Problème 1. Intersection de deux ensembles de points

L'objet de ce problème est de décrire un algorithme efficace pour calculer l'intersection de deux ensembles de points dans le plan.

Notations Pour un entier naturel n, on note D_n l'ensemble des entiers naturels $\{0, \ldots, 2^n - 1\}$. Les ensembles de points que l'on manipule dans ce sujet sont des sous-ensembles de $D_n \times D_n$ pour un certain entier n. Autrement dit, un *point* est un couple $(x, y) \in D_n \times D_n$. L'*abscisse* de (x, y) est x et son *ordonnée* est y. On représente un point (x, y) par un couple Python `(x,y)`, de type `tuple`. Un ensemble de points est représenté par une liste de points, sans répétition.

On utilise des entiers représentés dans différentes bases. On utilise la notation $\overline{}^b$ pour représenter un entier en base b, avec le bit de poids fort à gauche. Un entier écrit sans indication est en base 10. Par exemple, $37 = \overline{37}^{10} = \overline{100101}^2 = \overline{211}^4$.

1.1 Solutions naïves

Question 1.1. Solution naïve en Python

(a) Écrire une fonction `membre` qui prend en entrée un point et un ensemble de points, et teste si le point appartient à la liste. La fonction doit renvoyer un booléen.

(b) Écrire une fonction `intersection` qui prend en entrée deux ensembles de points et renvoie leur intersection.

(c) Quelle est la complexité de l'algorithme `intersection`, en nombres de comparaisons entre entiers ?

Question 1.2. Solution naïve en SQL On suppose maintenant que les points et les ensembles sont stockés dans une base de données constituées de deux tables. La table `POINTS` possède trois colonnes : `id` (clé primaire) qui est un entier naturel unique représentant le point, `x` et `y` qui sont des entiers naturels représentant respectivement l'abscisse et l'ordonnée du point. La table `MEMBRE` représente les ensembles de points et possède deux colonnes : `idpoint` qui est un entier naturel identifiant un point, et `idensemble` qui est un entier naturel identifiant un ensemble de points.

(a) Décrire le contenu de deux tables `POINTS` et `MEMBRES` pouvant représenter les deux ensembles de points $E_1 = \{(0, 0), (1, 1)\}$ et $E_2 = \{(1, 0), (1, 1)\}$.

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

(c) Écrire une requête SQL qui renvoie les coordonnées des points qui appartiennent à l'intersection des ensembles d'identifiants i et j.

(d) 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).

1.2 Solution efficace *via* le codage de Lebesgue

Pour décrire un algorithme efficace pour le problème d'intersection d'ensembles de points, on utilise une représentation spéciale pour les points $(x, y) \in D_n \times D_n$, appelée *codage de Lebesgue*. Le codage de Lebesgue d'un point $(x, y) \in D_n \times D_n$ s'obtient en entrelaçant les n bits des représentations binaires de x et y. On suppose que les bits de poids fort sont à gauche et on représente x et y sur n bits chacun, en ajoutant des 0 en tête si nécessaire. On commence l'entrelacement par le bit de poids fort de x.

Par exemple, si $n = 3$ et $(x, y) = (6, 3)$, alors $x = \overline{\mathbf{110}}^2$ et $y = \overline{011}^2$ en binaire. Leur codage de Lebesgue est l'entier $\overline{\mathbf{101101}}^2 = 45$. Le codage de Lebesgue d'un point $(x, y) \in D_n \times D_n$ est donc une suite de $2n$ bits. On peut regrouper ces bits deux par deux, et interpréter le codage de Lebesgue de (x, y) comme un nombre de n chiffres en base 4 : par exemple, le codage de $(6, 3)$ est $45 = \overline{\mathbf{10}}^2\overline{\mathbf{11}}^2\overline{\mathbf{01}}^2 = \overline{231}^4$.

