Análisi Auric Cypher

Tutor de la pràctica : Francesc Castro

Marc Sànchez Pifarré, GEINF (UDG-EPS)

27/10/2019

Contents

Auric cypher

Introducció

L'algoritme de xifratge que he generat consta de dues parts diferenciades.

- Algoritme de xifratge àuric inventat per mi.
- Algoritme de xifratge Rail Fence.

Context i estudi previ

Una part del temps invertit en aquesta pràctica ha sigut a l'estudi i les priopietats de la sèrie de fibonacci.

Fibonacci és la sèrie en la que els seus números estan compostos per la suma dels dos components anteriors a la mateixa sèrie, existeixen infinits nombres de la sèrie de fibonacci. L'estudi ha començat en veient quines propietats interessants em podia aportar realitzant una cerca de la sèrie de fibonacci per la xarxa, intentant esbrinar quines aplicacions s'han realitzat en criptografia.

Per sorpresa meva en la criptografia clàssica no existeixen algoritmes famosos, com els que hem vist a classe, per a l'aplicació d'aquesta série referent a criptografia, sí que s'han trobat moltes coincidències aplicades a disseny, dibuix i fins i tot relacionades amb la natura.

L'estudi l'he enfocat en cercar propietats d'aquesta mateixa sèrie mitjançant l'aplicació de l'aritmètica modular. Primerament s'ha realitzat l'algoritme i s'han cercat nombres de la sèrie, ja que l'algoritme és molt costós no s'han aconseguit gaires nombres i s'han cercat els 300 primers nombres de la mateixa per tenir un conjunt de nombres prou gran per estudiar.

```
1 # Declarem la constant que conté els 300 nombres de fibonacci
      utiltizats a aquesta pràctica
2 fibonacci300 =
      [1,1,2,3,5,8,13,21...,84885164052257330097714121751630835360966663883732297726360
```

La primera prova que vaig realitzar va ser la més significativa, la meva idea inicial va ser llençar un bucle de 1 fins a 50 classificant els 300 nombres de la sèrie de fibonacci dins d'un diccionari clau valor, on la clau era el nombre d'aparicions i el valor la llista de colisions dels mateixos nombres a Z/n sent n la variable del bucle i generant els histogrames per cada iteració. La idea va sortir per el que anomenem la propietat de la raó àurea.

$(a+b)/a = a/b$

En definitiva només m'he fixat en la propietat que dicta que donat un segment "c" compós per dos segments "a" i "b", si a i b son segments àuris, llavors c també ho és. Evidentment. M'ha fet pensar, què passa si estudio, en forma de simulació, (no em considero matemàticament prou bó com per treure una regla matemàtica en tota la meva existència, menys en 2 setmanes…) la relació que pugui haver-hi entre la els nombres d'aquesta sèrie i les seves congruències a Z/n.

Ajudant-me de l'algoritme següent que realitza aquesta classificació i torna el hashmap mostrarem una sèrie de gràfics interessants.

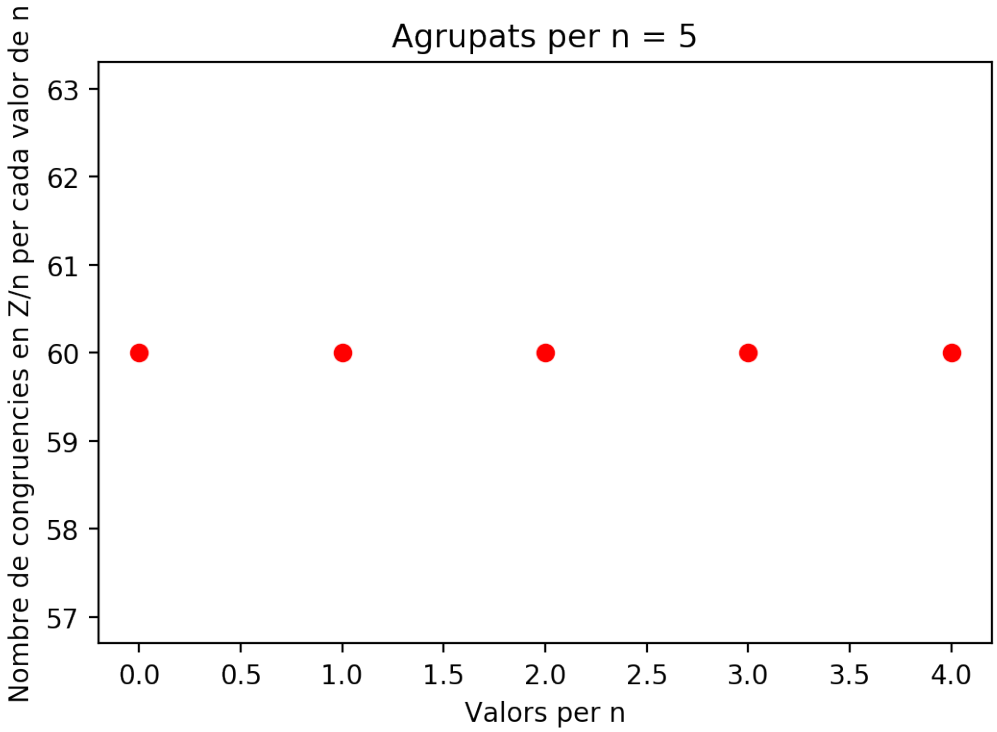
```
def generateAppearancesCount(fibonacci300, L):
    appearences = dict()
    for i in fibonacci300:
        number = i % L
        if number in appearences:
            appearences[number] += 1
        else:
            appearences[number] = 1
    return appearences
```

Generem els hashmap passant com a paràmetre els nombres de la sèrie i els valors per la n (mòdul) com a prova.

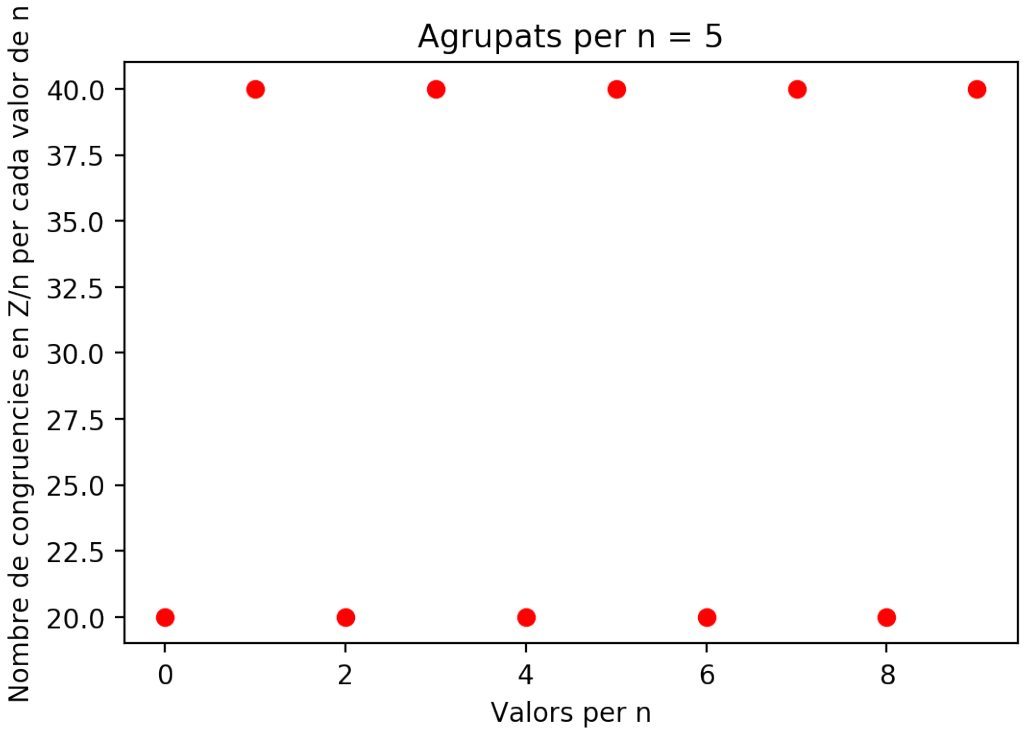
```
seq5 = generateAppearancesCount(fibonacci300, 5)
seq10 = generateAppearancesCount(fibonacci300, 10)
seq15 = generateAppearancesCount(fibonacci300, 15)
seq20 = generateAppearancesCount(fibonacci300, 20)
seq25 = generateAppearancesCount(fibonacci300, 25)
seq30 = generateAppearancesCount(fibonacci300, 30)
```

I mostrem els gràfics de com es comporten el nombre de colisions dels mateixos quan comprovem la seva congruència a Z/n.

