Informatique

TP - Révisions

PSI - Lycée Rabelais

★

Exercices "simples"

★

1 Liste pyramidale

Écrire une fonction `liste_pyramide(n)` qui construit une liste avec des entiers "en pyramide" allant de 1 à n en ordre croissant puis redescend à 0. Par exemple `liste_pyramide(5)` devra renvoyer `[1,2,3,4,5,4,3,2,1]`.

2 Le roi Dagobert

Écrire une fonction `Dagobert(mot)` qui écrit le mot `mot` à l'envers. Par exemple, `Dagobert('Roi Dagobert')` devra renvoyer la chaîne `'trebogaD ioR'`.

En déduire une fonction `palindrome(mot)` qui renvoie un booléen indiquant si la chaîne de caractère `mot` est un palindrome (`palindrome('kayak')` devra par exemple renvoyer `True`).

3 Min, Max et Moy

Écrire une fonction `minMaxMoy(L)` qui reçoit une liste`L` d'entiers et qui renvoie le minimum, le maximum et la moyenne de cette liste.

4 Calcul des racines

Écrire une fonction `Racine(a,b,c)` qui renvoie, si elles existent dans $\mathbb{R}$, les racines du polynôme $P(x) = a.x^2 + b.x + c$. La fonction renverra également `False`, s'il n'y a pas de racine dans $\mathbb{R}$.

★

Problème à faire absolument !

★

5 Codage de Hamming

L'utilisation de cartes magnétiques avec transmission de données par liaison RFID s'est généralisée ces dernières années. Lors de la transmission, des erreurs peuvent apparaitre dans les données. En pratique, il est donc indispensable de pouvoir détecter ces erreurs et, dans la mesure du possible, les corriger sans que cela ne nécessite une nouvelle transmission ; l'objet de ce problème est de mettre en place quelques algorithmes dans ce but.

On suppose dans la suite que l'on cherche à transmettre un nombre entier, noté `N`, codé en binaire avec 4 bits. On notera `L` la liste associée à la représentation binaire de `N` (on placera les bits de poids fort à gauche de la liste).

Question 1. Dans quel intervalle est compris `N` ? Écrire une fonction `dec2bin(N)` permettant la conversion décimale vers binaire (qui renvoie donc `L` et prend en argument `N`).

Question 2. Écrire une fonction `bin2dec(L)` permettant la conversion binaire vers décimale (qui renvoie donc `N` et prend en argument `L`).

5.1 Bit de parité

Une technique simple et très répandue pour s'assurer qu'une donnée binaire sera lue correctement par son récepteur est de lui adjoindre un bit de parité, égal, par définition à :

- 0 si la donnée contient un nombre pair de 1 (et donc si ses bits sont de somme paire) ;
- 1 si la donnée contient un nombre impair de 1 (et donc si ses bits sont de somme impaire).

Après la réception de la donnée, le récepteur recalcule le bit de parité et le compare à celui que l'émetteur lui a adressé. Si la donnée n'a pas été altérée lors de la transmission, alors les deux bits de parité sont identiques.

Question 3. Donner les bits de parité associés aux représentations binaires des entiers 5, 15 et 4.

Question 4. Écrire une fonction `parite(L)` prenant pour argument la liste `L` constituée d'entiers valant 0 ou 1 et retournant l'entier 0 ou 1 correspondant à son bit de parité.

Les techniques de vérification les plus simples consistent à découper la donnée en blocs et à joindre un bit de parité à

chaque bloc. Le premier protocole consiste à joindre à quatre bits un bit de parité et à transmettre l'ensemble des 5 bits.

On donne ci-dessous une fonction `trans` pour tester les transmissions avec présence éventuelle d'erreur :

```
import random as rd

### rd.randint(a, b) : renvoie un entier aléatoire entre a et b (inclus)

def trans(L):
    li = rd.randint(0, 1)
    i = rd.randint(0,len(L)-1)

    Ltrans = L[:]
    Ltrans[i] = li
    return Ltrans
```

Question 5. Commenter la fonction `trans`.

