

Corrige - Epreuve IPT - Mines - 2019

Autour des nombres premiers

Partie I. Preliminaires

Q1

```
1 from math import floor, ceil, log, sqrt
2 print(log(0.5))
```

Q2

```
1 def sont_proches(x, y):
2     atol, rtol = 1e-5, 1e-8
3     return abs(x - y) < atol + abs(y) * rtol
```

Q3 La valeur retournée par `mystere(1001, 10)` est 3. En effet, $\text{mystere}(1001, 10) = 1 + \text{mystere}(100.1, 10) = 2 + \text{mystere}(10.01, 10) = 3 + \text{mystere}(1.001, 10) = 3 + 0 = 3$

Q4 `mystere(x, b)` renvoie 0 si $x < b$ et k si $b^k < x < b^{k+1}$.

$$\text{On a donc } \text{mystere}(x, b) = \begin{cases} \lfloor \frac{\ln x}{\ln b} \rfloor = \lfloor \log_b(x) \rfloor & \text{si } x \geq b \\ 0 & \text{sinon} \end{cases}$$

remarque : il faut $b \in]0, 1[$ pour que le programme fonctionne.

Q5 Mathématiquement, on devrait avoir $x_1 = 100000 * 10^{-5} = 1 = x_2$. Le réel 10^{-5} n'est pas représenté de façon exacte sous forme de flottant car les flottants sont représentés avec un nombre limité de bits. Dans le calcul de $x_2 = \sum_{i=0}^{99999} 10^{-5}$, les erreurs d'arrondis s'additionnent à chaque passage dans la boucle, ce qui fait qu'en première approximation, l'erreur initiale est multipliée par 100000. Dans le cas du calcul de $x_1 = 100000 * 10^{-5}$, on effectue le produit de deux nombres représentés par des flottants, la précision du résultat est bien meilleure.

Partie II. Generation de nombres premiers

II.a Approche systematique

Q6 Un Giga-Octet, c'est $8 \cdot 10^9$ bits donc avec 4 Go, on peut travailler au maximum avec une liste de $\frac{4 \cdot 8 \cdot 10^9}{32} = 10^9$ éléments.

Q7 Les booléens peuvent être codés sur un seul bit. On peut dans ce cas travailler avec 32 fois plus d'éléments.

conclusion : Si on code les booléens sur 1 bit alors la valeur maximale de N est $32 \cdot 10^9$

Q8 Solution du rapport du concours :

```
1 def erato_iter(N):
2     liste_bool = N * [True]
3     liste_bool[0] = False
4     i=2
5     while i**2 <= N:
6         if liste_bool[i-1]:
7             for k in range(2, N//i+1):
8                 liste_bool[k*i-1] = False
9         i += 1
10    return liste_bool
```

Q9

On exécute $\lfloor \sqrt{N} \rfloor$, la boucle conditionnelle "while". Dans cette boucle, le coût :

- est constant si i n'est pas premier (on ne fait qu'une comparaison et une incrémentation de variable),
- proportionnelle à $\lfloor \frac{N}{i} \rfloor$ si i est premier. En effet, la boucle itérative pour marquer les multiples de i est en $O(\frac{N}{i})$

Au final, la complexité est en $O(\sqrt{N} + \sum_{\substack{p \text{ premier} \\ p \leq \sqrt{N}}} \frac{N}{p}) = O(\sqrt{N} + N \ln(\ln(\sqrt{N}))) = O(N \ln(\ln(N)))$

□ Q10 Si n est le nombre de chiffres de N , on a $10^{n-1} \leq N \leq 10^n$, on a donc

$$\underbrace{10^{n-1} \ln(\ln(10^{n-1}))}_{=O(n 10^n)} \leq N \ln(\ln(N)) \leq \underbrace{10^n \ln(\ln(10^n))}_{=O(n 10^n)}$$

La complexité de l'algorithme en fonction du nombre de chiffres n de N est $O(n 10^n)$

II.b Generation rapide de nombres premiers

□ Q11 Si x_i est impair à chaque itération alors $A = \sum_{i=0}^{N-1} 2^i = \frac{2^N - 1}{2 - 1} = 2^N - 1$.

