Banque « Agro- Veto »
AT - 0519

Algorithmique et Informatique
Durée : 45 mn

Si au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.
L'usage d'une calculatrice est interdit pour cette épreuve.

Le sujet s'intéresse au problème de détermination de la longueur de la plus longue sous-séquence commune à deux séquences ADN. Cette longueur est un indicateur de proximité d'espèces permettant de les comparer lors d'une étude phylogénétique.
Les parties 1, 2 et 3 sont indépendantes. On pourra utiliser les fonctions de la partie 1 dans la partie 3.

1 Questions préliminaires

a. Écrire une fonction `maximum` qui renvoie le plus grand des deux nombres passés en argument.

Par exemple, `maximum(2, 4)` *devra renvoyer* `4`.

b. Écrire une fonction `zeros` qui prend en argument deux entiers naturels n et p (supposés non nuls) et qui renvoie une liste de n listes contenant chacune p zéros, représentant ainsi la matrice nulle à n lignes et p colonnes.

Par exemple, `zeros(2,3)` *devra renvoyer* `[[0,0,0], [0,0,0]]`.

2 Plus longue sous-chaine commune

On considère $A = a_1...a_n$ et $B = b_1...b_p$ deux chaines de caractères non vides.
On appelle **sous-chaine de** A toute chaine de caractères $a_{i_1}...a_{i_k}$ où $1 \leqslant i_1 < \cdots < i_k \leqslant n$ (**ces caractères ne sont pas nécessairement consécutifs** dans A).
On appelle **plus longue sous-chaine commune** à A et B toute sous-chaine commune à A et B de longueur maximale. Si l'une des chaines A ou B est vide, ou si A et B n'ont aucune sous-chaine commune, on convient que la chaine vide est l'unique plus longue sous-chaine commune à A et B.
On s'intéresse alors au problème ci-dessous.

> *Étant données deux chaines de caractères A et B de longueurs respectives n et p, quelle est la longueur d'une plus longue sous-chaine commune à A et B ?*

Par exemple, les chaines de caractères `"AAA"` *et* `"TAA"` *sont des plus longues sous-chaines communes aux chaines de caractères* `chaine1 = "ATAGA"` *et* `chaine2 ="TAACA"`.
La chaine de caractères `sschaine = "ATGC"` *est une plus longue sous-chaine commune aux chaines de caractères* `chaine1 = "AATGCG"` *et* `chaine2 ="TATTAGC"`.

a. Quelle est la longueur d'une plus longue sous-chaine commune aux chaines `"AATGCG"` et `"TATTAGC"` ? *Justifier.*

b. Déterminer une plus longue sous-chaine commune aux chaines `"AATGCG"` et `"TATTAGC"` autre que `"ATGC"`.

c. Parmi les chaines de caractères ci-dessous, indiquez sur votre copie quelle est la seule plus longue sous-chaine commune aux chaines `"TCGTA"` et `"CTG"` ? *On ne demande pas de justifier la réponse.*

☐`"CTG"` ☐`"TCGT"`
☐`"CGT"` ☐`"CG"`

d. Déterminer toutes plus longues sous-chaines communes aux chaines `"TCGTA"` et `"CTG"`.

On ne demande pas de justifier la réponse.

e. Parmi les fonctions ci-dessous, déterminer les deux seules permettant de déterminer si une chaine de caractères `ssch` est une sous-chaine d'une chaine de caractères `ch`. *On ne demande pas de justifier la réponse.*

```
def estSousChaine1(ch,ssch):
    n = len(ch)
    p = len(ssch)
    i, j = 0, 0
    while i < n and j < p:
        if ch[i] == ssch[j]:
            j += 1
        i += 1
    return j == p
```

```
def estSousChaine2(ch,ssch):
    n = len(ch)
    p = len(ssch)
    i, j = 0, 0
    while i < n and j < p:
        if ch[j] == ssch[i]:
            j += 1
        i += 1
    return i == j
```

```
def estSousChaine3(ch,ssch):
    n, p = len(ch), len(ssch)
    for i in range(n):
        j = 0
        while j < p and ch[i+j]==ssch[j]:
            j += 1
        if j == p:
            return True
    return False
```

```
def estSousChaine4(ch,ssch):
    j = 0
    for i in range(len(ch)):
        if ch[i] == ssch[j]:
            j += 1
        if j == len(ssch):
            return True
    return False
```

f. Écrire alors une fonction booléenne `sousChaineCommune` qui prend en argument trois chaines de caractères `chaine1`, `chaine2` et `sschaine` qui renvoie `True` si `sschaine` est une sous-chaine commune à `chaine1` et `chaine2`, et `False` dans le cas contraire.