Question 6. Écrire une fonction `test1(L1)` où `L1` est la liste des 5 bits reçus (avec ou sans erreur) et qui renvoie l'entier `N` codé pour la transmission si la donnée a bien été transmise (en s'appuyant sur le bit de parité) et `False` sinon.

Question 7. Écrire les instructions permettant de simuler la transmission d'un entier codé sur 4 bits avec cette méthode en respectant notamment les étapes suivantes :

1- Définition de l'entier à transmettre.
2- Création de la liste `L1`.
3- Transmission de la donnée.
4- Vérification de la donnée reçu.
5- Affichage de l'entier reçu s'il n'y a pas eu d'erreur de transmission ou demande d'une nouvelle transmission (tant que le message n'a pas été transmis correctement).

Question 8. Que se passe-t-il s'il y a deux erreurs ?

5.2 Code de Hamming

D'autres protocoles, plus intéressants, permettent de détecter et de **corriger l'erreur !** Cela évite donc de transmettre à nouveau la donnée. Le code de Hamming est un exemple de code correcteur. Nous nous intéressons ici au code dit `(7 ; 4)`, ainsi appelé car il consiste à joindre trois bits de parité à quatre bits de données, ce qui donne un message d'un longueur totale de sept bits. Ces trois bits de parité sont définis ainsi :

si la donnée s'écrit `[d1; d2; d3; d4]` avec `di = 0` ou `di = 1`, alors :

- `p1` est le bit de parité de `[d1 ; d2 ; d4]` ;
- `p2` est le bit de parité de `[d1 ; d3 ; d4]` ;
- `p3` est le bit de parité de `[d2 ; d3 ; d4]`.

Le message encodé, que l'on transmet, s'écrit alors comme suit : `[p1 ; p2 ; d1 ; p3 ; d2 ; d3 ; d4]`.

Question 9. Écrire une fonction `encodeHamming(N)` prenant pour argument un entier `N` codé avec quatre bits représentés par les entiers 0 ou 1 et retournant la liste des bits contenant le message complet encodé selon le protocole de Hamming.

Le contrôle après réception d'un message ainsi encodé est relativement simple. On pourrait naturellement recalculer

les trois bits de parité de la donnée et les comparer aux valeurs transmises, mais la technique proposée par Hamming est de calculer les trois bits de contrôle suivants, notés `[c1 ; c2 ; c3]` à partir du message complet (données et bits supplémentaires), noté `[m1;m2;m3;m4;m5;m6;m7]` :

- `c1` est le bit de parité de `[m4;m5;m6;m7]` ;
- `c2` est le bit de parité de `[m2;m3;m6;m7]` ;
- `c3` est le bit de parité de `[m1;m3;m5;m7]`.

On montre que si le message a bien été encodé selon les règles précédentes et n'a pas été altéré, alors les trois bits de contrôle doivent être à 0. Si ce n'est pas le cas, alors il y a une erreur. L'intérêt de la technique de Hamming est que dans le cas particulier où l'erreur est unique, le mot de contrôle donne la représentation binaire de la position de cette erreur en numérotant à partir de 1. Par exemple, si `[c1;c2;c3] = [0;1;1]`, alors l'erreur porte sur le troisième bit du message. Il suffit ainsi d'inverser ce bit (le mettre à 0 s'il est à 1 et inversement) pour corriger l'erreur.

La donnée décodée est alors constituée des quatre bits `[d1;d2;d3;d4]` qui se trouvent respectivement en position 3, 5, 6 et 7 (toujours en numérotant à partir de 1) conformément à la description de l'encodage donnée ci-dessus.