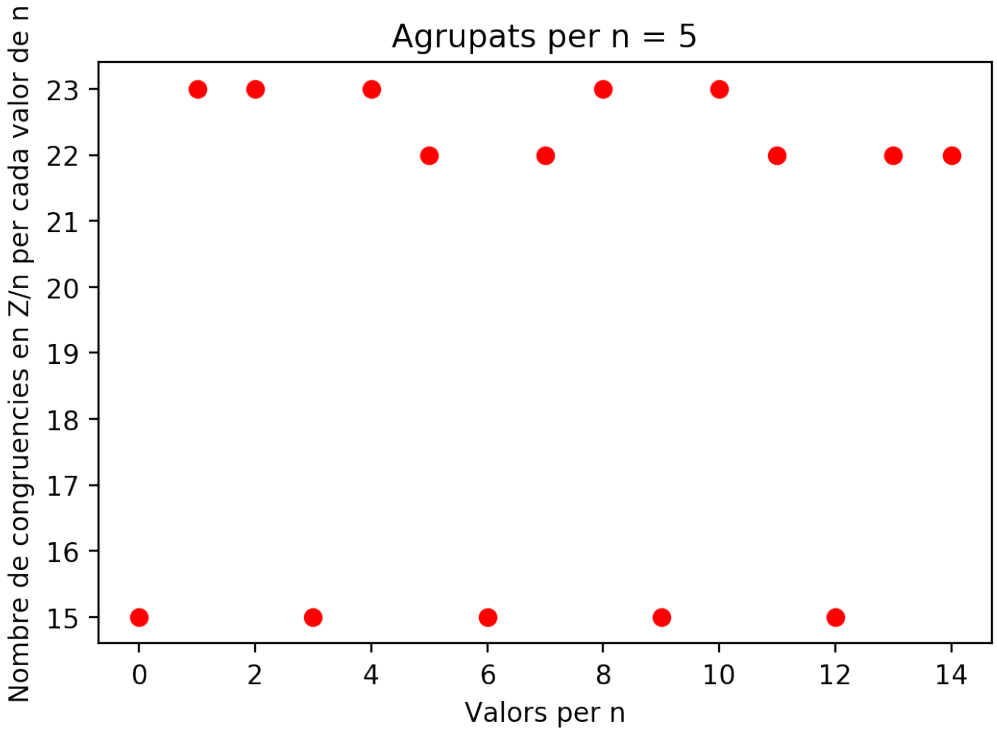
```
import matplotlib.pyplot as plt
plt.plot(seq5.keys(), seq5.values(), 'ro')
plt.title("Agrupats per n = 5")
plt.xlabel('Valors per n')
plt.ylabel('Nombre de congruencies en Z/n per cada valor de n')
plt.show()
```



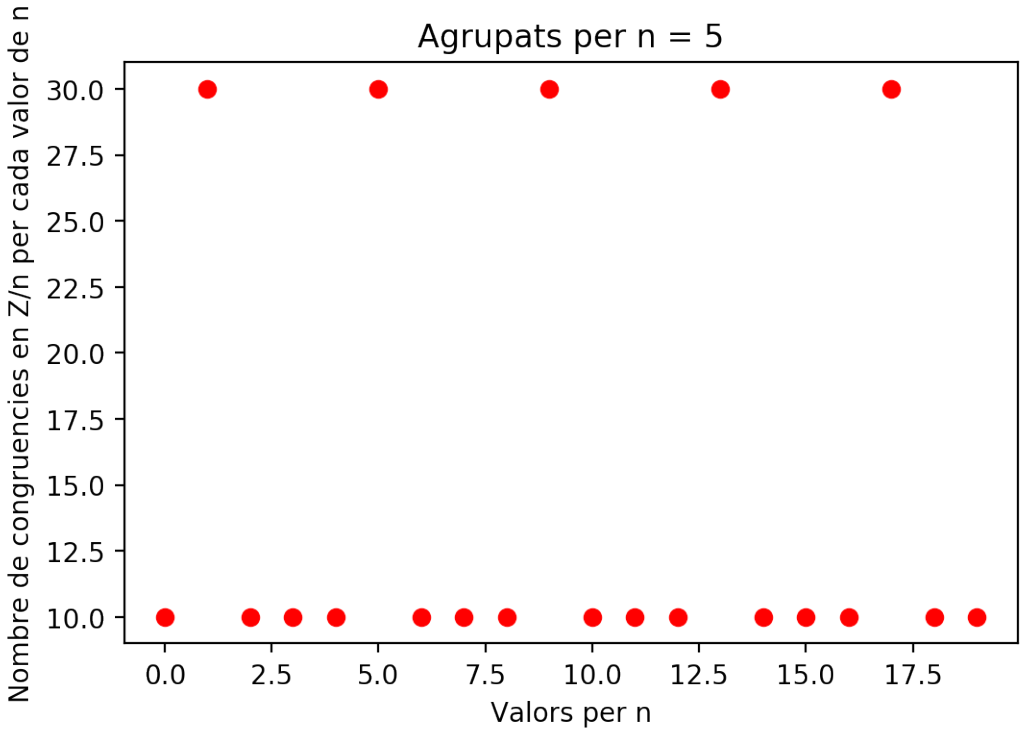

```
plt.plot(seq10.keys(), seq10.values(), 'ro')
plt.title(”Agrupats per n = 5”)
plt.xlabel('Valors per n')
plt.ylabel('Nombre de congruencies en Z/n per cada valor de n')
plt.show()
```



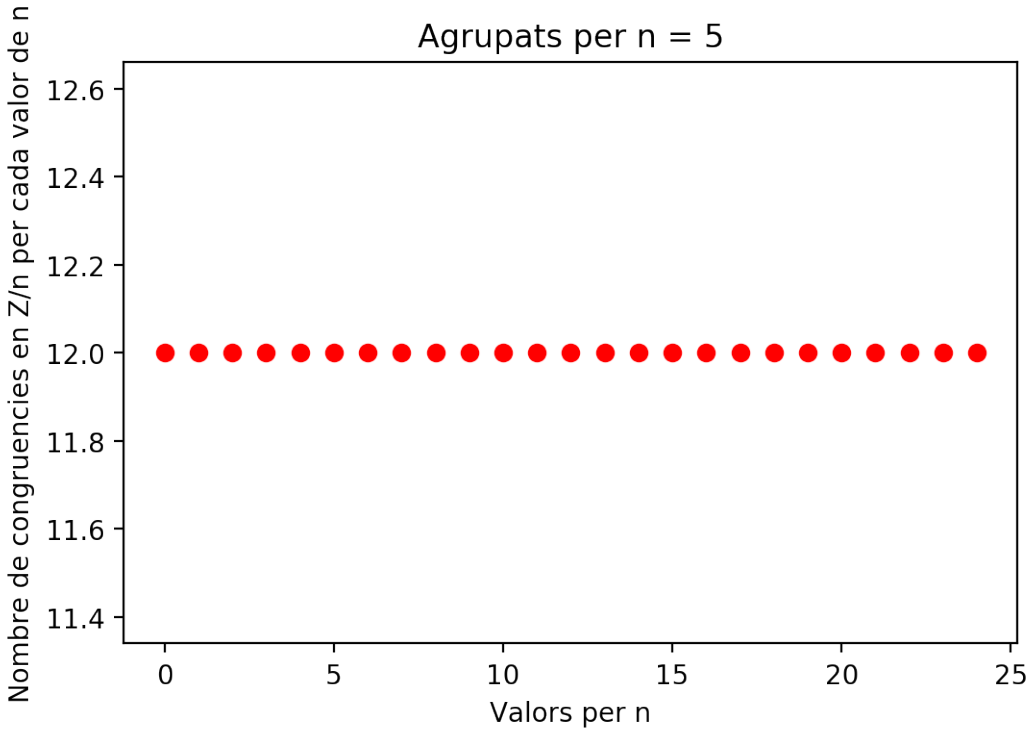

```
plt.plot(seq15.keys(), seq15.values(), 'ro')
plt.title(”Agrupats per n = 5”)
plt.xlabel('Valors per n')
plt.ylabel('Nombre de congruencies en Z/n per cada valor de n')
plt.show()
```



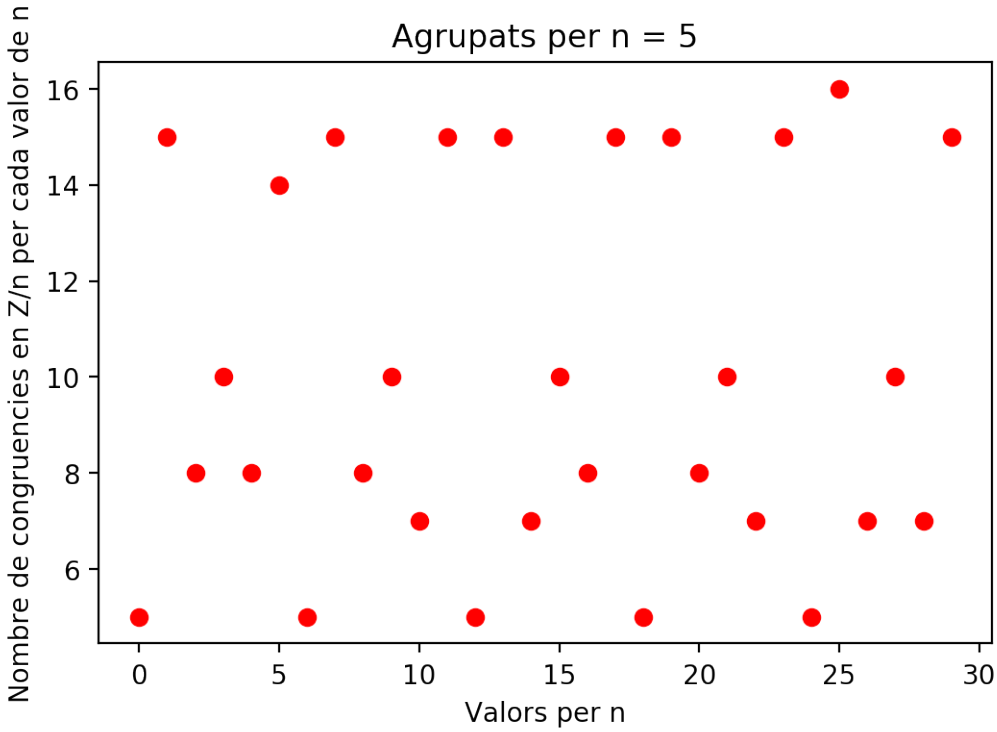

```
plt.plot(seq20.keys(), seq20.values(), 'ro')
plt.title(”Agrupats per n = 5”)
plt.xlabel('Valors per n')
plt.ylabel('Nombre de congruencies en Z/n per cada valor de n')
plt.show()
```




```
plt.plot(seq25.keys(), seq25.values(), 'ro')
plt.title(”Agrupats per n = 5”)
plt.xlabel('Valors per n')
plt.ylabel('Nombre de congruencies en Z/n per cada valor de n')
plt.show()
```




```
plt.plot(seq30.keys(), seq30.values(), 'ro')
plt.title(”Agrupats per n = 5”)
plt.xlabel('Valors per n')
plt.ylabel('Nombre de congruencies en Z/n per cada valor de n')
plt.show()
```



Es pot detectar que per els 300 primers valors de la sèrie de fibonacci sense contar el 0, els gràfics del 5 i del 25 tenen una linealitat molt curiosa. I és que per 300 valors, hi ha el mateix nombre de congruents amb els diferents valors de 0 - n per els valors 5 i 25, concretament amb n = 25 tenim 12 valors exactes de congruència a cada n. Curiosament amb 25 valors podem codificar gairebé totes les lletres de l'alfabet anglés.

