Informatique Tronc Commun
Devoir Surveillé 1

1 Chiffrement de César

Le chiffrement de César est une méthode de cryptographie élémentaire : pour coder un message, on décale chaque lettre d'un même nombre de rangs dans l'alphabet, nombre qu'on appelle la **clé**. Le message codé ainsi obtenu n'est alors plus compréhensible par quelqu'un ne disposant pas de la clé, et peut être décodé par quelqu'un en disposant.
Afin de pouvoir décaler toutes les lettres, on considère que a est le successeur de z. Par exemple, en codant le message "exemple" avec la clé 4, on obtient "ibiqtpi". Dans la suite, on supposera que les chaînes de caractères considérées ne contiennent que les lettres minuscules de a à z ou des espaces.

1. On suppose que la liste `alphabet = ['a','b','c','d', ... , 'y','z']` est définie comme variable globale, ce qui signifie qu'on peut l'utiliser dans n'importe quelle fonction sans la prendre en argument. Cette liste permet d'associer chaque lettre à son indice, qui est un entier de 0 à 25.
   (a) Écrire une fonction `int_vers_char` prenant en argument un entier i supposé entre 0 et 25, et renvoyant la lettre associée.
   Ainsi, `int_vers_char(3)` doit renvoyer 'd'.
   (b) Écrire une fonction `char_vers_int` prenant en argument un caractère c, et renvoyant l'entier associé. La fonction ne renverra rien si `c` est une espace.
   Ainsi, `char_vers_int('y')` doit renvoyer 24
   (c) Donner une nouvelle version de la fonction précédente fonctionnant par recherche dichotomique (on peut en Python comparer des caractères selon l'ordre alphabétique de la même façon qu'on compare des nombres).
   (d) Combien de caractères du tableau `alphabet` sont comparés à `c` au maximum lors d'un appel de la version dichotomique de `char_vers_int(c)` ?
2. Écrire une fonction `code_char(c,cle)` prenant en argument un caractère c et renvoyant la lettre obtenu en décalant `c` d'un nombre de rangs égal à `clé`, ou renvoyant une espace si c est une espace.
   Ainsi, `code_char('y', 5)` doit renvoyer 'd', et `code_char(' ', 4)` doit renvoyer `' '`.
3. Écrire une fonction `code` prenant en argument une chaîne de caractère `m` et un entier `cle`, et renvoyant le message obtenu par chiffrement de César en partant de m et en utilisant la clé `cle`.
   Ainsi, `code("top secret", 10)` doit renvoyer "dyz combod"
4. Si on connaît la clé, il suffit d'utiliser la fonction précédente avec la valeur opposée pour décoder. Pour décoder sans la clé un message suffisamment long, on peut essayer une attaque statistique. Si on suppose que le message est initialement écrit en français, la lettre la plus fréquente a de fortes chances de correspondre à "e" après décodage, ce qui permet de déterminer la clé.
   (a) Écrire une fonction `indice_max` prenant en argument une chaîne `m` et renvoyant l'entier associé à une lettre de nombre d'occurrences maximal dans `m`.

   Ainsi, `indice_max("huuhxu")` doit renvoyer 20 ;
   `indice_max("dyz combod")` peut renvoyer 3 ou 14.
   (b) En déduire une fonction `decode` prenant en argument une chaîne de caractère correspondant à un message codé, et tentant de le décoder par l'attaque statistique présentée.

2 Palindromes

1. On dit qu'une chaîne de caractères est un palindrome si elle se lit identiquement dans un sens ou dans l'autre, à l'exemple du mot *ressasser*.
   (a) Écrire une fonction `est_un_palindrome` prenant en argument une chaîne de caractères et déterminant (en renvoyant `True` ou `False`) si c'est un palindrome.
   (b) Déterminer le nombre de comparaisons entre caractères effectuées par cette fonction.
2. La longueur palindromique d'une chaîne de caractère est la longueur maximale d'une sous-chaîne palindrome de cette chaîne. Par exemple, `"truculent"` a pour longueur palindromique 3, atteinte pour la sous-chaîne palindrome `"ucu"`.
   Écrire une fonction prenant en argument une chaîne de caractères `c` et renvoyant sa longueur palindromique. Le nombre de comparaisons entre caractères effectuées par cette fonction devra être inférieur à $\texttt{len(c)}^3$, ce qu'on justifiera.

