Correction DS 2

Librement adapté de Centrale-Supelec 2017

Le sujet est globalement mal posé... les questions franchement pas toujours très claires.

Le sujet manipule la valeur `NULL` pour représenter un champ vide ce qui hors-programme en CPGE. Il faut donc faire au mieux avec ce qu'on sait faire en SQL.

La condition`NULL = NULL` teste si un champ non rempli vaut un autre champ non rempli. Il est difficile d'attribuer une valeur true ou false à cette condition, c'est pourquoi SQL a trois valeurs de vérité, true, false et unknown (Cf la norme SQL, partie II, section Boolean types). La condition `NULL = NULL` renvoie unknown. C'est pourquoi sont mis à disposition `IS NULL` et `IS NOT NULL` qui permettent de vérifier si une valeur vaut `NULL` .

Comme la notion est hors-programme, les réponses de la forme `X=NULL` seront évidemment acceptées.

1. Pour qu'une exploration soit en cours il faut d'après l'énoncé que `EX_DEB` ne soit pas `NULL` et `EX_FIN` soit `NULL`. On obtient la syntaxe suivante

```
SELECT EX_NUM FROM EXPLO WHERE EX_DEB IS NOT NULL AND EX_FIN IS NULL
```

   La syntaxe suivante sera acceptée même si elle n'est pas totalement rigoureuse en SQL

```
SELECT EX_NUM FROM EXPLO WHERE EX_DEB IS NOT NULL AND EX_FIN IS NULL
```

2. Soit `X` le numéro de l'exploration en cours. La liste des points d'intérêt s'obtient grâce à la table PI dont le `EX_NUM` correspond à celui de l'exploration `X` On propose la syntaxe suivante :

```
SELECT PI_NUM , PI_X, PI_Y FROM PI  WHERE EX_NUM = X
```

3. En mètre carré ! Zut les données sont en millimètres ! Attention aux conversions. Pour obtenir la zone déjà explorée il faut regarder les abscisses et les ordonnées minimale et maximale des PI déjà visité, c'est-à-dire ceux pour lesquels `PI_ARR` n'est pas `NULL`.

   On propose la syntaxe suivante :

```
SELECT EX_NUM, (MAX(PI_X)-MIN(PI_X))*(MAX(PI_Y) -MIN(PI_Y))*10**(-6)
FROM  PI JOIN EXPLO ON PI.EX_NUM=EXPLO.EX_NUM
WHERE PI_ARR IS NOT NULL
GROUP BY EXPLO.EX_NUM
```

4. Question théorique où il faut se demander quel est la taille maximale des entiers stockables. [1] Une exploration est incluse dans le carré de sommets : $(0,0),(0,max\_int),(max\_int,0)$, et (max_int,max_int), ca fait une surface maximale de $max\_int^2 * 10^{-6} m^2$ (on n'oublie pas que les unités sont pas les unites SI dans le sujet...) Si on stocke des entiers en 32-bits (ce qui semble classique en SQL pour le type int) on obtient $\max_i nt = 2^{31} - 1$, soit une surface maximale d'environ

$$3.4 \times 10^{32} m^2$$

   à comparer au $1.4 \times 10^{14} m^2$ de la planéte Mars...

5. Beaucoup de solutions avec des jointures différentes étaient possibles. Je propose :

```
SELECT IN_NUM, COUNT(*), SUM(IT_DUR)
FROM  EXPLO  JOIN ANALY ON EXPLO.EX_NUM = ANALY.EX_NUM
JOIN INTYP ON ANALY.TY_NUM = INTYP.TY_NUM
GROUP BY IN_NUM HAVING EX_DEB IS NOT NULL AND EX_FIN IS NULL
```