□ Q12

```

1 from time import time
2
3 def bbs(N):
4     p1 = 24375763
5     p2 = 28972763
6     M = p1*p2
7     h = time()
8     xi = floor((h - floor(h))*10**7) # Partie fractionnaire de h.
9     A = 0
10    for i in range(N):
11        if xi%2 == 1:
12            A = A + 2**i
13            xi = (xi**2)%M
14    return A

```

□ Q13

```

1 def premier_rapide(n_max):
2     trouve = False
3     while not(trouve):
4         trouve = True
5         N = mystere(n_max, 2)
6         p = bbs(N)
7         for a in [2, 3, 5, 7]:
8             if (a**(p-1))%p == 1:
9                 trouve = False
10                break
11    return p

```

□ Q14

```

1 def stats_bbs_fermet(N, nb):
2     liste_premier = erato_iter(N)
3     liste_erreur = []
4     for k in range(nb):
5         p = premier_rapide(N)
6         if not(liste_premier[p-1]):
7             liste_erreur.append(p)
8     return len(liste_erreur)/nb, liste_erreur

```

Partie III. Compter les nombres premiers

III.a Calcul de $\pi(n)$ via un crible

□ Q15

```
1 def Pi(N):
2     liste_premier = erato_iter(N)
3     compteur = 0
4     res = []
5     for n in range(1, N+1):
6         if liste_premier[n-1]: compteur += 1
7         res.append([n, compteur])
8     return res
```

□ Q16

```
1 def verif_Pi(N):
2     Pi_n = Pi(N)
3     for n in range(5393, N+1):
4         ValeurPi_n = Pi_n[n-1][1]
5         if (n/(log(n)-1)) >= ValeurPi_n :
6             return False
7     return True
```

III.b Calcul d'une valeur approchée de $\pi(n)$

Estimation de \ln par quadrature numérique

□ Q17 La fonction évaluée dépend du paramètre x et l'évaluation dépend du choix de pas . L'évaluation de l'intégrale par la méthode des rectangles est proportionnelle aux nombres de points de calcul de la fonction (égal au nombre de rectangles) soit en $O\left(\frac{x}{pas}\right)$

□ Q18 La méthode des rectangles centrés et la méthode des trapèzes ont la même complexité que la méthode des rectangles à droite donc dans ces deux nouveaux la complexité reste en $O\left(\frac{x}{pas}\right)$.

□ Q19

```
1 def inv_ln_rect_d(a, b, pas):
2     S = 0.
3     n = int((b-a)/pas)
4     for k in range(1, n+1):
5         S = S + 1/log(a+k*pas)
6     return S*pas
```

□ Q20

```
1 def li_d(x, pas):
2     if x == 1:
3         return float("inf")
4     elif x == 1 :
5         return inv_ln_rect_d(0, x, pas)
6     else : # x>1
7         return inv_ln_rect_d(0, 1-pas, pas) + inv_ln_rect_d(1+pas, x, pas)
```

Analyse des résultats de `li_d`

□ Q21 La fonction li s'annule pour $x_0 \simeq 1,4$. Dans la figure 2, on affiche une erreur relative, soit $\frac{li(x) - li_d(x)}{li(x)}$

or $li(x) \xrightarrow{x \rightarrow x_0} 0$ ce qui conduit $\frac{li(x) - li_d(x)}{li(x)} \xrightarrow{x \rightarrow x_0} +\infty$

□ Q22 On peut remarquer sur le dessin de la figure 4 que la méthode des rectangles à droite, de la part la monotonie et le signe de \ln sur $]0, 1[$ et $]1, +\infty[$, sur-évalue $\int_0^{1-\epsilon} \frac{dt}{\ln(t)}$ et sous-évalue $\int_{1+\epsilon}^x \frac{dt}{\ln(t)}$.

