Actividad 8 - Criptografía
Juguemos a los espías con la computadora

El objetivo de esta actividad es realizar programas que nos permitan encriptar y desencriptar mensajes con algunas estrategias desafiantes. Trabajaremos con diccionarios y archivos de texto.

Codificación romana

Tanto la diplomacia como las operaciones militares han hecho uso, históricamente, de distintas formas de *criptografía* para ocultar el contenido de un mensaje de modo de hacerlo ilegible para todos salvo para el destinatario que debe contar con una *clave* adecuada para descifrar el mensaje. Uno de los primeros métodos criptográficos conocidos nos llega a través del historiador romano Suetonio, que cuenta que Julio César codificaba algunos de sus mensajes cambiando cada letra por la que se encuentra tres lugares por delante. Por ejemplo: en lugar de A, escribía D. Al llegar al final del abecedario, volvía a comenzar, de modo que, por ejemplo, la letra Z se reemplazaba por C. Al reemplazo de unas letras por otros se lo denomina un *cifra*. La *cifra de César* es un caso particular de las cifras de desplazamiento (*shift*, en inglés), que consisten en reemplazar una letra por otra que se encuentra a una cierta cantidad prefijada de casilleros.

Codificación y decodificación de caracteres y cadenas

1. **Lectura del alfabeto**
   Implementar una función `leer_archivo(nombre)` que dado un `nombre` de archivo, abre el archivo para lectura y carga su contenido en una variable, y devuelve esa variable.

   Ayuda: Recordar que con `open(nombre, "r", encoding="utf-8")` se abre el archivo indicado en `nombre` en modo lectura, el cual puede o no contener su ruta completa (esto es, puede contener sólo el nombre del archivo, como `"alfabeto.txt"`, o incluir la ubicación en la computadora, como `"archivos/alfabeto.txt"`).

   Precondiciones: El archivo indicado existe y está disponible para lectura.

   Para probar en consola:

   ```
   alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
   alf[0] # a
   alf[26] # z
   ```

2. **Codificación de un carácter**
   Escribir una función `codificar_caracter(letra, alfabeto, k)` que dados un carácter `letra`, un `alfabeto` y un número `k` devuelve el carácter codificado con un salto de `k` caracteres.

   Para esto se debe identificar la posición de `letra` en el alfabeto, y hacer el desplazamiento indicado hasta obtener el valor cifrado de la letra.

   Ayuda: Crear una función `obtener_indice(caracter,cadena)` que, dado un `caracter`, devuelve la posición de primera aparición en `cadena`, y tiene como precondición que `caracter` está en la `cadena`.

   Precondiciones: La `letra` pertenece al `alfabeto` indicado, y el valor `k` es mayor o igual a cero y menor al tamaño de `alfabeto`.

   Para probar en consola:

   ```
   alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
   codificar_caracter("m",alf,23) # Nos movemos 23 posiciones, y se obtiene 'i'
   codificar_caracter("a",alf,23) # 'w'
   codificar_caracter("t",alf,23) # 'p'
   codificar_caracter("e",alf,23) # 'a'
   ```

3. **Normalización de cadena**
   Implementar una función `normalizar(mensaje)`, que dada una cadena `mensaje` devuelve el resultado de aplicar distintas reglas de normalización del texto. Esta función nos permitirá procesar un mensaje antes de que sea codificado.

   a) Se pasa todo a minúsculas.

   b) Las vocales con tilde se reemplazan con su versión sin tilde.

   c) El resto de los caracteres quedan igual.

   Ayuda: Recordar que dada una `cadena`, `cadena.lower()` obtiene una nueva cadena como la original, pero pasada a minúsculas.

   Precondiciones: Ninguna.

   Para probar en consola:

```
normalizar("Soy Marcelo") # "soy marcelo"
normalizar("Nada más") # "nada mas"
normalizar("Aló, Victoria!") # "alo, victoria!"
```

4. **Codificación de cadenas de caracteres**
   Implementar una función `codificar(mensaje, alfabeto, k)` que dados un `mensaje`, un `alfabeto` y salto `k`, convierte el `mensaje` entero con un desplazamiento `k`, y devuelve la cadena resultante.

   Se deberá usar la función `normalizar` para todo el mensaje antes de codificarlo carácter a carácter usando `codificar_caracter`.

   Ojo: Antes de llamar a la función `codificar_caracter` se deberá verificar que el carácter pertenezca al alfabeto, ya que la precondición de esa función es esa misma. Si el carácter no pertenece al alfabeto, no se debe usar `codificar_caracter`, y queda exactamente igual en el mensaje codificado.

   Precondiciones: El valor `k` es mayor o igual a cero y menor al tamaño de `alfabeto`.

   Para probar en consola:

```
alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
codificar("Ya!",alf,4) # "ce!"
codificar("Mica y Claudia toman mate",alf,0) # "mica y claudia toman mate"
codificar("La conclusión: perdieron.",alf,10) # "'uk mywmuecryw: zñbnrñbyw."
```

5. **Decodificación de un carácter**
   Escribir una función `decodificar_caracter(letra, alfabeto, k)` que dados un carácter codificado `letra`, un `alfabeto` y un número `k` devuelve el carácter original que fue codificado con un salto de `k` caracteres.

