teaching sciences UPSTI for innovation

Proposition de corrigé

Concours : Concours Commun Mines-Ponts

Année : 2020

Filière : MP - PSI - PC

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.

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

Images de vagues et de structures

Corrigé UPSTI

I - Création d'un objet dans la scène

I.1 - Le Chargement d'un modèle 3D à partir d'une base de données

id	nom
1	coque
2	bouée
3	échelle
4	moteur
...	...

(a) Relation maillages_bateau

numero	maillage	s1	s2	s3
1	3	1	2	3
2	3	2	4	3
3	2	3	12	5
...	...	...	...	...

(b) Relation faces

id	x	y	z
1	0.0	0.0	0.0
2	1.0	0.0	0.0
3	0.0	1.0	0.0
...	...	...	...

(c) Relation sommets

Question 1 Proposez une requête SQL permettant de compter le nombre de maillages que contient le modèle du bateau.

SELECT COUNT(*) FROM maillages_bateau

Question 2 Proposez une requête SQL permettant de récupérer la liste des numéros des facettes (numero) du maillage nommé « gouvernail ».

SELECT numero FROM faces
JOIN maillages_bateau
ON faces.maillage=maillages_bateau.id
WHERE maillages_bateau.nom= 'gouvernail'

OU avec sous-requête :

SELECT numero FROM faces
WHERE maillage= (SELECT id FROM maillages_bateau WHERE nom= 'gouvernail')

OU mais on aime moins.

SELECT numero FROM faces, maillages_bateau
WHERE faces.maillage=maillages_bateau.id AND maillages_bateau.nom= 'gouvernail'

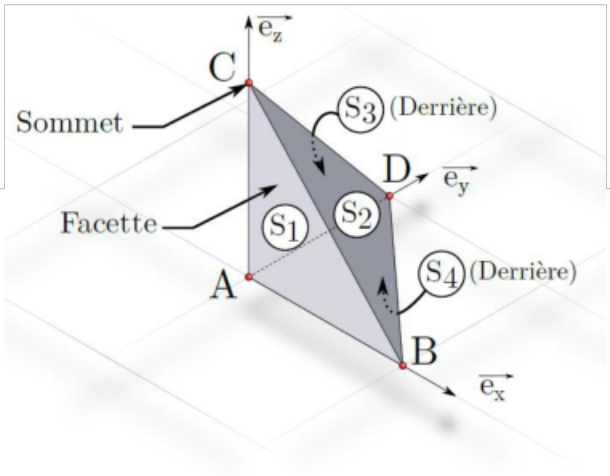
Question 3 Expliquez ce que renvoie la requête SQL suivante :

Requête SQL

```
1 SELECT (MAX(x)-MIN(x))
2 FROM sommets AS s JOIN faces AS f JOIN  maillages_bateau AS m
3 ON (s.id=f.s1 OR s.id=f.s2 OR s.id=f.s3) AND f.maillage=m.id
4 WHERE m.nom="coque"
```

La requête renvoie, pour le maillage « coque » la différence entre la valeur la plus grande et la valeur la plus petite des x de tous les sommets. Autrement dit la longueur de la coque suivant l'axe des x.

I.2 - Travail sur les facettes



(a) Un maillage simple : le tétraèdre

```
maillage_tetra = [ [[0.,0.,0.], [0.,0.,1.], [0.,1.,0.]],
                   [[0.,0.,0.], [0.,1.,0.], [1.,0.,0.]],
                   [[0.,0.,0.], [1.,0.,0.], [0.,0.,1.]],
                   [[1.,0.,0.], [0.,1.,0.], [0.,0.,1.]] ]
```

Question 4 A partir de la variable **maillage_tetra**, écrire une expression Python permettant de récupérer la coordonnée y du premier sommet de la première facette.

Maillage_tetra[0][0][1]

Question 5 A quel élément, sur la figure 2(a), correspond **maillage_tetra[1]** ?

maillage_tetra[1] correspond à la facette de sommets [0,0,0] (A) ; [0,1,0] (D) ; [1,0,0] (B) donc la facette S4.

Le sujet introduit un module **operations_vectorielles** contenant différentes fonctions (voir sujet) que l'on va pouvoir utiliser dans la suite.

