

2019 – INFO



ÉCOLE DES PONTS P RISTECH,
IS E-SUP ERO, ENST P RISTECH,
TELECOM P RISTECH, MINES P RISTECH,
MINES S INT-ÉTIENNE, MINES N NCY,
IMT atlantique, ENS E P RISTECH,
CHIMIE P RISTECH.

Concours Centrale-Supélec (Cycle International),
Concours Mines-Télécom, Concours Commun TPE/EIVP.

CONCOURS 2019

ÉPREUVE D'INFORM TIQUE COMMUNE

Durée de l'épreuve : 1 heure 30 minutes

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Cette épreuve est commune aux candidats des filières MP, PC et PSI

*Les candidats sont priés de mentionner de façon apparente
sur la première page de la copie :*

INFORM TIQUE COMMUNE

L'énoncé de cette épreuve comporte 9 pages de texte.

*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 est amené à prendre.*

Préambule

Chiffrer les données est nécessaire pour assurer la confidentialité lors d'échanges d'informations sensibles. Dans ce domaine, les nombres premiers servent de base au principe de clés publique et privée qui permettent, au travers d'algorithmes, d'échanger des messages chiffrés. La sécurité de cette méthode de chiffrement repose sur l'existence d'opérations mathématiques peu coûteuses en temps d'exécution mais dont l'inversion (c'est-à-dire la détermination des opérandes de départ à partir du résultat) prend un temps exorbitant. On appelle ces opérations « fonctions à sens unique ». Une telle opération est, par exemple, la multiplication de grands nombres premiers. Il est aisé de calculer leur produit. Par contre, connaissant uniquement ce produit, il est très difficile de déduire les deux facteurs premiers.

Le sujet étudie différentes questions sur les nombres premiers.

Les programmes demandés sont à rédiger en langage Python 3. Si toutefois le candidat utilise une version antérieure de Python, il doit le préciser. Il n'est pas nécessaire d'avoir réussi à écrire le code d'une fonction pour pouvoir s'en servir dans une autre question. Les questions portant sur les bases de données sont à traiter en langage SQL.

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 considéré comme premier.
- Un flottant est la représentation d'un nombre réel en mémoire.
- Quand une fonction Python est définie comme prenant un « nombre » en paramètre cela signifie que ce paramètre pourra être indifféremment un flottant ou un entier.
- On note $\lfloor x \rfloor$ la partie entière de x .
- `abs(x)` renvoie la valeur absolue de x . La valeur renvoyée est du même type de données que celle en argument.
- `int(x)` convertit vers un entier. Lorsque x est un flottant positif ou nul, elle renvoie la partie entière de x , c'est-à-dire l'entier n tel que $n \leq x < n + 1$.
- `round(x)` renvoie la valeur de l'entier le plus proche de x . Si deux entiers sont équidistants, l'arrondi se fait vers la valeur paire.
- `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 (supposé strictement positif).
- `log(x,n)` renvoie sous forme de flottant la valeur du logarithme de x en base n .
- La fonction `time()` du module `time` renvoie un flottant représentant le nombre de secondes depuis le 01/01/1970 avec une résolution de 10^{-7} seconde (horloge de l'ordinateur).
- L'opérateur usuel de division `/` renvoie toujours un flottant, même si les deux opérandes sont des multiples l'un de l'autre.
- L'infini $+\infty$ en Python s'écrit `float('inf')`.
- En Python 3, on peut utiliser des entiers illimités de plus de 32 bits avec le type `long`.

Partie I. Préliminaires

Q1 – Dans un programme Python on souhaite pouvoir faire appel aux fonctions `log`, `sqrt`, `floor` et `ceil` du module `math` (`round` est disponible par défaut). Écrire des instructions permettant d’avoir accès à ces fonctions et d’afficher le logarithme népérien de 0.5.

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

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

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 `mystere(1001, 10)` ? Le paramètre `x` est un nombre strictement positif et `b` un entier naturel non nul.

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

Q4 – Exprimer ce que renvoie `mystere` en fonction de la partie entière d’une fonction usuelle.

Q5 – On donne le code suivant :

```
1 pas = 1e-5
2
3 x2 = 0
4 for i in range(100000):
5     x1 = (i + 1) * pas
6     x2 = x2 + pas
7
8 print("x1:", x1)
9 print("x2:", x2)
```

L’exécution de ce code produit le résultat :

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

Commenter.

