TP

Titre EXO

Source EXO

Savoirs et compétences :

- ....

ALGO

ALG-000

Votre robot dispose de nombreux récepteurs et enregistre tous les signaux qui l'entourent. Cependant vous avez remarqué que certains de ces signaux sont très bruités. Vous décidez donc d'écrire un programme qui atténue le bruit de ces signaux, en effectuant ce que l'on appelle un lissage.

Une opération de lissage d'une séquence de mesures (des nombres décimaux) consiste à remplacer chaque mesure sauf la première et la dernière, par la moyenne des deux valeurs qui l'entourent.

Par exemple, si l'on part de la séquence de mesures suivantes : 1 3 4 5

On obtient après un lissage : 1 2.5 4 5

Le premier et dernier nombre sont inchangés. Le deuxième nombre est remplacé par la moyenne du 1er et du 3e, soit $(1+4)/2 = 2.5$, et le troisième est remplacé par $(3+5)/2 = 4$.

On peut ensuite repartir de cette nouvelle séquence, et refaire un nouveau lissage, puis un autre sur le résultat, etc. Votre programme doit calculer le nombre minimum de lissages successifs nécessaires pour s'assurer que la valeur absolue de la différence entre deux valeurs successives de la séquence finale obtenue ne dépasse jamais une valeur donnée, `diffMax`.

On vous garantit qu'il est toujours possible d'obtenir la propriété voulue en moins de 5000 lissages successifs.

On cherche à définir la fonction suivante `def lissage(t:list[float], diffmax:float) -> int :` où

- l'entrée est une liste `t` contenant les mesures, qui sont des flottants, et un flottant `diffmax`;
- la sortie est un entier qui correspond au nombre minimal de lissages nécessaire.

▪ **Exemple** `lissage([1.292, 1.343, 3.322, 4.789, -0.782, 7.313, 4.212], 1.120) -> 13.` ▪

ALG-001

Il existe de nombreuses traditions étranges et amusantes sur Algoréa, la grande course de grenouilles annuelle en fait partie. Il faut savoir que les grenouilles algoréennes sont beaucoup plus intelligentes que les grenouilles terrestres et peuvent très bien être dressées pour participer à des courses. Chaque candidat a ainsi entraîné sa grenouille durement toute l'année pour ce grand événement.
La course se déroule en tours et, à chaque tour, une question est posée aux dresseurs. Le premier qui trouve la réponse gagne le droit d'ordonner à sa grenouille de faire un bond. Dans les règles de la course de grenouilles algoréennes, il est stipulé que c'est la grenouille qui restera le plus longtemps en tête qui remportera la victoire. Comme cette propriété est un peu difficile à vérifier, le jury demande votre aide.
Ce que doit faire votre programme :

`nbg` numérotées de 1 à `nbg` sont placées sur une ligne de départ. À chaque tour, on vous indique le numéro de la seule grenouille qui va sauter lors de ce tour, et la distance qu'elle va parcourir en direction de la ligne d'arrivée.
Écrivez un programme qui détermine laquelle des grenouilles a été strictement en tête de la course à la fin du plus grand nombre de tours.

ENTRÉE : deux entiers `nbg` et `nbt` et un tableau `t`.
`nbg` est le nombre de grenouilles participantes.
`nbt` est le nombre de tours de la course.
`t` est un tableau ayant `nbt` éléments, et tel que chaque élément est un couple : (numéro de la grenouille qui saute lors de ce tour, longueur de son saut).
SORTIE : vous devez renvoyer un entier : le numéro de la grenouille qui a été strictement en tête à la fin du plus grand nombre de tours. En cas d'égalité entre plusieurs grenouilles, choisissez celle dont le numéro est le plus petit.

EXEMPLE :
entrée : (4, 6, [[2,2], [1,2], [3,3], [4,1], [2,2], [3,1]])
sortie : 2.

