DEVOIR SURVEILLE N°1 1.2014.TRIMESTRE.N°3		
Etablissement : OMAR IBN LKHATTAB	DATE : 02/05/2014 DUREE : 2H00MIN	Enseignante : OUSTOUH

Remarques générales:

- L'épreuve se compose de deux problèmes indépendants,
- Toutes les instructions et les fonctions demandées seront écrites en **PYTHON**.
- Les questions non traitées peuvent être admises pour aborder les questions ultérieures.
- Toutes utilisations du correcteur blanc seront sanctionnées

---

PROBLÈME I: REPRÉSENTATION DE GRANDS NOMBRES ENTIERS NATURELS

Contexte du problème :

Pour tout langage de programmation, la représentation des nombres par des types prédéfinis est très limitée (en python, une variable entière v de type **int** est bornée ainsi : **-2147483648<=v<=2147483647**).

Cependant, plusieurs problèmes (par exemple les systèmes CPS, la cryptographie ou encore les applications de gestion) ont besoin d'utiliser des nombres ayant des valeurs et des précisions qui dépassent largement les limites des types prédéfinis par les langages de programmation.

Pour toutes ces applications, les types de base ne conviennent plus et il faudra définir de nouvelles représentations pour ces nombres.

Le problème suivant s'inscrit dans ce contexte. Il s'intéresse à la représentation des nombres entiers positifs pouvant avoir des valeurs très grandes.

A : Représentation par des chaines de caractères

Dans cette partie, on représentera un nombre positif ou nul par une chaine de caractères contenant ses chiffres.

Notations

– On dira qu'une chaine de caractères S est une **ChaineChiffres** si sa longueur est strictement positive et tout caractère de S est un caractère chiffre (les caractères chiffres sont '0','1','2','3','4','5','6','7','8,'9').

– Soit **N** un nombre entier positif ou nul de **NC** chiffres (NC > 0), la valeur décimale (base 10) de N est: $N=C_{NC-1}C_{NC-2}...C_i...C_1C_0$, ($C_0$ chiffre des unités, C_1 chiffre des dizaines,...), on dit que N est représenté par une **chaineChiffre** S, si S contient les chiffres de N comme suit : S= "$C_{NC-1}C_{NC-2}...C_i...C_1C_0$"

Exemple :

Le nombre N=45009876156430987 de 17 chiffres sera représenté par la **ChaineChiffres** S= "45009876156430987" (S [0]='4 ', S[1]= '5 ', S[2]= '0 ' ,...S[16]= '7 ')

❖ Question 1 : Chaine de chiffres

Soit S une chaine de caractères quelconque déjà déclarée et initialisée.

✍ Définir une fonction d'entête **ChaineChiffres()** qui retourne 1 si S est une **ChaineChiffres** ou 0 (zéro) sinon.

Exemple :

– Si S= **"78500120360007"** alors l'appel à la fonction **ChaineChiffres()** retourne 1.
– Si S= "**856942a1478"** alors l'appel de la fonction **ChaineChiffres()** retourne 0.

❖ **Question 2 : Zéros non significatifs**

On suppose que S est une **ChaineChiffres** déjà déclarée et définie.

✍ Ecrire une fonction d'entête **supprimer_zeros()** qui supprime les zéros à gauche de la chaine S (les zéros non significatifs dans un nombre)

Exemple :
Si S= "0009760004300", après l'appel de **supprimer_zeros( ), S= "9760004300".**

Question 3 : Somme de deux chaines de chiffres.

Il s'agit de faire la somme de deux nombres entiers positifs représentés par leurs **ChaineChiffres**.

Pour ce faire:

✍ Écrire une fonction de prototype **additionner(S1, S2)** qui retourne la chaine **SOM** qui est la somme des nombres représentés par les **ChaineChiffres S1** et **S2**.

Rappel :

Les valeurs décimales des codes **ASCII** des caractères chiffres sont indiquées dans le tableau qui suit :

Caractère	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
Code ASCII	48	49	50	51	52	53	54	55	56	57

Exemple :

si S1= "129782004977" et S2= "754022234930" alors après l'appel de la fonction SOM=**additionner(S1,S2),** on aura SOM= "883804239907".

B : Représentation des nombres par des dictionnaires

Dans cette exercice, on utilisera des dictionnaires pour représenter des nombres entiers positifs ou nuls. Pour optimiser la représentation d'un nombre, on se propose de la décomposer en plusieurs parties. Chaque partie peut contenir 4 chiffres.