1. (Je suis pas complétement sûr que ca soit vraiment la réponse attendue, je suppute que n'importe quelle réponse pas trop délirante était valorisée.)

```
# correction ds dictionnaire.py

001| import math
002| import numpy as np
003| import random
004|
005| n=10
006| cmax=1000
007|
008|
009|
010| ### 1
011| def generer_PI(n,cmax):
012|     ''' n : entier - nombre de points à générer
013|         cmax : entier - coordonnées maximales autorisées
014|
015|         retourne un dictionnaire avec n entrées dont les valeurs
sont les coordonnées de n points distincts'''
016|
017|     d={}
018|     i=0
019|     while len(d)<n:
020|         x=random.randint(0,cmax) #randint inclus 0 et cmax
021|         y=random.randint(0,cmax)
022|         coordonnees=[x,y]   # crée des coordonnées aléatoires
023|         if coordonnees not in d.values():
024|             d[i]=coordonnees # enregistre les coordonées
différentes des précédentes.
025|             i+=1
026|     return(d)
027|
028| PI=generer_PI(n,cmax)
029| print(PI)
030| print('Il y a ', n, 'points d interet')
031| print('Ils sont compris dans le carré [0,100]^2')
032|
033| ### 2
034| '''
035| Il y a (cmax+1)^2 points à coordonnées entières distincts dans
le carré de la zone d'exploration. Les arguments doivent donc
vérifier :
036|                 n<= (cmax+1)^2
037|
038| '''
039|
040| ### 3
041|
042|
043|
044| def calculer_distances(PI):
045|     '''  PI : dictionnaire des points d'intérets de la forme
(clé,valeur):(numero du PI, coordonnées du PI)
046|
047|         retourne un dictionnaire de la forme
```

```
048|         (clé, valeur) : (numero du PI, liste des distance de la
clé aux autres PI) '''
049|
050|     d={}  #Crée un dictionnaire vide
051|     keys=PI.keys()
052|     for c1 in keys:
053|         X_1,Y_1=PI[c1] #Coordonnées du premier PI
054|         d_c1=[] #liste des distance de c1 aux autres PI
055|         for c2 in keys:
056|             X_2,Y_2=PI[c2] #Coordonnées du second PI
057|             d_c1 += [math.sqrt( (X_1-X_2)**2 +(Y_1-Y_2)**2)]
#ajoute la distance de c1 à c2
058|         d[c1]=d_c1 #ajoute la liste des distance de c1 aux
autres PI aux dictionnaire des distances.
059|     return(d)
060|
061|
062|
063| dist=calculer_distances(PI)
064|
065| # print(dist)
066| ### 4
067|
068| def longueur_chemin(chemin,d):
069|     '''
070|     chemin : list - suite des PI commencant par 0 et incluant
une et une seule fois tous les PI.
071|     d : dictionnaire des distances comme donné par
calculer_distances
072|
073|     retourne la longueur du chemin d
074|     '''
075|     distance=0
076|     position=chemin[0] #  position initiale, supposée égale à 0
077|     for prochain_pi in chemin[1:]:
078|         distance+= d[position][prochain_pi]
079|         position=prochain_pi # On actualise la position du
bureau.
080|
081|     return(distance)
082|
083|
084|
085|
086| ### 5
087| ''' Un chemin valide commence par 0 puis on a le choix entre
(n-1) PI, puis (n-2) PI et ainsi de suite. Au final il y a (n-1)!
chemins valides.
088| '''
089| ### 6
090| f=1
091| for i in range(1,20):
092|     f=f*i
093| # print('factiorel 19', f)
```

```
094| ''' 19!=121645100408832000 = 1.2 *10^17 C'est trés grand !
095| Si on suppose que pour calculer la longueur d'un chemin il faut
1 nano seconde (ce qui parait etre fort sous évalué), il faudra alors
10^8 secondes ce qui donne à peu prés 3.8 années...
096|  '''
097|
098| ### 7
099|
100| def plus_proche_voisin(d):
101|     c=[0] # on initialise le chemin à la position initiale du
robot, en 0.
102|     while len(c)<len(d):
103|
104|         position_du_robot=c[-1]
105|         non_visite=[]
106|         for pi in d.keys():
107|             if pi not in c:
108|                 non_visite+=[pi]
109|
110|
111|         plus_proche=non_visite[0]
112|         distance_plus_proche=d[position_du_robot][plus_proche]
113|         for pi in non_visite[1:]:
114|
115|             if distance_plus_proche>d[position_du_robot][pi]:
116|                 plus_proche=pi
117|                 distance_plus_proche=d[position_du_robot][pi]
118|
119|         c+=[plus_proche]
120|     return(c)
121|
122|
123|
124|
125| print('chemin obtenu avec l algorithme du plus proche voisin',
plus_proche_voisin(dist))
126|
127|
128| ### 8
129|
130| ''' La fonction calculer_distance utilise deux boucles
impbriquées de taille n, sa complexité est en O(n^2)
131| La fonction plus proche
132|
133| '''
134| ### 9
135| ''' {0 :[3,0], 1:[4,0], 2:[0,0], 3:[9,0]}
136| l'algorithme du proche voisin retourne [0,1,2,3] de longueur
1+4+9=14
137| Or le chemin [0,2,1,3] est de longueur 3+4+5=12.
138| Ainsi l'algorithme du plus proche voisin ne retourne pas le
meilleur chemin.
139| '''
140|
```

```
141|
142| ### 10
143| ''' On peut permuter 2 PI parmi les n-1 points , il y a donc
(n-1)(n-2)/2 chemins voisins'''
144|
145| ### 11
146|
147| def permutation_chemin(c):
148|     voisins=[]
149|     for i in range(1,len(c)):
150|         for j in range(i+1,len(c)):
151|             v=c[:i]+c[j:j+1]+c[i+1:j]+c[i:i+1]+c[j+1:]
152|             voisins+=[v]
153|     return(voisins)
154|
155|
156| ### 12
157| def meilleur_voisins(voisins,d):
158|     lmin=longueur_chemin(voisins[0],d)
159|     chemin_minimal=voisins[0]
160|     for chemin in voisins[1:]:
161|         ltest=longueur_chemin(chemin,d)
162|         if lmin>ltest:
163|             chemin_minimal=chemin
164|             lmin=ltest
165|     return(chemin_minimal)
166|
167| ### 13
168| def descente_locale(d):
169|     c_min=plus_proche_voisin(d)
170|
171|     l_min=longueur_chemin(c_min,d)
172|     voisins=permutation_chemin(c_min)
173|     c_test=meilleur_voisins(voisins,d)
174|     l_test=longueur_chemin(c_test,d)
175|     while l_min>l_test:
176|         c_min=c_test
177|         l_min=longueur_chemin(c_min,d)
178|         voisins=permutation_chemin(c_min)
179|         c_test=meilleur_voisins(voisins,d)
180|         l_test=longueur_chemin(c_test,d)
181|
182|     return(c_min)
183|
184|
185|
186| descente_locale(dist)
187|
188|
189|
190|
191|
192|
193|
```

```
194|
195|
196|
```