La idea de tot plegat és tenir una manera de poder homogeneitzar l'IC de l'encriptat i que no depengui del llenguatge amb el que està escrit. Després de veure això em vaig tirar a la piscina.

Algoritme inicial

Després de l'estudi inicial he realitzat diferents iteracions intentant lligar les propietats de l'aritmètica modular a aquesta característica de la sèrie de fibonacci. Per raons de temps no he explorat les propietats de la suma modular o la resta modular, simplement m'he quedat en la propietat :

Donats :

- a Congruent amb b en mòdul n
- c Com a nombre enter

Tenim que :

a + c Congruent amb b + c

Dit això he generat un algoritme d'encriptament que utilitza una taula de nombres de fibonacci per poder encriptar, i que utiltiza la propietat de la congruència de l'aritmètica modular per poder desencriptar.

Fites

Per motius òbvis de temps s'ha restringit molt l'algoritme inicial per simplificar-ne el seu funcionament. Es donen els següetns axiomes per a l'encriptació de qualsevol text.

- L'alfabet és de 25 caràcters on el primer és la "a" i l'últim la "y". Fent quadrar així el nombre de caràcters de l'alfabet amb el nombre de mòduls possibles a z/n.
- No es processen llavors cap caràcter que estigui fóra d'aquest rang, entre el 97 i el 97 + 25 - 1 en codi ascii.

Taula àurea.

Utilitzant els 300 valors s'ha aconseguit una taula com la següent :

	a	b	c	d	e	f	g	h	i	j	k	l
	0	1	2	3	4	5	6	7	8	9	10	11
0	75025	1	2	3	514229	5	4181	28657	8	34	610	17711
1	12586269025	1	377	3524578	591286729879	55	4052739537881	1,3049695E+15	233	2584	1,9039249E+14	317811
2	2,1114851E+15	4807526976	14930352	1836311903	3,0806152E+14	102334155	3,7889062E+16	3,4164546E+15	701408733	2178309	2,3416728E+16	63245986
3	3,5422485E+20	2,18923E+20	8,3621143E+19	4,494557E+13	7,5401138E+18	3,1940435E+19	1,1000878E+18	5,5279397E+15	6,7989164E+17	4,6600466E+18	2,596955E+17	2504730781961
4	5,9425115E+25	5,7314784E+20	1,5005205E+21	2,4278932E+21	4,073058E+26	3,9284138E+21	3,3116481E+24	2,2698374E+25	6,356307E+21	2,6925749E+22	4,8316295E+23	1,4028367E+25
5	9,9692167E+30	9,2737269E+20	2,9861113E+23	2,7917155E+27	4,6834098E+32	4,3566776E+22	3,2100568E+33	1,0336283E+36	1,8455183E+23	2,0467111E+24	1,5080434E+35	2,5172883E+26
6	1,6724458E+36	3,8079019E+30	1,1825896E+28	1,4544891E+30	2,4400655E+35	8,10559E+28	3,0010821E+37	2,7060741E+36	5,555654E+29	1,725375E+27	1,8547708E+37	5,0095301E+28
7	2,8057117E+41	1,7340252E+41	6,6233869E+40	3,5600076E+34	5,9723043E+39	2,5299087E+40	8,7134745E+38	4,3785198E+36	5,3852234E+38	3,691087E+39	2,0569723E+38	1,9839242E+33
8	4,7068901E+46	4,5397369E+41	1,1885186E+42	1,9230634E+42	3,2261504E+47	3,111582E+42	2,6230599E+45	1,797872E+46	5,0346454E+42	2,13271E+43	3,8269929E+44	1,111146E+46
9	7,8963258E+51	7,3454487E+41	2,3652117E+44	2,2112364E+48	3,7095923E+53	3,4507973E+43	2,5425924E+54	8,1870685E+56	1,4617812E+44	1,6211402E+45	1,1944772E+56	1,9938706E+47
10	1,3246955E+57	3,0161281E+51	9,3669477E+48	1,1520584E+51	1,9327047E+56	6,4202015E+49	2,3770697E+58	2,1434024E+57	4,4004716E+50	1,3666193E+48	1,4691098E+58	3,9679027E+49
11	2,2223224E+62	1,3734708E+62	5,2461917E+61	2,8197782E+55	4,7304881E+60	2,0038669E+61	6,9016891E+59	3,4680979E+57	4,2654784E+59	2,9236024E+60	1,6292678E+59	1,5714085E+54

m	n	o	p	q	r	s	t	u	v	w	x	y
12	13	14	15	16	17	18	19	20	21	22	23	24
987	13	89	6765	165580141	24157817	46368	144	1134903170	21	1597	2971215073	7778742049
5702887	2,777789E+13	4,9845401E+14	832040	956722026041	225851433717	121393	1346269	139583862445	10946	86267571272	53316291173	20365011074
433494437	1,6050064E+17	8,9443943E+15	9227465	7,272346E+13	6557470319842	196418	39088169	1548008755920	267914296	9,9194853E+16	1,061021E+13	32951280099
365435296162	1,220016E+19	1,7799794E+18	1,716768E+13	6,1305791E+16	1,9740274E+19	8,0651553E+14	1,1766903E+14	2,8800672E+18	1,4472334E+16	5,1680709E+19	4,2019614E+17	1,3530185E+20
7,8177408E+23	1,0284721E+22	7,0492525E+22	5,3583593E+24	1,311512E+29	1,9134702E+28	3,6726741E+25	1,140593E+23	8,9892371E+29	1,6641028E+22	1,264937E+24	2,3534128E+30	6,1613147E+30
4,5170905E+27	2,2002057E+34	3,9481089E+35	6,5903462E+26	7,5779162E+32	1,7889033E+32	9,6151855E+25	1,0663404E+27	1,1056031E+32	8,6700074E+24	6,8330028E+31	4,223028E+31	1,6130531E+31
3,433583E+29	1,2712788E+38	7,0845939E+36	7,308806E+27	5,7602132E+34	5,193981E+33	1,5557697E+26	3,0960599E+28	1,2261326E+33	2,122071E+29	7,8569351E+37	8,4040378E+33	2,6099748E+31
2,8945064E+32	9,6633913E+39	1,4098698E+39	1,3598019E+34	4,8558529E+37	1,5635696E+40	6,3881744E+35	9,3202208E+34	2,2812172E+39	1,1463114E+37	4,0934782E+40	3,3282511E+38	1,0716865E+41
6,1922045E+44	8,1462274E+42	5,5835073E+43	4,2442001E+45	1,0388104E+50	1,515604E+49	2,909018E+46	9,0343046E+43	7,1201126E+50	1,3180873E+43	1,0019197E+45	1,8640697E+51	4,8801977E+51
3,5778557E+48	1,7427188E+55	3,1271819E+56	5,2200211E+47	6,0022464E+53	1,4169382E+53	7,6159081E+46	8,4461715E+47	8,7571595E+52	6,86726E+45	5,4122222E+52	3,3449373E+52	1,2776524E+52
2,719641E+50	1,0069429E+59	5,6115003E+57	5,7890921E+48	4,5624969E+55	4,1140009E+54	1,2322798E+47	2,4522988E+49	9,7118387E+53	1,6808306E+50	6,2232492E+58	6,6565933E+54	2,0672849E+52
2,2926541E+53	7,6540905E+60	1,1167167E+60	1,0770594E+55	3,8461795E+58	1,2384579E+61	5,0598866E+56	7,3822751E+55	1,8068857E+60	9,0795981E+57	3,2423248E+61	2,6362106E+59	8,4885164E+61

On a cada columna s'hi sitúa el caràcter al que correspon l'encriptació i el seu valor de congruència a Z/n com a capçalera, mentre que a les files hi trobem els valors dels nombres de la sèrie de fibonacci. A cada columna hi ha els que son congruents entre ells i amb el nombre de la capçalera.

Aquest algoritme té un problema molt gros en termes de criptografia, i és que si algú coneix la taula o coneix el rang de valors que pot prendre la clau generada, és molt fàcil de desencriptar. Per aquest motiu utilitzant l'aritmètica modular s'ha plantejat una segona iteració (millora) en l'algoritme inicial.

Algoritme de xifrat àuric (Substitució)

No segueix cap esquema dels algoritmes que hem vist a classe, ha sigut invenció de l'autor.

L'algoritme de codificació es pot trobar al fitxer auric.py. Es parteix d'un algoritme cíclic, començant per la posició [0,0] de la taula, es llegeix el primer caràcter que es vol encriptar (sempre que estigui a dins del rang d'acceptació, és a dir, que formi part de l'alfabet), es transforma el seu valor en el valor decimal de la taula ascci i s'avança per la taula en horitzontal tantes caselles com el valor del caràcter ens marca, és cíclic per tant la posició 25 és la posició [25 % L][25 // L] on L = 25.

