COMPOSITION D'ITC n°3

Corrigé

⋆ ⋆ ⋆

Question 1 La paire `(inId, phId)` constitue la clé primaire de la table `Présent` : chacun de ces deux attributs pris isolément n'est pas une clé primaire. Il est donc possible d'avoir deux enregistrements avec la même valeur pour l'attribut `inId`, donc la même personne présente sur deux photos. De même plusieurs personnes différentes peuvent apparaitre sur une même photo.

Question 2 On a :

```
SELECT phId FROM Photos
WHERE phLarg / phHaut = 16 / 9
```

Question 3 On propose :

```
SELECT inNom FROM Individus AS in
JOIN Présent AS pr ON pr.inId = in.inId
JOIN Photos AS ph ON ph.phId = pr.phId
WHERE phAuteur = in.inId
```

Question 4 On propose :

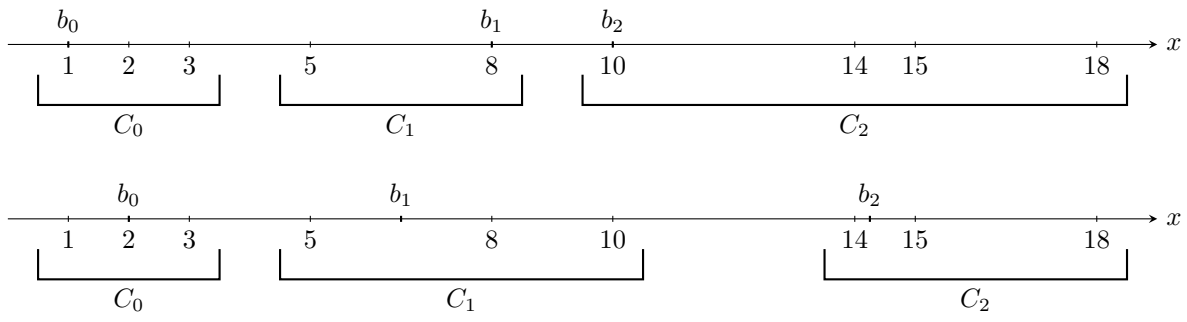
```
SELECT phId FROM Photos
WHERE phId NOT IN
   (SELECT phId FROM Présent)
```

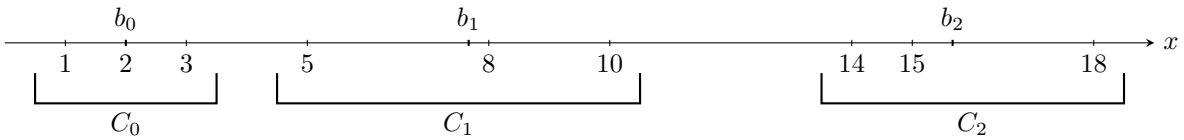
Question 5 On obtient :

```
SELECT phFichier FROM Photos AS ph
JOIN Description AS de ON ph.phId = de.phId
GROUP BY ph.phId
HAVING COUNT(motCle) >= 5
```

Question 6 Si $K = N$, chaque classe d'équivalence est de cardinal 1 et le score est nul, ce qui est bien minimal. Si $K = N - 1$, toutes les classes sont de cardinal 1, sauf une qui est de cardinal 2. Pour minimiser le score, il faut mettre dans la même classe les deux éléments les plus proches de E.

Question 7 On a l'exécution suivante :





Question 8 Les trois erreurs sont :

- ligne 5 : il faut vérifier que ni `i1`, ni `i2` n'a atteint la dernière valeur. On doit remplacer la condition booléenne par `if i2 == n2 or (i1 < n1 and L1[i1] <= L2[i2]):`
- entre la ligne 9 et 10 : il faut penser à incrémenter `i2` en rajoutant `i2 += 1`
- ligne 14 : le cas d'arrêt doit aussi prendre en compte le cas de la liste à un seul élément, sinon l'algorithme ne termine pas. Il faut remplacer cette ligne par `if n <= 1:`

