y de la clave que esté debajo de él, y consideramos el carácter de la matriz que esté en la columna que contiene a x en la primera fila y en la fila que contiene a y en la primera columna.

Por ejemplo, obtengamos la codificación de `'d'`, el primer carácter del texto:

- `'d'` está en la cuarta columna de la primera fila
- el primer carácter de la clave (`'c'`) está en la tercera fila de la primera columna
- el carácter que se encuentra en esas coordenadas es `'f'`.

El texto encriptado queda

```
fhdlhnncrveqjkwjyhxh rlaxmmpwcdybgdrvq ywmdlontxgqefltleos
```

1.2. Permutación por intercambios arborescentes

Este segundo método es independiente del alfabeto usado en el texto a encriptar, ya que se basa simplemente en cambiar caracteres de posición. Dado un texto T, el algoritmo de encriptación funciona de la siguiente manera. En primer lugar (paso 1) el texto se transforma en un árbol binario de caracteres A:

- si el texto es vacío, el árbol es vacío
- en caso contrario, el carácter situado en la posición central del texto ($longitud/2$) es la raíz del árbol, el subárbol izquierdo se obtiene recursivamente a partir los caracteres anteriores y el subárbol derecho se obtiene recursivamente a partir los caracteres posteriores

Ejemplo: consideremos el texto T `"Cuando duerma con la soledad."`. La siguiente figura muestra el árbol resultante.

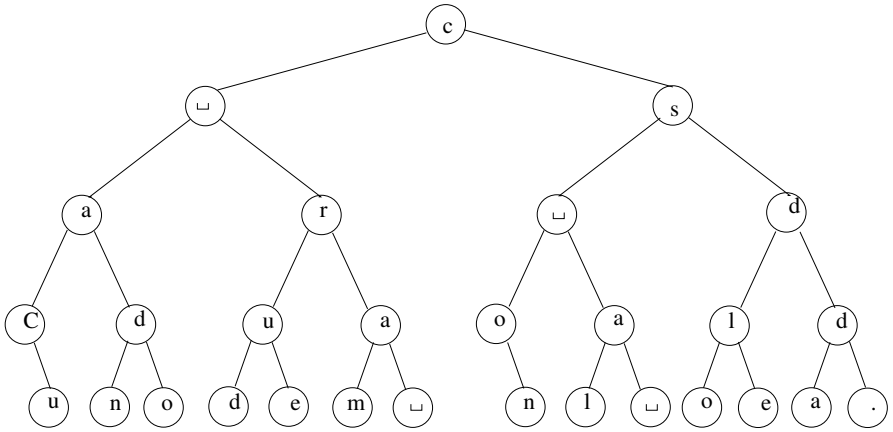

Figura 1: árbol del texto T

Una vez copiado el mensaje en el árbol, se obtiene un segundo árbol, con la misma forma que el primero, construido a base de intercambiar las posiciones de todos los nodos con sus hermanos (paso 2). Los nodos que no tengan hermanos, por ejemplo la raíz, no cambian de posición.

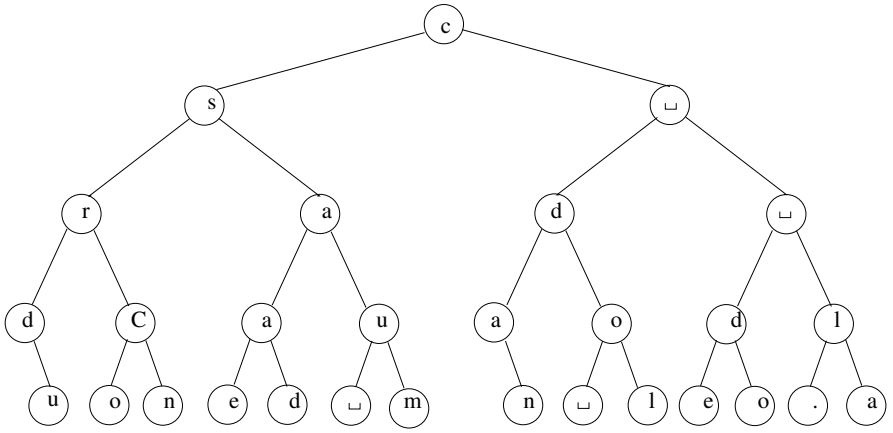

Figura 2: árbol del texto T con los hermanos intercambiados

Por último (paso 3), el texto encriptado se obtiene recorriendo en inorden este segundo árbol.

