teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun Mines-Ponts

Année : 2019

Filière : MP - PC - PSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

Mines Ponts 2019

Correction

Partie I : Préliminaire

Q1

```
from math import log, sqrt, floor, ceil
print("log népérien de 0.5 =", log(0.5))
```

Q2

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

Q3

mystere(1001, 10) renvoi 1 + mystere(100.1, 10)

mystere(100.1, 10) renvoi 1 + mystere(10.01, 10)

mystere(10.01, 10) renvoi 1 + mystère (1.001, 10)

mystere(1.001, 10) renvoi 0.

Donc le retour de l'ensemble donne 3.

Q4

```
def mystere2(x, b):
    return floor(log(x, b))
print("mystere2(1001,10) =", mystere2(1001,10))
```

Q5

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

X1 est le produit de la dernière valeur que prend i +1 le tout fois le pas c'est à dire $X1 = (99\,999+1).1.10^{-5} = 1$

X2 est la somme $X2 = \sum_{i=0}^{99\,999} 1.10^{-5}$. L'accumulation des erreurs de codage en binaire de 1.10^{-5} conduit donc à un résultat approché.

Partie II : Génération de nombres premiers

Q6 32 bits = 4 octets dont $N = {}^{4000000000}/_{4} = 1\,000\,000\,000$ valeurs.

Q7 Le plus petit espace mémoire pour stocker un booléen est 1 bit donc on peut gagner un facteur 32.

Q8

```
def erato_iter(N):
    liste_bool = [True] * N
    liste_bool[0] = False
    for i in range(2, int(sqrt(N)+1)):
        if liste_bool[i-1]:
            for j in range(i * 2, N+1, i):
                liste_bool[j-1] = False
    return liste_bool
N=20
print("Nombres premiers inférieurs ou égaux à", N)
liste_bool = erato_iter(N)
for i in range(N):
    if liste_bool[i]:
        print(i+1, end=" ")
    print()
```

Q9 La première boucle est exécutée $\sqrt{N}$ fois. La seconde boucle est effectuée N/i fois.
Donc la complexité est$O(n^{3/2}.ln(ln(N)))$

Q10 $n = Log(N)$en base10 donc la compléxité devient : $O(n^{3/2}.ln(n))$

Q11 $A = \sum_{i=1}^{N-1} 2^i = 1\frac{1-2^N}{1-2} - 1 = 2^N - 2$

Q12

```
from time import time
def bbs(N):
    p1 = 24375763
    p2 = 28972763
    M = p1 * p2
    A = 0
    xi = time()
    xi = int((xi - int(xi)) * 1e7)
    for i in range(N):
        if xi % 2 != 0:
            A = A + 2**i
        xi = xi**2 % M
return A
print(bbs(100))
```

Q13

```
premier_rapide(n_max):
    non_premier = True
    while non_premier:

        p = bbs(int(log(n_max+2)/log(2)))+1 # tirage d'un nombre aléatoire < n_max
        if p > 7:
            r2 = 2**(p-1) % p
            r3 = 3**(p-1) % p
            r5 = 5**(p-1) % p
            r7 = 7**(p-1) % p
            non_premier = (r2 != 1) or (r3 != 1) or (r5 != 1) or (r7 != 1)
    return p
```

Q14

```
def stats_bbs_fermat(N, nb):
    premiers = erato_iter(N
    erreur = 0
    faux = []
    for i in range(nb):
        p = premier_rapide(N)
        if not premiers[p-1]:
            erreur += 1
            faux.append(p)
    return erreur/nb, faux
```

Partie III : Compter les nombres premiers

Q15

```
def Pi(N):
    premiers = erato_iter(N)
    n = 0
    liste = []
    for i in range(N):
        if premiers[i]:
            n += 1
            liste.append([i+1, n])
    return liste
```

Q16

```
def verif_Pi(N):
    liste = Pi(N)
    test = True
    n = 1
    while test and n<=N:
        test = n / (log(n) - 1) < liste[n-1][1]
        n += 1
    return test
```

Q17 La méthode des rectangles doit calculer $\frac{1}{\varepsilon}$rectangles en determinant la hauteur du rectangle à chaque fois.

Q18 La méthode des rectangles centrés ne change pas le nombre de points à calculer. Pour la méthode des trapèzes, il faut à priori calculer les deux hauteurs de chaque trapèze, mais une des hauteurs est commune avec le trapèze précédent et donc n'as pas besoin d'être recalculée, elle peut être mémorisée, à part pour le premier trapèze.

Q19

```
def inv_ln_rect_d(a, b , pas):
    somme = 0
    k = 1
    x = a + pas
    while x <= b:
        somme += pas / log(x)
        k += 1
        x = a + k * pas
    return somme
```

Q20

```
def li_d(x, pas):
    if x < 1:
        return inv_ln_rect_d(0, x, pas)
    elif x == 1:
        return -float('inf')
    else:

        return inv_ln_rect_d(0, 1-pas , pas) + inv_ln_rect_d(1+pas, x , pas)
```

Q21 Au voisinage de 1.4, la valeur de li est quasiment nulle donc une faible erreur de calcul donne une erreur relative importante.

Q22 Etant donné que l'on utilise la méthode des rectangles à droite, les deux rectangles symétriques de part et d'autre de x=1 devraient avoir la même hauteur, mais comme ils sont calculés à droite, le premier a une ordonnée de $li(1-\varepsilon)$ et le second de $li(1+2.\varepsilon)$.

Q23 Pour résoudre ce problème on pourrait utiliser une méthode des rectangles centrés.

Q24

```
def li_dev(x):
    x = abs(log(x))
    ei = 0.577215664901 + log(x)
    eip = 10000
    kfact = 1
    k = 1
    MAXIT = 100
    while not sont_proches(ei, eip) or k >= MAXIT:
```

```
            eip = ei ei += x**k / (k * kfact)
            k += 1
            kfact *= k
        if sont_proches(ei, eip):
            return ei
        else:
            return False
```

Partie IV : Analyse de performance de code

Q25 Nom ne peut pas être une clé primaire car il n'identifie pas de manière unique un n-uplet.

Q26

```
SELECT COUNT(nom), AVG(ram) FROM ordinateurs ;
SELECT teste_sur FROM fonctions
WHERE id NOT IN (SELECT id FROM fonctions WHERE algorithme='rectangles') ;
SELECT algorithme, ordinateurs.nom, gflops FROM ordinateurs
JOIN fonctions ON ordinateurs.nom=teste_sur
WHERE fonctions.nom='Ei'
ORDER BY temps_exec DESC ;
```