   Para esto se debe identificar la posición de `letra` en el alfabeto, y hacer el desplazamiento indicado hacia atrás, hasta obtener el carácter original.

   Precondiciones: La `letra` pertenece al `alfabeto` indicado, y el valor `k` es mayor o igual a cero y menor al tamaño de `alfabeto`.

Para probar en consola:

```
alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
decodificar_caracter("i",alf,23) # 'm'
decodificar_caracter("w",alf,23) # 'a'
decodificar_caracter("p",alf,23) # 't'
decodificar_caracter("a",alf,23) # 'e'
```

6. **Decodificación de cadenas de caracteres**
   Implementar una función `decodificar(mensaje, alfabeto, k)` que dados un `mensaje`, un `alfabeto` y un desplazamiento `k`, devuelve el mensaje decodificado.

   Ayuda: Usar la función `decodificar_caracter` anterior.

   Ojo: Antes de llamar a la función `decodificar_caracter` se deberá verificar que el carácter pertenezca al alfabeto, ya que esa es una de las precondiciones de esa función (y somos responsables de verificar que eso ocurra). Si el carácter no pertenece al alfabeto no hace ninguna conversión, y queda exactamente igual en el mensaje codificado.

   Precondiciones: El valor `k` es mayor o igual a cero y menor al tamaño de `alfabeto`.

Para probar en consola:

```
alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
decodificar("ol orwr pr duudpfd :(",alf,3) # "mi moto no arranca :("
decodificar("hola, ¿que tal?",alf,0) # "hola, ¿que tal?"
mens = "el arte de programar!"
decodificar(codificar(mens,alf,10),alf,10) # "el arte de programar!"
```

Codificación de archivos de texto

Para codificar textos más extensos es conveniente procesar el texto tomado de un archivo y almacenar el resultado en otro archivo.

7. Implementar una función `codificar_archivo(nombre,alfabeto,k)`, que dados un `nombre` de archivo, el `alfabeto` a usar, y el desplazamiento `k`, devuelve una cadena que contiene el resultado de codificar el archivo completo, además de crear un archivo nuevo, con el mismo nombre que el anterior, pero al cual se le cambia la extensión a ".enc".

   Ayuda: Leer el archivo completo a una cadena, y convertirla usando la función `codificar`.

   Precondiciones: El archivo indicado en `nombre` existe y está disponible para lectura. El valor `k` es mayor o igual a cero y menor al tamaño de `alfabeto`.

Para probar en consola:

```
alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
codificar_archivo("original.txt",alf,3) # "ol orwr pr duudpfd :("
```

8. Implementar una función `decodificar_archivo(nombre,alfabeto,k)` que dados un `nombre` de archivo, que puede ser sólo el nombre o también incluir su ruta completa, el `alfabeto` a usar, y el desplazamiento `k`, devuelve una cadena que contiene el resultado de decodificar el archivo completo, además de crear un archivo nuevo, con el mismo nombre que el anterior, pero al cual se le cambia la extensión a ".dec".

Precondiciones: El archivo indicado en `nombre` existe y está disponible para lectura, y, además, se puede crear un nuevo archivo en esa misma ubicación. El valor `k` es mayor o igual a cero y menor al tamaño de `alfabeto`.

Para probar en consola:

```
alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
decodificar_archivo("original.enc",alf,3) # "mi moto no arranca :("
```

Cifras de reemplazo con clave

Ahora se quiere poder hacer una codificación, pero sin usar el cifrado por desplazamiento estándar. Así, dada una palabra clave, por ejemplo, *catapulta*, se contruirá una cifra eliminando las letras repetidas de la palabra clave (en este caso, queda *catpul*), y construimos un diccionario en donde la cifra comienza con esas letras y sigue con el resto del alfabeto en orden, sacando las que tenía la palabra clave.
En la tabla de abajo se muestra cada carácter del alfabeto y, debajo, el carácter cifrado que le corresponde:

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

Así, usamos una cifra arbitraria y sólo será necesario recordar la palabra clave para codificar/decodificar.

9. Implementar la función `quitar(elementos, lista)`, que dadas dos listas `elementos` y `lista`, elimina cada elemento contenido en `elementos` de `lista`, y el resto se mantiene en el orden en el que estaba originalmente.

   Precondiciones: Ninguna.

   Para probar en consola:

   ```
   alf2 = list("abcdefghijkl") # Una lista con sólo una parte de nuestro alfabeto
   quitar(['l','i','s','t','a'], alf2) #['b','c','d','e','f','g','h','j','k']
   quitar(['a','b','9'], alf2) #['c','d','e','f','g','h','i','j','k','l']
   quitar(['f','e','d'], alf2) #['a','b','c','g','h','i','j','k','l']
   ```

10. Implementar una función `sin_repetidos(cadena)`, que dada una `cadena`, devuelve la cadena resultante de eliminar los caracteres repetidos, sólo quedándose con la primera aparición.