En el caso del ejemplo, el resultado es `"duroCnseada umcand ol edo .la"`.

La encriptación mediante intercambios arborescentes también se puede aplicar por bloques consecutivos, dado un cierto valor entero $b > 0$ (el tamaño del bloque).

- Si $b \geq long\_texto$, se aplica el método que acabamos de describir, es decir, solo se construye un árbol, de tamaño $long\_texto$
- Si $b < long\_texto$, entonces el texto se divide en $\lceil long\_texto/b \rceil$ bloques, todos de tamaño b excepto quizás el último, que puede tener un tamaño menor o igual a b. Cada bloque se encripta por separado mediante su propio árbol

2. Funcionalidades

2.1. Decisiones sobre los datos

Con el objetivo de simplificar los procesos de encriptación y desencriptación, los mensajes, alfabetos y claves que usaremos para probar la práctica estarán formados únicamente por caracteres de la tabla ASCII con índices entre 32 (espacio en blanco) y 126 ($\sim$). En particular, consideraremos lo siguiente:

- cada mensaje, clave y alfabeto estará formado por una sola línea de texto, no vacía
- un alfabeto será una permutación de un subconjunto de caracteres válidos (abc es distinto que bca)
- se puede distinguir un tipo especial de alfabeto, consistente en un intervalo de caracteres válidos, en la misma permutación que en la tabla ASCII (abc es especial, mientras que bca no lo es); puede comprobarse que en este caso el método de la sustitución polialfabética admite soluciones aún más simples y eficientes

Los datos del programa se dividen en dos partes. Un primer grupo de alfabetos y mensajes se usará para la inicialización del sistema. Una vez realizada dicha inicialización, el programa ofrecerá un repertorio de funcionalidades. Cada funcionalidad podrá requerir a su vez unos datos y producirá una salida que describimos con más precisión en el siguiente apartado.

En todo momento, se leerán solo datos sintácticamente correctos y no será necesario comprobar dicha corrección. Por ejemplo, si decimos que inicialmente el sistema contiene M mensajes, el programa recibirá un entero M mayor o igual que 0 y a continuación exactamente M mensajes, con las restricciones mencionadas a lo largo de este enunciado. Los alfabetos serán asimismo sintácticamente correctos, por ejemplo sus caracteres siempre estarán entre los ASCII ya comentados. Los mensajes y las claves siempre estarán escritos usando el alfabeto que les acompaña en cada funcionalidad.

2.2. Programa principal: estructura y comandos

El programa principal empezará leyendo un número inicial de alfabetos $A \geq 0$ y una secuencia de A alfabetos. Cada alfabeto irá precedido por un string que será su identificador. Se garantiza que los identificadores de los alfabetos no están repetidos.

A continuación se leerá un número inicial de mensajes $M \geq 0$ y una secuencia de M mensajes. Cada mensaje vendrá precedido por un string, que servirá como identificador del mismo y del identificador del alfabeto al que pertenece. Se garantiza que los identificadores de los mensajes no están repetidos y que los alfabetos están entre los anteriormente leídos.

Notemos que las magnitudes M y A no serán constantes ya que posteriormente se podrán añadir o borrar nuevos mensajes o nuevos alfabetos.