Question 6 On souhaite utiliser les fonctions de ce module depuis un autre fichier Python. Complétez le code ci-dessous afin qu'il fournisse le résultat attendu :

code Python

```
1 from ??? import ?? as ?
2 vect_1 = [1., 2., 3.]
3 vect_2 = [2., 3., 4.]
4 scal12 = ps(vect_1, vect_2) #Calcul du produit scalaire de vect_1 avec vect_2
```

from operations_vectorielles **import** prod_scalaire **as** ps

Dans ce module, une fonction n'a pas été documentée :

code Python

```
1 def mystere1(V):
2     return (V[0]**2 + V[1]**2 + V[2]**2)**0.5
```

Question 7 Que fait la fonction ***mystere1*** ?

La fonction *mystere1* renvoie la norme euclidienne du vecteur V.

Question 8 Créer la fonction ***multiplie_scalaire***, prenant comme argument un flottant $\boldsymbol{a}$ et un vecteur $\vec{\boldsymbol{V}}$ et renvoyant un nouveau vecteur correspondant à $\boldsymbol{a}\vec{\boldsymbol{V}}$.

```
def multiplie_scalaire(a,V):
    aV=[] #initialisation du vecteur renvoyé
    for i in range(len(V)):
        aV.append(a*V[i])
    return aV
```

La fonction barycentre (incomplète) est définie ci-dessous.

code Python

```
1 def barycentre(F):
2     G = [0,0,0]
3     for i in range(3):   #Pour chaque point de F
4         .....    # Ligne à compléter
5         .....    # Ligne à compléter
6     return G
```

Question 9 Compléter les lignes 4 et 5 permettant de calculer le barycentre.

4 V=multiplie_scalaire(1/3,F[i])
5 G=addition(G,V)

OU

4 for j in range(3) :
5 G[i]+=F[j][i]/3

Question 10 Pour une facette F=(A,B,C) d'aire non-nulle, proposer une fonction ***normale***, prenant comme argument une facette **F** et renvoyant le vecteur unitaire normal

$$\vec{n} = \frac{\overrightarrow{AB} \wedge \overrightarrow{AC}}{\|\overrightarrow{AB} \wedge \overrightarrow{AC}\|}$$

```
def normale(F):
    AB=soustraction(F[1],F[0])
    AC=soustraction(F[2],F[0])
    pVect=produit_vectoriel(AB,AC)
    coef=1/mystere1(pVect)
    return multiplie_scalaire(coef,pVect)
```

I.3 - Liste des sommets

Question 11 Compte tenu de la représentation limitée des nombres réels en machine, deux sommets S_1 et S_2 supposés être au même endroit peuvent avoir des coordonnées légèrement différentes. Proposer une fonction ***sont_proches***, prenant comme arguments deux sommets $\boldsymbol{S_1}$ et $\boldsymbol{S_2}$ (représentés par leur vecteur position) et un flottant positif **eps**, et qui renvoie **True** si S1 et S2 sont proches (i.e. si leur distance au sens de la norme Euclidienne est inférieure à eps) et **False** sinon.

```
def sont_proches(S1,S2,eps):
    V=soustraction(S2,S1)
    return mystere1(V)<eps
```

Soient les fonctions suivantes :

code Python

```
1  def mystere2(S1, L):
2      for S2 in L:
3          if sont_proches(S1, S2, 1e-7):
4              return True
5      return False
6
7
8  def mystere3(maillage):
9      res = []
10     for facette in maillage:
11         for sommet in facette :
12             if not mystere2(sommet, res):
13                 res.append(sommet)
14         return res
```

Question 12 Sous quelle condition la fonction **mystere2** renvoie-t-elle **True** ?

mystere2 renvoie TRUE si un des sommets de L est proche de S1 (distance inférieures à 10^{-7} m)

Question 13 Donner (sans justification) ce que renvoie **mystere3(maillage_tetra)**, dans le cas où **maillage_tetra** est la variable définie précédemment.

Il semblerait que l'indentation de la ligne 14, correspondant au contenu « return res » de mystere3, soit erroné. En l'état mystere3 renvoie les sommets de la première facette [[0.,0.,0.],[0.,0.,1.],[0.,1.,0.]], rendant la boucle for externe inutile .

Dans la suite de ce corrigé, nous considérerons que cette ligne est d'indentation identique à la ligne 10.

mystere3 renvoie la liste des 4 sommets du tétraèdre (sans doublons), à savoir *[[0.,0.,0.],[0.,0.,1.],[0.,1.,0.],[1.,0.,0.]].*

Question 14 Pour une liste **L** de longueur **n**, discuter la complexité de la fonction **mystere2**. En déduire la complexité de **mystere3**, pour un maillage contenant **m** facettes triangulaires. On distinguera le meilleur et le pire des cas.