$$\begin{aligned} \int_{1-\epsilon}^{1+\epsilon} \frac{dt}{\ln(t)} &= \lim_{\epsilon \rightarrow 0} \int_{1-\epsilon}^1 \frac{dt}{\ln(t)} + \int_{1+\epsilon}^x \frac{dt}{\ln(t)} = \lim_{\epsilon \rightarrow 0} \int \frac{du}{\ln(1-u)} + \int \frac{dv}{\ln(1+v)} \\ &= \lim_{\epsilon \rightarrow 0} \int \left(\frac{1}{\ln(1-u)} + \frac{1}{\ln(1+u)} \right) du = \int_0^\epsilon \frac{\ln(1+u) + \ln(1-u)}{\ln(1-u)\ln(1+u)} du \end{aligned}$$

La fonction $h : u \mapsto \frac{\ln(1+u) + \ln(1-u)}{\ln(1-u)\ln(1+u)}$ est bien intégrable sur $]0, 1[$ car

$$h(u) \sim_0 \frac{u - \frac{u^2}{2} - u + \frac{u^2}{2} + o(u^2)}{(u^2 + o(u^2))} \sim_0 1$$

On en déduit aussi que $\lim_{\varepsilon \rightarrow 0} \int_{1-\varepsilon}^{1+\varepsilon} \frac{dt}{\ln(t)} = 0$.

L'aire du dernier rectangle avant $x = 1$ est $\frac{1}{\ln(1-\varepsilon)} \xrightarrow{\varepsilon \rightarrow 1} 1$ et l'aire du premier rectangle après $x = 1$ est

$$\frac{1}{\ln(1+\varepsilon)} \xrightarrow{\varepsilon \rightarrow 1} \frac{1}{2}.$$

En conclusion, avec la méthode des rectangles à droite, on introduit un biais systématique dans le calcul de $\int_0^x \frac{dt}{\ln(t)}$ pour $x > 1$.

□ **Q23** Pour résoudre ce problème, il suffit d'évaluer les deux intégrales ($\int_0^{1-\varepsilon} \frac{dt}{\ln(t)}$ et $\int_{1+\varepsilon}^x \frac{dt}{\ln(t)}$) par la méthode des rectangles centrés qui respecte mieux la propriété de pseudo-symétrie autour de $x = 1$.

remarque : on peut aussi choisir la méthode des trapèze en faisant attention à l'évaluation de la fonction $\frac{1}{\ln(t)}$ en 0.

Estimation de li via Ei

□ **Q24**

```

1 def Ei(x):
2     if x == 0:
3         return False
4     gamma, MAXINT = 0.577215664901, 100
5     k, z = 1, x
6     S = gamma + log(x)
7     fact = 1 # pour n
8     x_n = x # pour x**n
9     while not(sont_proches(S, S+z)) and k < MAXINT:
10        S = S+z
11        k = k+1
12        fact = fact*k
13        x_n = x_n * x
14        z = x_n / (k*fact)
15    if k == MAXINT:
16        return False
17    else :
18        return S
19
20 def li_dev(x):
21    return Ei(log(x))

```

Partie IV. Analyse de performance de code

□ **Q25** Il n'est pas possible d'utiliser l'attribut `nom` comme clé primaire de la table `fonction` car plusieurs enregistrements peuvent avoir le même nom (une même fonction peut avoir été testé plusieurs fois).

□ **Q26**

1.

```
1 SELECT COUNT(*) AS nb_ordi, VG(ram) AS ram_moy FROM ordinateurs
```

2.

```
1 SELECT nom FROM ordinateurs WHERE nom NOT IN(
2     SELECT teste_sur FROM fonctions WHERE nom = "li" AND algorithme="rectangles")
```

3.

```
1 SELECT algorithme, teste_sur, ram, gflops
2     FROM ordinateurs JOIN fonctions ON teste_sur = ordinateurs.nom
3     WHERE fonctions.nom = "Ei" ORDER BY temps_exec DESC
```