Concours blancs

ÉPREUVE D'INFORMATIQUE

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.

⋆ ⋆ ⋆

Ave Cesar (zud bdrzq)

On cherche à crypter un texte t de longueur n composé de caractères en minuscules (soit 26 lettres différentes) représentés par des entiers compris entre 0 et 25 ($0 \leftrightarrow$ `a`, $1 \leftrightarrow$ `b`,... $25 \leftrightarrow$ `z`). Nous ne tenons pas compte des éventuels espaces.
Ainsi, le texte `ecolepolytechnique` est représenté par le tableau suivant où la première ligne représente le texte, la seconde les entiers correspondants.

Le texte	e	c	o	l	e	p	o	l	y	t	e	c	h	n	i	q	u	e
contenue de la liste `t`	4	2	14	11	4	15	14	11	24	19	4	2	7	13	8	16	20	4

Codage de César

Ce codage est le plus rudimentaire que l'on puisse imaginer. Il a été utilisé par Jules César (et même auparavant) pour certaines de ses correspondances. Le principe est de décaler les lettres de l'alphabet vers la gauche de 1 ou plusieurs positions. Par exemple, en décalant les lettres de 1 position, le caractère `a` se transforme en `z`, le `b` en `a`, ... le `z` en `y`. Le texte `avecesar` devient donc `zudbdrzq`.

Q1 Que donne le codage du texte `maitrecorbeau` en utilisant un décalage de 5 ?

Q2 Écrire la fonction `codageCesar`(t, d) qui prend en arguments un liste t et un entier d ; et qui retourne un tableau de même taille que t contenant la liste t décalé de d positions.

Q3 Écrire de même la fonction `decodageCesar`(t, d) prenant les mêmes arguments mais qui réalise le décalage dans l'autre sens.

Pour réaliser ce décodage, il faut connaître la valeur du décalage. Une manière de la déterminer automatiquement est d'essayer de deviner cette valeur. L'approche la plus couramment employée est de regarder la fréquence d'apparition de chaque lettre dans le texte crypté. En effet, la lettre la plus fréquente dans un texte suffisamment long en français est la lettre `e`.

Q4 Écrire la fonction `frequences`(t', n) qui prend en argument une liste t' de taille n représentant le texte crypté ; et qui retourne un tableau de taille 26 dont la case d'indice i contient le nombre d'apparitions du nombre i dans t' ($0 \leqslant i < 26$).

Q5 Écrire la fonction `decodageAuto`(t', n) qui prend en argument la liste t' représentant le texte crypté ; et qui retourne la liste t d'origine (en calculant la clé pour que la lettre `e` soit la plus fréquente dans le texte décrypté).

Codage de Vigenère

Au XVIème siècle, Blaise de Vigenère a modernisé le codage de César très peu résistant de la manière suivante. Au lieu de décaler toutes les lettres du texte de la même manière, on utilise un texte clé qui donne une suite de décalages.
Prenons par exemple la clé `concours`. Pour crypter un texte, on code la première lettre en utilisant le décalage qui envoie le `a` sur le `c` (la première lettre de la clé). Pour la deuxième lettre, on prend le décalage qui envoie le `a` sur le `o` (la seconde lettre de la clé) et ainsi de suite. Pour la huitième lettre, on utilise le décalage `a` vers `s`, puis, pour la neuvième, on reprend la clé à partir de sa première lettre. Sur l'exemple `ecolepolytechnique` avec la clé `concours`, on obtient : (la première ligne donne le texte, la seconde le texte crypté et la troisième la lettre de la clé utilisée pour le décalage)

texte d'origine	e	c	o	l	e	p	o	l	y	t	e	c	h	n	i	q	u	e
texte codé	g	q	b	n	s	j	f	d	a	h	r	e	v	h	z	i	w	s
Lettre utilisée pour la clé	c	o	n	c	o	u	r	s	c	o	n	c	o	u	r	s	c	o

Q6 Donner le codage du texte `becunfromage` en utilisant la clé de codage `jean`.

Q7 Écrire la fonction `codageVigenere`(t, c) qui prend comme arguments une liste t représentant le texte à crypter, et un tableau d'entiers c donnant la clé servant au codage ; et qui retourne un e liste contenant le texte crypté t'.

