Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Réseaux & Système

Corrections – TP n°2

Python

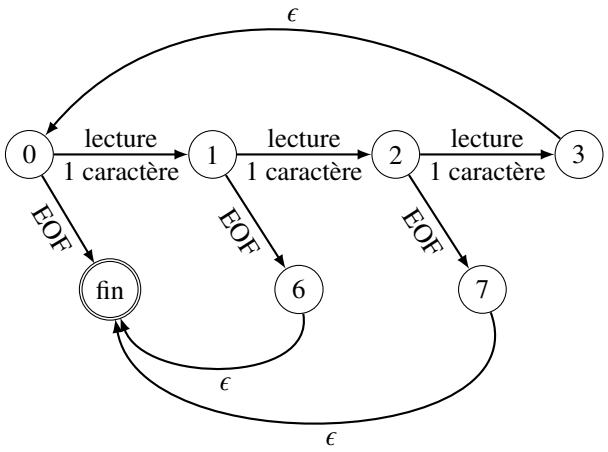
Manipulation binaire

1 – Utilisation d'un GCL, « générateur à congruence linéaire », est un générateur de nombres pseudo-aléatoires basé sur des congruences et une fonction affine :

```
#!/bin/python3

# parametres
# Knuth
m = 2**64
a = 6364136223846793005
c = 1442695040888963407

valeur_courante = 0

def init_lcg():
    global valeur_courante
    valeur_courante = 0

def lcg():
    global valeur_courante
    valeur_courante = (a*valeur_courante+c) % m
    return valeur_courante

def sequence_aleatoire():
    lcg()
    notation_hexa = "{0:08X}".format(valeur_courante)
    octets_hexa = [notation_hexa[n]+notation_hexa[n+1] for n in range(0,len(nota
tion_hexa),2)]
    print (octets_hexa)
    octets_entier = [int(x,16) for x in octets_hexa]
    return octets_entier

def chiffrer(m):
    message = [ord(c) for c in m]
    print ("message :", message)
    nb_sequences = int(len(m)/8) + 1
    seq_alea = []
    for i in range(0,nb_sequences):
        seq_alea += sequence_aleatoire()
    operation_chiffre = list(zip(message,seq_alea))
    print ("Operation chiffrement:", operation_chiffre)
    chiffre = [c^v for (c,v) in operation_chiffre]
    return chiffre

message_chiffre = chiffrer("Salut c'est moi !")
print (message_chiffre)

init_lcg()
message_clair = chiffrer(''.join([chr(x) for x in message_chiffre]))
print (message_clair)
```

2 – Écrire un programme réalisant l'**encodage base64** d'un fichier conformément à la **RFC 2045**.

Automate gérant les cas de terminaison d'abord :



état	opérations
1	1 caractère lu : 8bits, écriture d'un caractère sur 6bits, reste 2bits
6	reste 2bits: écriture d'un caractère sur 6bits avec les 2bits complétés avec '0000' et de '=='
2	1 caractère lu : 8bits, écriture d'un caractère sur 6bits avec les 2bits précédents, reste 4bits
7	reste 4bits: écriture d'un caractère sur 6bits avec les 4bits complétés avec '00' et de '='
3	1 caractère lu : 8bits, écriture de deux caractères sur 6bits avec les 4bits restants

```
#!/usr/bin/python3

import sys

base64 = [chr(x) for x in range(ord('A'),ord('A')+26)] + [chr(x) for x in range(ord('a'),ord('a')+26)]
+ [chr(x) for x in range(ord('0'),ord('0')+10)]+ ['+', '/']

def binaire(c):
    rep_binaire = bin(ord(c))[2:]
    rep_binaire = '0'*(8-len(rep_binaire))+rep_binaire
    return rep_binaire

try:
    entree = open("td1_exo12.py","r")
    sortie = open("td1_exo12.py.base64","w")
except Exception as e:
    print (e.args)
    sys.exit(1)

while True:
    car1 = entree.read(1)
    if not car1:
        break
    rep_binaire = binaire(car1)
    sortie.write(base64[int('00'+rep_binaire[:6],2)])
    bits_restants = rep_binaire[6:]
    car2 = entree.read(1)
    if not car2:
        sortie.write(base64[int('00'+bits_restants+'0000',2)])
        sortie.write('==')
        break
    rep_binaire = bits_restants + binaire(car2)
    sortie.write(base64[int('00'+rep_binaire[:6],2)])
    bits_restants = rep_binaire[6:]
    car3 = entree.read(1)
    if not car3:
        sortie.write(base64[int('00'+bits_restants+'00',2)])
        sortie.write('=')
        break
    rep_binaire = bits_restants + binaire(car3)
    sortie.write(base64[int('00'+rep_binaire[:6],2)])
    sortie.write(base64[int('00'+rep_binaire[6:],2)])
entree.close()
sortie.close()
```