Épreuve d'informatique

Durée : 2h

Calculatrices interdites

Le sujet se compose de deux exercices indépendants. Si, au cours de l'épreuve, un-e candidat-e repère ce qui lui semble être une erreur d'énoncé, il/elle le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il/elle est amené-e à prendre.

1 Machines à café connectées

Un fabricant de machines à café connectées a mis en place un système de notation. Après avoir paramétré sa machine et préparé un café, chaque client a la possibilité de renseigner une note permettant d'évaluer la qualité du café produit.

Le fabricant dispose ainsi d'une base de données constituée de deux tables :

- la table `clients`, comportant les détails relatifs à chaque client, avec pour colonnes
  - `id` : un identifiant unique à chaque client ;
  - `nom` : le nom du client ;
  - `email` : l'adresse mail du client, et
- la table `retours`, dont chaque ligne correspond à un retour d'expérience sur une tasse de café, avec pour colonnes
  - `id_client` : l'identifiant du client ayant rentré une évaluation ;
  - `note` : un entier compris entre 0 et 5 ;
  - `masse_café` : la masse de café moulu par la machine, en grammes ;
  - `masse_eau` : la masse d'eau utilisée pour préparer le café, en grammes ;
  - `granulométrie` : la taille des particules une fois le café moulu, le diamètre d'une particule étant supposé compris entre 0 et 1 mm.

1.1 Requêtes sur la base de données

Le fabricant souhaite mener une étude sur les pratiques de ses clients afin de réaliser une enquête de satisfaction.

Les requêtes de cette partie devront êtres rédigées en langage SQL.

Q. 1 Écrire une requête permettant d'obtenir le nombre total de clients.

Q. 2 Écrire une requête permettant d'obtenir la liste des notes attribuées par le client dont le nom est "Potter" (on pourra considérer que c'est le seul client qui porte ce nom)

Q. 3 Écrire une requête permettant d'obtenir, pour chaque client ayant rentré au moins une évaluation, la moyenne des évaluations que ce client a rentrées.

Q. 4 Compléter la requête précédente pour ne conserver que les lignes correspondant aux clients ayant rentré des notes dont la moyenne soit inférieure ou égale à 2.

Q. 5 Compléter la requête précédente pour obtenir les noms et les emails des clients concernés.

1.2 Interpolation des données par la méthode des k plus proches voisins

À partir de la base de données, on peut construire un nuage de points dans $[0,1] \times [0,1]$, chaque point ayant une abscisse correspondant à une granulométrie, une ordonnée correspondant au rapport entre la masse de café et la masse d'eau utilisées, et une étiquette correspondant à la note rentrée par le client ayant choisi les réglages.

Le fabricant souhaite alors construire une fonction de prédiction `note_prédite(g: float, r: float) -> int` qui, pour une granulométrie et un rapport de masses donnés, n'ayant a priori pas été déjà évalués par les clients, prédise la note que les clients sont susceptibles d'attribuer au réglage concerné, l'idée étant bien sûr d'une part d'améliorer la machine, et d'autre part de suggérer aux clients les meilleurs réglages possibles.

Pour ce faire, l'algorithme retenu est celui des k plus proches voisins : k étant un entier strictement positif choisi, la note prédite pour un point non nécessairement étiqueté $p = (g, r)$ sera la note majoritaire parmi les k points étiquetés les plus proches de p.

Les instructions de cette partie devront être rédigées en langage Python.

Q. 6 Écrire une fonction `dist(p: tuple, q: tuple) -> float` qui, étant donné deux points p et q du plan identifiés à des couples de coordonnées, renvoie la distance euclidienne entre ces deux points.

Q. 7 Écrire une fonction `plus_proche(p: tuple, l: list) -> tuple` qui, étant donné un point p du plan et une liste ℓ de points du plan, renvoie un point élément de ℓ qui soit le plus proche possible de p au sens de la distance euclidienne.

On suppose que le nuage de points est représenté par une variable `note`, de type dictionnaire, dont les clés sont les points `p = (g, r)` identifiés à des couples de coordonnées, et les valeurs les notes attribuées par les clients aux réglages correspondants.

Q. 8 Comment extraire du dictionnaire `note` la liste des points du nuage ?

Q. 9 Écrire une fonction `note_prédite_1NN(g: float, r: float, note: dict) -> int` faisant intervenir les fonctions précédentes et mettant en œuvre la méthode des k plus proches voisins avec $k = 1$.