    Ayuda: Como primer paso conviene pasar toda la cadena a minúscula, como se hizo anteriormente.

    Precondiciones: Ninguna.

    Para probar en consola:

    ```
    sin_repetidos("wololo") # "wol"
    sin_repetidos("python") # "python"
    sin_repetidos("bieeeeeeen") # "bien"
    ```

11. Implementar una función `crear_codificacion(palabra, alfabeto)` que dada una `palabra` clave y un `alfabeto`, devuelve el diccionario resultante de generar una cifra usando `palabra` como palabra clave tomando como base el `alfabeto`, de acuerdo a lo explicado anteriormente.

A diferencia de lo que se hizo anteriormente, ahora no se usará la codificación del César con una desplazamiento `k`. sino que se armará el diccionario de codificación, que permite poner como clave el carácter original y como valor el carácter codificado.

Ayuda: Usar la función `sin_repetidos` para procesar la palabra.

Precondiciones: Ninguna.

Para probar en consola:

```
alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
clave = "catapulta"
crear_codificacion(clave, alf) # {'a':'c','b':'a','c':'t','d':'p',...,'z':'z'}
```

Uso de cifras de reemplazo con clave (tarea)

12. Implementar una función `codificar_con_dicc(mensaje,diccionario)`, que dados un `mensaje` y un `diccionario` de codificación, realiza la codificación del `mensaje` de acuerdo a lo indicado en el `diccionario`.

    El `diccionario` de codificación tendrá como clave el carácter original y como valor el carácter al cual se lo convertirá.

    Como se hizo anteriomente, antes de realizar la codificación, la cadena del mensaje se normalizará (esto es, se pasa a minúscula y se quitan letras con tildes) por medio de la función `normalizar`. Todos los caracteres que no estén incluidos en el diccionario no serán convertidos, y se dejarán sin modificar en el mensaje codificado resultado.

    Precondiciones: El `diccionario` debe tener como claves y como significados a algo de tipo `string`.

    Para probar en consola:

    ```
    alf = leer_archivo("alfabeto.txt") # alf es "abcdefghijklmnñopqrstuvwxyz"
    clave = "catapulta"
    dicc = crear_codificacion(clave, alf) # {'a':'c','b':'a','c':'t',...,'z':'z'}
    codificar_con_dicc("El ABC de la programación!",dicc) # "uh cat pu hc nombocictemj!"
    ```

13. Implementar una función `decodificar_con_dicc(mensaje,diccionario)`, que dados un `mensaje` codificado y un `diccionario` de codificación, realiza la decodificación del `mensaje` de acuerdo a lo indicado en el `diccionario`.

    Ayuda: El diccionario para realizar la decodificación es el mismo que se usó para la codificación. Para encontrar más fácil las claves (cada carácter) convendría invertir el diccionario haciendo lo siguiente, para que quede el carácter codificado como clave y como valor el original:
    `dicc_dec = dict(zip(dicc.values(),dicc.keys()))`

    Precondiciones: El `diccionario` debe tener como claves y como significados a algo de tipo `string`.

    Para probar en consola:

    ```
    dicc = {'a':'1','b':'9','c':'2','d':'-'}
    decodificar_con_dicc("9o21 juniors", dicc) # "boca juniors"
    decodificar_con_dicc("river pl1te", dicc) # "river plate"
    decodificar_con_dicc("y -ijo, nun21 m1s!", dicc) # "y dijo, nunca mas!"
    ```

14. Implementar una función `codificar_archivo_con_dicc(nombre,diccionario)`, que dados un `nombre` de archivo y un `diccionario` de codificación, devuelve una cadena que contiene el resultado de codificar el archivo completo, además de crear un archivo nuevo, con el mismo nombre que el anterior, pero al cual se le cambia la extensión a ".enc".

    Ayuda: Recordar que se puede reemplazr parte de una `cadena` haciendo `cadena.replace(a_buscar, reemplazo)`, donde se reemplaza todas las apariciones de la cadena `a_buscar` con la cadena `reemplazo`.

    Precondiciones: El `nombre` de archivo indicado existe y está disponible para lectura, y el `diccionario` debe tener como claves y como significados a algo de tipo `string`.

    Para probar en consola:

    ```
    dicc = {'a':'1','b':'9','c':'2','d':'-'}
    codificar_archivo("original.txt",dicc) # "mi moto no 1rr1n21 :("
    ```

15. Implementar una función `decodificar_archivo_con_dicc(nombre,diccionario)`, que dados un `nombre` de archivo y un `diccionario` de codificación, devuelve una cadena que contiene el resultado de decodificar el archivo completo, además de crear un archivo nuevo, con el mismo nombre que el anterior, pero al cual se le cambia la extensión a ".dec".

    Precondiciones: El `nombre` de archivo indicado existe y está disponible para lectura y, además, se puede crear un nuevo archivo en esa misma ubicación. El `diccionario` debe tener como claves y como significados a algo de tipo `string`.

    Para probar en consola:

    ```
    dicc = {'a':'1','b':'9','c':'2','d':'-'}
    decodificar_archivo("original.enc",dicc) # "mi moto no arranca :("
    ```