teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun Mines-Ponts

Année : 2021

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

En cas de doute, n'hésitez pas à nous contacter à : corrigesconcours@upsti.fr.

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

Marchons, marchons, marchons...

Corrigé UPSTI

Partie I. Randonnée

Objectif

Base de données : requêtes SQL Python : utilisation de fichier texte et exploitation des données

Question 1 Compter le nombre de participants nés entre 1999 et 2003 inclus.

```
SELECT COUNT(*) FROM Participant WHERE ne BETWEEN 1999 AND 2003 ;
```

Question 2 Calculer la durée moyenne des randonnées pour chaque niveau de difficulté.

```
SELECT diff, AVG(duree)FROM Rando GROUP BY diff;
```

Question 3 Extraire le nom des participants pour lesquels la randonnée n°42 est trop difficile

```
SELECT pnom FROM Participant WHERE diff_max > (SELECT diff FROM Rando WHERE rid = 42) ;
```

Question 4 Extraire les clés primaires des randonnées qui ont un ou des homonymes sans redondance.

```
SELECT DISTINCT rid FROM Rando WHERE rnom IN (SELECT rnom FROM Rando GROUP BY rnom
HAVING COUNT(*) > 1 ) ;
```

Question 5 Implémenter la fonction importe_rando(nom_fichier) qui réalise l'importation en retournant une liste

```
def importe_rando(nom_fichier):
    coords = []
    fichier = open(nom_fichier, "r")
    ligne = fichier.readline()
    ligne = fichier.readline()
    while ligne:
            valeurs = ligne.split(',')
            coords.append([float(v) for v in valeurs])
            ligne = fichier.readline()
    fichier.close()
    return coords
```

Question 6 Implémenter la fonction plus_haut(coords) qui renvoie la liste(lattitude, longitude) du point le plus haut de la randonnée.

```
def plus_haut(coords):
    lat, lon = coords[0][0], coords[0][1]
    maxi = coords[0][2]
    for pt in coords[1:]:
        if pt[2] > maxi:
            lat, lon  = pt[0], pt[1]
            maxi = pt[2]
    return [lat, lon]
```

Question 7 Implémenter la fonction deniveles(coords) qui calcule les dénivelés cumulés positif et négatif sous forme d'une liste de deux flottants.

```
def deniveles(coords):
    positif, negatif = 0, 0
    for i in range(len(coords)-1):
        delta = coords[i+1][2] - coords[i][2]
        if delta > 0:
            positif += delta
        else:
            negatif -= delta
    return [positif, negatif]
```

Question 8 Implémenter la fonction distance(c1, c2) qui calcule en mètres la distance entre deux points de passage.

```
def distance(c1, c2):
    Rt = 6371e3 + (c1[2] + c2[2]) /2

    phi_1, lambda_1 = radians(c1[0]), radians(c1[1])
    phi_2, lambda_2 = radians(c2[0]), radians(c2[1])

    d_phi = (phi_2 - phi_1) / 2
    d_lambda = (lambda_2 - lambda_1) /2

    racine = (sin(d_phi))**2 + cos(phi_1) * cos (phi_2) * (sin(d_lambda))**2
    d = 2 * Rt * asin(sqrt(racine))

    h = c1[2] - c2[2]

    return sqrt(d**2 + h**2)
```

Question 9 Implémenter la fonction distance_totale(coords) qui calcule la distance en mètres parcourue au cours de la randonnée.

```
def distance_totale(coords):
    d = 0
    for i in range(len(coords)-1):
        d += distance(coords[i], coords[i+1])
    return d
```

Partie II. Mouvement Brownien d'une petite particule

Objectif

Utiliser la méthode d'Euler

Question 10 Implémenter la fonction vma(v1, a, v2) (multiplication-addition sur des vecteurs) avec v1 et v2 des listes de flottants qui renvoie une nouvelle liste de flottants correspondant à $\vec{v} = \vec{v_1} + a\vec{v_2}$

```
def vma(v1, a, v2):
    assert len(v1) == len(v2)
    v = []
    for i in range(len(v1)):
        v.append(v1[i] + a * v2[i]
    return v
```

Question 11 Implémenter la fonction derive(E) qui renvoie la dérivée du vecteur d'état passé en paramètre d'après l'équation différentielle décrite en introduction de la partie

```python
from random import uniform, gauss
from math import cos, sin, sqrt, pi
def derive(E):
    norme = gauss(MU, SIGMA)
    direction = uniform(0, 2*pi)
    fb = [norme * cos(direction) / M, norme * sin(direction) / M]
    E1 = E[2] / M , E[3] / M
    E2 = vma(fb, -ALPHA, E1)
    return E[2:4] + E2
```

Question 12 Implémenter la fonction euler(E0, dt, n) pour résoudre numériquement l'équation différentielle par la méthode d'Euler.

```python
def euler(E0, dt, n):
    Es = [E0]

    for i in range(n):
        E1 = derive(Es[-1])
        Es.append([Es[-1][0] + dt * E1[0],
                   Es[-1][1] + dt * E1[1],
                   Es[-1][2] + dt * E1[2],
                   Es[-1][3] + dt * E1[3],])
    return Es
```

Partie III. Marche auto-évitante

Objectif

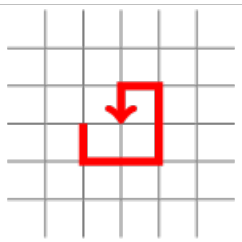
Manipulation de liste

Question 13 Implémenter la fonction positions_possibles(p, atteints) qui construit la liste des positions suivantes possibles à partir du point p.

```python
from random import randrange, choice

def position_possibles(p, atteints):
    possibles = []
    for dx, dy in ((-1,0), (1,0), (0,-1), (0,1)):
        if [p[0] + dx, p[1] + dy] not in atteints:
            possibles.append((p[0] + dx, p[1] + dy))
    return possibles
```

Question 14 Mettre en évidence graphiquement un exemple de CAE le plus court possible pour lequel, à une étape donnée, la fonction position_possible va renvoyer une liste vide.