Question 9 Il suffit juste de faire attention aux indices.

```
def moyenne(E, i, j):
    mu = 0
    for k in range(i, j):
        mu += E[k]
    return mu / (j - i)
```

Question 10 Le code ressemble au précédent en faisant d'abord le calcul de la moyenne.

```
def somme_emc(E, i, j):
    S = 0
    mu = moyenne(E, i, j)
    for k in range(i, j):
        S += (E[k] - mu) ** 2
    return S
```

Question 11 La liste `P` associée à la partition $\mathcal{P} = \{\{0,1,2\},\{3,4\},\{5,6,7\},\{8\}\}$ est `P = [0, 3, 5, 8]`. La partition associée à la liste `P = [0, 1, 4, 5]` est $\mathcal{P} = \{\{0\},\{1,2,3\},\{4\},\{5,6,7,8\}\}$.

Question 12 Il faut ici faire attention à la manière dont sont délimitées les classes d'équivalence. On traite la dernière classe à part, qui contient les valeurs jusqu'à x_{N-1}.

```
def score(E, P):
    SP = somme_emc(E, P[len(P) - 1], len(E))
    for i in range(len(P) - 1):
        SP += somme_emc(E, P[i], P[i + 1])
    return SP
```

Question 13 Il suffit d'une double boucle pour déterminer les indices des plus petits éléments de C_1 et C_2. On calcule le score à chaque étape et on garde la partition de score minimal.

```
def clustering3(E):
    N = len(E)
    Pmin = [0, 1, 2]
    for i in range(1, N - 1):
        for j in range(i + 1, N):
            P = [0, i, j]
            if score(E, P) < score(E, Pmin):
                Pmin = P
    return Pmin
```

Question 14 On donne les complexités des fonctions précédentes :

- `moyenne(E, i, j)` est en $\mathcal{O}(j-i)$, au même titre que `somme_emc(E, i, j)` ;
- `score(E, P)` est en $\mathcal{O}(N)$, car on calcule `somme_emc` pour chaque classe, dont la somme des cardinaux est N ;
- on en déduit que `clustering3(E)` est en $\mathcal{O}(N^3)$, car on fait le calcul d'un score de l'ordre de $\mathcal{O}(N^2)$ fois.

Question 15 On commence par calculer la liste des moyennes des éléments des classes (toujours en faisant attention à traiter la dernière classe à part), puis on compare les différences entre deux moyennes consécutives en gardant l'indice qui permet d'atteindre le minimum.

```
def classes_plus_proches(E, P):
    N, K = len(E), len(P)
    Lmu = [0] * K
    for i in range(K - 1):
        Lmu[i] = moyenne(E, P[i], P[i + 1])
    Lmu[K - 1] = moyenne(E, P[K - 1], N)
    iopt = 0
    for i in range(K - 1):
        if Lmu[i + 1] - Lmu[i] < Lmu[iopt + 1] - Lmu[iopt]:
            iopt = i
    return iopt
```

Question 16 On crée une nouvelle liste de taille $K-1$, qu'on remplit en fonction de `iopt`. Il s'agit juste de supprimer l'élément d'indice `iopt + 1` (mais on crée ici une nouvelle liste pour ne pas modifier l'ancienne).

```
def fusion_classes(P, iopt):
    K = len(P)
    nouv_P = [0] * (K - 1)
    for i in range(K - 1):
        if i <= iopt:
            nouv_P[i] = P[i]
        else:
            nouv_P[i] = P[i + 1]
    return nouv_P
```

Question 17 On se contente d'appliquer l'algorithme décrit précédemment avec les fonctions déjà écrites.

```
def CHA(E, K):
    N = len(P)
    P = [i for i in range(N)]
    while len(P) > K:
        iopt = classes_plus_proches(E, P)
        P = fusion_classes(P, iopt)
    return P
```

Question 18 Lorsque $k=1$, $D(n,k) = S_{\text{emc}}(0,n)$ (il n'y a qu'une seule classe). Ce score peut être calculé en $\mathcal{O}(n)$.