Terminadas las inicializaciones, se procesará una serie de comandos acabados en un comando `fin`. La estructura general del programa principal será la siguiente:

```
lectura del conjunto inicial de alfabetos;
lectura del conjunto inicial de mensajes;
lee comando;
while (comando != "fin") {
    procesa comando;
    lee comando;
}
```

Los comandos aceptados se describen a continuación. Todo ellos se presentan en dos versiones, una con el nombre completo y otra con el nombre abreviado. La sintaxis exacta de la entrada y la salida de cada comando se podrá derivar del juego de pruebas público del ejercicio creado en el Jutge para cada entrega.

1. `nuevo_mensaje` idm ida: lee y añade un nuevo mensaje con identificador idm escrito mediante el alfabeto de identificador ida. El comando admite la forma abreviada `nm`. Si no existe un alfabeto con identificador ida se imprime un mensaje de error. Si ya existía un mensaje en el sistema con el mismo identificador se imprime un mensaje de error. En caso contrario se imprime el número de mensajes M en el sistema después de añadirlo.

2. `nuevo_alfabeto` ida: lee y añade un nuevo alfabeto con identificador ida. El comando admite la forma abreviada `na`. Si ya existía un alfabeto en el sistema con el mismo identificador se imprime un mensaje de error. En caso contrario se imprime el número de alfabetos A en el sistema después de añadirlo.

3. `borra_mensaje` idm: elimina del sistema un mensaje con identificador idm. El comando admite la forma abreviada `bm`. Si no existe un mensaje con identificador idm se imprime un mensaje de error. En caso contrario se imprime el número de mensajes M después de borrarlo. Si más tarde se añade otro mensaje con el mismo nombre es como si el anterior no hubiera existido.

4. `borra_alfabeto` ida: elimina del sistema un alfabeto con identificador ida. El comando admite la forma abreviada `ba`. Si no existe un alfabeto con identificador ida se imprime un mensaje de error. Si existen mensajes con dicho alfabeto imprime un mensaje de error. En caso contrario se imprime el número de alfabetos A después de borrarlo. Si más tarde se añade otro alfabeto con el mismo identificador es como si el anterior no hubiera existido.

5. `listar_mensajes`: se listan los mensajes del sistema en orden alfabético de identificador. El comando admite la forma abreviada `lm`.

6. `listar_alfabetos`: se listan los alfabetos del sistema en orden creciente de identificador. El comando admite la forma abreviada `la`.

7. `codificar_sustitucion_guardado` idm cl: codifica el mensaje de identificador idm mediante la clave cl. El comando admite la forma abreviada `csg`. Si el mensaje idm no existe se imprime un mensaje de error. En caso contrario se imprime el mensaje codificado.

8. `codificar_sustitucion` ida cl: lee un mensaje escrito con el alfabeto de identificador ida, lo codifica mediante la clave cl. El comando admite la forma abreviada `cs`. Si el alfabeto ida no existe se imprime un mensaje de error. En caso contrario se imprime el mensaje codificado.

9. `decodificar_sustitución` ida cl: lee un mensaje escrito con el alfabeto de identificador ida y lo decodifica mediante la clave cl. El comando admite la forma abreviada `ds`. Si el alfabeto ida no existe se imprime un mensaje de error. En caso contrario se imprime el mensaje decodificado.

10. `codificar_permutacion_guardado` idm b: codifica el mensaje con identificador idm, dividiendo el mensaje en bloques de tamaño b. El comando admite la forma abreviada `cpg`. Si el mensaje idm no existe se imprime un mensaje de error. En caso contrario se imprime el mensaje codificado.

11. `codificar_permutacion` b: lee y codifica un mensaje, dividiéndolo en bloques de tamaño b. El comando admite la forma abreviada `cp`. Se imprime el mensaje codificado.

12. `decodificar_permutacion` b: lee y decodifica un mensaje, dividiéndolo en bloques de tamaño b. El comando admite la forma abreviada `dp`. Se imprime el mensaje decodificado.

13. `fin`: termina la ejecución del programa