Per exemple, Volem codificar "aa":

- Per la primera lletra "a" s'avancen 97 valors i es cau a la columna 97%L i la fila 97//L. Obtinguent el seu corresponent valor de la sèrie de fibonacci.
- Es porta un comptador dels passos que s'han realitzat per a cada caràcter processat. De moment al llegir només la "a" el comptatge està a 97.
- Per la segona lletra "a" s'avancen 97 valors més des de l'última casella on ens hem situat anteriorment i es cau a la casella [(97+97)%L,(97+97)/L].

Es realitza aquesta metodologia successivament fins a arrivar al final del text xifrat on el resultat del xifrat anterior és "wt" i s'ha generat una clau corresponent a l'últim valor obtingut entre 0 i 299. Aquesta clau és la que marca d'inici del desencriptat.

Per tant la sortida consta de :

- text xifrat
- valor de la clau entre 0 i 299

Algoritme de desxifrat (Substitució)

Utiltizem la mateixa taula per que hem utilitzat per encriptar, ara per desencriptar. L'algoritme per desencriptar és més simple peró té una gràcia afegida que no tenen els altres, i és que per poder desencriptar cal que hi hagi la relació de congruència entre un nombre de fibonacci de la taula i caràcter que segueix al caràcter que estem mirant.

Per poder començar requerim la clau, i és que la casella de sortida serà la casella corresponent al final de l'encriptació, en aquest cas [clau%L][clau//L]. Donada aquesta casella cal :

- Invertir la cadena encriptada (reverse) comencem per l'últim caràcter fins al primer.

- Llegir el següent caràcter, avançar de forma inversa a la que s'ha encriptat per la taula en sentit contrari 97 caselles, ja que hem de moure'ns a dins dels 25 possibles valors de la "a" a la "y".
- Un cop situats hem de començar a comparar la congruència del caràcter següent de la cadena inversa, amb els següents 25 valors existents en forma de cercle, pararem quan es trobi una congruència entre el valor de fibonacci de la casella per la que avancem i el valor de la capçalera de la taula.

Seguin l'exemple anterior i situats a la casella (97+97), si retrocedim 97 valors apareixem a la casella 97, comprovem si el valor de fibonacci de la casella és congruent amb el 22 corresponent al valor de la w xifrada, ho és per tant ja tenim la primera "a", per trobar la segona "a" simplement hem de transformar el valor restant en un caràcter.

Podriem dir que :

donats :

- un caràcter c
- una posició n a la taula
- la posició i del caràcter c al text xifrat

El valor del desxifrat serà el primer valor congruent dins de la primera sèrie de 25 valors a des de n-97 fins a n-97-25, tal que el valor de fibonacci sigui congruent amb la codificació numèrica del caràcter i+1 del text. (veure l'algoritme o cridar-me a tutories per més informació.).

IC En el codificat àuric

Estudiem l'IC del text de frankenstain codificat amb el primer algoritme, algoritme àuric, algoritme que en aquesta segona fase no existeix ja que s'ha modificat per aconseguir les mateixes propietats sense la sèrie de fibonacci.

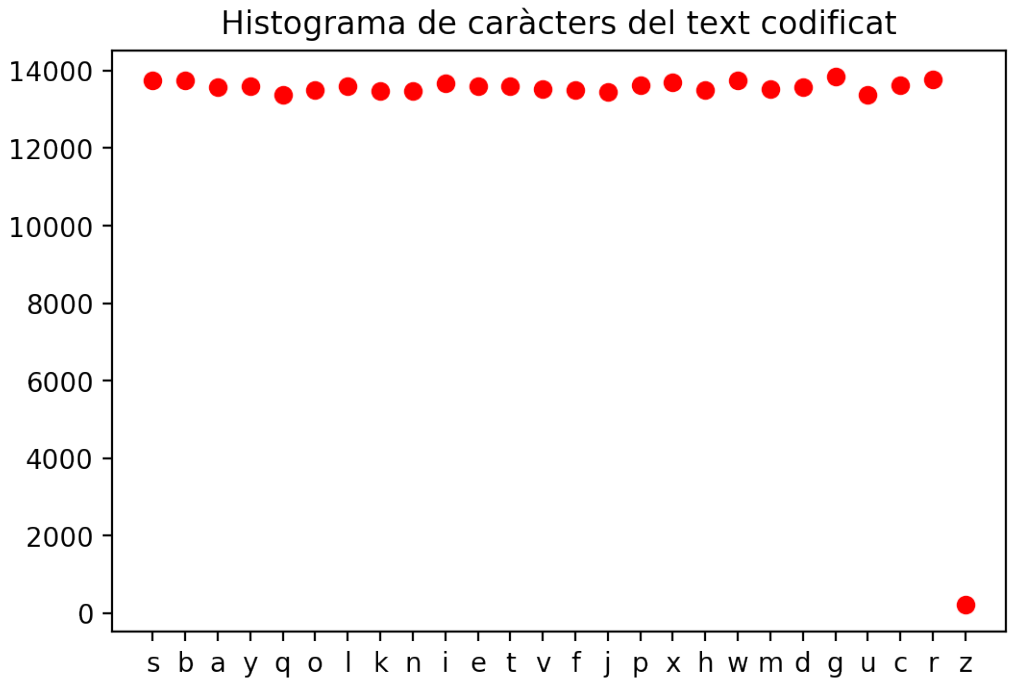
Suposarem un text en anglés per a l'encriptat i el posterior anàlisi de l'Index de coincidència. El text proposat és el llibre de frankenstain que es pot trobar a la referència [1] o bé al directori txt sota el nom de frankenstain.

Després de codificar el text i desar-lo amb el nom de codificat.txt, obrim el fitxer que acabem de generar i filtrem tots els caràcters de l'alfabet fent un histograma de les aparicions.

```
1 histograma = dict()
2 book = ""
3 with open("txt/codificat-auri.txt", 'r', encoding='utf-8') as fileobj:
4     for line in fileobj:
5         for ch in line:
6             if ch >= 'a' and ch <= 'z':
```

```
                if ch in histograma:
                    histograma[ch] += 1
                else:
                    histograma[ch] = 1

plt.plot(histograma.keys(), histograma.values(), 'ro')
plt.title("Histograma de caràcters del text codificat")
plt.xlabel("")
plt.ylabel("")
plt.show()
```



Com podem veure l'IC dels 25 caràcters és molt bo, si fem el càlcul serà molt proper a 1, deixant de banda que la z no s'utilitza pel xifrat.

Algoritme encriptació final (Substitució)

Utitilitzant la mateixa lógica que en l'anterior explicació s'ha realitzat una evolució de l'algoritme àuric per aconseguir desacoblar la clau de l'algoritme. En aquesta segona iteració s'han incorporat els següents features :

- Ús d'una clau o password per poder xifrar i desxifrar.
- Eliminació de la taula àurea i generació d'una taula mitjançant el password.

- El desencriptat no va de redera a endavant sinó que el text final tindrà n+1 caràcters on n és el nombre de caràcters del text inicial (no xifrat).
- Ús d'un delimitador al final del text xifrat que s'utiltiza per desxifrar.
- Incorporació de un caràcter com a marca d'inici de la taula ascii i un caràcter com a marca de fi. (el rang de caràcters que codifiquem.)
- S'han descartat els primers 32 caràcters de la taula ascii peró el rang és modificable.

Així doncs, qualsevol text que estigui en ascii i que contingui valors d'entre el 32 fins el 127 serà processat per l'algoritme, tots els altres caràcters es deixen tal i com estan i l'algoritme rail fence explicat més endavant se n'encarrega de desordenar-los.

La gràcia està en que l'aritmètica modular es pot adaptar de manera concisa de la mateixa manera que s'utilitzava anteriorment peró sense haver de tenir la sèrie de fibonacci sinó amb nombres d'un rang totalment aleatori, peró usant el mateix tipus de procediment.

Ús de la clau

La clau s'utilitza com a llavor per generar els nombres aleatòris que composaran la taula de xifrat. També s'utilitza per marcar el punter de sortida, és a dir, quina casella de la taula serà l'inicial, en l'algoritme àuric sempre era la primera casella de la taula mentre que en aquest és una casella a l'atzar d'entre les possibles.

S'utiltiza la llargada de la clau en nombre de caràcters per a estipular els rails del railfence.

Algoritme de xifrat (Transformació)

Un cop s'ha aplicat l'algoritme de xifrat per subsitució s'aplica l'algoritme de xifrat railfence.

L'aplicació d'aquest algoritme és mitjançant la clau. s'utilitzen un nombre de rails variable entre 1 i |clau| % 25 + 1, per tant hi haurà entre 1 i 25 rails. S'ha optat per aquest sistema per integrar la clau al xifratge per transformació. S'ha volgut aplicar el 25 com a màxim nombre de rails degut a que ha sigut un nombe significant a la realització de la pràctica. El gran esforç en aquesta pràctica s'ha destinat a l'algoritme de substitució.

S'ha utilitzat el mateix algoritme que s'ha vist a classe a les transparències amb la variant que aquest algoritme no requereix alfabet i treballa sobre tots els caràcters. Aquest fet ens ajuda a amagar els caràcters que no es poden xifrar amb l'algoritme de substitució.