Soit N un nombre entier positif ou nul de NC chiffres. $N=C_{NC-1}C_{NC-2}...C_i...C_1C_0$, ($C_0$ chiffre des unités, C_1 chiffre des dizaines,...), Alors N sera décomposé ainsi :

N = $\boxed{C_{NC-1}C_{NC-2}C_{NC-3}C_{NC-4}}$ $\boxed{C_{NC-5}C_{NC-6}C_{NC-7}C_{NC-8}}$ ... $\boxed{...C_1C0}$
1ère partie 2ème partie ... dernière partie

Exemple : le nombre N= **8002590300407896420003** peut être décomposé ainsi :

N= 8002 5903 0040 7896 4200 03
1ère partie 2ème partie 3ème partie 4ème partie 5ème partie 6ème partie

Pour représenter un nombre en le découpant ainsi, on va utiliser un dictionnaire définie somme suit :
dic=**{indice_partie :partie}**

Chaque partie du nombre est un élément du dictionnaire contenant la valeur de la partie et son indice. Le nombre $N=C_{NC-1}C_{NC-2}...C_i...C_1C_0$ sera représenté par un dictionnaire comme suit :

dic={1 :$C_{NC-1}C_{NC-2}C_{NC-3}C_{NC-4}$, 2 : $C_{NC-5}C_{NC-6}C_{NC-7}C_{NC-8}$,…,p : $...C_1C0$}

La valeur du champ partie étant déclarée de type **int** , les 0(**zéros**) non significatifs (à gauche) n'apparaitrons pas dans la valeur de la partie (voir exemple).

Exemple : Le nombre **8002590300407896420003** peut être décomposé ainsi :

N= 8002 5903 40 7896 4200 3
1ère partie 2ème partie 3ème partie 4ème partie 5ème partie 6ème partie

et sera représenté par un dictionnaire dic comme suit : dic={1 :8002,2 :5903,3 :40, 4 :7896,5 :4200,6 :3}

❖ **Question 4: Création d'un dictionnaire des parties d'un nombre**

Soit un nombre entier positif ou nul défini par une **ChaineChiffres S** (voir définition de). On se propose de le représenter par un dictionnaire qui contiendra les différentes parties du nombre. Pour ce faire :

✍ Ecrire une fonction de prototype : **`regrouperChiffres(S)`** ; qui permet de créer un dictionnaire pour représenter le nombre défini par la **ChaineChiffres S** (en paramètre) . Cette fonction retourne ce dictionnaire.

Exemple : soit le nombre **330104660000027** représenté par une **ChaineChiffres S="330104660000027"**

L'appel de la fonction **`regrouperChiffres(S)`**, va retourner le dictionnaire dic={1 :3301,2 :466,3 :0,4 :27}

PROBLEME II : ALGORITHME DE CHIFFREMENT RC4

Préambule

Le "WIFI" est une technologie de transmission radio qui permet d'échanger des données, sans se servir de fils. Cette technologie est utilisée pour connecter des ordinateurs à une borne afin d'accéder à un réseau local ou **Internet**.

La propagation des ondes radio à travers l'air étant naturellement non protégée, beaucoup d'études et de recherches ont été réalisées pour sécuriser les données transmises dans les réseaux sans fils.

C'est dans cette perspective que le protocole **WEP (WIRED Eqivalent Privacy)** a été conçu pour assurer la sécurité des réseaux **WIFI**. Ce protocole implémente l'algorithme **RC4** objet de ce problème.

Principe général de l'algorithme RC4

RC4 "Rives Chiffer 4" est un algorithme de chiffrement (algorithme de cryptage) conçu par Ronald Rives en 1987. Cet algorithme transforme à l'aide d'une clé secrète appelée clé de chiffrement, un texte clair en un texte incompréhensible (dit texte chiffré ou crypté) pour celui qui ne dispose pas de la clé de chiffrement.

Il est largement déployé dans les réseaux sans fils pour garantir leurs sécurités et dans plusieurs applications **E_COMMERCE** pour assurer la confidentialité des transactions **Internet**.

Description

RC4 permet de chiffrer un texte en générant une suite d'octets pseudo aléatoires. Cette suite produira le texte chiffré.