3 Recherche d'une solution par programmation dynamique

Si $A = a_1...a_n$ et $B = b_1...b_p$ sont deux chaines de caractères non vides, on note $\ell_{i,j}$ la longueur d'une plus longue sous-chaine commune aux chaines $a_1a_2...a_i$ et $b_1b_2...b_j$ (respectivement composées des i premiers et j premiers caractères de A et B).
On peut montrer que la suite double $(\ell_{i,j})_{(i,j)\in[\![0,n]\!]\times[\![0,p]\!]}$ vérifie l'initialisation et la relation de récurrence :

$$\forall (i,j) \in [\![0,n]\!] \times [\![0,p]\!],\ \ell_{i,j} = \begin{cases} 0 & \text{si } i = 0 \text{ ou } j = 0 \\ 1 + \ell_{i-1,j-1} & \text{sinon si } a_i = b_j \\ \max(\ell_{i-1,j}, \ell_{i,j-1}) & \text{sinon si } a_i \neq b_j. \end{cases}$$

On cherche donc à calculer la longueur d'une plus longue sous-chaine commune à $A = a_1...a_n$ et $B = b_1...b_p$, c'est-à-dire le coefficient $\ell_{n,p}$.
Le principe est de calculer de proche en proche chaque coefficient $\ell_{i,j}$ (en respectant l'ordre défini par la relation de récurrence) en les mémorisant dans une matrice.

a. Quel est la taille (nombres de lignes et de colonnes) de la matrice où on mémorisera les coefficients $\ell_{i,j}$?

b. Compléter en annexe le code de la fonction `lplscc` calculant la longueur d'une plus longue sous-chaine commune à deux chaines de caractères `chaine1` et `chaine2` passées en argument.

c. Donner la matrice des coefficients $(\ell_{i,j})$ dans le cas où `A="CTG"` et `B="TCGT"`.

d. Expliquer (sans l'implémenter) comment adapter l'algorithme donné par la fonction `lplscc` pour construire une plus longue sous-chaine commune.

e. On sait qu'une chaine de n caractères admet au plus 2^n sous-chaines distinctes (en comptant la chaine vide).

Expliquez pourquoi la méthode programmée à la question précédente est plus efficace qu'en comparant chaque sous-chaine de A avec chaque sous-chaine de B.

FIN DU SUJET

Annexe : programme principal

```
def lplsch(chaine1, chaine2):
    n, p = len(chaine1), len(chaine2)
    l = zeros(..... , .....)
    for i in range(..... , n+1):
        for j in range(1, .....):
            if chaine1[i-1] == chaine2[j-1]:
                l[i][j] = .....
            else:
                l[i][j] = maximum(..... , .....)
    return l[.....][.....]
```

Banque « Agro-Véto »
A TB - 0721

ALGORITHMIQUE ET INFORMATIQUE

Durée : 45 minutes

L'usage d'abaques, de tables, de calculatrice et de tout instrument électronique susceptible de permettre au candidat d'accéder à des données et de les traiter par les moyens autres que ceux fournis dans le sujet est interdit.

Chaque candidat est responsable de la vérification de son sujet d'épreuve : pagination et impression de chaque page. Ce contrôle doit être fait en début d'épreuve. En cas de doute, le candidat doit alerter au plus tôt le surveillant qui vérifiera et, éventuellement, remplacera le sujet.

Ce sujet comporte 2 pages numérotées de 1 à 2.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

***Rappel** : si* `a` *et* `b` *désignent deux nombres entiers en Python, alors l'instruction* `min(a,b)` *renvoie le minimum des entiers* `a` *et* `b`. *Par exemple* `min(-4, 5)` *renvoie* `-4`.