ALG-002

Un marchand de légumes très maniaque souhaite ranger ses petits pois en les regroupant en boîtes de telle sorte que chaque boîte contienne un nombre factoriel de petits pois. On rappelle qu'un nombre est factoriel s'il est de la forme 1, 1×2, $1\times 2\times 3$, $1\times 2\times 3\times 4\cdots$ et qu'on les note sous la forme suivante :

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Il souhaite également utiliser le plus petit nombre de boîtes possible.
Ainsi, s'il a 17 petits pois, il utilisera :
2 boîtes de $3! = 6$ petits pois
2 boîtes de $2! = 2$ petits pois
1 boîte de $1! = 1$ petits pois.
ce qui donne bien $2 \times 3! + 2 \times 2! + 1 \times 1! = 12 + 4 + 1 = 17$.

D'une manière générale, s'il a `nbp` petits pois, il doit trouver une suite `a_1, a_2⋯, a_p` d'entiers positifs ou nuls avec `a_p`> 0 et telle que : `nbp=a_1 x 1!+a_2 x 2!+⋯+a_p x p!` avec `a_1+⋯+a_p` minimal.
Remarque mathématique : si à chaque étape on cherche le plus grand entier `k` possible tel que `k!` soit inférieur au nombre de petits pois restant, on est sûrs d'obtenir la décomposition optimale : en termes informatiques, on dit que l'algorithme *glouton* est optimal.

ENTRÉE : un entier, `nbp`, le nombre total de petits poids.
SORTIE : un couple constitué de l'entier `p` et du tableau `[ a_1, a_2, ⋯, a_p ]`.

EXEMPLE :
entrée : `17`
sortie : `(3, [ 1, 2, 2 ] )`.

ALG-003

Rien de tel que de faire du camping pour profiter de la nature. Cependant sur Algoréa, les moustiques sont particulièrement pénibles et il faut faire attention à l'endroit où l'on s'installe, si l'on ne veut pas être sans cesse piqué.

Vous disposez d'une carte sur laquelle est indiquée, pour chaque parcelle de terrain, si le nombre de moustiques est supportable ou non. Votre objectif est de trouver le plus grand camping carré évitant les zones à moustiques qu'il est possible de construire.

ENTRÉE : un tableau `t` de n éléments correspondant à des lignes, chacune de ces lignes contenant p élements, qui ne sont que des 0 et des 1. 0 signifie qu'il n'y a pas de moustiques et 1 qu'il y a des moustiques.
SORTIE : un entier : la taille maximale du côté d'un carré ne comportant que des 0 et dont les bords sont parallèles aux axes.

EXEMPLE 1 :

entrée : `t`=
```
[ [1, 0, 0, 1, 0, 0, 1],
  [0, 0, 0, 0, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0],
  [0, 1, 0, 0, 0, 0, 1],
  [1, 0, 0, 0, 1, 0, 1] ]
```
sortie : 4.

EXEMPLE 2 :

entrée : `t`=
```
[ [0, 0, 0, 1, 1, 1, 1],
  [0, 0, 0, 1, 1, 1, 1],
  [0, 0, 0, 1, 1, 1, 1],
  [1, 1, 1, 1, 1, 0, 0],
  [1, 1, 1, 1, 1, 0, 0] ]
```
sortie : 3.

ALG-004

On ne s'intéresse pas ici à la validité d'un nombre écrit en chiffre romains, mais à sa valeur. On rappelle quelques principes de base. Les sept caractères de la numération romaine sont :

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

Certaines lettres sont dites d'*unité*. Ainsi on dit que I est une unité pour V et X, X est une unité pour L et C, C est une unité pour D et M.

Pour trouver la valeur d'un nombre écrit en chiffres romains, on s'appuie sur les règles suivantes :
- toute lettre placée à la droite d'une lettre dont valeur est supérieure ou égale à la sienne s'ajoute à celle-ci ;
- toute lettre d'unité placée immédiatement à la gauche d'une lettre plus forte qu'elle indique que le nombre qui lui correspond doit être retranché au nombre qui suit ;
- les valeurs sont groupées en ordre décroissant, sauf pour les valeurs à retrancher selon la règle précédente ; 1
- la même lettre ne peut pas être employée quatre fois consécutivement sauf M.

Par exemple, DXXXVI = 536, CIX = 109 et MCMXL = 1940.

