Corrigé Mines-Ponts 2022

Informatique tronc commun 2ème année

1)

```
from math import exp
from math import tanh
from random import randrange
from random import random
```

2) À partir de l'équation 2, on obtient l'équation suivante :

$$m - tanh(m/t) = 0$$

En posant x = m, on obtient la fonction $f : x, t \mapsto x - tanh(x/t)$.

```
def f(x,t) :
    return x - tanh(x/t)
```

3)

On veut calculer un zéro à eps près, donc quand notre intervalle b-a est de taille eps, on peut arrêter la fonction et renvoyer a.

```
def dicho(f,t,a,b,eps) :
    if b - a < eps :
        return a

    m = (a + b) / 2
    if f(m,t) * f(a,t)  < 0 :
        return dicho(f,t,a,m,eps)

    else :
        return dicho(f,t,m,b,eps)
```

Récursif

```
def dicho(f,t,a,b,eps) :
    while b - a > eps :
        m = (a + b) / 2

        if f(m,t) * f(a,t)  < 0 :
            b = m

        else :
            a = m
    return a
```

Itératif

4)

La fonction **dicho** précédente coupe en deux l'intervalle de recherche à chaque appel récursif (ou tour de boucle). La fonction consiste à trouver dans l'intervalle [a,b] le sous-intervalle où f s'annule parmi [a,a+eps[, [a+eps, a + 2eps[.... [a+Neps, b[. Les sous-intervalles sont au nombre maximum de $\frac{b-a}{eps} + 1$. À chaque appel (ou tour de boucle), grâce à la dichotomie, on écarte la moitié des intervalles. la complexité est donc en $O(log_2(\frac{b-a}{eps}))$.

5)

```
def construction_liste_m(t1, t2) :
    a = 0.001
    b = 1.0
    eps = 0.000001
    reponse = []
    for i in range(0,500) :
        t = ((499 - i) * t1 + i * t2) / 499
        if t > 1 :
            reponse.append(0)
        else :
            reponse.append(dicho(f,t,a,b,eps))
    return reponse
```

6)

```
SELECT nom
FROM materiaux
WHERE t_curie < 500
```

7)

```
SELECT nom_fournisseur, 4.5 * prix_kg
FROM prix
JOIN fournisseurs ON fournisseurs.id_fournisseurs = prix.id_four
WHERE prix.id_mat = 8713
```

8)
On ajoute ces lignes à la fin de la requête précédente :

```
AND prix_kg = (SELECT min(prix_kg)
               FROM prix
               JOIN fournisseurs ON fournisseurs.id_fournisseurs = prix.id_four
               WHERE id_mat = 8713)
```

Nous sommes obligés de rajouter une requête imbriquée si on veut tous les fournisseurs proposant un prix minimum

9)

```
SELECT nom, AVG(prix_kg) AS prix_moyen
FROM materiaux
JOIN prix ON materiaux.id_materiau = prix.id_mat
GROUP BY nom
HAVING prix_moyen < 50
```

10)

```
def initialisation() :
    reponse = [1 for _ in range (n)]
    return reponse
```

11)

```
def initialisation_anti() :
    reponse = []
    for i in range(h) :
        for j in range(h) :
            # i commence à zéro
            # La première ligne aura bien des 1
            reponse.append((-1) ** i)
    return reponse
```

12)

```
def repliement(s) :
    reponse = []
    for i in range(h) :
        ligne_suiv = []
        for j in range(h) :
            ligne_suiv.append(s[i * h + j])
        reponse.append(ligne_suiv)
    return reponse
```

13)

```
def liste_voisins(i) :
    gau = (i - 1) % h + (i//h)*h
    droi = (i + 1) % h + (i//h)*h
    sous = (i + h) % n
    sur = (i - h) % n

    return [gau, droi, sous, sur]
```

14)

```
def energie(s) :
    somme = 0
    for i in range(n) :
        voisins = liste_voisins(i)
        for j in voisins :
            somme = somme + s[i] * s[j]

    return -(1 / 2) * somme
```

15)

```
def test_boltzmann(delta_e,T) :
    if delta_e <= 0 :
        return True

    p = random()
    if p < exp(-delta_e / T) :
        return True
    return False
```

16)

La solution la plus efficace est la deuxième. La première solution fait une copie du tableau s, modifie un spin et recalcule l'énergie de tout le tableau pour calculer la différence ΔE. La complexité est en O(n).

La deuxième fonction ne calcule le delta que de manière locale, à l'endroit où le spin est modifié. La complexité de cette deuxième fonction est donc en O(1).

17)

```
def monte_carlo(s,T,n_tests) :
    for _ in range (nb_tests) :
        i = randrange(0,n)
        delta_e = calcul_delta_e2(s,i)
        if test_boltzmann(delta_e,T) :
            s[i] = s[i] * (-1)
```

18)

```
def aimantation_moyenne(n_tests,T) :
    s = initialisation()
    monte_carlo(s,T,n_tests)

    somme_spin = 0
    for si in s :
        somme_spin = somme_spin + si

    return somme_spin / n
```

19)

initialisation est en O(n).

monte_carlo est en O(n_test) *(si on a bien choisi calcul_delta_e2, sinon, avec l'autre, c'est en grand O((n_test) * n).*

Enfin, la somme des spins se fait en O(n).

La complexité de **aimantation_moyenne(n_tests,T)**, qui réalise de manière séquentielle les opérations précédentes, est en $O(n\_tests + n)$.

20)

Oui, la complexité aurait été modifiée. Celle de la fonction **calcul_delta** n'aurait plus été constante mais en $O(n)$. On aurait alors une complexité pour **monte_carlo** en $O(n\_tests * n)$, donc la même pour **aimantation_moyenne(n_tests,T)**.

21)

Plus la température augmente et plus les zones continues de même spin sont petites. En effet, à basse température, il n'y a que quatre zones continues de spins positifs, alors qu'à haute température, il y en a beaucoup plus.

22)

```
def explorer_voisinage(s,i,weiss,num) :
    weiss[i] = num
    voisin = liste_voisins(i)
    for si in voisin :
        if weiss[si] == -1 and s[si] == s[i] :
            explorer_voisinage(s,si,weiss,num)
```

23)

```
def explorer_voisinage_pile(s,i,weiss,num,pile) :

    while pile != [] :
        etud = pile.pop()
        weiss[etud] = num
        voisin = liste_voisins(etud)
        for si in voisin :
            if s[si] == s[etud] and weiss[si] == - 1 :
                pile.append(si)
```

24)

```
def construire_domaine_weiss(s) :
    num = 0
    weiss = [-1 for _ in range(n)]
    for i in range(n) :
        if weiss[i] == -1 :
            pile = [i]
            explorer_voisinage_pile(s,i,weiss,num,pile)
            num = num + 1
    return weiss
```