Maintenant, on suppose disposer d'un texte t' assez long crypté par la méthode de Vigenère, et on veut retrouver le texte t d'origine. Pour cela, on doit trouver la clé c ayant servi au codage. On procède en deux temps : 1) détermination de la longueur k de la clé c, 2) détermination des lettres composant c.
La première étape est la plus difficile. On remarque que deux lettres identiques dans t espacées de $\ell \times k$ caractères (où ℓ est un entier et k la taille de la clé) sont codées par la même lettre dans t'. Mais cette condition n'est pas suffisante pour déterminer la longueur k de la clé c puisque des répétitions peuvent apparaître dans t' sans qu'elles existent dans t. Par exemple, les lettres `t` et `n` sont toutes deux codées par la lettre `h` dans le texte crypté à partir de `ecolepolytechnique` avec `concours` comme clé. Pour éviter ce problème, on recherche les répétitions non pas d'une lettre mais de séquences de lettres dans t' puisque deux séquences de lettres répétées dans t, dont les premières lettres sont espacées par $\ell \times k$ caractères, sont aussi cryptées par deux mêmes séquences dans t'.
Dans la suite de l'énoncé, on ne considère que des séquences de taille 3 en supposant que toute répétition d'une séquence de 3 lettres dans t' provient exclusivement d'une séquence de 3 lettres répétée dans t. Ainsi, la distance séparant ces répétitions donne des multiples de k.
La valeur de k est obtenue en prenant le PGCD de tous ces multiples. Si le nombre de répétitions est suffisant, on a de bonnes chances d'obtenir la valeur de k. On suppose donc que cette assertion est vraie.

Q8 Écrire la fonction `pgcd`(a, b) qui calcule le `PGCD` des deux entiers strictement positifs a et b par soustractions successives de ses arguments.

Q9 Écrire la fonction `pgcdDesDistancesEntreRepetitions`(t', i) qui prend en argument le texte crypté t' et un entier i $(0 \leqslant i < n - 2)$ qui est l'indice d'une lettre dans t' ; et qui retourne le `pgcd` de toutes les distances entre les répétitions de la séquence de 3 lettres $\langle t[i], t[i+1], t[i+2] \rangle$ dans la suite du texte $\langle t[i+3], t[i+4], \ldots t[n-1] \rangle$. Cette fonction retourne 0 s'il n'y a pas de répétition.

Q10 Écrire la fonction `longueurDeLaCle`(t', n) qui prend en argument le texte crypté t' ; et qui retourne la longueur k de la clé de codage.

Q11 Donner le nombre d'opérations réalisées par la fonction `longueurDeLaCle` en fonction de la taille n de la liste t' ? (On ne comptera que le nombre d'appels à la fonction `PGCD`).

Q12 Une fois la longueur de la clé connue, donner une idée d'algorithme permettant de retrouver chacune des lettres de la clé. (Il s'agit de décrire assez précisément l'algorithme plutôt que d'écrire le programme).

Q13 Écrire la fonction `decodageVigenereAuto`(t') qui prend en argument le tableau t' représentant le texte crypté ; et qui retourne le texte t d'origine. (On n'hésitera pas à recopier des parties de texte dans des tableaux intermédiaires).

* *

*

Concours blancs

Correction Q1 le codage de "maitrecorbeau" avec un décalage de 5 donne le message codé : "hvdomzxjmwzvp".

Q2 On remplit le tableau représentant le texte codé avec les " lettres" du texte initial (représenté par un tableau d'entiers) décalés de d vers la gauche, ce qui se fait modulo le nombre de lettres dans l'alphabet soit 26.

```
def codageCesar(t, d):
    tt=[]
    for i in t:
        tt.append((i-d)%26
    return tt
```

Q3 On peut utiliser la fonction précédente en effectuant un décalage dans l'autre sens de la manière suivante :

```
def codageCesar(t, d):
    return codageCesar(t,26-d)
```

Q4 On commence par créer et initialiser le tableau des fréquences puis on parcourt le tableau représentant le texte. À la lecture de la i-ème lettre du texte, on incrémente la case correspondante dans le tableau des fréquences.

```
def frequences(t', n):
    res=[0]*26
    for i in t':
        res[i]=res[i]+1
    return res
```

Q5 dans decodageAuto, on commence par récupérer le tableau des fréquences des apparitions des lettres dans le texte codé. On parcourt le tableau des fréquences pour repérer l'indice de la fréquence d'apparition maximale du tableau correspondant. On calcule la clé en prenant en compte le fait que 'e' est la cinquième lettre de l'alphabet c'est-à-dire l'entier 4 et enfin on décode le texte.

```
def decodageAuto(t', n):
    freq=frequences(t',n)
    maxi=0
    for i in range(1,26):
        if freq[i]>freq[maxi]:
            maxi=i
    c=(4-maxi)%26
    return decodageCesar(t',c)
```

Q6 Le codage du texte "becunfromage" en utilisant la clé de codage "jean" est "kichwjrbvegr".