Exemples d'execució

En el director txt hi ha els fitxers que s'han utiltizat com a jocs de proves. En el fitxer main.py hi ha 2 jocs de proves preparats per ser executats. A continuació es mostra l'execució del fitxer main.py en aquest informe autogenerat.

```
import os
import sys
sys.path.append(os.getcwd())
from auric import encode
from auric import decode
from RailFence import codifica
from RailFence import descodifica

def doAction(fileName, key, firstChar, lastChar):
    text = ""
    with open(fileName, 'r', encoding='utf-8') as fileobj:
        for line in fileobj:
            for ch in line:
                text += ch

    # matrix, columns, length = auric.generateMatrix(text, L)
    print("")
    encoded = encode(text, key, firstChar, lastChar)
    print("ENCODED TEXT By SUBSTITUTION :")
    print("-----------------------------------------------")
    print(encoded)

    print("")
    fenced = codifica(encoded, len(key))
    print("ENCODED TEXT By SUBSTITUTION + TRANSFORMATION :")
    print("-----------------------------------------------")
    print(fenced)

    print("")
    defenced = descodifica(fenced, len(key))
    print("DECODED TEXT By SUBSTITUTION + TRANSFORMATION :")
    print("-----------------------------------------------")
    print(defenced)

    print("")
    decoded = decode(defenced, key, firstChar, lastChar)
    print("DECODED TEXT By SUBSTITUTION + TRANSFORMATION :")
```

```
    print("-----------------------------------------------")
    print(decoded)


key = "frankenstain"
firstChar = 32
lastChar = 127
shortFileName = "txt/short.txt"
doAction(shortFileName, key, firstChar, lastChar)

print("
    ------------------------------------------------------------------------------
    ")

longFileName = "txt/long.txt"
doAction(longFileName, key, firstChar, lastChar)
```

```

ENCODED TEXT By SUBSTITUTION :
-----------------------------------------------
[L_oz!%:ZBXms#&,?Gn#C*=?NZ`o$9?IX%EHb#pr&@`Xhu#7L\kqu)+2Gg0w(-EO^(H<
    EKXeky

ENCODED TEXT By SUBSTITUTION + TRANSFORMATION :
-----------------------------------------------
[sNbL(kL#Z#\-y_&`pkEo,orqOz?$&u^!G9@)(%n?`+H:#IX2<ZCXhGEB*%ugKX=E#0Xm?
    H7we

DECODED TEXT By SUBSTITUTION + TRANSFORMATION :
-----------------------------------------------
[L_oz!%:ZBXms#&,?Gn#C*=?NZ`o$9?IX%EHb#pr&@`Xhu#7L\kqu)+2Gg0w(-EO^(H<
    EKXeky

DECODED TEXT By SUBSTITUTION + TRANSFORMATION :
-----------------------------------------------
Project Gutenberg's Frankenstein, by Mary Wollstonecraft (Godwin)
Shelle.
------------------------------------------------------------------------------


ENCODED TEXT By SUBSTITUTION :
-----------------------------------------------
```

```
[OX^~02H\bg4Tlrx*4CKw8:INn$-3Bbfv&;ETjpuBb%n/DM]s{%:Zr|2;[dt(;K^+KY
s(@FLa”/16P|=R[]r3M]s4HQaw%*JMS`jp(.Nhx/BbMcw-7FLx9QZjx9Scy-MP]cw,2
79O^s4=?Ddx)IR\dmz5U^n}.DW]b/OQ`e&>GWe&@Pf'4D[af3Skm”BDdh{”$9Obh)-/@BER
Xho02RVis”(H`isw!AP`ou69Od%:CIint,6Cclv%9?LSsw(>KPpy{39Yjp$5;Pcez!&
T9?ATtlv$1;=Kl-28:MSg|=@MSg{”'GKT^kp2Rr\|1AQ`!5>@MZz/5;[u&<\^fhr”BL
[djl$*9e&>GM`f'?Eey#%2?_an{<?Eenp”3M)IKZ_ 5>@Uuy*9M]jp%ESY&FN^hw @BVv
`bp1FVv+AHOUh)3;JZhr”<\^mr38>@U^-

Oqaj7WBXl”,;Ab#C*:MU_v|=KQqx)<\eg~)8@`gw+K[jp1?O]cr(HMWk!4J^sy~?Yi
NFOi*/9>^x)?_cs#*0DX8Xx[q'GL\|,<Qq 0FYh5UZ`bu6>H[h7Ww\l-<La”)/1Dr3S
=U_ly:K^nr ”,:f'p1IS`m.?Rby @Zj!4T^m|-17FJP~?_Ii”,9Fftz(=]r{”BVk{+
ENTVi~3Scj+EUk~?ETZhrx-MPj+9Ss)/1DXxz*/O`su06I],Llfv-Majly'GVf{<AKQ
rl|3_ .Hhy')CJP]jz3_ .Hhl|+<>MWgvCcq,L`j~4:My:KQdn#,L\k,AJPp%)+29IV[
|k{=]}l|>^~h)8>U[n/3CYfk,@Vi!+BHh|-MVfy-7:GMmo0>H\cs'<Ragi

ENCODED TEXT By SUBSTITUTION + TRANSFORMATION :
-----------------------------------------------
[TNTs(LrJBZwD^eDhBsotsjTKM2@^$E?_j vhUXUejk
?[0H”l' 7Fkcr16lrCl,d%=Ui-sOlnj{;a3Mbj,dn&[{E”u,wp9lSRMf*eE
p@+r^l_gp!N_qF[)yp@Ff{jxDIylJ|Ln)][!7'Xr$p%K”MSMx2x}>a”R(66($?-grZh9ye5
   %BA”-”v~14Fc'Yh/:1ZJt++-X]'|P+`#+}n+:<^x-u:^/]`c9
).Gf$
H9C>5A2{\zre#n>EVH<
,|)?JOsGh71KIjPz
EMx,G3]<j,2l/BGR~*3BZ+1sjwS7IDW39X`OcK;T8”|/”&%p@SvO\
;=8O^i#L5WD^S!~(EUPzLV_j>~L9|3HMa04BbrK64p-c9RWeSOhidlPPt:'15B>2”UY
U^OAK@]s**\Uwrn`4?=Nkj*lf
zM4\I>Chmg2Cb%|YPH(7yO\]&kbos%vpclMGA;LG?3u&`hmqbQ`cy/0|Z\3rmT_]T~+/f
   {.3W:kV^Y|oiHKfn2
|Q.F-^db@mh0w:%yevSKQ[
M_MyFb)ra#qgr~9D,`lS
.^IrV?9Ov<H_gM,[~f-0\wv/;s=aNLMsm/P”)2!C9{z$gT`u[`a)*Np33jCxw(?>X<b-
”?mi{iES`-Ah vyA
hkM>b8&D[(RwhxP4zOfB-
RAI?3!1|^!&dfnI9^1;87*)+HY^8Qu<=,R|””~TssMKy.C:J|),VHg:;Md@[%x9]=5Q'D/
   VPiL9&;=k5<j'{KMhFJ>W:<KMixXq6LU:b-,B3Z)uaQ'HcKPk8@f\4IE]tF]*/Qc?U`4
   d@i`nSY
=@p>\l?<Z]wVZ@BM\[W )x >a_fy19VSh/0j
)hqQp{>Vyc

DECODED TEXT By SUBSTITUTION + TRANSFORMATION :
-----------------------------------------------
[OX^~02H\bg4Tlrx*4CKw8:INn$-3Bbfv&;ETjpuBb%n/DM]s{%:Zr|2;[dt(;K^+KY
```

```
s(@FLa”/16P|=R[]r3M]s4HQaw%*JMS`jp(.Nhx/BbMcw-7FLx9QZjx9Scy-MP]cw,2
79O^s4=?Ddx)IR\dmz5U^n}.DW]b/OQ`e&>GWe&@Pf'4D[af3Skm”BDdh{”$9Obh)-/@BER
Xho02RVis”(H`isw!AP`ou69Od%:CIint,6Cclv%9?LSsw(>KPpy{39Yjp$5;Pcez!&
T9?ATtlv$1;=Kl-28:MSg|=@MSg{”'GKT^kp2Rr\|1AQ`!5>@MZz/5;[u&<\^fhr”BL
[djl$*9e&>GM`f'?Eey#%2?_an{<?Eenp”3M)IKZ_ 5>@Uuy*9M]jp%ESY&FN^hw @BVv
`bp1FVv+AHOUh)3;JZhr”<\^mr38>@U^-

Oqaj7WBXl”,;Ab#C*:MU_v|=KQqx)<\eg~)8@`gw+K[jp1?O]cr(HMWk!4J^sy~?Yi
NFOi*/9>^x)?_cs#*0DX8Xx[q'GL\|,<Qq 0FYh5UZ`bu6>H[h7Ww\l-<La”)/1Dr3S
=U_ly:K^nr ”,:f'p1IS`m.?Rby @Zj!4T^m|-17FJP~?_Ii”,9Fftz(=]r{”BVk{+
ENTVi~3Scj+EUk~?ETZhrx-MPj+9Ss)/1DXxz*/O`su06I],Llfv-Majly'GVf{<AKQ
rl|3_ .Hhy')CJP]jz3_ .Hhl|+<>MWgvCcq,L`j~4:My:KQdn#,L\k,AJPp%)+29IV[
|k{=]}l|>^~h)8>U[n/3CYfk,@Vi!+BHh|-MVfy-7:GMmo0>H\cs'<Ragi

DECODED TEXT By SUBSTITUTION + TRANSFORMATION :
-----------------------------------------------
She paused, weeping, and then continued, ”I thought with horror, my
sweet lady, that you should believe your Justine, whom your blessed
aunt had so highly honoured, and whom you loved, was a creature
capable
of a crime which none but the devil himself could have perpetrated.
Dear William! dearest blessed child!  I soon shall see you again in
heaven, where we shall all be happy; and that consoles me, going as I
am to suffer ignominy and death.”

”Oh, Justine!  Forgive me for having for one moment distrusted you.
Why did you confess?  But do not mourn, dear girl.  Do not fear.  I
will proclaim, I will prove your innocence.  I will melt the stony
hearts of your enemies by my tears and prayers.  You shall not die!
You, my playfellow, my companion, my sister, perish on the scaffold!
No!  No!  I never could survive so horrible a misfortune.
```

En cas de voler executar la codificació i descodificació es poden realitzar les següents instruccions.

```
python3 encode.py txt/frankenstein.txt text-codificat.txt
   lamevaparauladepas
```

```
python3 decode.py text-codificat.txt text-descodificat.txt
   lamevaparauladepas
```

Propietats de l'algoritme presentat

- Tant la clau com el text poden fer canviar la configuració de l'algortime.
- Es basa en la clau per generar les taules de coincidència per a l'aritmètica modular.
- Cada caràcter, en funció del caràcter que té al costat pot prendre un valor o un altre, tinguent en compte les taules generades.
- Es cerca homogeneitzar l'IC mitjançant aritmètica modular i s'aconsegueix fent que l'IC sigui molt proper a 1.
- L'aritmètica modular també s'utilitza per desxifrar.
- Hi ha un tractament especial de l'últim caràcter, el text xifrat sempre tindrà n + 1 caràcters sent n el nombre de caràcters del text desxifrat.

