

## ÉCOLE POLYTECHNIQUE - ÉCOLES NORMALES SUPÉRIEURES

CONCOURS D'ADMISSION 2017

FILIÈRES MP, PC, PSI

## ÉPREUVE D'INFORMATIQUE

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.

Le langage de programmation sera **obligatoirement** Python.

**Intersection de deux ensembles de points**

Soit  $n$  un entier naturel. on note  $D_n$  l'ensemble des entiers naturels compris entre 0 et  $2^n - 1$ . On appelle « point de  $D_n \times D_n$  » tout couple d'entiers  $(x, y) \in D_n \times D_n$ . Soient  $P$  et  $Q$  deux parties de  $D_n \times D_n$ . On cherche à calculer efficacement l'intersection des ensembles de points  $P$  et  $Q$ . La résolution de ce problème a des applications en simulation numérique, en robotique ou encore dans l'implémentation d'interfaces utilisateurs.

Ce sujet est découpé en cinq parties. La partie I porte sur une solution naïve en PYTHON, la partie II sur une solution naïve en SQL. Les parties III, IV et V conduisent à la réalisation d'une solution efficace en PYTHON. La partie I et la partie II sont totalement indépendantes l'une de l'autre et du reste du sujet. Les parties suivantes ne sont pas indépendantes.

**Remarques préliminaires**

**Complexité.** La complexité, ou le temps d'exécution d'un programme  $P$  (fonction ou procédure) est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc.) nécessaires à l'exécution de  $P$ . Lorsque cette complexité dépend de plusieurs paramètres  $n$  et  $m$ , on dira que  $P$  a une complexité  $O(\phi(m, n))$ , lorsqu'il existe trois constantes  $A$ ,  $n_0$ , et  $m_0$  telles que la complexité de  $P$  soit inférieure ou égale à  $A \times \phi(m, n)$ , pour tout  $n > n_0$  et  $m > m_0$ . Lorsqu'il est demandé de garantir une certaine complexité, le candidat devra justifier cette dernière en raisonnant sur la structure du code.

**Rappels de PYTHON.** Si  $a$  est une liste alors  $a[i]$  désigne le  $i$ -ième élément de cette liste où l'entier  $i$  est supérieur ou égal à 0 et strictement plus petit que la longueur  $\text{len}(a)$  de la liste. La commande  $a[i]=e$  affecte la valeur de l'expression  $e$  au  $i$ -ième élément de la liste  $a$ . L'expression  $[]$  construit une liste vide. L'expression  $n * [k]$  construit une liste de longueur  $n$  contenant  $n$  occurrences de  $k$ . La commande  $a = \text{list}(b)$  construit une copie de la liste  $b$  et l'affecte à la variable  $a$ . La commande  $a.append(x)$  modifie la liste  $a$  en lui rajoutant un nouvel élément final contenant la valeur de  $x$ . **Important** : seules les opérations sur les listes apparaissant dans ce paragraphe sont autorisées dans les réponses. Si une fonction PYTHON standard

est nécessaire, elle devra être réécrite.

*Nous attacherons la plus grande importance à la lisibilité du code produit pas les candidats ; aussi, nous encourageons les candidats à utiliser des commentaires et à introduire des procédures ou des fonctions intermédiaires pour faciliter la compréhension du code.*

## Partie I. Une solution naïve en PYTHON

Pour commencer, un point de coordonnées  $(x, y) \in D_n \times D_n$  est représenté en PYTHON par une liste de deux entiers naturels  $[x, y]$ . Un ensemble de points est représenté par une liste de points sans répétition, donc comme une liste de listes d'entiers naturels de longueur 2.

**Question 1.** Écrire une fonction `membre(p, q)` qui renvoie `True` si le point `p` est dans l'ensemble représenté par la liste `q` et qui renvoie `False` dans le cas contraire.