1 Introduction au problème des coupes de somme minimale

Dans tout le sujet, on s'intéresse aux **listes dont les éléments sont des entiers relatifs**.
Pour une telle liste `t` $= [t_0, \dots, t_{n-1}]$, on appelle **coupe** de `t` toute sous-liste non vide d'éléments consécutifs de `t`. Toute coupe d'une liste `t` est donc une liste de la forme `t`$[i : j] = [t_i, \dots, t_{j-1}]$ où $0 \leqslant i < j \leqslant n$.

À toute coupe `t`$[i : j]$, on associe la somme $s_{i,j} = \sum_{k=i}^{j-1} t_k$ de ses éléments. Le but du problème est de construire un algorithme efficace pour déterminer la valeur minimale des sommes des coupes d'une liste `t`.
Par exemple, la valeur minimale des sommes des coupes de la liste `t=[4, -1, -9, -10, 9, -2, 8, -9]` *est -20, atteinte pour la coupe* `t[1:4]`.

a. Quelle est la valeur minimale des sommes des coupes de la liste `t=[4,-4,1,-1,-9,8,-3]` ?

Pour quelle coupe cette somme est-elle atteinte ?

b. Écrire une fonction `somme` prenant en argument une liste `t`, ainsi que deux entiers i et j, et qui renvoie la somme $s_{i,j}$ de la coupe `t`$[i : j]$. *On ne vérifiera pas dans le code de la fonction que les indices i et j sont valides : on supposera qu'ils vérifieront toujours la propriété* $0 \leqslant i < j \leqslant n$, *où n est la longueur de la liste* `t`.

2 Résolution par force brute

Le principe est de calculer la somme de toutes les coupes `t`$[i : j]$ et d'en déterminer le minimum.

a. Parmi les quatre fonctions ci-après, nommer l'unique fonction qui renvoie la valeur minimale des sommes des coupes de la liste passée en argument. *On ne demande pas de justifier la réponse.*

```
def somme_min_brute1(t):
    n = len(t)
    s_min = 0
    for i in range(n-1):
        for j in range(i+1,n+1):
            s = somme(t,i,j)
            if s < s_min:
                s_min = s
    return s_min
```

```
def somme_min_brute2(t):
    n = len(t)
    s_min = t[n-1]
    for i in range(n):
        for j in range(i,n):
            s = somme(t,i,j)
            if s < s_min:
                s_min = s
    return s_min
```

```
def somme_min_brute3(t):
    n = len(t)
    s_min = t[0]
    for i in range(n):
        for j in range(i+1,n+1):
            s = somme(t,i,j)
            if s < s_min:
                s_min = s
    return s_min
```

```
def somme_min_brute4(t):
    n = len(t)
    s_min = t[0]
    for i in range(n):
        for j in range(i+1,n):
            s = somme(t,i,j)
            if s_min < s:
                s_min = s
    return s_min
```

b. En déduire le code d'une fonction `coupe_min` qui prend en argument une liste d'entiers `t` et qui renvoie les indices i et j d'une coupe $\texttt{t}[i : j]$ de somme minimale.

3 Résolution par les suffixes

La méthode par force brute répond au problème mais recalcule plusieurs fois les mêmes sommes. On se propose ici de résoudre le problème pour toutes les coupes qui commencent par t_i, puis d'en déduire la solution.

a. Sans utiliser la fonction `somme`, écrire une fonction `min_coupe` prenant en argument une liste d'entiers `t` et un entier i, et qui renvoie la valeur minimale des sommes des coupes de `t` **qui commencent par** $\texttt{t}_\texttt{i}$, c'est-à-dire de la forme $\texttt{t}[i : j] = [t_i, \dots, t_{j-1}]$.

b. Recopier sur la copie et compléter le code de la fonction ci-dessous de manière à ce qu'elle renvoie la valeur minimale des sommes des coupes de la liste passée en argument.

```
def somme_min_coupe(t):
    n = len(t)
    s_min = ...
    for i in range(...):
        s = min_coupe(..., ...)
        if ... :
            ...
    return s_min
```

4 Résolution par programmation dynamique

Pour une liste $\texttt{t} = [t_0, \dots, t_{n-1}]$ et un entier naturel $j \in [\![1, n]\!]$, on note :

- x_j la valeur minimale des sommes des coupes de la sous-liste $\texttt{t}[0 : j] = [t_0, \dots, t_{j-1}]$;
- y_j la valeur minimale des sommes des coupes de la sous-liste $\texttt{t}[0 : j] = [t_0, \dots, t_{j-1}]$ **finissant par** $\texttt{t}_{\texttt{j}-1}$ (c'est-à-dire d'une coupe de la forme $\texttt{t}[i : j] = [t_i, \dots, t_{j-1}]$).