Partie II. Génération de nombres premiers

II.a Approche systématique

Le crible d'Ératosthène est un algorithme qui permet de déterminer la liste des nombres premiers appartenant à l'intervalle $\llbracket 1, n \rrbracket$. Son pseudo-code s'écrit comme suit :

Données : N , entier supérieur ou égal à 1

Résultat : *liste_bool*, liste de booléens

début

```
liste_bool ← liste de N booléens initialisés à Vrai;
Marquer comme Faux le premier élément de liste_bool;
pour entier  $i \leftarrow 2$  à  $\lfloor \sqrt{N} \rfloor$  faire
    si  $i$  n'est pas marqué comme Faux dans liste_bool alors
        | Marquer comme Faux tous les multiples de  $i$  différents de  $i$  dans liste_bool;
        n
    n
retourner liste_bool
n
```

Algorithme 1 : Crible d'Ératosthène

À la fin de l'exécution, si un élément de *liste_bool* vaut Vrai alors le nombre codé par l'indice considéré est premier. Par exemple pour $N=4$ une implémentation Python du crible renvoie [False True True False].

Q6 – Sachant que le langage Python traite les listes de booléens comme une liste d'éléments de 32 bits, quel 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 permettant de coder les booléens non pas sur 32 bits mais dans le plus petit espace mémoire possible pour ce type de données (on demande de le préciser) ?

Q8 – Écrire la fonction `erato_iter(N)` qui implémente l'algorithme 1 pour un paramètre N qui est un entier supérieur ou égal à 1.

Q9 – Quelle est la complexité algorithmique du crible d'Ératosthène en fonction de N ? On admettra que :

$$\sum_{p < N, p \text{ premier}} \frac{1}{p} \simeq \ln(\ln(N)) \quad (1)$$

La réponse devra être justifiée.

Q10 – Quand on traite des nombres entiers il est intéressant d'exprimer la complexité d'un algorithme non pas en fonction de la valeur N du nombre traité 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.

II.b Génération rapide de nombres premiers

L'approche systématique qui précède est inefficace car elle revient à attendre d'avoir généré la liste de tous les nombres premiers inférieurs à une certaine valeur pour en choisir ensuite 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, au sens probabiliste, si un nombre

réussit un de ces tests alors la probabilité qu'il ne soit pas premier est prouvée être inférieure à un seuil calculable.

En suivant cette idée, une nouvelle approche est la suivante :

1. générer un entier pseudo-aléatoire (voir ci-dessous)
2. vérifier si cet entier a de fortes chances d'être premier
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 Blum Blum Shub, décrit comme suit. On initialise A à zéro au début de l'algorithme et pour chaque itération ($i \geq 1$) on calcule :

$$x_i = \text{reste de la division euclidienne de } x_{i-1}^2 \text{ par } M \quad (2)$$

où M est le produit de deux nombres premiers quelconques et x_0 une valeur initiale nommée « graine » choisie aléatoirement. On utilise ici l'horloge de l'ordinateur comme source pour x_0 .

Puis, pour chaque x_i , s'il est impair, on additionne 2^i à A .

Q11 – On répète (2) pour i parcourant $\llbracket 1, N-1 \rrbracket$, quelle sera la valeur de A si x_i est impair à chaque itération ?

Q12 – Compléter (avec le nombre de lignes que vous jugerez nécessaire) la fonction `bbs(N)` donnée ci-dessous qui réalise ces itérations. La graine est un entier représentant la fraction de secondes du temps courant, par exemple 1528287738.7931523 donne la graine 7931523. Le paramètre N est un entier non nul.

```

1 ...
2 ...
3 def bbs(N):
4     p1 = 24375763
5     p2 = 28972763
6     M = p1 * p2
7     calculer la graine
8     ... (a compléter)
9     A = 0
10    for i in range(N):
11        if ... (a compléter) si xi est impair
12            A = A + 2**i
13            calculer le nouvel xi
14            xi = ... (a compléter)
15    return(A)
```

Le test probabiliste de primalité le plus simple est le test de primalité de Fermat. Ce test utilise la contraposée du petit théorème de Fermat qu'on peut évoquer comme suit : si $a \in \llbracket 2, p-1 \rrbracket$ 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.

Q13 – 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 inférieur strictement à n_max qui a de fortes chances d'être premier. Le paramètre `n_max` est un entier supérieur à 12.

Q14 – On souhaite caractériser le taux d'erreurs de `premier_rapide`. Écrire une fonction `stats_bbs_fermat(N, nb)` qui contrôle pour `nb` nombres, inférieurs ou égaux à N , générés par `premier_rapide`, qu'ils sont réellement premiers. Cette fonction renvoie le taux relatif d'erreur ainsi que la liste des faux nombres premiers trouvés. Les paramètres N et `nb` sont des entiers strictement positifs.