**Question 2.** Écrire une fonction `intersection(p, q)` qui renvoie une liste représentant l'intersection des ensembles représentés par `p` et `q`. On implémentera l'algorithme qui consiste à itérer sur tous les points de `p` et à insérer dans le résultat ceux qui sont aussi dans `q`

**Question 3.** Si la comparaison entre deux entiers naturels est prise comme opération élémentaire, quelle est la complexité de l'algorithme de la question précédente exprimée en fonction de la longueur de `p` et `q` ?

## Partie II. Une solution naïve en SQL

On suppose maintenant que l'on représente les points du problème à l'aide d'une base de données. Cette base comporte deux tables. La tables `POINTS` contient trois colonnes :

- `id` (clé primaire) qui est un entier naturel unique représentant le point ;
- `x` qui est un entier naturel représentant son abscisse ;
- `y` qui est un entier naturel représentant son ordonnée.

On suppose qu'il n'existe pas deux points d'identifiants distincts et de mêmes coordonnées.

La relation d'appartenance à un ensemble de points est représentée par la table `MEMBRE` à deux colonnes :

- `idpoint`, un entier naturel qui identifie un point ;
- `idensemble`, un entier naturel qui identifie un ensemble de points.

**Question 4.** Écrire une requête SQL qui renvoie les identifiants des ensembles auxquels appartient le point de coordonnées  $(a, b)$ .

**Question 5.** Écrire une requête SQL qui renvoie les coordonnées des points qui appartiennent à l'intersection des ensembles d'identifiants  $i$  et  $j$ .

**Question 6.** Écrire une requête SQL qui renvoie les identifiants des points appartenant à au moins un des ensembles auxquels appartient le point de coordonnées  $(a, b)$ .

## Partie III. Codage de Lebesgue

On souhaite implémenter en PYTHON une solution efficace au problème de calcul de l'intersection entre deux ensembles de points. La solution proposée s'appuie sur une structure de données appelée AQL. Cette structure de données suppose que les coordonnées des points sont représentées par leur codage de Lebesgue.

Le codage de Lebesgue d'un point de coordonnées  $(x, y) \in D_n \times D_n$  s'obtient par entrelacement des bits des représentations binaires de  $x$  et  $y$  en commençant pas les bits de  $x$ . On suppose que les bits de poids fort sont situés à gauche dans les représentations binaires des entiers naturels.

Par exemple, si  $n = 3$ , si  $x$  vaut 6 (donc  $\overline{110}^2$  en binaire) et si  $y$  vaut 3 (donc  $\overline{011}^2$  en binaire) alors le codage de Lebesgue du point de coordonnées  $(x, y)$  est  $\overline{101101}^2$ , c'est-à-dire 45.

Le codage de Lebesgue d'un point peut être vu comme un nombre écrit dans une base formée des chiffres  $\overline{00}^2$ ,  $\overline{01}^2$ ,  $\overline{10}^2$  et  $\overline{11}^2$ . Ainsi, si  $n = 3$ , le point  $(6, 3) = (\overline{110}^2, \overline{011}^2)$  est codé par le nombre  $\overline{10}^2 \overline{11}^2 \overline{01}^2$ .

De plus, on utilisera la notation décimale 0, 1, 2 et 3 pour représenter les chiffres  $\overline{00}^2$ ,  $\overline{01}^2$ ,  $\overline{10}^2$  et  $\overline{11}^2$ , et on notera  $\overline{c_{n-1} \dots c_0}^\ell$  la représentation en base 4 du codage de Lebesgue d'un point de  $D_n \times D_n$ . Par exemple, pour  $n = 3$ , le codage de Lebesgue du point  $(6, 3)$  sera écrit  $\overline{231}^\ell$ .

En PYTHON, la séquence des chiffres d'un codage de Lebesgue d'un point de  $D_n \times D_n$  est stockée dans une liste de longueur  $n$  triées par poids décroissants : le chiffre de poids le plus fort se trouve en première position, le chiffre de poids le plus faible en dernière position. Ainsi, si  $n = 3$ , le codage de Lebesgue du point  $(6, 3)$  est représenté en PYTHON par la liste `[2, 3, 1]`.