Question 19 Une partition des n éléments de taille k consiste en une partition de $i < n$ éléments en $k-1$ classes, à laquelle on rajoute une classe formée des $n-i$ derniers éléments. Comme aucune classe ne doit être vide, on considère $i \geqslant k-1$. La partition optimale atteint le minimum parmi toutes les partitions possibles de cette forme, ce qui donne bien la formule voulue.

Question 20 On écrit une fonction auxiliaire `D(E, n, k, dico)` qui prend en argument l'ensemble E, des entiers n et k et un dictionnaire mémoïsant les résultats et renvoie $D(n, k)$. L'initialisation et l'hérédité se font selon les deux questions précédentes. Une fois cette fonction écrite, il suffit de lancer un appel avec $n = N$ et $k = K$, en utilisant un dictionnaire vide.

```
def D(E, n, k, dico):
    if (n, k) not in dico:
        if k == 1:
            dico[(n, k)] = somme_emc(E, 0, n)
        else:
            dico[(n, k)] = D(E, k - 1, k - 1, dico) + somme_emc(E, k - 1, n)
            for i in range(k, n):
                d = D(E, i, k - 1, dico) + somme_emc(E, i, n)
                dico[(n, k)] = min(dico[(n, k)], d)
    return dico[(n, k)]

def clustering_dynamique(E, K):
    return D(E, len(E), K, {})
```

Question 21 On remarque qu'il y a de l'ordre de $N \times K$ valeurs qui sont calculées dans le dictionnaire. De plus, chaque valeur nécessaire de calculer le minimum par la boucle `for`. Cette boucle, de taille $n - k$, fait un appel à `somme_emc`, de complexité $\mathcal{O}(n - i)$. En combinant tout ça, on obtient une complexité totale en $\mathcal{O}(K \times N^3)$.

Question 22 Dans le calcul du minimum, on peut garder en mémoire l'indice `i` qui permet d'atteindre ce minimum, ce qui correspond au plus petit élément de la classe C_{k-1} dans une partition de taille k. On peut alors reconstruire une solution complète en utilisant les valeurs présentes dans le dictionnaire.

Question 23 On remarque les formules suivantes :

- $\mu(i, i+1) = x_i$;
- $\mu(i, n+1) = \dfrac{(n-i)\mu(i,n) + x_n}{n+1-i}$.

Ces formules peuvent être utilisées pour calculer tous les $\mu(i, n)$, $i < n$, en temps $\mathcal{O}(N^2)$ au total. Dès lors, on remarque, en adaptant la formule admise que :

- $S_{\text{emc}}(i, i+1) = 0$;
- $S_{\text{emc}}(i, n+1) = S_{\text{emc}}(i, n) + \frac{n-i}{n+1-i}(x_n - \mu(i, n))^2$.

À nouveau, on peut utiliser ces formules pour calculer les $S_{\text{emc}}(i, n)$ en temps $\mathcal{O}(N^2)$ au total.

Question 24 On fait d'abord une fonction pour calculer les moyennes, puis sur le même schéma une fonction pour calculer les S_{emc}.

```
def calcul_mu(E):
    N = len(E)
    mu = {}
    for i in range(N):
        mu[(i, i + 1)] = E[i]
        for n in range(i + 1, N):
            mu[(i, n + 1)] = ((n - i) * mu[(i, n)] + E[n]) / (n + 1 - i)
    return mu
```

```
def calcul_Semc(E):
    N = len(E)
    mu = calcul_mu(E)
    Semc = {}
    for i in range(N):
        Semc[(i, i + 1)] = 0
        for n in range(i + 1, N):
            Semc[(i, n + 1)] = Semc[(i, n)] +
                    (n - i) / (n + 1 - i) * (E[n] - mu[(i, n)]) ** 2
    return Semc
```

⋆ ⋆ ⋆