Devoir Surveillé n°5 d'Informatique

Samedi 25 Mai 2024 *(Durée : 2 heures)*

Exercice

Les données concernant les pensionnaires d'un zoo ont été stockées sous la forme d'une liste de triplets `(nom,age,espece)` nommée `Zoo`. Voici à titre d'exemple le début de la liste :

```
Zoo=[('Flipper',15,'Dauphin'),('Sophie',32,'Girafe'),('Serge',53,'Lama'),...]
```

Évidemment, les données ont été saisies dans le désordre, ce qui ne permet pas recenser les animaux par espèce. On se propose d'établir la liste des différentes espèces présentes au zoo, puis de regrouper les animaux par espèce et enfin de calculer l'âge moyen pour chaque espèce.

1) Écrire une fonction `Especes(Zoo)` qui prend en argument la liste `Zoo` et renvoie la liste des espèces présentes dans le zoo.

2) Créer un dictionnaire `ZooDico` ayant pour clefs les différentes espèces et pour valeurs associées les listes de couples `(nom,age)` des pensionnaires de chaque espèce. Par exemple, on aura :

   ```
   ZooDico={'Ours':[('Winnie',1),('Michka',7)],'Dauphin':[('Flipper',15), ...], ... }
   ```

   Attention, il ne faut pas utiliser la fonction de la question 1 ici.

3) Créer un dictionnaire `AgeMoyen` ayant pour clefs les différentes espèces et pour valeurs associées les âges moyens des pensionnaires de chaque espèce.

Problème

Définitions, rappels et notations

- Un nombre premier est un entier naturel qui admet exactement deux diviseurs : 1 et lui-même. Ainsi, 1 n'est pas premier ... et 0 non plus.
- On note $\lfloor x \rfloor$ la partie entière de x.
- `abs(x)` renvoie la valeur absolue de x.
- `int(x)` convertit un flottant en entier en tronquant sa partie décimale.
- `floor(x)` renvoie la valeur du plus grand entier inférieur ou égal à x.
- `ceil(x)` renvoie la valeur du plus petit entier supérieur ou égal à x.
- `log(x)` renvoie sous forme de flottant la valeur du logarithme népérien de x.
- `log(x,b)` renvoie sous forme de flottant la valeur du logarithme de x en base b.
- La fonction `time()` du module time renvoie un flottant représentant le nombre de secondes depuis le 01/01/1970.

Partie 1 : préliminaires

Q1) Dans un programme `Python`, on souhaite pouvoir faire appel aux fonctions `log`, `sqrt`, `floor` et `ceil` du module `math`. Écrire des instructions permettant d'avoir accès à ces fonctions et d'afficher le logarithme népérien de 0,5 dans la console.

Q2) Écrire une fonction `sont_proches(x,y)` qui renvoie `True` si la condition suivante est remplie et `False` sinon :

$$|x - y| \leqslant atol + |y| \times rtol$$

où *atol* et *rtol* sont deux constantes, à définir dans le corps de la fonction, valant respectivement 10^{-5} et 10^{-8}. Les paramètres x et y sont des nombres quelconques.

Q3) On donne la fonction `mystere` ci-dessous. Que renvoie l'instruction `mystere(1001,10)` ? On suppose que x est un nombre réel strictement positif et b un entier naturel non nul.

```
def mystere( x , b ) :
    if x < b:
        return 0
    else:
        return 1 + mystere( x/b , b )
```

Q4) Exprimer ce que renvoie `mystere` à l'aide de la partie entière et d'une fonction usuelle. Comment aurait-on pu obtenir ceci directement en `Python` ? Réécrire la fonction.

Q5) On donne le code suivant :

```
pas = 1e-5
x2 = 0
for i in range (100000):
    x1 = ( i + 1) * pas
    x2 = x2 + pas
print ("x1 : " , x1)
print ("x2 : " , x2)
```

L'exécution de ce code produit le résultat suivant :

```
x1: 1.0
x2: 0.9999999999980838
```

Commenter.

Partie 2 : Génération de nombres premiers

Le crible d'Eratosthène est un algorithme qui permet de déterminer la liste des nombres premiers appartenant à l'intervalle $[\![0 ; N]\!]$. Voici son pseudo-code :

Données : N, entier supérieur ou égal à 1
Résultat : `liste_bool`, liste de booléens
début
  `liste_bool` ← liste de (N + 1) booléens tous initialisés à `True` ;
  Remplacer par `False` les deux premiers éléments de `liste_bool` ;
  pour p entier allant de 2 à $\left\lfloor \sqrt{N} \right\rfloor$ **faire**
    si le terme d'indice p de `liste_bool` vaut `True` **alors**
      Remplacer par `False` tous les termes de `liste_bool` dont les indices sont des multiples de p différents de p ;
    fin
  fin
retourner `liste_bool`
fin

À la fin de l'exécution, les nombres premiers sont les indices des termes de `liste_bool` égaux à `True`. Par exemple, pour N=4, l'algorithme renvoie `[False,False,True,True,False]`.

Q6) Par défaut, le langage `Python` code les booléens sur 32 bits.
Quelle est (approximativement) la valeur maximale de N pour laquelle `liste_bool` est stockable dans une mémoire vive de 4 Go ?

Q7) Quel facteur peut-on gagner sur la valeur maximale de N en utilisant une bibliothèque qui permettrait de coder les booléens non pas sur 32 bits, mais dans le plus petit espace mémoire possible (à préciser, en justifiant) ?