En Python, on représentera le codage de Lebesgue d'un point $(x, y) \in D_n \times D_n$ par un n-uplet d'entiers ne contenant que des entiers entre 0 et 3 (inclus). Par exemple, la représentation de $(6, 3)$ sera donc `(2,3,1)`.

Question 1.3. Représentation d'un point

(a) Donner la représentation Python du codage de Lebesgue du point $(1, 6) \in D_3 \times D_3$. Donner la représentation du même point $(1, 6)$, mais vu cette fois comme élément de $D_4 \times D_4$.

On suppose disposer d'une fonction `bit` qui prend en entrée deux entiers x et k et renvoie le coefficient de 2^k dans l'écriture binaire de x.

(b) Écrire une fonction `code(n, p)` qui prend en entrée un entier n et un point p de $D_n \times D_n$ représenté par ses coordonnées et renvoie le codage de Lebesgue de p.

(c) Écrire une fonction `decode(n, c)` qui effectue l'opération inverse.

Question 1.4. Représentation d'un ensemble de points On représente un ensemble de points par une liste de points, chacun représentés par leur codage de Lebesgue. Afin d'obtenir des algorithmes rapides, on maintient la liste triée selon l'ordre lexicographique : soit $c = \overline{c_{n-1} \cdots c_0}^4$ et $d = \overline{d_{n-1} \cdots d_0}^4$ deux codages de Lebesgue de points de $D_n \times D_n$, alors $c < d$ dans l'ordre lexicographique s'il existe $i \in \{0, \ldots, n-1\}$ tel que $c_i < d_i$ (pour l'ordre naturel des entiers) et $c_j = d_j$ pour $j > i$.

(a) Trier l'ensemble suivant selon l'ordre lexicographique : $\{\overline{311}^4, \overline{000}^4, \overline{012}^4, \overline{101}^4, \overline{233}^4\}$.

(b) Écrire une fonction `compare_code` qui prend en entrée un entier naturel n et deux codages de Lebesgue c et d de points de $D_n \times D_n$, et renvoie 0 s'ils sont égaux, 1 si $c < d$ et -1 sinon.

(c) Écrire une fonction `lebesgue` qui prend en entrée un entier n et une liste de points de $D_n \times D_n$ représentés par leurs coordonnées, et renvoie la liste triée de leurs codages de Lebesgue. *On utilisera une recherche dichotomique pour insérer chaque codage de Lebesgue dans la liste triée.*

(d) Écrire une fonction `delebesgue` qui effectue l'opération inverse. *L'ordre des points dans la liste résultat sera quelconque.*

Question 1.5. Intersection efficace L'idée pour calculer efficacement l'intersection de deux ensembles S et T triés est similaire à l'algorithme de *fusion* de deux listes triées pour le tri fusion (qui calcule une union). On maintient deux indices i et j, un par ensemble, initialisés tous les deux à 0. À chaque étape, on compare S_i avec T_j : s'ils sont égaux, cet élément est ajouté à l'intersection et les deux indices sont incrémentés ; sinon, on incrémente l'indice du plus petit des deux éléments.

(a) Écrire une fonction `intersection_lebesgue` qui prend en entrée un entier n et deux ensembles de points de $D_n \times D_n$ représentés par les listes triées de leurs codages de Lebesgue, et renvoie l'intersection des deux ensembles dans la même représentation.

(b) Écrire une fonction `intersection2` qui prend en entrée un entier n et deux ensembles de points de $D_n \times D_n$ représentés par les listes de leurs coordonnées et renvoie l'intersection des deux ensembles dans la même représentation.

(c) Quelle est la complexité de la fonction `intersection_lebesgue` ?