1. Écrire une fonction `valeur (caractere)` qui retourne la valeur décimale d'un caractère romain. Cette fonction doit renvoyer 0 si le caractère n'est pas l'un des 7 chiffres romains.
2. Écrire la fonction principale `conversion (romain)` qui permet de convertir un nombre romain en nombre décimal. Cette fonction doit prendre en argument une chaîne de caractères `romain`. Si cette chaîne est écrite en majuscule et correspond à un nombre romain correctement écrit, la fonction doit renvoyer le nombre décimal égal au nombre romain passé en argument. Sinon, la fonction doit renvoyer -1.

ALG-004-cor

1. Nous pouvons proposer la fonction programme suivante :

```
def valeur (caractere) :
    """ renvoie la valeur décimale d'un caractè↙
        re
    s'il est utilisé dans la numération romaine
    renvoie 0 sinon.
    Précondition : caractere est une lettre de l'↙
        alphabet
    latin """
```

```
    if caractere == "I" :
        return 1
    elif caractere == "V" :
        return 5
    elif caractere == "X" :
        return 10
    elif caractere == "L" :
        return 50
    elif caractere == "C" :
        return 100
    elif caractere == "D" :
        return 500
    elif caractere == "M" :
        return 1000
    else :
        return 0
```

Notons que cette fonction serait plus courte en utilisant la notion de *dictionnaire* python, qui ne figure pas au programme.

2. Nous pouvons proposer la fonction programme suivante :

```
def conversion (romain) :
    L = ["I","V","X","L","C","D","M"]
    decimal = 0
    n = len(romain)
    if n==0 :
        return -1
    for k in range(n) :
        if romain[k] in L :
            v = valeur(romain[k])
            if k+1 < n and v < valeur(romain[k+1]) :
                decimal -= v
            else :
                decimal += v
        else :
            return -1
    return decimal
```

ALG-005

Pour tout $n \in \mathbb{N}\setminus\{0,1\}$, on pose $S_n = \sum_{k=2}^{n} \frac{(-1)^k}{k \ln k}$. On admet que la suite $(S_n)_{n\in\mathbb{N}\setminus\{0,1\}}$ a une limite finie ℓ en $+\infty$, et que pour tout $n \in \mathbb{N}\setminus\{0,1\}$, $u_{2n+1} \leq \ell \leq u_{2n}$.

1. Écrire un script Python donnant une valeur approchée de ℓ à 10^{-8} près.
2. Démontrer que le résultat donné par cet script est correct. On donnera clairement les éventuels invariants et variants des boucles intervenant dans le programme.

ALG-005-cor

Correction 1. Nous pouvons proposer le programme suivant :

```
from math import log

def limite () :
    """ renvoie une approxiamtion à10(-8) près de
    la limite de la suite (S_n) """
    k = 2
    u = (-1)**k/(k*log(k)) # u = S_k
    v = u + (-1)**(k+1)/((k+1)*log(k+1)) # v = S_{k+1}
    while abs(u-v) >= 2*10**(-8) :
        u = v
        k += 1
        v = u + (-1)**(k+1)/((k+1)*log(k+1))
        # invariant en sortie : u=S_k et v=S_{k+1}
        # variant : abs(u-v), réel positif
        # de limite nulle
    return (u+v)/2


limite()
```

2. Démontrons l'invariant : à chaque tour de boucle, $u = S_k$ et $v = S_{k+1}$.
Initialement, à l'entrée du premier tour de boucle, nous avons $k = 2$, $u = \frac{(-1)^2}{2\ln 2} = S_2$ et $v = u + \frac{(-1)^{k+1}}{(k+1)\ln(k+1)} = S_3$. L'invariant est donc vérifié.
Supposons l'invariant vérifié à l'entrée d'un tour de boucle donné. Notons alors k_1, u_1 et v_1 les valeurs à l'entrée de ce tour de boucle, et k_2, u_2 et v_2 leurs valeurs à la sortie. Alors $k_2 = k_1+1$, $u_2 = v_1 = S_{k_1+1} = S_{k_2}$ et $v_2 = u_2 + \frac{(-1)^{k_2+1}}{(k_2+1)\ln(k_2+1)} = S_{k_2+1}$: l'invariant est donc vérifié en sortie de boucle.
Par principe de récurrence, l'invariant est donc bien vérifié à chaque tour de boucle.