**Question 7.** Soit  $n = 3$ , quelle liste PYTHON représente le codage de  $(1, 6)$  ?

**Question 8.** On suppose que l'on dispose d'une fonction `bits(x, k)`, qui prend en arguments deux entiers naturels  $x$  et  $k$ , et qui renvoie la valeur du bit de coefficient  $2^k$  dans la représentation binaire de  $x$ .

Écrire une fonction `code(n, p)` qui prend en arguments un entier strictement positif  $n$  et un point  $p$  représenté par une liste de longueur 2 dont les deux coordonnées sont prises dans  $D_n$ . Cette fonction renvoie le codage de Lebesgue de  $p$  représenté sous la forme d'une liste PYTHON.

## Partie IV. Représentation d'un ensemble de points

On utilise l'ordre lexicographique (autrement dit, l'ordre du dictionnaire) pour trier les codages. Soient  $c = \overline{c_{n-1} \dots c_0}^\ell$  et  $d = \overline{d_{n-1} \dots d_0}^\ell$  deux codages de Lebesgue de points de  $D_n \times D_n$ . On note  $c < d$  pour "c est strictement plus petit que d" si

$$\exists i, 0 \leq i < n \text{ tel que } \begin{cases} \forall j, j > i \Rightarrow c_j = d_j \\ c_i <_{\mathbb{N}} d_i \end{cases}$$

où  $<_{\mathbb{N}}$  est l'ordre usuel des entiers naturels.

**Question 9.** Trier les codages suivants par ordre croissant pour l'ordre lexicographique :

$$\{\overline{311}^\ell, \overline{000}^\ell, \overline{012}^\ell, \overline{101}^\ell, \overline{233}^\ell\}$$

**Question 10.** Écrire une fonction `compare_pcodes(n, c1, c2)`, qui prend en arguments deux codages de Lebesgue de  $D_n \times D_n$  et renvoie 0 s'il sont égaux, 1 si `c2` est plus grand par l'ordre lexicographique que `c1` et  $-1$  sinon.

Nous allons maintenant représenter un ensemble  $P$  de points de  $D_n \times D_n$  sous la forme d'une liste triée pour l'ordre lexicographique des codages de Lebesgue des points de  $P$ . En guise d'exemple, nous allons coder les points  $S_0 = \{(0, 0), (1, 0), (1, 1), (2, 2), (3, 0), (0, 1)\}$  de  $D_2 \times D_2$ . Ces points représentés en noir dans la figure 1 ci-dessous, dont l'origine est en bas à gauche, les abscisses croissent de gauche à droite et les ordonnées du bas vers le haut.

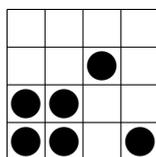


FIGURE 1 – L'ensemble des points  $S_0$

On considère d'abord les représentations binaires des coordonnées des points de  $S_0$  :

$$\overline{S_0}^2 = \{(\overline{00}^2, \overline{00}^2), (\overline{01}^2, \overline{00}^2), (\overline{01}^2, \overline{01}^2), (\overline{10}^2, \overline{10}^2), (\overline{11}^2, \overline{00}^2), (\overline{00}^2, \overline{01}^2)\}$$

à partir desquelles on calcule le codage de Lebesgue de chaque point

$$\overline{S_0}^\ell = \{\overline{00}^2\overline{00}^2, \overline{00}^2\overline{10}^2, \overline{00}^2\overline{11}^2, \overline{11}^2\overline{00}^2, \overline{10}^2\overline{10}^2, \overline{00}^2\overline{01}^2\} = \{\overline{00}^\ell, \overline{02}^\ell, \overline{03}^\ell, \overline{30}^\ell, \overline{22}^\ell, \overline{01}^\ell\}$$