1.3 Intersection d'ensembles compacts

On cherche à représenter les ensembles de points de manière plus compacte, tout en ayant un algorithme rapide pour le calcul d'intersection. L'idée pour cela est de voir $D_n \times D_n$ comme un ensemble de cases et d'utiliser un découpage récursif de $D_n \times D_n$ en quatre zones numérotées de 0 à 3, chacune isomorphe à $D_{n-1} \times D_{n-1}$, comme sur la figure 1 dont l'origine $(0, 0)$ est en bas à gauche les abscisses croissant de gauche à droite et les ordonnées de bas en haut. On appelle *quadrant* une zone du découpage. Le découpage est récursif au sens où chaque quadrant $D_{n-1} \times D_{n-1}$ est lui-même découpé en quatre quadrants $D_{n-2} \times D_{n-2}$, et ainsi de suite jusqu'aux quadrants $D_0 \times D_0$ qui sont de simples cases.

Question 1.6. Changement de coordonnées On peut identifier un quadrant $D_k \times D_k$ de $D_n \times D_n$ par le *chemin* des numéros de quadrants à suivre pour l'atteindre. Par exemple, le quadrant grisé de la figure 1 est identifié par le chemin 30 puisqu'il est dans le quadrant $D_2 \times D_2$ numéroté 3, puis à l'intérieur de ce quadrant il s'agit du quadrant numéroté 0. Les cases sont donc identifiées des chemins de longueur n : la case de coordonnées $(4, 5)$, en haut à gauche du quadrant grisé, est identifiée par le chemin 301.

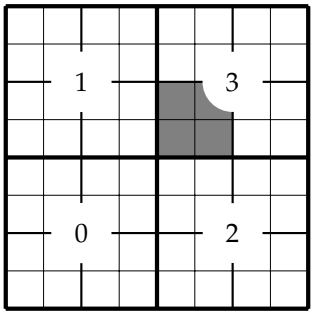

FIGURE 1 – Découpage de $D_3 \times D_3$ en quatre quadrants $D_2 \times D_2$, avec un quadrant $D_1 \times D_1$ grisé.

(a) Quel est le chemin à suivre pour atteindre la case $(2,1)$ de $D_3 \times D_3$?
Quelle case atteint-on en suivant le chemin 032 dans $D_3 \times D_3$?

(b) Montrer que chaque case de $D_n \times D_n$ est identifiée de manière unique par un chemin tel que défini ci-dessus.

(c) Montrer que le chemin d'une case $(x, y) \in D_n \times D_n$ est égal à son codage de Lebesgue, c'est-à-dire qu'une case identifiée par un chemin $c_{n-1} \cdots c_0$ possède le codage de Lebesgue $\overline{c_{n-1} \cdots c_0}^4$.

(d) Montrer que tout codage de Lebesgue de longueur k identifie de manière unique un quadrant $D_{n-k} \times D_{n-k}$ de $D_n \times D_n$.

Question 1.7. Représentation compactée Pour compacter la représentation d'un ensemble de points, on utilise l'identification unique des quadrants de $D_n \times D_n$ par des codages de Lebesgue de longueur $\leq n$. Si un ensemble S contient tous les points d'un quadrant $D_k \times D_k$ au sein de $D_n \times D_n$, on représente cet ensemble de points par le codage de Lebesgue du quadrant. Par exemple, l'ensemble $S_0 = \{(0,0),(1,0),(1,1),(2,2),(3,0),(0,1)\} \subset D_2 \times D_2$, représenté en figure 2, contient tous les points du quadrant $D_1 \times D_1$ de codage de Lebesgue $\overline{0}^4$. On peut donc représenter S_0 par l'ensemble $\{\overline{0}^4, \overline{22}^4, \overline{30}^4\}$. On appelle cette représentation la *représentation compactée* de S_0.

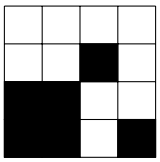

FIGURE 2 – Représentation graphique de l'ensemble S_0.