L'algorithme s'effectue en deux phases :

– **Une phase d'initialisation** visant à créer la table d'état (un tableau de 256 entiers) en utilisant la clé de chiffrement. Cet algorithme est appelé **Key Scheduling Algorithm, (KSA).**

– Une phase de génération d'octets pseudo aléatoires. Cet algorithme est appelé **Pseudo Random Generator Algorithme (PRGA).**

Le problème suivant s'intéresse à l'implémentation de l'algorithme **RC4 en python.**

A : Chiffrement d'une chaîne de caractères par l'algorithme RC4.

Dans cette partie on se propose de chiffrer (crypter) selon l'algorithme **RC4**, un texte sous forme d'une chaîne de caractères.

Déclarations globales préliminaires.

Dans toutes les questions de la **partie A**, on suppose avoir déjà déclaré les **variables globales** suivantes :

- **L1** et **L2** deux variables entières déjà définies (0<L1<256) et (0<L2<256)
- **Claire** une chaîne de **L1** caractères contenant le texte à chiffrer (crypter).
- **Clef** une chaîne de **L2** caractères contenant la clé de chiffrement.
- **Chiffree** une chaîne de **L1** caractères destinée à contenir le texte après son chiffrement (cryptage).
- **K** une chaîne de **256** caractères
- **S** un tableau de **256** entiers (la table d'état)

Phase 1: Algorithme d'ordonnancement clé (Key Scheduling Algorithm (KSA)

L'algorithme **(KSA)** est décrit comme suit :

– Initialiser d'abord le tableau S(table d'état) avec l'identité
– Initialiser le tableau K avec la chaine Clef (représentant la clé de chiffrement)
– Mélanger les éléments de S de la façon suivante :
– déclarer deux indices entiers i et j et initialiser j à 0
– pour tout i variant de 0 à 255, faire les 2 instructions suivantes :
  – Instruction 1 **: j=(j+S[i]+K[i]) modulo 256**
  – Instruction 2: **Permuter S[i] et S[j]**

Rappel : En python, l'opérateur modulo est représenté par le symbole **%** il s'agit dans cette **phase 1** de mettre en œuvre l'algorithme **(KSA)** en le décomposant en plusieurs fonctions.

❖ **Question 1 : Initialisation du tableau S avec l'identité**

✍ Ecrire la fonction de prototype **`identiteS( )`** ; qui permet d'initialiser les éléments du tableau **S** de telle sorte que chaque élément S[i] (0<= i <256) soit initialisé par la valeur de l'indice i (S[0]=0, S[1]=1,… S[i]=i,… S[255]=255).

❖ **Question 2 : Initialisation du tableau K avec la clé**

✍ Ecrire la fonction de prototype **`initialiserK( )`** ; qui permet d'initialiser la chaîne **K** avec la chaîne Clef comme suit :

K[0]=Clef[0], K[1]=Clef[1],….., K[L2-1]=Clef[L2-1], K[L2]=Clef[0], K[L2+1]=Clef[1]….., K[255]=Clef[…]

Exemple :

Soit la clé de longueur 3 (L2=3) suivante : Clef="ABC" alors le tableau K sera initialisé ainsi :

K[0]='A', K[1]='B', K[2]='C', K[3]='A', K[4]='B', K[5]='C',… , K[255]='A'

❖ **Question 3 : Permutation**

✍ Ecrire une fonction de prototype **`permuterS(i, j )`** ; qui permet de permuter S[i] avec S[j] (on suppose 0<= i <256 et 0<= j <256)

❖ **Question 4 : Algorithme KSA**

✍ Ecrire une fonction de prototype **`KSA( )`** ; qui réalise l'algorithme d'ordonnancement-clé (KSA), défini ci-dessus au début de ce problème.

Phase 2: Pseudo Random Generator Algorithm (PRGA)

L'algorithme **(PRGA)** est décrit comme suit :

- Déclarer 3 entiers **i**, **j** et **a** puis initialiser **i** à **0** et initialiser **j** à **0**
- Déclarer un entier de nom **octet** puis initialiser **octet** à **0** (**octet** contiendra l'octet généré)
- Pour **a** variant de **0** à **(L1-1),** faire les instructions suivantes :
  - **j = ((j+Claire[a]) modulo 256)**
  - **Permuter S[i] et S[j]**
  - **Octet = S[(S[i] + S[j]) modulo 256]**
  - **Chiffree[a]=(Clef[a]) OU EXCLUSIF (octet)**

Pour réaliser l'algorithme (PRGA), on se propose de le découper en plusieurs fonctions.

