Code Inverse, César et César affine

par Léo Peyronnet

Décembre 2022

1 Rappel des méthodes

1.1 Code Inverse

Cette méthode consiste à écrire le mot codé à l'envers. Pour ce faire, nous pouvons associer à chaque x lettre du mot à crypter un entier $i \in [0, n[$ avec n le nombre de lettres qui compose le mot. De fait, nous pouvons aussi associer à chaque c lettre du mot crypté un entier $j \in [0, n[$. Ainsi, la relation entre les deux mots est :

$$x_i \equiv c_{n-i-1} \Leftrightarrow c_j \equiv x_{n-j-1}$$

Ainsi, le même algorithme peut être utilisé pour crypter et décrypter un mot.

Exemple:

0	1	2	i	$n-3$	$n-2$	$n-1$
C	R	Y	...	A	G	E
$x_0 = c_{n-1}$	$x_1 = c_{n-2}$	$x_2 = c_{n-3}$	$x_i = c_{n-i-1}$	$x_{n-3} = c_2$	$x_{n-2} = c_1$	$x_{n-1} = c_0$
E	G	A	...	Y	R	C
0	1	2	j	$n-3$	$n-2$	$n-1$

1.2 Code César

Cette méthode consiste à décaler chaque lettre d'un mot par rapport à un alphabet. Pour ce faire, pour un mot x, déterminons α un alphabet incluant toutes les lettres de x et n la taille de α. Soit $i \in [0, p[$ l'index d'une lettre de x et p la taille du mot, alors x_i correspond à l'indice de cette lettre dans α. Ainsi, pour $b \in \mathbb{Z}$ le décalage à appliquer dans le code, nous avons y le mot codé tel que :

$$y_i \equiv x_i + b[n]$$

Le décryptage avec x le mot crypté et y le mot décrypté correspond alors à :

$$y_i \equiv x_i - b[n]$$

1.3 Code César affine

Cette méthode consiste à appliquer à chaque lettre d'un mot une autre valeur via une fonction affine. Soit x,y,i,α et n comme vu à la méthode précédente. Soit a et b des entiers relatifs, alors :

$$y_i \equiv ax_i + b[n]$$

Le décryptage avec x le mot crypté et y le mot décrypté correspond alors à :

$$y_i \equiv a'(x_i - b)[n]$$

avec a' l'inverse modulaire de a.

2 Présentation des programmes

2.1 codeInverse()

Fonction relative à la méthode Code Inverse (cf. 1.1).

```
def codeInverse(mot):
    result=""
    for i in range(len(mot)-1,-1,-1): # len(mot)-1 >= i
        >= 0
        # soit j un compteur incrémenté à chaque passage
            de boucle (sousentendu ici car boucle for)
        result+=mot[i] # la i-ème lettre du mot en clair
            devient la j-ème lettre du mot crypté
    return result
```

2.2 scan()

Fonction retournant l'indice d'une lettre dans un alphabet. Utilisé pour la méthode César et César affine.

```
def scan(lettre, alphabet):
    y=0
    while (lettre!=alphabet[y]): #si la i-ème lettre du
        mot est dans l'alphabet (à l'index y), alors fin
        de boucle
            y+=1 #passage à la lettre suivante de l'
                alphabet
            if y==len(alphabet): #si la i-ème lettre n'
                est pas dans l'alphabet
                print("attention:␣un␣des␣caractères␣du␣
                    mot␣n'est␣pas␣dans␣l'alphabet␣!")
                exit()
    return y
```

2.3 codeCesar() et decodeCesar()

Fonction de codage et décodage relative à la méthode Code César (cf. 1.2).

```
def codeCesar(alphabet,mot,cle):
    result=""
    for i in range(len(mot)): # 0 <= i <= len(mot)-1
        y=scan(mot[i],alphabet) # position de la i-ème
            lettre du mot dans l'alphabet ; copions la
            valeur de y à cette étape sous le nom x
        y=(y+cle)%len(alphabet) # décalage avec la clé cé
            zar (y congru à x + cle modulo la taille de l'
            alphabet)
        result+=alphabet[y] #la i-ème lettre du mot en
            clair d'index x dans l'alphabet devient la i-è
            me lettre du mot crypté d'index y.
    return result

def decodeCesar(alphabet,mot,cle):
    result=""
    for i in range(len(mot)): # 0 <= i <= len(mot)-1
        y=scan(mot[i],alphabet) # position de la i-ème
            lettre du mot dans l'alphabet ; copions la
            valeur de y à cette étape sous le nom x
        y=(y-cle)%len(alphabet) # décalage avec la clé cé
            zar (y congru à x - cle modulo la taille de l'
            alphabet)
        result+=alphabet[y] #la i-ème lettre du mot en
            clair d'index x dans l'alphabet devient la i-è
            me lettre du mot crypté d'index y.
    return result
```

2.4 codeCesarAff()

Fonction de codage relative à la méthode Code César affine (cf. 1.3).

```
def codeCesarAff(alphabet,mot,a,b):
    result=""
    for i in range(len(mot)): # 0 <= i <= len(mot)-1
        y=scan(mot[i],alphabet) # position de la i-ème
            lettre du mot dans l'alphabet ; copions la
            valeur de y à cette étape sous le nom x
        y=(a*y+b)%len(alphabet) # cryptage affine (y
            congru à a*x+b modulo la taille de l'alphabet)
        result+=alphabet[y] #la i-ème lettre du mot en
            clair d'index x dans l'alphabet devient la i-è
            me lettre du mot crypté d'index y.
    return result
```

2.5 euclideEtt() et inv()

Fonctions employées par decodeCesarAff() (cf. 2.6) :

— euclideEtt() : Algorithme d'Euclide étendu. Renvoie l'identité de Bézout ($a.u + b.v = pgcd(a, b)$) sous forme d'une liste de trois entiers. r1 : pgcd ; u1 : u et v1 : v.

— inv() : Revoie u si r1=1.

```
def euclideEtt(a,b):
    r1=b
    r2=a
    u1=0
    v1=1
    u2=1
    v2=0
    while r2!=0:
        q=r1//r2

        r3=r1
        u3=u1
        v3=v1

        r1=r2
        u1=u2
        v1=v2

        r2=r3-q*r2
        u2=u3-q*u2
        v2=v3-q*v2
    return [r1,u1,v1] # retourne l'identité de bézout (r1
        :pgcd;u1:u et v1:v)

def inv(a,modulo):
    t=euclideEtt(a,modulo) #import de l'identité de bé
        zout
    if t[0]==1: #si le pgcd=1
        return t[1] # retourne l'inverse modulaire de a
            :(u)
    else: #si le pgcd!=1
        print("attention:␣le␣coefficient␣A␣de␣la␣fonction
            ␣affine␣n'est␣pas␣premier␣avec␣la␣taille␣de␣l'
            alphabet␣!")
        exit()
```

2.6 decodeCesarAff()

Fonction de décodage relative à la méthode Code César affine (cf. 1.3).

```
def decodeCesarAff(alphabet,mot,a,b):
    result=""
    for i in range(len(mot)): # 0 <= i <= len(mot)-1
        y=scan(mot[i],alphabet)  # position de la i-ème
            lettre du mot dans l'alphabet ; copions la
            valeur de y à cette étape sous le nom x
        y=(inv(a,len(alphabet))*(y-b))%len(alphabet) # dé
            cryptage affine (y congru à a'*(x-b) modulo la
             taille de l'alphabet)
        result+=alphabet[y] #la i-ème lettre du mot en
            clair d'index x dans l'alphabet devient la i-è
            me lettre du mot crypté d'index y.
    return result
```