Pràctica 5 - Treball sobre l'algoritme.

Ens basem en el resultat obtingut al codificar el llibre frankenstain vist en els apartats anteriors.

Anàlisi a fer

- Tipus d'algoritme.
- Taula de freqüències i càlcul de l'IC.
- Ràtio absoluta del llenguatge.
- Ràtio verdadera del llenguatge.
- Càlcul de la redundància del llenguatge.

Tipus d'algoritme

És un algoritme de clau secreta. S'utiltiza la clau per construïr l'estructura de dades que permet realitzar el xifrat. En aquest algoritme la fortalesa rau en el desconeixement de la clau.

Taula de freqüències i Càlcul de l'IC.

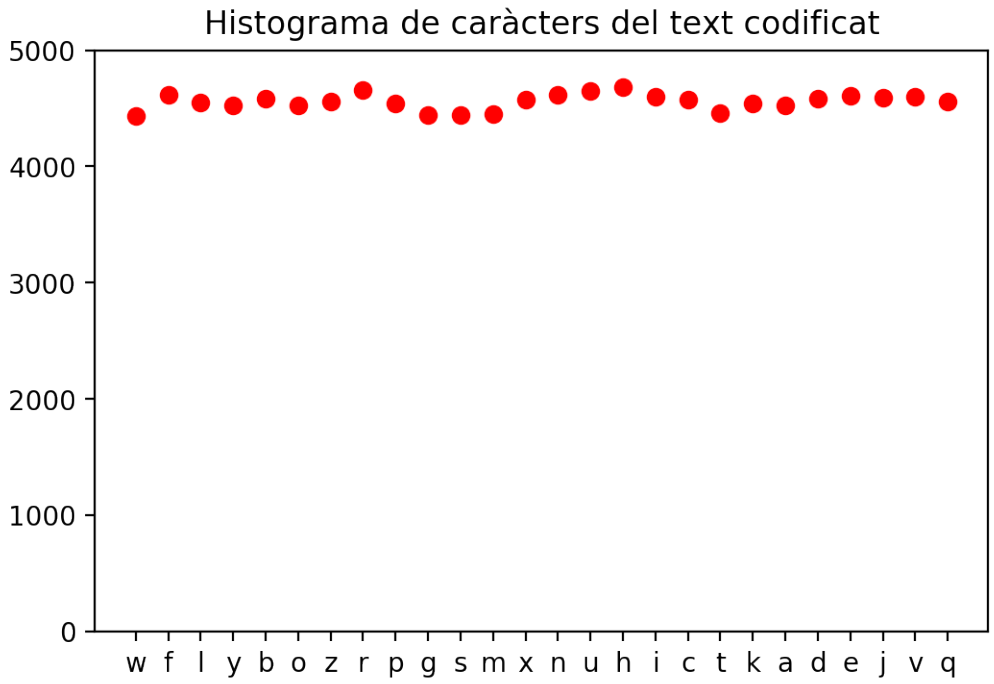
Taula de freqüències.

Realitzem el càlcul de l'IC del resultat de l'encriptació. Primerament mirem l'histograma d'aparicions de l'algoritme final, en aquest cas utiltizem el fitxer txt/codificat-modular.txt per veure fins a quin punt l'histograma de caràcters ens és prou bo per poder distingir o no si l'IC serà l'esperat, en general de

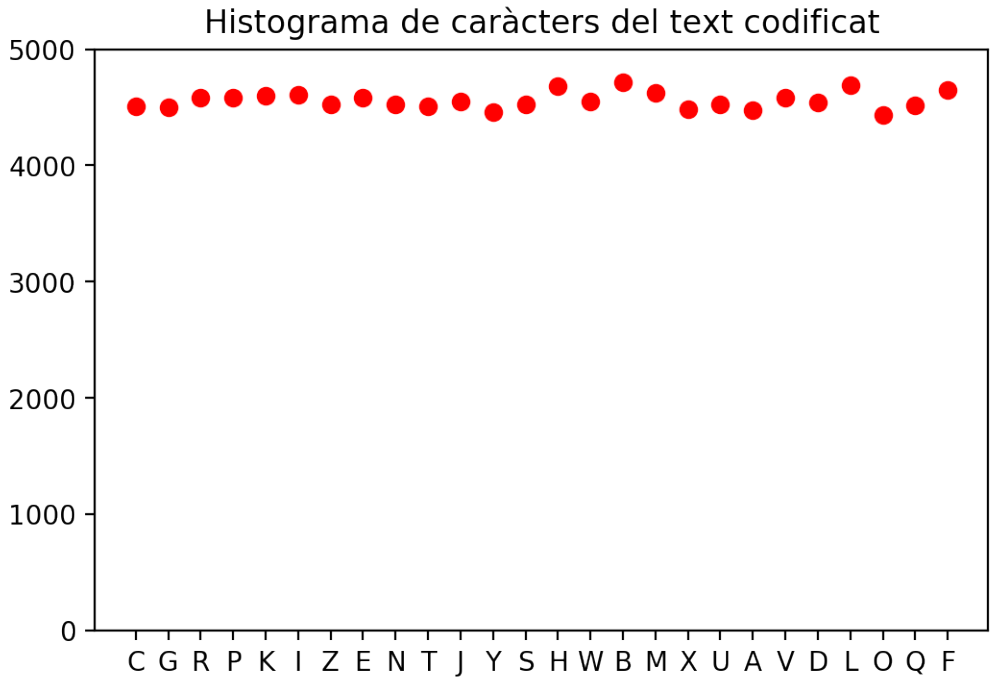
la mateixa manera que el resultat de l'histograma de caràcters en el text codificat-auri.txt ha sigut bo per nosaltres, s'espera que utilitzant l'algoritme nou aquest també ho sigui.

```
histograma = dict()
book = ""
with open("txt/codificat-modular.txt", 'r', encoding='utf-8') as
    fileobj:
     for line in fileobj:
         for ch in line:
             if ch >= 'a' and ch <= 'z':
                 if ch in histograma:
                     histograma[ch] += 1
                 else:
                     histograma[ch] = 1

plt.plot(histograma.keys(), histograma.values(), 'ro')
plt.title("Histograma de caràcters del text codificat")
axes = plt.gca()
axes.set_ylim([0,5000])
plt.xlabel("")
plt.ylabel("")
plt.show()
```



```
1
2  histograma = dict()
3  book = ””
4  with open(”txt/codificat-modular.txt”, 'r', encoding='utf-8') as
       fileobj:
5      for line in fileobj:
6          for ch in line:
7              if ch >= 'A' and ch <= 'Z':
8                  if ch in histograma:
9                      histograma[ch] += 1
10                 else:
11                     histograma[ch] = 1
12
13 plt.plot(histograma.keys(), histograma.values(), 'ro')
14 plt.title(”Histograma de caràcters del text codificat”)
15 axes = plt.gca()
16 axes.set_ylim([0,5000])
17 plt.xlabel(””)
18 plt.ylabel(””)
19 plt.show()
```



Com podem observar els caràcters es mouen entre les 4400 i les 4800 aparicions! **l'histograma, igual**

que hem vist en l'apartat de l'auric és molt prometedor per el càlcul de l'IC!