❖ **Question 5 : Décomposition binaire**

✍ Ecrire une fonction de prototype **`decomposer(bit,nombre)`** qui met dans le tableau **bit** (1er paramètre), la représentation binaire sur un octet (8 bits) de l'entier **nombre** (2ème paramètre). On suppose (0<= nombre<256).

Dans cette décomposition, **bit[0]** contiendra le bit de poids faible (voir exemple).

Exemple :

Après l'appel de la fonction **`decomposer(bit , 11),`** on aura :

bit[0]=1, bit[1]=1, bit[2]=0, bit[3]=1, bit[4]=0, bit[5]=0, bit[6]=0, bit[7]=0

❖ **Question 6 : Puissance de deux**

✍ Définir la fonction d'entête : **`deuxPuissance(n)`** qui retourne la valeur $\mathbf{2^n}$ (2 à la puissance n) (on suppose n>=0) (Bonus (1,5 pt) : En utilisant la **récursivité)**

❖ **Question 7 : valeur décimale d'un nombre en binaire**

✍ Ecrire une fonction de prototype **`decimale(bit)`** qui retourne la valeur décimale (base 10) de l'entier représenté en binaire par le tableau **bit** (le tableau **bit** est supposé être initialisé avec **8** entiers de valeurs **0** ou **1**).

Exemple :

Soit le tableau bit initialisé ainsi : bit[0]=1, bit[1]=0, bit[2]=0, bit[3]=1, bit[4]=0, bit[5]=1, bit[6]=0, bit[7]=0

Alors l'appel de la fonction **`decimale(bit)`** retourne le nombre **41**.

❖ **Question 8 : Opérateur OU EXCLUSIF (XOR)**

Soit x et y deux entiers tels que (0<= x<=255) et (0<= y<=255).

✍ Ecrire une fonction **`ouExclusif(x,y)`** qui retourne le résultat de l'opération **x ouExclusif (XOR) y** (cette opération est effectuée bit à bit sur les représentations binaires de x et y selon les règles suivantes :

0 ouExclusif 0 vaut 0	**1 ouExclusif 0 vaut 1**
0 ouExclusif 1 vaut 1	**1 ouExclusif 1 vaut 0**

Remarque : Il ne faut pas utiliser l'opérateur **^** (symbole de **XOR** en python).

Exemple : L'appel de la fonction **ouExclusif(5,6)** retourne **3**

(En binaire 5 = 00000101, 6 = 00000110 et **5 OU EXCLUSIF 6**=00000011 = 3).

❖ **Question 9 : Algorithme PRGA**

✍ Ecrire la fonction **`PRGA( )`** qui permet de générer le texte chiffré en utilisant l'algorithme **PRGA** décrit ci-dessus pour chiffrer la chaîne **Claire** en utilisant la clé de chiffrement Clef, le texte sera mis dans la chaine **Chiffree**.

B: Chiffrement d'un fichier texte par l'algorithme RC4

Il s'agit de reprendre l'algorithme RC4 pour chiffrer les lignes d'un fichier texte donné.

Pour ce faire on suppose avoir définie la fonction d'entête suivante :

`RC4Chaine(Claire, L, Clef)`

Cette fonction a pour paramètres :

- **Claire** : Une chaine de caractères contenant le texte clair qu'on désire chiffrer
- **L** : Un entier positif contenant la longueur de la chaine Claire
- **Clef** : Une chaine de caractères contenant la clé de chiffrement

La fonction **`RC4Chaine`** retourne une chaine de caractères contenant le texte chiffré par l'algorithme **RC4**, de la chaine Claire en utilisant la clé de chiffrement **`Clef`**.

❖ **Question 10 : Chiffrement d'un fichier**

En utilisant la fonction RC4Chaine décrite ci dessus,

✍ Écrire la fonction de prototype **:**

`RC4Fichier(fich, Clef, fichChiffre)` ;

Cette fonction permet de chiffrer en utilisant la chaine **Clef**(2ème paramètre) les lignes du fichier texte de nom physique **fich** (le fichier **fich** en premier paramètre est supposé existant).

Les lignes chiffrées du fichier **fich** seront mises dans un nouveau fichier texte de nom physique **fichChiffre**(3ème paramètre) sera crée dans la fonction.

FIN DE L'EPREUVE