Question 10. On cherche, dans cette question, à vérifier **à la main** les affirmations précédentes à partir du message `[d1;d2;d3;d4] = [1;0;1;1]` qui permet de coder l'entier 11.

a - montrer que le message à transmettre est `[0,1,1,0,0,1,1]`.

b - s'il n'y a **pas d'erreur de transmission**, cela signifie que le message reçu est le même que le message envoyé à savoir `[0,1,1,0,0,1,1]`. Vérifier alors que le mot de contrôle `[c1;c2;c3]` est bien égal à `[0;0;0]` et que l'on peut retrouver l'entier 11 codé avec la méthode de Hamming.

c - s'il y a **une erreur de transmission**, cela signifie que le message reçu est différent du message envoyé. On peut imaginer recevoir, par exemple, la liste `[0,1,1,0,1,1,1]`. Bien entendu, le receveur du message n'est pas censé savoir quel est le message avant la transmission ! Quel serait l'entier décodé si l'on ne vérifie pas si une erreur a été commise ? Vérifier alors que le mot de contrôle `[c1;c2;c3]` est égal à `[1;0;1]` et que $(101)_2 = (5)_{10}$ indique bien l'emplacement de l'erreur. Indiquer alors comment corriger l'erreur commise durant la transmission et retrouver l'entier codé.

Question 11. Écrire une fonction `testHamming(LH)` où `LH` est la liste des 7 bits reçus (avec ou sans erreur) et qui renvoie l'entier `N` codé pour la transmission. En cas d'erreur, on affichera en plus à l'écran un avertissement indiquant la position du bit affecté (et bien entendu, on effectuera la correction nécessaire). On supposera dans cette question que s'il y a une erreur, alors elle est unique.

Question 12. Écrire les instructions permettant de simuler la transmission d'un entier codé sur 4 bits avec cette méthode.

Question 13. Réécrire une nouvelle fonction `trans2` pour que celle-ci génère deux erreurs **à chaque transmission.** Tester avec les instructions de la question précédente. Quel a été l'effet de la "correction" sur la donnée dans ce cas ?

Question 14. Sans coder, proposer un moyen simple de différencier une double erreur d'une erreur unique au moyen d'un bit de parité supplémentaire et expliquer comment cela permet d'éviter le problème mis en évidence à la question précédente. On s'appuiera sur les techniques introduites dans cette partie. On ne demande pas d'essayer de corriger la double erreur.

★

Exercices "difficiles"

★

6 Fourmi ivre

Une fourmi ivre suit les lignes d'un quadrillage dont les intersections sont représentées par les couples de nombres entiers. Lorsqu'elle arrive à une intersection, alle a autant de chance de continuer tout droit, de tourner à gauche, à droite ou de revenir sur ses pas. On note C_a, le carré formé des couples $[x, y]$ avec $-a \leq x \leq a$ et $-a \leq y \leq a$.

Q1. Écrire une fonction `avancer` sans argument qui renvoie l'un des quatre couples `[1,0]`, `[0,-1]`, `[0,1]` ou `[-1,0]` avec équiprobabilité (on pourra utiliser la fonction `randint` du module `random`).

Q2. Simuler une trajectoire de la fourmi en partant de $[0,0]$ puis en faisant afficher les nœuds du quadrillage par où elle passe, jusqu'à ce qu'elle sorte du carré C_2.

Q3. Écrire une fonction `traj` d'argument `a` qui renvoie une trajectoire tirée au hasard, partant du centre $[0,0]$ et s'arrêtant lorsque la fourmi tente de sortir du carré C_a.

Q4. Tracer sur un même graphique 6 trajectoires différentes pour $a = 10$.

Q5. Soit L_a la longueur du trajet effectué par une fourmi pour sortir du carré C_a en partant de son centre. Définir une fonction `LM` de deux arguments `a` et `N` qui renvoie la moyenne des longueurs de trajet de N fourmis (N réalisations de la variable aléatoire L_a).

Q6. Tracer pour a entier compris entre 1 et 20, une estimation de $E(L_a)$, l'espérance de L_a.

7 Transformation

Q1. On considère un nombre n. Que donne la commande `list(str(n))` ?

