CCinP ITC 2026 - Correction

Q1 Un pixel prend au minimum 8 bits (1 octet). Il y a 3049472 pixels (de tête !) ce qui donne autant d'octets et donc environ 3 Mo.

Q2 On a besoin de 10 flottants par nm, et il y 540 nm en tout. Donc on doit avoir 5400 flottants qui prennent chacun 8 octets, ce qui donne 43200 octets et environ 43 Ko.

Q7 Par exemple :

```
def distance(S1, S2):
    s = 0
    N = len(S1)
    for i in range(N):
        s = s + (S1[i] - S2[i])**2
    return (S/N)**(1/2)
```

Q8 Par exemple :

```
def positions_a_rejeter(S1, S2):
    N = len(S1)
    Ind = []
    for i in range(N):
        if S1[i] == None or S2[i] == None:
            Ind.append(i)
    return Ind
```

Q9 Par exemple :

```
def est_absent(element, L):
    for i in range(len(L)):
        if L[i] == element:
            return False
    return True
```

Dans le pire des cas, `element` n'est pas dans `L` et on doit parcourir toute la liste. La complexité est linéaire en r : $O(r)$.

Q10 Par exemple : `d` contient la valeur $\sum_{\substack{k=0 \\ \text{k n'est pas rejeté}}}^{i-1} (S_{1,i} - S_{2,i})^2$.

Q11 Pour `distance2` : on répète N fois la boucle `for` et à chaque itération, on fait un appel en $O(r)$ à `est_absent`. La complexité est donc $O(Nr)$.
Pour `distance3` : on fait deux copies en $O(N)$ puis une boucle en $O(r)$ puis une boucle en $O(N)$. Comme $r \leq N$, on a une complexité en $O(N)$.

Q23 Depuis D, l'arrête la plus légère mène à C (poids égal à 1).
A partir de D,C, l'arrête la plus légère est celle de C à A (poids égal à 2, poids total égal à 3).
A partir de D,C,A, l'arrête la plus légère est celle de A à B (poids égal à 1, poids total égal à 4).
A partir de D,C,A,B, l'arrête la plus légère est celle de C à G (poids égal à 3, poids total égal à 7).
A partir de D,C,A,B,G, l'arrête la plus légère est celle de A à F (poids égal à 4, poids total égal à 11).
A partir de D,C,A,B,G,F, l'arrête la plus légère est celle de F à E (poids égal à 5, poids total égal à 16).
L'arbre final contient donc les arrêtes DC, CA, AB, CG, AF et FE, et la somme de ses pondérations vaut 16.

Q24 Depuis F, l'arrête la plus légère mène à A (poids égal à 4).
A partir de F,A, l'arrête la plus légère est celle de A à B (poids égal à 1, poids total égal à 5).
A partir de F,A,B, l'arrête la plus légère est celle de A à C (poids égal à 2, poids total égal à 7).
A partir de F,A,B,C, l'arrête la plus légère est celle de C à D (poids égal à 1, poids total égal à 8).
A partir de F,A,B,C,D, l'arrête la plus légère est celle de C à G (poids égal à 3, poids total égal à 11).
A partir de F,A,B,C,D,G, l'arrête la plus légère est celle de F à E (poids égal à 5, poids total égal à 16).
L'arbre final contient donc les arrêtes FA, AB, AC, CD, CG et GF, et la somme de ses pondérations vaut 16.
On retrouve le même poids, mais l'arbre n'est pas le même (ce ne sont pas les mêmes arrêtes qui ont été choisies).

Q25 Par exemple :

```
def poids(coords, i, j):
    return (coords[i][0] - coords[j][0])**2 + (coords[i][2] - coords[j][1])**2 + (coords[i][2] - c
```

Q26 Par exemple :

```
def matrice_adjacence(coords):
    M = []
    N = len(coords)
    for i in range(N):
        L = [poids(coords, i, j) for j in range(N)]
        M.append(L)
    return M
```

Q27 Par exemple :

```
def initialisation_distance(G, depart):
    N = len(G)
    d = {i : float('inf') for i in range(N)}
    d[depart] = 0
    return d
```

Q28 On a :

```
if dist[gal] < mindist:
    mindist = dist[gal]
    candidat = gal
```

Q29 `len(dist)` est un variant de boucle : c'est un entier qui décroît strictement à chaque itération à cause du `del` et il doit rester positif ou nul.

Q30
- L'appel à `initialisation_distance` est en $O(N)$.
- La longueur de `dist` diminue de 1 à chaque itération donc il y a N itérations.
- L'appel à `recherche_distance_minimal` est en $O(\texttt{len(dist)}) = O(N-i)$ (où i est le numéro de l'itération).
- L'appel à `mise_a_jour` est en $O(N)$.

Donc la complexité est $O(N^2)$.

Q31 Par exemple :

```
def dico\_galaxies\_avec\_successeurs(arbre):
    d = {}
    for gal in arbre:
        pred = arbre[gal]
        if pred in d:
            d[pred] = d[pred] + 1
        else:
            d[pred] = 1
    return d
```

Q32 A chaque étape de la boucle, la fonction récupère l'inventaire des prédecesseurs et de leur nombre de successeurs, et reconstruit un nouvel arbre dans lequel elle retire les sommets n'ayant aucun successeur, donc qui sont au bout d'une branche de l'arbre. Cette étape est répété un nombre donné (`nb_etapes`) de fois.

Si `nb_etapes` n'est pas trop grand, on retire les "branches courtes" de l'arbre, ce qui réalise bien le travail demandé. Les "branches longues" sont également un peu raccourcies, mais si `nb_etapes` est bien choisi, on peut supposer que ce raccourcissement est négligeable devant la taille des filaments de ces branches et on conservera la structure principale de l'arbre.

Q33 Par exemple :

```
def separation(G, arbre, alpha):
    lm = 0
    for gal in arbre:
        pred = arbre[gal]
        lm = lm + G[pred][gal]
    lm = lm/len(arbre)
    d = {}
```

```
    for gal in arbre:
        pred = arbre[gal]
        if G[pred][gal] <= alpha*lm:
            d[gal] = pred
    return d
```