Q. 10 Écrire une fonction `plus_proches(p: tuple, l: list, k: int) -> list` qui, étant donné un point p du plan, une liste ℓ de points du plan, et un entier strictement positif k, renvoie une liste de k points, éléments de ℓ, qui soient les plus proche possibles de p au sens de la distance euclidienne. On précisera, sans nécessairement la justifier en détail, la complexité asymptotique de la fonction choisie, assimilée au nombre de comparaisons effectuées.

Q. 11 Écrire une fonction `note_prédite_kNN(g: float, r: float, note: dict, k: int) -> int` faisant intervenir les fonctions précédentes et mettant en œuvre la méthode des k plus proches voisins avec k arbitraire.

2 Escaliers de poids minimal

On considère un graphe non orienté $G = (S, A)$ dont les sommets sont disposés sur une grille carrée formée de n lignes et n colonnes. Chaque sommet est donc repéré par un couple d'entiers (i, j) avec $0 \leq i < n$ et $0 \leq j < n$. Dans ce graphe, tous les sommets qui sont séparés par une distance euclidienne de 1 sont reliés par une arête; ainsi un sommet (i, j) possède au plus 4 voisins : un vers le haut $(i + 1, j)$, un vers le bas $(i - 1, j)$, un vers la droite $(i, j + 1)$ et un vers la gauche $(i, j - 1)$ (sous réserve d'existence de ces couples). De plus chaque arête $((i, j), (a, b))$ possède un poids qui est un **entier positif** noté $P(i, j, a, b)$. Le graphe étant non orienté, on considère que les poids vérifient la condition $P(i, j, a, b) = P(a, b, i, j)$. Dans l'exemple ci-dessous $P(1, 3, 2, 3) = 5$.

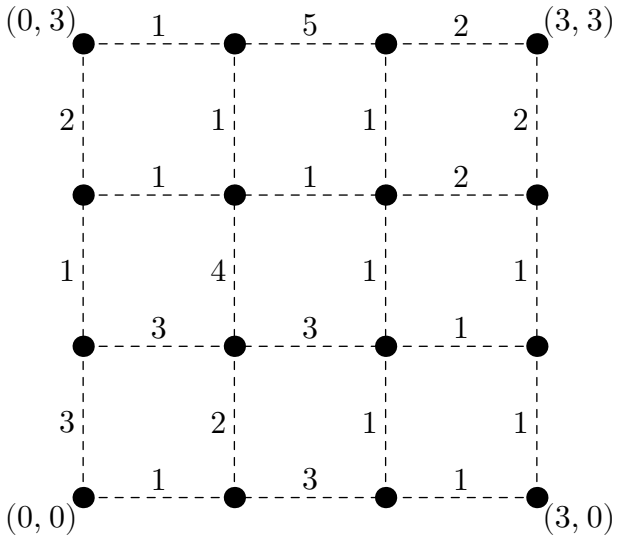

Figure 1 Un exemple de graphe G pour $n = 4$

Un *chemin* de longueur $p \in \mathbb{N}$ dans le graphe G est une liste de sommets $[(i_0, j_0), \ldots, (i_p, j_p)]$ tel que deux sommets consécutifs de la liste sont reliés par une arête dans G, c'est-à-dire que $\forall k \in [\![0, p - 1]\!]$, $((i_k, j_k), (i_{k+1}, j_{k+1})) \in A$. Le sommet (i_0, j_0) est appelé *origine* du chemin et le sommet (i_p, j_p) est la *destination* du chemin. On appelle *poids* du chemin la somme des poids des arêtes qui le constitue.

On appelle *escalier* un chemin dans G d'origine $(0, 0)$ qui n'utilise que des déplacements d'une unité vers le haut $(+1, 0)$, ou d'une unité vers la droite $(0, +1)$. La figure suivante donne un exemple d'escalier menant au sommet $(3, 2)$.