La fonction **mystère2** est constituée d'une boucle for et d'un test utilisant la fonction **sont_proches** qui est en O(1). Dans le meilleur des cas, le test est toujours vrai et la boucle n'est exécutée qu'une seule fois (le premier point de L est proche de S1) : la complicité est en O(1). Dans le pire des cas, le test n'est jamais vrai et la boucle est exécutée n fois (aucun point de L n'est proche de S1) : la complexité est en O(n).

La fonction **mystère3** est constituée de deux boucles for imbriquées, l'une externe exécutée m fois et l'autre interne exécutée 3 fois (facettes triangulaires). Le corps de boucle de la boucle interne est constituée d'opération à cout constant et de la fonction **mystère2** précédente.

Au meilleur des cas, **mystère2** est en O(1) donc on obtient une complexité de **mystere3** en O(m) (tous les points du maillage sont proches ($<10^{-7}$m) les uns des autres).

Au pire des cas, **mystère2** est en O(len(res)), res étant une liste qui augmente d'un élément à chaque exécution de la boucle interne, soit une complexité de la forme 1+2+3+…+3m, donc on obtient une complexité de **mystere3** en O(m^2) (tous les points du maillage sont loin ($>10^{-7}$m) des autres).

II - Génération de vagues

La scène est composée de 350 images. Le plan d'eau est composé de 200x200 sommets. Chaque hauteur $h_{i,j}$ est un flottant codé sur 64 bits.

Question 15 Quel est l'espace occupée en mémoire vive par l'ensemble des données (en Mo).

Mémoire = 350x200x200x8(octets)=7*16Mo=112Mo (en prenant 1Mo=1000000 octets)

Question 16 Ecrire une fonction ***mat2str*** qui prend en argument une liste de listes (représentant un **mat_h**) et renvoie les données qu'elle contient sous forme d'une chaîne de caractères qui respecte le format suivant :

$$\left|\begin{array}{l} h_{01};h_{02}; \ \ldots \ ;h_{0m} \\ h_{11}; \ \ldots \quad ;h_{1m} \\ \ldots \\ h_{m0}; \ \ldots \quad ;h_{mm} \end{array}\right.$$

```
def mat2str(M):
    chaine="" #initialisation de la chaine de caractere
    for i in range(len(M)): #on boucle sur les lignes
        ligne="" #initialisation d'une ligne
        for j in range(len(M[0])):
            ligne+=str(M[i][j])+";" #on ajoute "hij;" à la chaine
        #on enleve le ";" en fin de ligne et on ajoute '\n'
        ligne=ligne[:len(ligne)-1]+"\n"
        chaine+=ligne
    return chaine
```

Question 17 En s'appuyant sur **mat2str**, proposer un code Python qui permet de sauvegarder le contenu de **liste_vagues** dans un fichier nommé fichier_vagues.txt (dans le répertoire courant), en séparant la représentation de chaque **mat_h** par deux sauts de lignes consécutifs.

```
with open('fichier_vague.txt','w') as fichier:
    for math in liste_vagues:
        fichier.write(mat2str(math))
        fichier.write('\n\n')
```

Ou

```
f=open('fichier_vague.txt','w')
for math in liste_vagues:
        f.write(mat2str(math))
        f.write('\n\n')
f.close()
```

Question 18 Après avoir défini judicieusement les types des éléments contenus dans I, J puis N, estimer la taille (en octets) que prendra une matrice ayant p éléments non-nuls, au format « Coordinate Format », dans le fichier.

I et J seront de type **Entier** et N sera de type **Flottant**.

Pour les coordonnées comprises entre 0 et 200, en supposant que chaque coordonnée est écrite avec 3 caractères (même pour les nombres de 0 à 99), chaque point est stocké avec : 2 coordonnées (6 caractères) + 1 flottant 15 caractères+2 « ; »+le caractère de fin de ligne « \n » soit 24 caractères codés sur 1 octet.

La matrice aura donc une taille de 24*p octets.

Question 19 En déduire à partir de combien d'éléments non-nuls il devient moins avantageux d'enregistrer une matrice creuse qu'une matrice complète classique.