Q2. Comment récupérer le chiffre des unités d'un nombre entier donné ? Celui des dizaines ? Tester cette méthode sur le nombre 2015.

Q3. Écrire une fonction `transforme` d'argument un entier naturel n et qui renvoie le nombre obtenu par la méthode suivante :

- les chiffres de rang impair (en partant des unités) sont inchangés.
- les chiffres de rang pair (en partant des unités) sont doublés, si ce double est supérieur ou égal à 10, on le remplace par la somme de ses chiffres (par exemple, 3 est remplacé par 6 et 7 est remplacé par 5).

Exemple : 43281 est transformé en 46271.

Q4. Déterminer les nombres inférieurs à 10000 invariants par la fonction `transforme`. Expliquer le résultat obtenu.

8 Type J

Dans la liste des entiers naturels non nuls, on barre un nombre sur 2 en commençant par barrer le deuxième :

$$1, \cancel{2}, 3, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, 9, \ldots$$

Puis dans la liste restante, on barre un nombre sur 3 en commençant par barrer le troisième :

$$1, 3, \cancel{5}, 7, 9, \cancel{11} \ldots$$

Et ainsi de suite... A l'infini, on obtient "*la liste des nombres de type J*".

Q0. Créer une liste comprenant tous les entiers allant de 1 à 100 (inclus).

Pour la suite, on s'interdit l'utilisation de l'instruction `pop`.

Q1. Écrire une fonction `enlever` de deux arguments, une liste `L` et un entier naturel `i`, qui renvoie une liste `S` construite en ne gardant dans la liste `L` que les éléments dont le rang n'est pas multiple de `i`. Par exemple :

`enlever([1,3,5,7,9,11,13],3)` doit donner `[1,3,7,9,13]`.

Q2. Écrire une suite d'instructions donnant *la liste des nombres de type J* inférieurs ou égaux à 100.

Q3. Écrire une fonction `LJ` d'argument `n` renvoyant *la liste des nombres de type J* inférieurs ou égaux à `n`.

Q4. Écrire une fonction `U` d'argument `n` renvoyant u_n, le nombre de *nombres de type J* inférieurs ou égaux à `n`.

Q5. Vers quelle limite l_i semble tendre $4.n/u_n^2$ quand n tend vers l'infini ?

Q6. Déterminer le premier n pour lequel la différence en valeur absolue entre $4.n/u_n^2$ et l_i est inférieure à 10^{-3}.

9 Jeu de carte

On considère un jeu de 32 cartes. Il est formé de couples de 8 valeurs ordonnées, `valeurs=["7","8", "9", "10", "V", "D", "R", "A"]`, et de 4 "couleurs", `couleurs=["T", "K", "C", "P"]` (Trèfle, Carreau, Cœur, Pique). On distribue au hasard une "main", c'est-à-dire 5 cartes distinctes, et on s'intéresse à des mains particulières :

- les "une paire" (2 cartes de même valeur et 3 cartes de valeurs différentes) ;
- les "deux paires" (2 cartes d'une même valeur, 2 cartes d'une même autre valeur, et une carte d'une troisième valeur) ;
- les "fulls" (3 cartes d'une même valeur et 2 cartes d'une même autre valeur) ;
- les "carrés" (les 4 cartes d'une même valeur, et une autre).

Q1. Construire la liste `cartes` des 32 cartes à partir des deux listes `valeurs` et `couleurs`, chaque carte étant représentée par la paire `[valeur,couleur]`. Vérifier que le nombre de cartes obtenu est correct.

Q2. Écrire une fonction `tirerMain` sans argument qui renvoie une liste de 5 cartes distinctes tirées au hasard. On pourra utiliser la fonction `sample` du module `random` dont la documentation est fournie ci-dessous.