Càlcul de l'IC

En el cas d'aquest algoritme s'utilitzen els caràcters només des del nº 32 en codi ascii fins el 127 en codi ascii. Per tant el nombre de caràcters de l'alfabet l'establim com 127 - 32, és a dir **L = 95**. Primer de tot generem l'histograma sencer per tots els caràcters.

```
histograma = dict()
book = ””
with open(”txt/codificat-modular.txt”, 'r', encoding='utf-8') as
    fileobj:
     for line in fileobj:
         for ch in line:
             if ch >= chr(32) and ch <= chr(126):
                 if ch in histograma:
                     histograma[ch] += 1
                 else:
                     histograma[ch] = 1

print(histograma)
```

```
{'_': 4648, '~': 4629, '^': 4617, 'w': 4436, 'C': 4504, 'G': 4502,
'R': 4582, '(': 4614, '\\': 4670, 'f': 4616, '*': 4627, 'l': 4546,
'P': 4583, 'y': 4527, 'K': 4602, ',': 4584, 'b': 4581, 'I': 4606, 'o':
4523, 'Z': 4526, 'E': 4579, 'z': 4558, 'r': 4654, 'N': 4525, '&':
4576, 'p': 4544, '7': 4526, '{': 4515, 'g': 4444, 'T': 4506, '|':
4648, 'J': 4550, '-': 4463, 's': 4444, 'm': 4446, '9': 4616, 'x':
4577, '%': 4543, 'n': 4613, '1': 4515, 'u': 4645, ']': 4577, '6':
4750, '`': 4660, 'Y': 4457, ”'”: 4411, 'h': 4680, 'i': 4601, 'S':
4526, '}': 4610, 'c': 4571, 'H': 4678, 'W': 4551, '[': 4488, '2':
4519, 't': 4460, 'B': 4716, ':': 4473, ';': 4472, 'M': 4627, '$':
4536, '>': 4515, 'X': 4484, 'k': 4543, '?': 4645, 'a': 4521, 'U':
4524, '<': 4443, ')': 4553, 'd': 4583, '.': 4425, '@': 4487, 'A':
4474, 'V': 4582, '3': 4580, 'D': 4540, '5': 4523, ' ': 4480, '”':
4623, '!': 4505, 'e': 4609, '+': 4537, 'L': 4692, '0': 4512, 'j':
4588, 'O': 4434, '/': 4654, '8': 4494, 'Q': 4513, 'v': 4603, '#':
4531, '=': 4623, 'q': 4558, 'F': 4650, '4': 4534}
```

Definim :

- L : Com el nombre de caràcters diferents del llenguatge, on $L = 25$.

- n : Com el nombre de parells de lletres iguals, on $n = \sum_{i=1}^{L} f(x_i)(f(x_i) - 1)$.
- c : Com el nombre de parells de lletres possibles, on $c = N(N - 1)$.

Establim el càclul de l'IC com :

$IC = \frac{\sum_{i=1}^{L} f(x_i)(f(x_i)-1)}{N(N-1)}$

En el nostre cas el calculem de la següent manera :

```
1 import Utils
2 ic = Utils.ic_calculation("txt/codificat-modular.txt", 32, 126)
3 print("IC = " + str(ic))
```

```
1 L=95
2 IC = 0.9648614759341749
```

Podem veure com el nostre IC s'acosta molt a 1, de fet és inferior a 1, això ens indica que és un xifrat polimòrfic.

Càlcul de l'Entropia i de les ratios.

Es vol calcular l'entropia del llenguatge anglés i comparar-lo després amb el fitxer que s'ha codificat. Per fer-ho el primer que hem de fer és aconseguir una relació de paraules en anglés. Per fer-ho (en màquines unix) podem consultar directament el fitxer /usr/share/dict/words. On hi ha totes les paraules escrites en anglés. La idea és comparar les ratios, del llenguatge en anglés, i del text xifrat per veure o per intentar deduïr amb quin idioma està escrit el text xifrat.

Volquem el fitxer en un fitxer de text dins d'aquest directori.

```
1 cat /usr/share/dict/words > ~/Projects/seguretat/seguretat-p3/txt/words
    .txt
```

Hem de cercar el nombre de caràcters de l'alfabet. Considerant que estan tots els mots en minúscoles, fem ús del fitxer que acabem de generar amb tots els mots de l'alfabet anglés. Tot seguit contem el nombre de caràcters que apareix en aquest fitxer i generm el diccionai d'aparicions de paraules de llargada n.

```
1 # Generem el diccionari d'aparicions dels caràcters.
2 histogram, length = Utils.getCharactersDict("txt/words.txt")
3 L = len(histogram.keys())
4 print("L=" + str(L))
```

```
L=53
```

Veiem que existeixen 53 caràcters per processar, de fet son els caràcters referents a les majúscoles, minúscoles i algun signe així com el signe "-".

Sabem que considerant el nombre de símbols que apareixen al llenguatge que estem estudiant, la seva entropia és $H(X) = log_2(L)$ on $L = 53$.

Per tant la seva ràtio absoluta és :

```
print("H(X)=" + str(Utils.getAbsoluteRatio(L)))
```

```
H(X)=5.7279204545632
```

Aquesta ràtio ens pot parlar de quines llargades tenen els mots en el llenguatge anglés, recordem que estem fent aquest procés per què és l'idioma amb el que s'ha codificat el llibre de frankenstain.

Anem a calcular la ratio verdadera del llenguatge anglés.

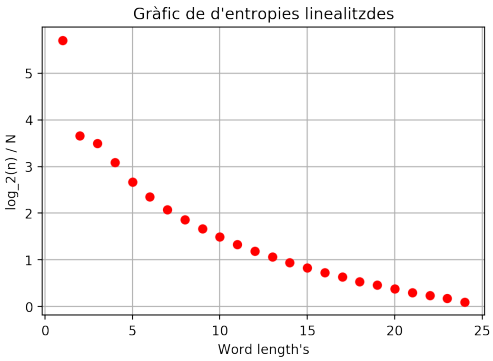
Generem el diccionari d'aparicions de mots de llargada n. Per fer-ho ens ajudem altre cop del fitxer Utils on hi tenim les funcions que realitzen les tasques de indexació. Un cop fet hem d'extreure els valors de r per cada mot aparegut.

```
histograma_mots, max = Utils.getWordsLengthDict("txt/words.txt")
print(histograma_mots)
```

```
{1: 52, 2: 160, 3: 1420, 5: 10230, 4: 5272, 8: 29989, 7: 23869, 9:
32403, 6: 17706, 11: 26013, 10: 30878, 12: 20462, 14: 9765, 16: 3377,
15: 5925, 20: 198, 19: 428, 17: 1813, 13: 14939, 18: 842, 21: 82, 22:
41, 23: 17, 24: 5}
```

Vegem les aparicions en forma de gràfic un cop calculats els seus valors de ratio verdadera per tots els nombres d'aparicions, $log_2(n)/N$.

```
rs = Utils.evalDictionary(histograma_mots, max)
Utils.plotGraphic(histograma_mots.keys(), rs, "Gràfic de d'entropies
    linealitzdes", "Word length's", "log_2(n) / N")
```



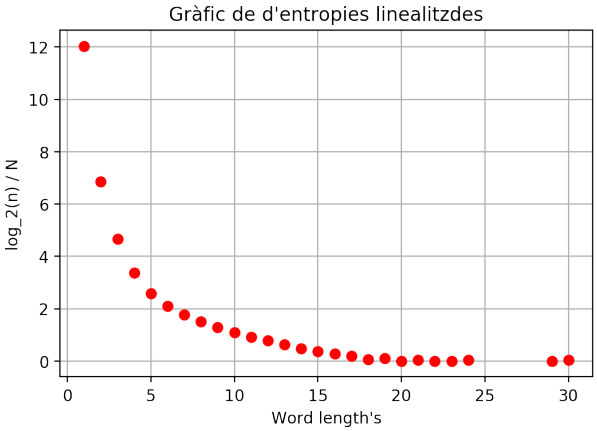
$\mu = 100,\ \sigma = 15$

Veiem que el gràfic presenta linealitat tal i com s'espera en un llenguatge donat la seva ràtio verdadera, en aquest cas tenim més nombres de paraules i la L és més grossa, de moment encara no ens afecta.

Tot seguit fem els càlculs amb el llibre de frankenstain sense xifrar per poder-lo comparar amb el llibre de frankenstain xifrat.

```
histograma_mots_fran, max_fran = Utils.getWordsLengthDictSpaceSeparator
    (”txt/frankenstein.txt”)
rsf = Utils.evalDictionary(histograma_mots_fran, max_fran)
print(rsf)
Utils.plotGraphic(histograma_mots_fran.keys(), rsf, ”Gràfic de d'
    entropies linealitzdes”, ”Word length's”, ”log_2(n) / N”)
```

```
[1.7801016931630305, 0.9296738357080714, 0.6257251015554559,
6.867883773633319, 3.3688066944355666, 0.4766018101408211,
1.5090165755209042, 2.593769351779259, 4.669447025746551,
2.1141612875239795, 0.7869119579873941, 1.3022493620319517,
0.19540753499337427, 1.1035486451292154, 12.034455095327186,
0.27451983892367254, 0.3781616894647664, 0.10526315789473684,
0.041666666666666664, 0.05555555555555555, 0.0, 0.03333333333333333,
0.047619047619047616, 0.0, 0.0, 0.0]
```