Dans la matrice classique on stocke chaque hauteur par un flottant (15 caractères) auquel on ajoute « ; » ou « \n » donc sa taille en fichier texte est de

Taille=200*200*16=640 ko

Ainsi, la valeur maximale de p pour que le nouveau stockage soit intéressant est

$$p_{max} = ENT\left(\frac{640000}{24}\right) \approx 26\,666$$

Question 20 Proposer un code permettant de construire, pour un tableau **mat_h** donné, les listes Python I,J et N. On considérera nulles les hauteurs inférieures à 10^{-3} (en valeur absolue).

```
I,J,N=[],[],[] #initialisation des listes
for i in range(len(mat_h)):#on parcours les lignes
    for j in range(len(math[0])): #on parcours les colonnes
        if abs(mat_h[i][j])>1e-3: #on test la hauteur
        # si la hauteur est suffisante, on stock les valeurs
            I.append(i)
            J.append(j)
            N.append(mat_h[i][j])
```

III - Mouvement de flottaison

III.1 - Estimation de la poussée d'Archimède

Question 21 Proposer une fonction **lister_FI** prenant comme argument un maillage **M** et renvoyant la liste des facettes immergées (i.e dont le centre de gravité est sous la surface définie par **hauteur**). On pourra utiliser les fonctions de la partie I.

```
def lister_FI(M):
    liste_facette=[] #initialisation de la liste
    for facette in M:  #on parcours les facette
        G=barycentre(facette) #on determine les coordonnées du barycentre
        if hauteur(G[0],G[1])>G[2]: #on compare la hauteur d'eau et la hauteur du CG
            liste_facette.append(facette)
    return liste_facette
```

Question 22 Proposer une fonction **force_facette** prenant en argument une facette **F**, et renvoyant le vecteur force appliqué par l'eau sur cette facette. On pourra utiliser les fonctions définies précédemment.

```
def force_facette(F):
    #on défini les constantes
    rho=1000
    g=9.81
    #on calcul les valeurs utiles
    S=aire(F)
    G=barycentre(F)
    p=rho*g*(hauteur(G[0],G[1])-G[2])
    forceF=multiplie_scalaire(-p*S,normale(F))
    return forceF
```

Question 23 Définir la fonction **resultante** prenant comme argument une liste L de facettes (supposées immergées), renvoyant la somme des forces sur l'axe $\vec{z}$ de l'eau, appliquée sur l'ensemble des surfaces.

```
def resultante(L):
    Resultante=0 #on initialise la résultantes
    #on parcourt les points de la liste et on additionne la composante sur z
    for i in range(len(L)):
        Resultante+=force_facette(L[i])[2]
    return Resultante
```

III.2 - Tri des facettes

code Python

```
1 def fusion(L1, L2):
2     # À compléter (sur une ou plusieurs lignes)
3
4 def trier_facettes(L):
5     # À compléter (sur une ou plusieurs lignes)
6
7 grandesFacettes = # À compléter
```

Question 24 Compléter la fonction fusion, prenant comme argument deux listes de facettes L1 et L2 (supposée chacune triée par aire décroissante) et renvoyant une nouvelle liste composée des facettes de L1 et L2 triées par aire décroissante.

```
def fusion(L1,L2):
    if len(L1)<1:return L2
    if len(L2)<1:return L1
    if aire(L1[0])<aire(L2[0]) :
        return [L2[0]]+fusion(L1,L2[1:])
    return [L1[0]]+fusion(L1[1:],L2)
```

Question 25 Compléter la fonction récursive **trier_facettes**, prenant comme argument une liste de facettes **L**, et renvoyant une nouvelle liste de facettes triées dans l'ordre des aires décroissantes, par la méthode du tri-fusion.

```
def trier_facette(L):
    if len(L)<=1 : return L
    L1=L[:len(L)//2]
    L2=L[len(L)//2:]
    return fusion(trier_facette(L1),trier_facette(L2))
```

Question 26 Affecter à une nouvelle variable grandesFacettes la liste des facettes de maillageG, privée de la moitié des facettes les plus petites (en cas de nombre impair d'eléments, on inclura la facette médiane).

grandesFacettes =trier_facette(maillageG) [:(len(maillageG)+1)//2]

III.3 - Mouvement vertical de la gondole

code Python

```
1 def nouvelle_hauteur(posG, vitG, mailG):
2     dt=1.0/25.0 # Pas de temps correspondant à une image du film.
3     facettes_immergees = lister_FI(mailG)
4     posG = posG + .........     # à compléter
5     vitG = vitG + ..........    # à compléter
6     return posG, vitG
```

Question 27 Compléter les lignes 4 et 5 du code précédent conformément à la méthode d'Euler.

On supposera que m et g sont des variables globales déjà définies.

4 posG=posG+vitG*dt
5 vitG=vitG+(résultante(facette_immergees)/m-g)*dt