> ***Syntax :*** *random.sample(sequence, k)*
>
> ***Parameters:***
> ***sequence****: Can be a list, tuple, string, or set.*
> ***k****: An Integer value, it specify the length of a sample.*
>
> ***Returns:*** *k length new list of elements chosen from the sequence.*

Q3. Écrire une fonction `resultat` en vous aidant de la fonction tri appelée `sorted` et déjà implémentée sur `Python`. La fonction `resultat` doit renvoyer :

- 0 si la meilleure combinaison est un "une paire" ; ;
- 1 si la meilleure combinaison est un "deux paires" ;
- 2 si la meilleure combinaison est un "full" ;
- 3 si la meilleure combinaison est un "carré" ;
- 4 sinon.

Q4. À partir de 50000 tirages aléatoires de mains, estimer les probabilités d'obtenir une "une paire", "deux paires", un full et un carré.

Q5. Comparer ces estimations avec les probabilités obtenues par dénombrement (ou par recherche sur internet).

10 Échiquier

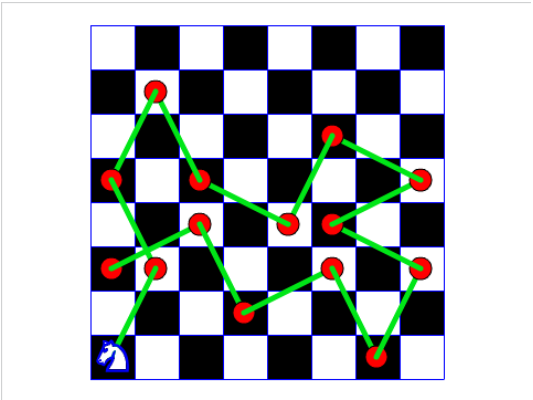
Un échiquier est un plateau de 8 lignes et 8 colonnes. Ces lignes et ces colonnes seront,dans cet exercice, numérotées de 0 à 7. Une position de l'échiquier est un couple $[i, j]$ d'entiers compris entre 0 et 7 inclus, avec i le numéro de ligne et j le numéro de colonne. Un cavalier placé sur l'échiquier se déplace en bougeant de 2 cases dans une direction (verticale ou horizontale) et de une case perpendiculaire. Si le cavalier est loin des bords de l'échiquier, il y a 8 possibilités de déplacements, mais il en a moins s'il est près des bords.

Q1. Illustrer sur un brouillon les deux cas énoncées précédemment.

Q2. Écrire une fonction `valide` prenant en argument deux entiers relatifs i et j et vérifiant que le couple $[i, j]$ est bien une position de l'échiquier. `valide` doit renvoyer un booléen.

Q3. Écrire une fonction `coupsSuivants` prenant en argument une position $[i, j]$ et renvoyant la liste des positions que peut atteindre un cavalier placé en $[i, j]$ en un seul coup.

On veut écrire un script permettant de résoudre le problème du cavalier. Ce problème algorithmique consiste à déplacer un cavalier sur le plateau pour parcourir toutes les cases sans passer deux fois sur la même.



On notera `T`, une liste de liste de dimension 8×8 définie telle que :

```
T = [[False,False,False,False,False,False,False,False], \
     [False,False,False,False,False,False,False,False], \
     [False,False,False,False,False,False,False,False], \
     [False,False,False,False,False,False,False,False], \
     [False,False,False,False,False,False,False,False], \
     [False,False,False,False,False,False,False,False], \
     [False,False,False,False,False,False,False,False], \
```

```
8         [False,False,False,False,False,False,False,False]]
```

On notera `False` lorsque la case n'a pas été parcourue et on remplira le tableau au fur et à mesure en remplaçant `False` par le numéro du n-ième déplacement.

Pour le déplacement du cavalier, on choisira aléatoirement celui-ci parmi les différents coups possibles (on pourra éventuellement modifier la fonction `valide`). On utilisera également la fonction `randint` du module `random` pour générer un entier aléatoire.

Q4. Écrire une liste d'instructions afin de remplir le tableau `T`. On remplira le tableau tant qu'un déplacement est possible.

Q5. Combien de déplacements sont nécessaires pour remplir entièrement l'échiquier ? Votre code permet-il de trouver une solution à ce problème ?