Démontrons maintenant la terminaison de l'algorithme : puisque nous savons que la suite (S_n) a une limite, alors à partir d'un certain rang la valeur absolue de la différence entre deux termes consécutifs de la suite est strictement inférieure à 2.10^{-8}. Puisqu'à l'entrée de chaque tour de boucle, $u = S_k$ et $v = S_{k+1}$, et puisque k augmente de 1 à chaque tour de boucle, alors à partir d'un certain nombre de tours de boucle, nous aurons $|u-v| < 2.10^{-8}$, et l'algorithme se termine donc.

Pour finir, démontrons la correction de l'algorithme : à la sortie du dernier tour de boucle, nous avons $|u-v| < 2.10^{-8}$. Or l'énoncé assure que la limite ℓ se trouve entre u et v. Ainsi, cette limite est à une distance inférieure à $\frac{|u-v|}{2}$ du milieu de l'intervalle $[u,v]$ (ou $[v,u]$). En renvoyant ce milieu, c'est-à-dire $\frac{u+v}{2}$, l'algorithme renvoie bien une approximation à 10^{-8} près de ℓ.

ALG-006

On appelle *nombre parfait* tout entier naturel non nul qui est égal à la somme de ses diviseurs stricts, c'est-à-dire de ses diviseurs autres que lui-même.

Par exemple, 26 n'est pas parfait, car ses diviseurs stricts sont 1, 2 et 13, et $1+2+13=16\neq 28$. Mais 28 est parfait, car ses diviseurs stricts sont 1, 2, 4, 7 et 14, et $1+2+4+7+14=28$.

QUESTION Écrire une fonction **??** prenant en entrée un entier naturel non nul `n`, et renvoyant un booléen donnant la valeur de vérité de l'assertion « `n` est parfait ».

QUESTION Écrire les éventuels variants et invariants permettant de montrer que cette fonction renvoie le bon résultat.

ALG-007

QUESTION Écrire un script permettant de calculer le plus petit entier naturel n tel que $n!>123456789$.

QUESTION Écrire les éventuels variants et invariants de boucle permettant de montrer que le code écrit précédemment donne le bon résultat.

QUESTION Montrer que les variants et/ou invariants donnés à la question précédente sont bien des variants/invariants de boucle et justifier que le script écrit termine et donne le bon résultat.

ALG-008

On s'intéresse au problème du codage d'une suite de bits (représentée sous la forme d'un tableau de 0 ou 1), de manière à pouvoir réparer une erreur de transmission. On fixe un entier naturel non nul **??**. Un tableau de bits **??** sera codé avec un niveau de redondance **??** en répétant chaque bit $2k+1$ fois. Pour décoder un tableau avec un niveau de redondance **??**, on le découpe en blocs de $2k+1$ bits. Dans chaque bloc, on effectue un « vote » et l'on considère la valeur majoritaire.

Exemple : Avec $k=2$ (et donc un niveau de redondance de 2), le tableau

$$b=[0,1,0]$$

sera codé en

$$c=[\underbrace{0,0,0,0,0}_{5\text{ bits}},\underbrace{1,1,1,1,1}_{5\text{ bits}},\underbrace{0,0,0,0,0}_{5\text{ bits}}].$$

Imaginons qu'après transmission, le tableau reçu soit

$$c'=[\underbrace{0,0,0,1,0}_{1\text{ erreur}},\underbrace{0,1,1,0,1}_{2\text{ erreurs}},\underbrace{0,1,0,1,1}_{3\text{ erreurs}}].$$

On le décode alors en

$$b'=[0,1,1].$$

QUESTION Écrire une fonction **??** renvoyant le tableau codant un tableau **??**, avec un niveau de redondance **??**.

QUESTION Écrire une fonction **??** renvoyant le tableau décodant un tableau **??**, avec un niveau de redondance **??**.