Question 15 Implémenter la fonction genere_chemin_naif(n) qui construit la liste des points représentant le chemin auto évitant de longueur n

```
def genere_chemin_naif(n):
    chemin = [[0, 0]] # on part de l'origine
    echoue = False
    while len(chemin) < n and not echoue:
        possibles = position_possibles(chemin[-1] , chemin)
        if len(possibles) == 0:
            echoue = True
        else:
            chemin.append(choice(possibles))
    if echoue:
        return None
    return chemin
```

Question 16 Evaluer avec soin la compléxité temporelle asymptotique dans le pire des cas de la fonction genere_chemin_naif(n)

n appel de positions possibles, pour chacun 4 tests provoquant une recherche séquentielle.

Donc la complexité est en $O(n^2)$

Question 17 Décrire ce que représente ce graphique et interpréter sa signification pour la méthode naïve.

Ce graphique représente la probabilité moyenne (avec 10000 essais) que la recherche d'un chemin échoue (la fonction position_possibles retourne None) en fonction de la longueur du chemin de 0 à 350 étapes.

Question 18 Rappeler la complexité temporelle asymptotique dans le pire des cas attendue de sorted. Donner le nom d'un algorithme possible pour son implémentation.

La complexité de sorted est $O(n.\log n)$. Un algorithme utilisable est le tri fusion.

Question 19 Implémenter la fonction est_CAE(chemin) qui vérifie si un chemin est auto-évitant en se basant sur sorted et qui renvoie un résultat booléen.

```
def est_CAE(chemin):
    ordone = sorted(chemin)
    auto_evitant = True
    i = 1
    while i < len(chemin) and auto_evitant:
        auto_evitant = (ordone[i] != ordone[i-1])
        i += 1
    return auto_evitant
```

La complexité de cet algorithme, après le sorted, est linéaire $O(n)$.

Question 20 Implémenter la fonction rot(p, q, a) qui renvoie le point image du point q par la rotation de centre p et d'angle défini par la valeur de a o pour π, 1 pour $\frac{\pi}{2}$, 2 pour $-\frac{\pi}{2}$.

```
def rot(p, q, a):
    dx = p[0] - q[0]
    dy = p[1] - q[1]
    if a==0: # pi
        return [p[0] + dx, p[1] + dy]
    elif a == 1: # +pi/2
        return [p[0] + dy, p[1] + dx]
    else: # # -pi/2
        return [p[0] - dy, p[1] - dx]
```

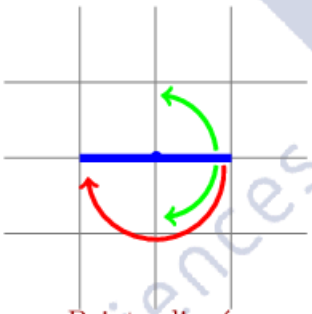
Question 21 Implémenter la fonction rotation(chemin, i_piv, a) qui renvoie un nouveau chemin identique à chemin jusqu'au pivot d'indice i_piv, et ayant subit une rotation de a autour du pivot sur la partie strictement après le pivot.

```
def rotation(chemin, i_piv, a):
    pivot = chemin[i_piv]
    for i in range(i_piv+1, len(chemin)):
        chemin[i] = rot(pivot, chemin[i], a)
    return chemin
```

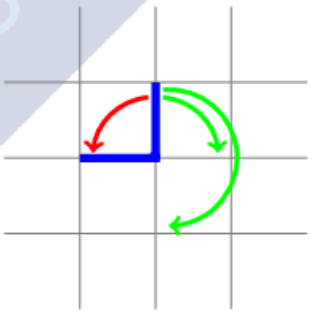
Question 22 Implémenter la fonction genere_chemin_pivot(n, n_rot) permettant de générer un chemin auto évitant de longueur n en appliquant n_rot rotations.

```
def genere_chemin_pivot(n, n_rot):
    chemin = [[i, 0] for i in range(n)]
    for i in range(n_rot):
        auto_evitant = False
        while not auto_evitant:
            copie = chemin.copy()
            i_piv = randrange(1, n-1)
            a = randrange(0, 3)
            rotation(copie, i_piv, a)
            auto_evitant = est_CAE(copie)
        chemin = copie
    return chemin
```

Question 23 On considère un pivot, son point précédent et son point suivant. Quel est l'impact prévisible sur les rotations admissibles ? Suggérer un moyen de prendre en compte cette propriété pour améliorer l'algorithme.



Points alignés



Points à 90°

Il faut tenir compte de l'orientation du dernier segment par rapport au précédent pour ne choisir que parmi les rotations possibles.