Cet ensemble, une fois trié pour l'ordre lexicographique, s'écrit

$$\{\overline{00}^\ell, \overline{01}^\ell, \overline{02}^\ell, \overline{03}^\ell, \overline{22}^\ell, \overline{30}^\ell\}$$

ce que l'on représente en PYTHON par la liste

$$[[0, 0], [0, 1], [0, 2], [0, 3], [2, 2], [3, 0]]$$

Remarquons que nous venons d'effectuer un changement de système de coordonnées. Le codage de Lebesgue du point  $(x, y)$  représente le chemin à emprunter pour atteindre  $(x, y)$  dans l'espace récursivement divisé en quadrants. En effet, en suivant la numérotation des quadrants donnée en figure 2 ci-dessous, on s'aperçoit par exemple que le point de coordonnées  $(1, 1)$  dans le système de coordonnées usuel est situé dans le quadrant 0 de  $D_2 \times D_2$  et qu'à l'intérieur de ce quadrant subdivisé à son tour, le point  $(1, 1)$  est dans le quadrant 3.

1	3
0	2

FIGURE 2 – Numérotation des quadrants

On peut vérifier que ces coordonnées  $[0, 3]$  dans ce nouveau système correspondent bien au codage de Lebesgue du point de coordonnées  $(1, 1)$  du système usuel.

Formellement, pour  $k < n$ , le quadrant atteint dans  $D_n \times D_n$  par le chemin  $c = d_1 \cdots d_2 \cdots d_k$  (avec  $d_i \in \{0, 1, 2, 3\}$ ) est défini comme suit :

- Si  $k$  vaut 0 alors le chemin  $c$  est vide et le quadrant atteint dans  $D_n \times D_n$  par  $c$  est l'ensemble des points de  $D_n \times D_n$ .
- Si  $k > 0$  alors le chemin est de la forme  $d_1 \cdot d_2 \cdots d_k$ . Dans ce cas, le quadrant atteint par  $d_1 \cdot d_2 \cdots d_k$  dans  $D_n \times D_n$  est l'ensemble des points de  $D_n \times D_n$  dont le codage de Lebesgue est de la forme  $\overline{d_1 d_2 \dots d_k c_{n-k-1} \dots c_0}^\ell$  pour  $c_{n-k-1}, \dots, c_0 \in \{0, 1, 2, 3\}$ .

**Question 11.** On pose pour cette question  $n = 2$ . Donner la représentation sous forme de codage de Lebesgue trié par ordre lexicographique de l'ensemble de points

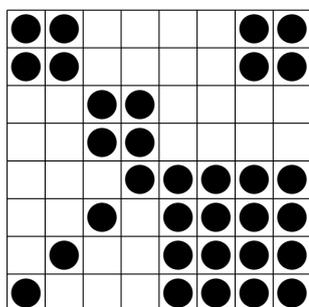
$$S_1 = \{(0, 0), (3, 3), (3, 2), (1, 1), (1, 2), (2, 2), (2, 3)\}$$

## Partie V. Calcul efficace de l'intersection d'ensembles de points

**Compaction par codage des quadrants.** Soit  $S$  un ensemble de points de  $D_n \times D_n$  représenté par la liste  $L$  triée par ordre lexicographique des codages de Lebesgue de ses points (comme dans la partie précédente). En se dotant d'un symbole supplémentaire, notons le 4, on compacte la liste  $L$  en représentant chaque sous-séquence (maximale)  $L'$  correspondant à un quadrant de chemin  $d_1 \cdots d_k$  par l'unique mot  $\overline{d_1 \cdots d_k \cdot 4 \cdots 4}^\ell$  (dans lequel on a rajouté  $n - k$  fois le symbole 4).

Notons qu'un codage de Lebesgue compacté représente *un ensemble de points* et non un unique point comme c'est le cas avec les codages de Lebesgue non compactés. Notons aussi que la liste des codages reste triée par ordre lexicographique.