$\mu = 100,\ \sigma = 15$

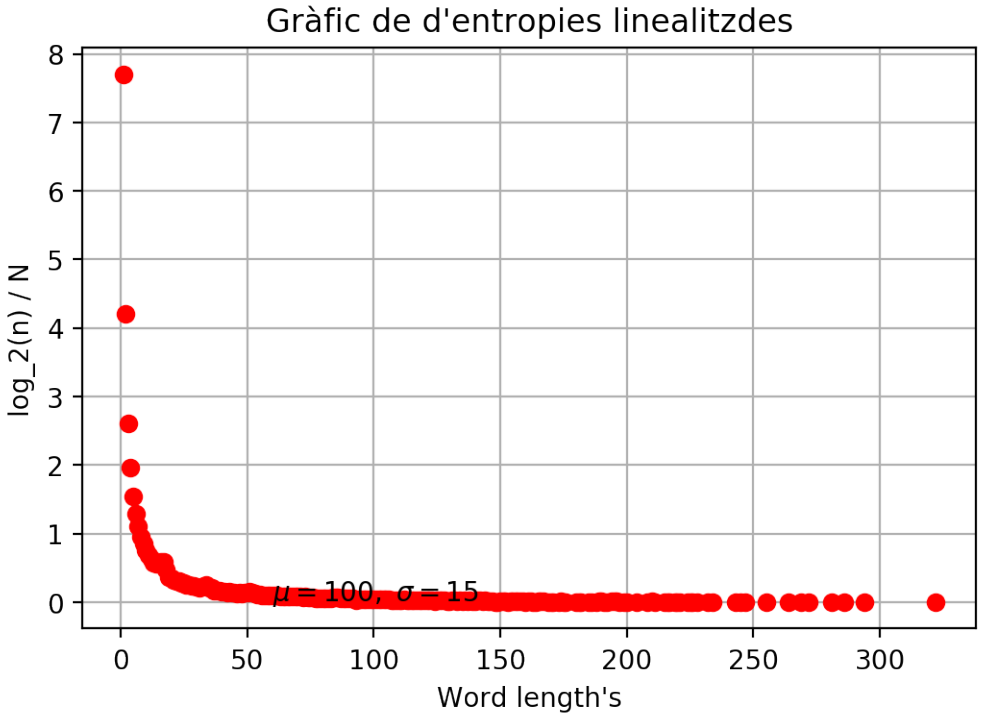
Amb aquestes dades veiem com ens afecta ara la ràtio d'entropia del llenguatge que hem generat amb l'encriptació. Fem els mateixos passos peró amb el fitxer que s'ha generat codificat-modular.txt que és el resultat de codificar el llibre de frankenstain. En aquest cas hem de fer ús d'una funció que ens separi les paraules per cada línia. Aquest gràfic comença a fer podor de logaritme. I és que tenim moltes dades atípiques, no estem contant els signes de puntuació que al cap i a la fi ens treuran la linealitat del gràfic.

```
histograma_mots_fran, max_fran = Utils.getWordsLengthDictSpaceSeparator
    (”txt/codificat-modular.txt”)
rsf = Utils.evalDictionary(histograma_mots_fran, max_fran)
print(rsf)
Utils.plotGraphic(histograma_mots_fran.keys(), rsf, ”Gràfic de d'
    entropies linealitzdes”, ”Word length's”, ”log_2(n) / N”)
```

```
[0.1341784600673061, 0.5827612841636713, 0.6846710418651193,
0.28626519664930855, 0.857899909856354, 1.1116228162178088,
0.27816785241547504, 7.700439718141093, 0.20681156398071657,
4.204695468068851, 1.961372512736094, 0.9554820237218405,
0.2369091282614967, 0.2549752546184465, 0.24442694516538704,
0.5547952063258241, 0.24892718744915238, 0.1490556031659931,
0.05583886201296061, 0.0425531914893617, 0.30906628072948567,
0.7584962500721156, 0.58398693894602, 0.06072476243909466,
0.11498987714127687, 0.03008201407510694, 0.09503443026584577,
0.14752082266778457, 0.6345591536762673, 2.608849495763639,
0.05283208335737188, 0.07731309849006472, 0.09105883363405963,
0.1389256106542368, 0.3283246553956164, 0.06537920352773423,
0.14285714285714285, 1.2968932855874435, 0.04365583934229659,
0.08812185314712614, 0.47568013467894854, 0.0748932975385939,
0.044594316186372975, 0.15619818783608966, 0.09546534090938666,
```

```
0.005494505494505495, 0.019435808276098917, 0.04528162291524496,
0.09828344341622573, 0.013793103448275862, 0.06280818322531284,
0.08983124201144367, 0.250181018598647, 0.04207010960978582,
0.5581272279440741, 0.14326774427675235, 0.06874949785473432,
0.13526706392804436, 0.1327940099299602, 0.04550440427180916,
0.15330249314298167, 0.08472481910804124, 0.3707575852293923,
0.17182572985454284, 0.23409245898848705, 0.015903617088269605,
0.22992350014431515, 0.12156497161978734, 0.0879143360935909,
0.037231473227163504, 0.5817109557093133, 0.1348019820101727,
0.21658856508600394, 1.538697391499865, 0.09754046613836388,
0.054032812524584875, 0.16210072038172724, 0.31340080742308835,
0.017860985345287402, 0.06077583276335486, 0.10783158795568885,
0.09407887119620552, 0.20510289701316814, 0.08258440966972264, 0.14,
0.22691183073525345, 0.022970471024944296, 0.020950409866101524,
0.05704308341271118, 0.17289093706134073, 0.05803750000912857,
0.3522197059679227, 0.049004743061948325, 0.08719526518429499,
0.1301651565300747, 0.06461003642740794, 0.04045645250898652,
0.034263330723528634, 0.004901960784313725, 0.04661654413885754,
0.08816254620311222, 0.08105475355159816, 0.0630961017988616,
0.020583928580794237, 0.020062816464824763, 0.04858307520404474,
0.08958501888531309, 0.021932460328575033, 0.0, 0.0,
0.04549397994618277, 0.04916915169627188, 0.0, 0.03064070513436886,
0.0426438584735802, 0.025158134932081844, 0.030381038141704717,
0.05281392941731565, 0.024960039381435532, 0.05953661781762467,
0.041666666666666664, 0.01909765253100411, 0.030199346317157844,
0.021430190244714535, 0.024014583344259943, 0.009108979889202048,
0.02484384886776641, 0.03255742163007099, 0.06997236684591447,
0.09901502087324078, 0.0309048491441479, 0.028590343955680147,
0.009844487582118984, 0.054383312422406065, 0.0, 0.012345679012345678,
0.03350432243664632, 0.008341907898532402, 0.006711409395973154,
0.010309278350515464, 0.014598540145985401, 0.02306894510338446,
0.06694030054276763, 0.04287915017920478, 0.010582010582010581,
0.008086543371026308, 0.005405405405405406, 0.009490793417491953,
0.005681818181818182, 0.0, 0.030125735300177786, 0.029384938530501285,
0.0328895642267996, 0.02330827206942877, 0.027675452949098383,
0.030084875757244653, 0.0291397201305909, 0.0058823529411764705, 0.0,
0.008128012824211057, 0.019230769230769232, 0.005025125628140704,
0.02112676056338028, 0.03408521506927218, 0.0, 0.01498018125733782,
0.018715699480384027, 0.0, 0.016129032258064516, 0.02355977372260541,
0.010095302552364053, 0.014072291484165831, 0.020052535157554317,
0.017118956958418252, 0.014603321351492844, 0.006097560975609756,
0.006535947712418301, 0.01527584272952212, 0.034300494793311744,
0.02003846899783842, 0.017199467369536016, 0.005714285714285714, 0.0,
```

```
0.006756756756756757, 0.0, 0.0, 0.0, 0.0075474404796245535, 0.0,
0.013987518643899773, 0.0, 0.005917159763313609, 0.0, 0.00625, 0.0,
0.0, 0.004424778761061947, 0.005376344086021506, 0.0, 0.0, 0.0, 0.0,
0.016704518668254405, 0.0, 0.011083653851196897, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```



Veiem que la ratio d'entropies verdaderes del llenguatge difereix molt, ja podem parlar ara no, d'una funció que tendeixi a la linealitat sinó d'una que tendèix al logaritme invers.

Per tant podem determinar que el nostre xifrat no dóna indicis clars de que el text estigui escrit amb el mateix idioma que l'anglés. Aquest fet es pot denotar ja que els nostres espais formen part del llenguatge de codificació dels caràcters i qualsevol lletra o número pot convertir-se en espai o en signe de puntuació i viceversa.

Redundància del llenguatge resultant

Podem parlar de la redundància d'un llenguatge com la distribució dels parells, tripletes i les paraules del mateix dins del text. De fet tal com es cita a les transparències és la diferència entre la ratio absoluta i la ràtio verdadera.

$D = R - r$

En el nostre cas la redundància s'esquiva, a l'hora de la transformació seguim sense contar que existeixi redundància en els caràcters ja que pot ser totalment variable. Els caràcters no tenen motiu per asimilar-se directament a altres caràcters, de la mateixa manera que les paraules no tenen per què semblar-se entre elles, son totalment aleatòries.

Análisi dels caràcters en funció del següent

Com ja hem vist, l'algoritme es basa en codificar un caràcter en funció de l'anterior, és a dir en funció de on vam deixar en la taula l'anterior caràcter podem assegurar on anirà el següent sempre que es conegui quina és la representació de la taula de caràcters, la mida de la mateixa, i sobretot, el més important, sempre que es conegui de quins nombres està formada la taula.

S'ha fet processos iteratius per jugar a descodificar mitjançant les restes dels caràcters aviam si n'hi havia algun que es pogués utilitzar com a caràcter bo peró com que l'algoritme es centra en la modularitat no podem saber on corresponen els diferents caràcters.

Deduim doncs que finalment la força de l'algoritme recau en la clau.

Complexitats

Complexitat de la codificació.

El cost de l'encriptació és de O(n) on n es correspon a la llargada del text.

Complexitat de la descodificació

El cost de la descodificació és O(n) on n es correspon a la llargada del text.

Complexitat de descodificació força bruta.

El més òptim seria llençar un algoritme que pogués generar una taula per cada possible mòdul de 0 a 94 en aquest cas. Per cada taula generada s'hauria de llençar l'algoritme a cada una de les posicions possibles d'inici de la taula.

Per tant l'algoritme de descodificació per força bruta acaba sent de O(n^2) el que representa una broma de mal gust per el creador de l'algoritme. La única manera de fer que l'algoritme fos escalable a descodificació per força bruta exponencial podria ser en la generació de la taula. A l'hora de generar la taula s'hauria d'aplicar difrents iteracions desordenant les columnes de la taula, aquestes iteracions s'haurien de poder generar amb la clau.

L'actual algoritme és molt fluix conta la força bruta, es podria descodificar amb complexitat O(n^2), per millorar-ho s'haurien de desordenar les columnes de la taula.

Referències

- Taula ASCII
- Gutenberg
- Secció àurea, Wikipedia
- série de fibonacci