ALG-009

On considère un fichier `points.csv` contenant n lignes, chaque ligne contenant n entiers parmi 0 ou 1, séparés par des virgules. Ce fichier représente donc un tableau de nombres. Par exemple, le fichier

```
0,1,0,0
1,1,0,1
1,0,1,0
0,0,0,0
```

sera représenté (en `Python`) par le tableau bidimensionnel `t` (construit comme un tableau de tableaux) :

```
t = [ [0,1,0,0],
      [1,1,0,1],
      [1,0,1,0],
      [0,0,0,0] ]
```

QUESTION Écrire une fonction `lit_fichier(nom_de_fichier)` qui, à un tel fichier `nom_de_fichier`, renvoie le tableau associé.

On voit ce tableau comme décrivant des points dans le plan. Étant donné un tel tableau `t`, on considère que l'on a un point aux coordonnées `(i,j)` si et seulement si `t[i][j]` vaut 1. Par exemple, avec le tableau précédents, la liste `L` des points décrits est

```
L = [ (0,1) , (1,0) , (1,1) , (1,3) , (2,0) , (2,2) ]
```

QUESTION Écrire une fonction `lit_tableau(t)` qui, à un tel tableau bidimensionnel `t`, renvoie la liste des points décrits.

QUESTION Complexité ou invariants ?

QUESTION Écrire une fonction `d(a,b)` qui, pour deux couples d'entiers `a`, `b`, dont nous noterons les coordonnées repectivement (x_a,y_a) et (x_b,y_b), renvoie la valeur $(x_a-x_b)^2+(y_a-y_b)^2$.

On veut maintenant, étant donné un entier naturel k non nul et un couple d'entiers `c`, trouver les k couples de la liste de points les plus proches de `c`. S'il y a égalité entre plusieurs points, on n'en garde que k.

Par exemple [...]

On considère le code suivant.

```
def kNN(L,c,k,d):
    """k plus proches voisins du point c dans L
       d : fonction de distance"""
    v = []
    for j in range(L) :
        a = L[j]
        if len(v) < k :
            v.append(a)
        if d(a,c) < d(v[-1],c) :
            v[-1] = a
        i = len(v) - 1
        while i >= 1 and v[i] < v[i-1] :
            v[i-1],v[i] = v[i],v[i-1]
            i = i-1
    return v
```

QUESTION Donner l'invariant pour la boucle while et faire montrer que c'en est un.

QUESTION Donner l'invariant pour la boucle for.

QUESTION Complexité.

ALG-010

On s'intéresse au problème d'insertion d'un nombre dans un tableau de nombres trié par ordre croissant.

Soit `t` $= [t_0, \dots, t_{n-1}]$ un tableau de nombres trié par ordre croissant, c'est-à-dire que

$$t_0 \leq t_1 \leq \cdots \leq t_{n-1}.$$

On dit qu'un nombre x s'insère en position $i \in [\![0, n-1[\![$ dans le tableau `t` si $t_i \leq x \leq t_{i+1}$. Si $x < t_0$, alors x s'insère en position -1 dans `t` et, si $x > t_{n-1}$, alors x s'insère en position $n-1$ dans `t`.

QUESTION Écrire une fonction `position_insertion(t,x)` prenant en argument un tableau de nombres `t` trié par ordre croissant et un nombre `x` et renvoyant une position où `x` s'insère dans `t`.

ALG-011

Soit $n \geq 2$ un entier. Un diviseur strict de n est un entier $1 \leq d \leq n-1$ qui divise n (c'est-à-dire que le reste de la division euclidienne de n par d est nul).

Deux entiers n_1 et n_2 sont dits *amicaux* si la somme des diviseurs stricts de n_1 vaut n_2 et si la somme des diviseurs stricts de n_2 vaut n_1.

QUESTION Écrire une fonction `amicaux(n,m)` qui prend en argument deux entiers naturels `n` et `m` et renvoie la valeur de vérité de « `n` et `m` sont amicaux ».

On pourra écrire une fonction auxiliaire, au besoin.