Corrigé - Épreuve IPT - Mines - 2019

Autour des nombres premiers

Partie I. Préliminaires

❑ **Q1**

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

❑ **Q2**

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

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

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

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

remarque : il faut $b \in \mathbb{N} \setminus \{0,1\}$ 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 `x2` $= \sum_{i=0}^{99999} 10^{-5}$, 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 $x1 = 100000 * 10^{-5}$, on effectue le produit de deux nombres représentés par des flottants, la précision du résultat est bien meilleure.

Partie II. Génération de nombres premiers

II.a Approche systématique

❑ **Q6** Un Giga-Octet, c'est 8.10^9 bits donc avec 4 Go, on peut travailler au maximum avec une liste de $\frac{4.8.10^9}{32} = 10^9$ 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 32.10^9

❑ **Q8** Solution du rapport du concours :

```
def erato_iter(N):
    liste_bool = N * [True]
    liste_bool[0] = False
    i=2
    while i**2 <= N:
        if liste_bool[i - 1]:
            for k in range(2, N//i + 1):
                liste_bool[k*i - 1] = False
        i += 1
    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 $O(\sqrt{N} + \sum_{\substack{p \text{ premier} \\ p<\sqrt{N}}} \frac{N}{p}) = O(\sqrt{N} + N \ln(\ln(\sqrt{N}))) = O(N \ln(\ln(N)))$

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

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

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

II.b Génération rapide de nombres premiers

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

❑ **Q12**

```
from time import time

def bbs(N):
    p1 = 24375763
    p2 = 28972763
    M = p1*p2
    h = time()
    xi = floor((h-floor(h))*10**7 ) # Partie fractionnaire de h.
    A = 0
    for i in range(N):
        if xi%2 == 1:
            A = A + 2**i
        xi = (xi**2)%M
    return A
```

❑ **Q13**

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

❑ **Q14**

```
def stats_bbs_fermat(N, nb):
    liste_premier = erato_iter(N)
    liste_erreur = []
    for k in range(nb):
        p = premier_rapide(N)
        if not(liste_premier[p-1]):
            liste_erreur.append(p)
    return len(liste_erreur)/nb,liste_erreur
```

Partie III. Compter les nombres premiers

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

❑ **Q15**

```
def Pi(N):
    liste_premier = erato_iter(N)
    compteur = 0
    res=[]
    for n in range(1,N+1):
        if liste_premier[n-1]: compteur += 1
        res.append([n,compteur])
    return res
```

❑ **Q16**

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

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

Estimation de li 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 $\underline{O(\frac{x}{pas})}$

❑ **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 $\underline{O(\frac{x}{pas})}$.

❑ **Q19**

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

❑ **Q20**

```
def li_d(x, pas):
    if x == 1:
        return -float("inf")
    elif x < 1 :
        return inv_ln_rect_d(0,x,pas)
    else : # x>1
        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 $\frac{li(x)-\text{li\_d}(x)}{li(x)}$ or $li(x) \xrightarrow[x\to x_0]{} 0$ ce qui conduit $\left|\frac{li(x)-li\_d(x)}{li(x)}\right| \xrightarrow[x\to x_0]{} +\infty$

❑ **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 $\left|\int_0^{1-\varepsilon} \frac{\mathrm{d}t}{\ln(t)}\right|$ et sous-évalue $\int_{1+\varepsilon}^{x} \frac{\mathrm{d}t}{\ln(t)}$.

$$\begin{aligned}\int_{1-\varepsilon}^{1+\epsilon} \frac{\mathrm{d}t}{\ln(t)} &= \lim_{\alpha\to 0^+} \int_{1-\varepsilon}^{1-\alpha} \frac{\mathrm{d}t}{\ln(t)} + \int_{1+\alpha}^{1+\varepsilon} \frac{\mathrm{d}t}{\ln(t)} = \lim_{\alpha\to 0^+} \int_{\alpha}^{\varepsilon} \frac{\mathrm{d}u}{\ln(1-u)} + \int_{\alpha}^{\epsilon} \frac{\mathrm{d}v}{\ln(1+v)} \\ &= \lim_{\alpha\to 0^+} \int_{\alpha}^{\varepsilon} \left(\frac{1}{\ln(1-u)} + \frac{1}{\ln(1+u)}\right) \mathrm{d}u = \int_0^{\varepsilon} \frac{\ln(1+u)+\ln(1-u)}{\ln(1-u)\,\ln(1+u)}\ \mathrm{d}u\end{aligned}$$

La fonction $h : u \mapsto \dfrac{\ln(1+u)+\ln(1-u)}{\ln(1-u)\,\ln(1+u)}$ est bien intégrable sur $]0,\varepsilon]$ 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 $\lim\limits_{\varepsilon\to 0}\displaystyle\int_{1-\varepsilon}^{1+\varepsilon}\frac{\mathrm{d}t}{\ln(t)} = 0$.

L'aire du dernier rectangle avant $x=1$ est $\dfrac{\varepsilon}{\ln(1-\varepsilon)} \xrightarrow[\varepsilon\to 1]{} -1$ et l'aire du premier rectangle après $x=1$ est $\dfrac{\varepsilon}{\ln(1+2\,\varepsilon)} \xrightarrow[\varepsilon\to 1]{} \dfrac{1}{2}$.

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

❑ **Q23** Pour résoudre ce problème, il suffit d'évaluer les deux intégrales ($\int_0^{1-\varepsilon}\frac{\mathrm{d}t}{\ln(t)}$ et $\int_{1+\varepsilon}^{x}\frac{\mathrm{d}t}{\ln(t)}$) 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**

```
def Ei(x):
    if x <= 0:
        return False
    gamma, MAXINT = 0.577215664901, 100
    k,z  = 1, x
    S = gamma + log(x)
    fact = 1 # pour n!
    x_n = x  # pour x**n
    while not(sont_proches(S,S+z)) and k < MAXIT:
        S = S+z
        k = k+1
        fact = fact*k
        x_n = x_n * x
        z = x_n /(k*fact)
    if k == MAXIT:
        return False
    else :
        return S

def li_dev(x):
    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.

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

2.

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

3.

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