Q7 Même principe que pour le codage de César sauf que le décalage dépend de la position de la lettre du texte à coder modulo k et s'obtient à partir des lettres de la clé. Le décalage pour la i-ème lettre du texte à coder est égal à $c[i\%k]$ (avec $k =$`len(c)` : le modulo permet de rendre la liste cyclique). Le décalage se fait cette fois-ci vers la droite.

```
def codageVigenere(t, c):
   t'=[]
   for i in range(len(t)):
      t'.append((t[i]+c[i%len(c)])%26)
   return t'
```

Q8 On programme un classique calcul de pgcd en utilisant les propriétés suivantes (les nombres sont supposés positifs) :

$$pgcd(0,b) = b, \quad pgcd(a,0) = a, \text{ et } \quad pgcd(a,b) = \begin{cases} pgcd(a, b-a) \text{ si } a \leqslant b \\ pgcd(a-b, b) \text{ si } a > b \end{cases}.$$

```
def pgcd(x, y):
   a,b=x,y
   while (a!=0) and (b!=0):
      if a<=b:
         b=b-a
      else:
         a=a-b
   if a==0:
      return b   return a
```

Q9 On mémorise la répétition des trois lettres $(tt[i], tt[i+1], tt[i+2])$ du texte codé dans un variable *test*, puis on fait une boucle pour regarder si la séquence $(tt[j], tt[j+1], tt[j+2])$ coïncide avec la séquence de départ pour j variant de $i+1$ à $n-4$. Si c'est le cas, on modifie la variable `pgcddistance` qui contiendra le résultat final. On y met `pgcd`($pgcddistance$, $j-i$). On utilise la propriété $pgcd(a_1, \ldots, a_p) = pgcd(pgcd(a_1, \ldots, a_{p-1}), a_p)$ et la propriété $pgcd(0, a) = a$.

```
def pgcdDesDistancesEntreRepetitions(t', i):
   pgcddistance=0
   test=(t'[i],t'[i+1],t'[i+2])
   for j in range (i+1,n-3):
      if test==(t'[j],t'[j+1],t'[j+2]):
         pgcddistance=pgcd(pgcddistance,j-i)
   return pgcddistance
```

Q10 Dans cette fonction, on essaye de trouver la longueur de la clé. Pour cela, on calcule les distances de répétition de $\langle tt[i], tt[i+1], tt[i+2] \rangle$ pour i variant de 0 à $n-4$. Comme on a supposé que toute répétition d'une séquence de 3 lettres dans le texte codé t' provient exclusivement d'une séquence de 3 lettres répétée du texte original t, ces distances seront toutes des multiples de la longueur de la clé. On retourne donc le *pgcd* de toutes les distances non nulles pour obtenir le plus petit multiple possible de k (on espère trouver ainsi la longueur exacte de la clé avec ce procédé).

```
def longueurDeLaCle(t'):
   longueur=0
   for j in range (0,n-3):
      longueur=pgcd(longueur,pgcdDesDistancesEntreRepetitions(t',i))
   return longueur
```

Q11 Dans la fonction `longueurDeLaCle`, on a au plus $n-2$ appels à la fonction `pgcd`. Mais il ne faut pas oublier que la fonction `pgcdDesDistancesEntreRepetitions` fait aussi appel à la fonction `pgcd` et que le nombre de ces appels est majoré par $n-i-3$.

On peut donc majorer le nombre total d'appels à la fonction `pgcd` par $n-2+\sum_{i=0}^{n-2}(n-i-3) =$

$n+(n-2)(n-3)-\dfrac{(n-2)(n-1)}{2}$ ce qui nous donne une complexité quadratique (en $O(n^2)$).

Q12 Une fois que l'on a k la longueur de la clé, on peut retrouver chacune des lettres de la clé en appliquant la méthode d'analyse des fréquences aux mots construits à partir du mot codé en prenant une lettre sur k. Pour le mot codé $\langle t'_0, \ldots, t'_{n-1}\rangle$ et $0 \leqslant j < k$, on fait une analyse en fréquence sur le mot $\langle t'_j, t'_{j+k}, \ldots, t'_{j+m.k}\rangle$, où $m = \lfloor \frac{((n-1)-j)}{k} \rfloor$ de la question 5. Même si le codage de César est différent de celui de Vigenere. La fonction fera son office.

Q13 On commence par récupérer la longueur k de la clé que l'on obtient grâce à la fonction `longueurDeLaCle`. À partir d'une liste d'éléments vides aussi grande que t'. On va décoder automatiquement la sous-liste extraite et introduire le résultat dans notre liste au bon endroit.

```
def decodageVigenereAuto(t'):
    k=longueurDeLaCle(t')
    res=[]*len(t')
    for j in range (k):
        listedecodee=decodageAuto(t'[j::k])
        for l in range (len(listedecodee)):
            res[j+l*k]=listedecodee[l]
    return res
```