Q8) Écrire une fonction `eratosthene(N)` qui implémente l'algorithme du crible d'Eratosthène.

Q9) Quelle est la complexité algorithmique du crible d'Eratosthène en fonction de N ?
On admettra que $\sum_{p\leqslant \mathrm{N}, p \text{ premier}} \frac{1}{p} \simeq \ln(\ln(\mathrm{N}))$

Q10) Quand on traite des nombres entiers, il est intéressant d'exprimer la complexité d'un algorithme en fonction non pas du nombre traité N, mais de son nombre de chiffres n.

Donner une approximation du résultat de la question précédente en fonction de n, en précisant la base choisie.

Partie 3 : Génération rapides de nombres premiers

L'approche systématique qui précède est inefficace, car elle revient à générer d'abord la liste de tous les nombres premiers inférieurs à une certaine valeur avant d'en choisir quelques uns au hasard.

Une meilleure idée est d'utiliser des tests probabilistes de primalité. Ces tests ne garantissent pas vraiment qu'un nombre est premier. Cependant, on montre que si un nombre passe un de ces tests, alors la probabilité qu'il ne soit pas premier est inférieure à un seuil calculable. En suivant cette idée, voici une nouvelle approche :

1) générer un entier pseudo-aléatoire (voir l'algorithme ci-dessous)
2) vérifier si cet entier a de fortes chances d'être premier (test de primalité de Fermat)
3) recommencer tant que le résultat n'est pas satisfaisant

Pour générer un entier pseudo-aléatoire A, on se base sur un certain nombre d'itérations de l'algorithme **Irigovitch-Angowski** (ou IA), dont voici le principe.

- Au début, on pose A = 0, on fixe le produit M de deux nombres premiers quelconques et on choisit aléatoirement une valeur $x_0 \in \mathbb{N}$ nommée « graine ».
- Pour tout $i \geqslant 1$, on définit x_i comme le reste de la division euclidienne de $(x_{i-1})^2$ par M.
- Pour chaque valeur de $i \geqslant 0$, si l'entier x_i est impair, on additionne alors 2^i à A.

Q11) Soit $\mathrm{N} \geqslant 1$ un nombre entier ; on répète les calculs pour $i \in [\![1 ; \mathrm{N}-1]\!]$.
Quelle sera la valeur finale de A si l'entier x_i est impair à chaque itération ?

C'est donc la valeur maximale renvoyée par l'algorithme pour une valeur donnée de N.

Q12) On choisit pour graine l'entier représentant la fraction de secondes (sur 7 chiffres) du temps t écoulé depuis le 01/01/1970 : par exemple, $t = 1528287738.7931523$ donne la graine $x_0 = 7931523$.

Compléter le code de la fonction `IA(N)` ci-contre (avec le nombre de lignes nécessaires) afin qu'elle implémente l'algorithme IA.

```
...... (à compléter)
def IA(N) :
    p1 = 24375763
    p2 = 28972763
    M= p1 * p2
    # calculer la graine
    x= ...... (à compléter)
    A = 0
    for i in range(N) :
        # si xi est impair
        if ...... (à compléter)
            A = A + 2**i
        # calculer le nouvel xi
        x = ...... (à compléter)
    return A
```

Q13) Le test probabiliste de primalité le plus simple est le test de Fermat. Ce test, qui utilise la contraposée du petit théorème de Fermat, peut s'énoncer comme suit : si $a \in [\![2 ; p-1]\!]$ est premier et que le reste de la division euclidienne de a^{p-1} par p vaut 1, alors il y a de fortes chances pour que p soit premier.

En combinant les résultats du test de primalité de Fermat pour $a = 2$, $a = 3$, $a = 5$ et $a = 7$, écrire une fonction `premier_rapide(n_max)` qui renvoie un nombre aléatoire strictement inférieur à `n_max` ayant de fortes chances d'être premier.

Le paramètre `n_max` est ici un entier supérieur à 12.

Indication : penser à utiliser le résultat de la question 11 afin de prendre une valeur de `N` *pour laquelle la fonction* `IA(N)` *renvoie toujours un entier plus petit que* `n_max`.

Q14) On souhaite caractériser le taux d'erreurs de la fonction `premier_rapide`. Pour cela, écrire une fonction `stats(N,nb)` qui va vérifier si `nb` nombres générés par `premier_rapide(N)` sont réellement premiers.

Cette fonction renvoie :

- le taux relatif d'erreur
- la liste des faux nombres premiers trouvés.

Partie 4 : Compter les nombres premiers

On étudie dans cette partie les propriétés de la fonction $\pi(n)$, qui renvoie le nombre de nombres premiers appartenant à $[\![1 ; n]\!]$.

Q15) Écrire une fonction `pi(N)` qui calcule la valeur de $\pi(n)$ pour tout entier $n \in [\![0 ; \mathrm{N}]\!]$ et qui retourne la liste des couples $(n, \pi(n))$. Le paramètre N est un entier supérieur à 1.

Par exemple, l'instruction `pi(4)` renvoie la liste `[(0,0),(1,0),(2,1]),(3,2),(4,2)]`.

Q16) Il a été prouvé que $\pi(n) > \dfrac{n}{\ln(n) - 1}$ pour tout entier $n \geqslant 5393$. On souhaite vérifier ceci.

Écrire une fonction `verif_pi(N)` qui renvoie `True` si l'inégalité est vérifiée jusqu'à N inclus, `False` sinon. Le paramètre N est un entier supérieur ou égal à 5393.