Par exemple, l'ensemble des points  $S_0$  est compactable en  $[[0, 4], [2, 2], [3, 0]]$ . Le codage  $[0, 4]$  représente le quadrant 0 situé en bas à gauche de la figure 1. Enfin, pour illustrer un cas où  $n > 2$ , la figure 3 décrit le codage compacté d'un ensemble de points de  $D_3 \times D_3$ .



est compacté en

$$\{000^\ell, 003^\ell, 030^\ell, 033^\ell, 114^\ell, 124^\ell, 244^\ell, 334^\ell\}$$

FIGURE 3 – Un ensemble de points de  $D_3 \times D_3$  et sa représentation compactée

**Structure de données d'AQL.** On appelle « AQL de l'ensemble de points  $S$  » la liste triée et compactée des codages de Lebesgue des points de l'ensemble  $S$ .

**Question 12.** Donner l'AQL de l'ensemble  $S_1$  de la question 11.

**Question 13.** Écrire une fonction `ksuffixe(n,k,q)` qui prend en arguments un entier  $n$  strictement positif, une liste  $q$  représentant le codage de Lebesgue compacté d'un quadrant de  $D_n \times D_n$  et un entier naturel  $k$  inférieur strictement à  $n$ . Si les  $k$  derniers chiffres de la liste  $q$  ont pour valeur 4, cette fonction renvoie une nouvelle liste semblable à la liste  $q$  mais dont les  $k+1$  derniers chiffres valent 4. Sinon, cette fonction renvoie  $q$  inchangée.

Ainsi, `ksuffixe(4,2,[0,1,4,4])` renvoie `[0,4,4,4]`, et `ksuffixe(4,2,[0,1,2,4])` renvoie `[0,1,2,4]`.

**Question 14.** L'algorithme de compaction d'une liste triée de codages de Lebesgue consiste à parcourir  $n$  fois la liste représentant l'ensemble de points. L'itération  $k$  vise à remplacer quatre codages successifs formant un quadrant complet de côté  $2^{k+1}$  par la représentation compactée de ce quadrant.

Écrire une fonction `compacte(n,s)` qui prend en arguments un entier strictement positif  $n$  et un ensemble de points  $P$  de  $D_n \times D_n$  représenté par la liste triée  $s$  des codages de Lebesgue de ses points. Cette fonction renvoie l'AQL de l'ensemble de points  $P$ .

**Question 15.** On remarque que l'ordre lexicographique  $<$  défini plus haut s'adapte sans changement aux codages de Lebesgue compactés. Cependant, on souhaite comparer deux codages de Lebesgue compactés en terme de relations d'inclusion et d'exclusion des ensembles de points qu'ils représentent.

Écrire une fonction `compare_ccodes(n,p,q)` qui prend en arguments un entier strictement positif  $n$ , une liste  $p$  contenant le codage de Lebesgue compacté d'un quadrant  $P$  de  $D_n \times D_n$  et une liste  $q$  contenant le codage de Lebesgue compacté d'un quadrant  $Q$  de  $D_n \times D_n$ . Cinq valeurs de retour sont possibles :

- l'entier 0 si les quadrants sont égaux ;
- l'entier 1 si les quadrants  $P$  et  $Q$  sont disjoints et  $p < q$  ;
- l'entier  $-1$  si les quadrants  $P$  et  $Q$  sont disjoints et  $q < p$  ;
- l'entier 2 si  $P \subset Q$  ;
- l'entier  $-2$  si  $Q \subset P$

Par exemple, `compare_ccodes(3,[1,4,4],[2,4,4])` renvoie 1 et `compare_ccodes(3,[1,2,4],[1,4,4])` renvoie 2.

**Question 16.** Pour calculer efficacement l'intersection de deux ensembles de points représentés par leur AQL respectif, on *fusionne* les deux listes triées qui leur correspondent.

En utilisant `compare_ccodes`, écrire une fonction `intersection(n,p,q)` qui prend en argument un entier strictement positif  $n$  ainsi que deux AQL  $P$  et  $Q$  représentant respectivement deux ensembles  $P$  et  $Q$  de points de  $D_n \times D_n$ . Cette fonction renvoie un AQL représentant  $P \cap Q$ . Le nombre d'appels à la fonction `compare_ccodes` effectués par `intersection(n,p,q)` doit être  $O(\text{len}(p) + \text{len}(q))$ .