On peut montrer que, pour toute liste $\texttt{t} = [t_0, \dots, t_{n-1}]$ et tout $j \in [\![1, n-1]\!]$,

$$y_{j+1} = \min(y_j + t_j, t_j) \text{ et } x_{j+1} = \min(x_j, y_{j+1}).$$

a. Pour une liste $\texttt{t} = [t_0, \dots, t_{n-1}]$, que vaut y_1 ? et x_1 ? *On justifiera rapidement la réponse.*

b. Le code de la fonction ci-dessous contient plusieurs erreurs. Corriger sur la copie le code de la fonction ci-dessous de manière à ce qu'elle renvoie la valeur minimale des sommes des coupes de la liste passée en argument.

```
def somme_min_dyn(t):
    x = 0
    y = 0
    for j in range(1,n+1):
        x = min(x, y)
        y = min(y + j, x)
    return y
```

FIN DU SUJET

Banque « Agro-Véto »
A TB - 0723

ALGORITHMIQUE ET INFORMATIQUE

Durée : 45 minutes

L'usage d'abaques, de tables, de calculatrice et de tout instrument électronique susceptible de permettre au candidat d'accéder à des données et de les traiter par les moyens autres que ceux fournis dans le sujet est interdit.

Chaque candidat est responsable de la vérification de son sujet d'épreuve : pagination et impression de chaque page. Ce contrôle doit être fait en début d'épreuve. En cas de doute, le candidat doit alerter au plus tôt le surveillant qui vérifiera et, éventuellement, remplacera le sujet.

Ce sujet comporte 5 pages numérotées de 1 à 5.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Ce sujet comporte trois indépendantes, mais qui peuvent faire appel au Préliminaire.

Préliminaire : Loi de Bernoulli

On rappelle qu'une variable aléatoire X qui suit une loi de Bernoulli de paramètre $p \in [0, 1]$ renvoie 1 avec probabilité p et 0 sinon.

On supposera que bibliothèque `random` a été importée.

```
from random import *
```

On rappelle que `random()` renvoie un nombre pseudo-aléatoire dans $[0, 1[$ avec une probabilité uniforme.

1. Écrire une fonction `ber(p)` qui simule une variable aléatoire X qui suit une loi de Bernoulli

```
def ber(p):
    ...
return(...)
```

On pourra utiliser librement dans les parties suivantes la fonction `ber(p)`, *même si on n'a pas répondu à cette question.*

Partie 1 : Trafic routier dans un tunnel

Soit n un entier naturel supérieur ou égal à 2.

Considérons n cases disposées en ligne représentant un tunnel à sens unique.

Chaque case représente un emplacement qui peut contenir un véhicule (représenté par un 1) ou pas de véhicule (représenté par 0).

Nous représenterons une telle configuration par une liste contenant des 0 ou des 1. Par exemple :

$$T = [0, 1, 1, 0, 1, 0, 1, 1]$$

2. Que vaut `len(T)` ?

3. Que renvoie `T[1]` ?

On supposera pour simplifier que les règles d'évolution dans ce tunnel sont les suivantes :

- Une voiture avance vers sa droite si la case à sa droite est libre, sinon la voiture reste immobile.
- Si une voiture se trouve complètement à droite (à la sortie du tunnel), elle sort du tunnel et disparaît de la liste.

Par exemple, la configuration après

$$T = [0, 1, 1, 0, 1, 0, 1, 1]$$

est

$$L = [0, 1, 0, 1, 0, 1, 1, 0]$$

4. Donner sous forme d'une liste l'état du trafic après la configuration $[0, 1, 0, 1, 0, 1, 1, 0]$.

5. Recopier en complétant la fonction `tunnel(T)` en langage Python qui, à partir d'une configuration T (une liste), renvoie la configuration suivante.

```
def tunnel(T):
    L=len(T)*[0]
    for i in range(0,len(T)-1):
        if T[i]==1 and T[i+1]==0:
            ...
        if T[i]==1 and T[i+1]==1:
            ...
    return(L)
```

On ajoute une nouvelle règle. Dans le cas où la case d'indice 0 est vide (contient un 0) à l'étape k, un véhicule s'introduit dans la case 0 avec probabilité $p \in [0, 1]$ à l'étape $k + 1$.