3 Simulation d'un jeu de pari (d'après un oral de Centrale)

On considère le jeu suivant, opposant $n \geq 3$ joueurs $J_0 \dots J_{n-1}$ pour se partager la somme de S euros : une pièce équilibrée va être lancée $N \in \mathbb{N}^*$ fois, et avant les lancers chaque joueur va établir une liste de prédictions sur le résultat de chaque lancer. Les gagnants sont les joueurs ayant le plus de prédictions correctes parmi l'ensemble des joueurs, et ces gagnants se partagent équitablement les S euros.
On notera dans la suite 0 pour pile et 1 pour face. Par exemple, si $n = 3$ et $N = 4$, le joueur J_0 pourrait prédire 1100, J_1 0101 et J_2 0001. Si le résultat des lancers est 0110, alors J_0 et J_1 ont 2 prédictions correctes, tandis que J_2 n'en a qu'une. J_0 et J_1 gagnent donc chacun $S/2$, et J_2 ne gagne rien.
On cherche dans cet exercice à simuler informatiquement ce jeu, pour étudier l'effet de stratégies que les joueurs peuvent mettre en oeuvre.

1. Importer la bibliothèque `random` sous le nom `rd`. On utilisera dans la suite la fonction `randint` de cette bibliothèque prenant en argument deux entiers a et b et renvoyant un nombre tiré uniformément au hasard entre a et b inclus.
2. Écrire une fonction `partie` prenant en argument `S`, `N` et une liste `P` telle que `P[i]` est la liste des prédictions du joueur J_i, simulant la partie et renvoyant la liste des gains de chaque joueur.
   Ainsi, en reprenant l'exemple précédent, `partie1(10, 4, [[1,1,0,0], [0,1,0,1], [0,0,0,1]])` doit renvoyer `[5,5,0]` si le résultat des lancers pendant la partie est 0110.
3. On suppose dans un premier temps que chaque joueur établit ses prédictions aléatoirement et indépendamment des autres joueurs.
   Écrire une fonction `strategies1` prenant en argument `n` et `N`, et renvoyant la liste `P` des listes de prédictions de chaque joueur.
4. On peut montrer par un simple argument de symétrie que l'espérance de gain de chaque joueur avec ces stratégies est S/n. On souhaite vérifier ce résultat expérimentalement.
   On fixe dans cette question $n = 4, N = 10, S = 100$.
   Écrire un programme répétant 100000 une partie précédée du tirage des stratégies de chaque joueur, et affichant le gain moyen de chaque joueur à l'issue de toutes ces parties.
5. On suppose à présent que J_0 et J_1 coopèrent de la façon suivante : J_0 continue de tirer ses prédictions au hasard, mais J_1 choisit alors les prédictions inverses. Ainsi, si J_0 prédit 1001, J_1 prédit 0110. Les autres joueurs continuent à établir leurs prédictions aléatoirement et indépendamment.
   Écrire une fonction `strategies2` modifiant `strategies1` pour prendre en compte ces modifications.
6. On note $E(G_i)$ l'espérance de gain du joueur i. Avec cette nouvelle stratégie, on peut montrer mathématiquement que, lorsque N est impair, on a :

$$E(G_0) = E(G_1) = \frac{S}{n-1}\left(1 - \frac{1}{2^{n-1}}\right)$$

   et lorsque N est pair, avec $N = 2p$, on a :

$$E(G_0) = E(G_1) = \frac{S}{n-1}\left(1 - \frac{\tau_p^n}{nq_p} + \frac{\tau_{p-1}^n}{nq_p}\right)$$

   où $\forall i \in [\![0, N]\!],\ q_i = \binom{N}{i}\frac{1}{2^N}$ et $\tau_i = \sum_{k=0}^{i} q_k$

   Écrire une fonction `esperance` prenant en argument `n`, `N` et `S` et renvoyant l'espérance mathématique de gain de J_0.
   On pourra utiliser la fonction `comb` du module `math` après l'avoir importée, `comb(p,k)` calculant $\binom{p}{k}$.
7. On fixe $S = 100$ et $n = 4$. Utiliser, après les avoir importées, les fonctions `plot` et `show` de la bibliothèque `matplotlib.pyplot` pour tracer, en fonction de N variant entre 1 et 10, l'espérance de gain théorique de J_0 puis le gain moyen de J_0 sur 10000 parties observées expérimentalement.