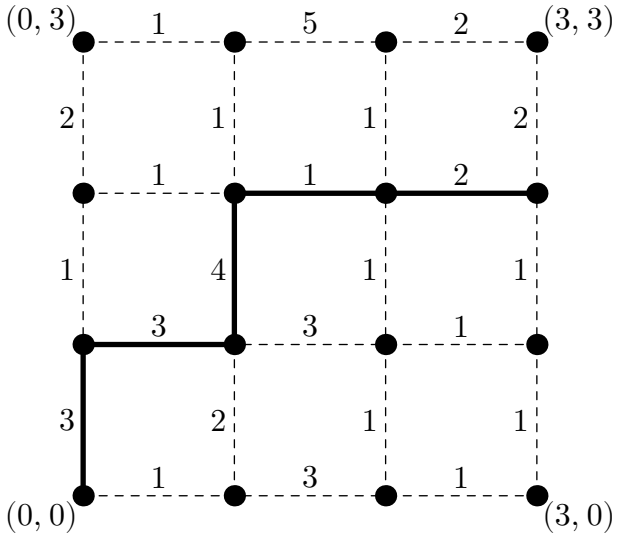

Figure 2 Un escalier menant à $(3, 2)$ de poids 13

Dans ce problème, on cherche à déterminer un escalier de poids minimal menant au sommet $(n-1, n-1)$. Pour tout $0 \leq i < n$ et tout $0 \leq j < n$, on notera $F(i, j)$ le poids minimal d'un escalier menant à (i, j). On dira d'un tel escalier qu'il est *optimal*.

Q. 12 Sans justifier, donner les valeurs de $F(2, 3)$, $F(3, 2)$ et $F(3, 3)$ pour l'exemple de la Figure 1.

En Python, on suposera que les informations de poids du graphe G sont codées dans une variable globale `P` qui est un *dictionnaire* dont les *clefs* sont des quadruplets (i, j, a, b) et les *valeurs* le poids de l'arête $((i, j), (a, b))$. Ainsi, la valeur de $P(i, j, a, b)$ pourra être obtenue en écrivant `P[i, j, a, b]`.

Q. 13 Écrire une fonction `poids_chemin(c)` prenant en entrée un chemin dans G supposé valide et retournant en sortie le poids de ce chemin.

Q. 14 Justifier la formule de récurrence suivante :

$$\forall i > 0, \quad \forall j > 0, \quad F(i, j) = \min(F(i-1, j) + P(i-1, j, i, j),\ F(i, j-1) + P(i, j-1, i, j))$$

On pourra admettre cette formule de récurrence dans la suite, même si on n'est pas parvenu à la justifier.

Q. 15 Sans justifier, donner les expressions permettant de calculer $F(0, 0)$, $F(i, 0)$ pour $i > 0$ et $F(0, j)$ pour $j > 0$.

Q. 16 On considère une méthode récursive de calcul des $F(i, j)$ de type :

```
def F(i, j):
    if i = 0 and j = 0:
        return 0
    if i = 0 and j > 0:
        (...)
```

```
    if j = 0 and i > 0:
        (...)
    if i > 0 and j > 0:
        a = F(i-1, j) + P[i-1, j, i, j]
        b = F(i, j-1) + P[i, j-1, i, j]
        return min(a, b)
```

Expliquer pourquoi cette méthode de calcul des $F(i, j)$ n'est pas efficace.

Q. 17 Proposer une meilleure définition de la fonction `F` précédente en utilisant la *programmation dynamique descendante*, c'est-à-dire en utilisant le principe de *mémoïsation*.

Q. 18 On souhaite implémenter le calcul de $F(n-1, n-1)$ en utilisant la *programmation dynamique ascendante*. Compléter le programme suivant :

```
F = np.zeros(n, n)
for i in range(1, n):
    F[i, 0] = (...)
for j in range(1, n):
    F[0, j] = (...)
for i in range(1, n):
    for j in range(1, n):
        (...)
print("Le poids d'un escalier optimal est", F[n-1, n-1])
```

Q. 19 Donner la complexité en temps de ce programme en fonction de n.

Le programme obtenu jusqu'à présent permet d'obtenir le poids minimal d'un escalier mais ne nous permet pas de savoir quel chemin a été utilisé. Nous souhaitons donc *reconstruire* l'escalier optimal dont la valeur a été déterminée par programmation dynamique. Pour cela, on se propose de construire pendant l'algorithme de programmation dynamique une seconde matrice D telle que, pour tout $i > 0$ et tout $j > 0$:

- $D(i, j) = 1$ si l'escalier optimal calculé arrive en (i, j) depuis la gauche (c'est-à-dire que l'avant-dernier sommet est $(i-1, j)$)
- $D(i, j) = 2$ si l'escalier optimal calculé arrive en (i, j) depuis le bas (c'est-à-dire que l'avant-dernier sommet est $(i, j-1)$)

les valeurs pour de $D(0, j)$ et $D(i, 0)$ ne sont pas utiles pour la suite, on pourra les fixer à 0 par exemple.

Q. 20 Réécrire le programme de la Question 18 en ajoutant les instructions pour construire et remplir la matrice D.

Q. 21 Écrire une fonction `reconstruire(n, D)` qui à partir de la matrice D retourne une liste qui est un escalier optimal menant de $(0, 0)$ à $(n-1, n-1)$.