6. Recopier en complétant la fonction `tunnel2(T,p)` en langage Python qui, à partir d'une configuration T (une liste) et d'une probabilité $p \in [0, 1]$, renvoie la configuration suivante en tenant compte de la nouvelle règle précédente.

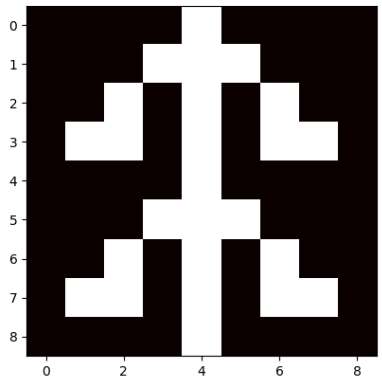
```
def tunnel2(T,p):
    L=len(T)*[0]
    for i in range(0,len(T)-1):
        if T[i]==1 and T[i+1]==0:
            ...
        if T[i]==1 and T[i+1]==1:
            ...
    if T[0]==0:
        ...
    return(L)
```

7. Soit $n \in \mathbb{N}^*$. Recopier en complétant la fonction `evol(T,n,p)` ci-dessous qui, à partir d'une configuration initiale T et d'une probabilité $p \in [0,1]$, affiche l'évolution du trafic dans le tunnel après n étapes :

```
def evol2(T,n,p):
    L=T
    print(L)
    for i in range(n):
        L= ...
        print(L)
```

Partie 2 : Motif sur un coquillage

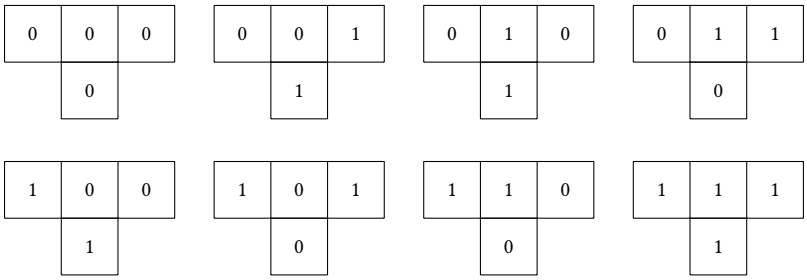
Nous nous intéressons à la formation de motifs sur un coquillage. On modélisera la surface du coquillage par une grille carrée à n lignes et n colonnes dont chaque case peut contenir du blanc (représenté par un 1) ou du noir (représenté par un 0). Par exemple la grille suivante :



sera représentée par la liste de listes :

```
[[0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 1, 1, 0, 0, 0],
 [0, 0, 1, 0, 1, 0, 1, 0, 0],
 [0, 1, 1, 0, 1, 0, 1, 1, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 1, 1, 0, 0, 0],
 [0, 0, 1, 0, 1, 0, 1, 0, 0],
 [0, 1, 1, 0, 1, 0, 1, 1, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0]]
```

Soit $k \in \mathbb{N}$. Les règles d'évolution s'effectuent de haut en bas : à partir de la ligne k, on obtient la ligne $k+1$. Le contenu d'une case à la ligne $k+1$ dépend du contenu des trois cases voisines à la ligne k.



Par défaut les cases à la ligne $k+1$ qui ne peuvent subir d'évolution (faute de voisin) contiennent des 0. Donnons un exemple d'évolution :

0	0	0	0	1	1	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0

8. À partir des règles précédentes, compléter à partir de la ligne k ci-dessous la ligne $k+1$:

0	0	1	0	1	0	0	1	1	1	0

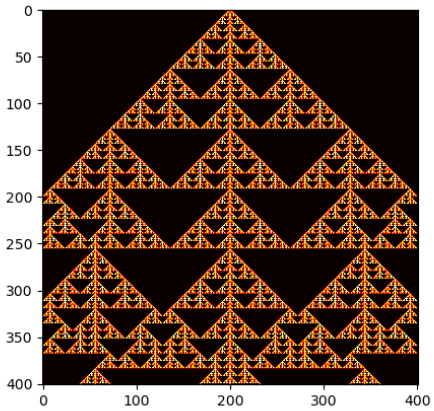
9. Recopier en complétant la fonction `coq(T)` qui, à une liste T représentant une ligne de la grille, renvoie une liste représentant la ligne suivante.

```
def coq(T):
    L=len(T)*[0]
    for i in range(1,len(T)-1):
        ...
    return(L)
```

10. Recopier en complétant la fonction `evol4(T,n)` qui, à partir d'une liste initiale T, renvoie la liste de listes A contenant l'évolution successive des lignes sur n étapes.

```
def evol4(T,n):
    A=[T]
    L=T
    for i in range(n):
        L= ...
        ...
    return(A)
```

En partant de la liste T contenant 401 zéros sauf un 1 au milieu, on obtient la figure suivante :



Les motifs qui apparaissent sont très similaires au coquillage *Conus textile* :

11. On souhaite connaître la proportion de 1 contenus dans la liste de listes M représentant la grille. Parmi les fonctions suivantes, laquelle permet d'effectuer ce calcul ?

```
def prop1(M):
    S=0
    for i in range(len(M)):
    for j in range(len(M[i])):
        S+=M[i][j]
    return(S/len(M)**2)
```

```
def prop2(M):
    S=0
    for i in range(len(M)):
        for j in range(len(M[i])):
            S+=M[i][j]
        return(S/len(M)**2)
```

```
def prop3(M):
    S=0
    for i in range(len(M)):
        for j in range(len(M[i])):
            S+=M[i][j]
    return(S/len(M)**2)
```

```
def prop4(M):
    S=0
    for i in range(len(M)):
    for j in range(len(M[i])):
        S+=M[i][j]
        return(S/len(M)**2)
```

```
def prop5(M):
    S=0
    for i in range(len(M)):
        for j in range(len(M[i])):
            S+=M[i][j]
            return(S/len(M)**2)
```

```
def prop6(M):
    S=0
    for i in range(len(M)):
    for j in range(len(M[i])):
            S+=M[i][j]
    return(S/len(M)**2)
```

Partie 3 : Épidémie dans une étable

Soit n un entier naturel supérieur ou égal à 3. On s'intéresse à la transmission de la tuberculose dans une étable au sein d'une population de bovins.

L'étable est modélisée par une grille carrée comportant n^2 cases. Dans chaque case se trouve un bovin qui peut être infecté (état 1) par la tuberculose ou sain (état 0).

On représentera informatiquement cette grille par une liste de listes comportant des 0 ou des 1 comme dans l'exemple ci-dessous :

0	0	1	1	1
0	0	1	1	1
0	0	1	1	1
0	0	1	1	1
0	0	1	0	1

```
[[0,0,1,1,1],
 [0,0,1,1,1],
 [0,0,1,1,1],
 [0,0,1,1,1],
 [0,0,1,0,1]]
```

Le voisinage d'un bovin au sein de l'étable dépend de sa position dans la grille :

V	V	V
V	B	V
V	V	V

Dans cet exemple, le bovin B possède 8 voisins V.

V	V
B	V

Ici, le bovin B est dans un coin de la grille, il possède 3 voisins V.

V	V
B	V
V	V

Le bovin B se trouve sur un coté de la grille sans être dans un coin et possède 5 voisins V.

Soit $p \in \left[0, \frac{1}{10}\right]$.

Les règles de transmission de la maladie sont les suivantes :

- Un bovin à l'état 1 reste à l'état 1.
- Un bovin à l'état 0 passe à l'état 1 avec la probabilité kp où k est le nombre de bovins infectés dans son voisinage.

12. Proposer un algorithme permettant de coder une fonction `epi(T,p)` qui, à partir d'une grille T et d'une probabilité $p \in [0, \frac{1}{10}]$, renvoie une grille A représentant l'évolution de l'étable à l'étape suivante.

On pourra utiliser la fonction `ber(p)`.

Le codage en Python de `epi(T,p)` n'est pas demandé, mais peut éventuellement constituer une manière de répondre à la question.

FIN DU SUJET