En Python, on ne souhaite pas manipuler des uplets de longueurs variables. Pour que tous les codages de Lebesgue dans $D_n \times D_n$ soient des n-uplets, on introduit un nouveau symbole, `4`, et on *complète* les codages de Lebesgue de longueur $< n$ par autant de `4` que nécessaire : ainsi, S_0 est représenté en Python par la liste `[(0,4),(2,2),(3,0)]`.

(a) Donner la représentation compactée de l'ensemble $S_1 = \{(0,0),(3,3),(3,2),(1,1),(1,2),(2,2),(2,3)\}$ de points de $D_2 \times D_2$, sous forme mathématique et en Python.

(b) Écrire une fonction `etend` qui prend en entrée un entier n et un n-uplet d'entiers de $\{0,\ldots,4\}$ et renvoie la liste, triée par ordre lexicographique, des codages de Lebesgue de toutes les cases du quadrant identifié par le n-uplet. *Par exemple,* `etend(3, (3, 2, 4))` *renvoie la liste* `[(3,2,0),(3,2,1),(3,2,2),(3,2,3)]`.

(c) Écrire une fonction `decompresse` qui prend en entrée un entier n et la représentation compactée d'un ensemble de points de $D_n \times D_n$ et renvoie sa représentation classique, c'est-à-dire la liste triée des codages de Lebesgue de cet ensemble.

Question 1.8. Compaction d'une représentation L'algorithme de compaction d'une liste triée de codages de Lebesgue de points de $D_n \times D_n$ consiste à parcourir n fois la liste. À la $k^{\text{ème}}$ itération, on cherche à remplacer quatre codages successifs formant un quadrant complet $D_k \times D_k$ par la représentation compactée de ce quadrant.

(a) Justifier l'utilisation d'une liste triée en entrée de cet algorithme.

(b) Soit S_i, S_{i+1}, S_{i+2} et S_{i+3} quatre codages de Lebesgues consécutifs dans une liste triée de codages de Lebesgue. Montrer qu'ils forment un quadrant complet si et seulement s'il existe c_1, ..., $c_t \in \{0, 1, 2, 3\}$ tels que $S_i = (c_1, \ldots, c_t, 0, 4, \ldots, 4)$ et $S_{i+3} = (c_1, \ldots, c_t, 3, 4, \ldots, 4)$.

(c) Écrire une fonction `ksuffixe(n, k, q)` qui prend en entrée deux entiers n et k, et le codage de Lebesgue compacté `q` d'un quadrant de $D_n \times D_n$. Si `q` représente un quadrant $D_k \times D_k$, la fonction doit renvoyer un codage de Lebesgue compacté du quadrant $D_{k+1} \times D_{k+1}$ qui contient `q`. Sinon, la fonction renvoie `q` lui-même.

(d) Écrire une fonction `compacte(n, S)` qui prend en entrée un entier n et un ensemble `S` de points de $D_n \times D_n$ sous la forme d'une liste triée de leurs codages de Lebesgue, et renvoie la représentation compactée de l'ensemble.

Question 1.9. Comparaisons de codages de Lebesgue compactés On souhaite comparer deux codages de Lebesgue compactés du point de vue des relation d'inclusion existant entre les deux ensembles de points qu'ils représentent.

(a) Soit q_1 et q_2 deux quadrants de $D_n \times D_n$ et S_1 et S_2 les ensembles de points qu'ils représentent, respectivement. Montrer qu'il n'existe que trois possibilités : soit S_1 et S_2 sont identiques, soit $S_1 \cap S_2 = \emptyset$, soit $S_1 \subset S_2$, soit $S_2 \subset S_1$.

(b) Écrire une fonction `compare_inclusion(n, q1, q2)` qui prend en entrée un entier n et deux quadrants `q1` et `q2` de $D_n \times D_n$ représentant des ensembles de points S_1 et S_2 et renvoie un entier $r \in \{-2, -1, 0, 1, 2\}$ avec la signification suivante : $r = 0$ si $S_1 = S_2$, $r = 1$ si $S_1 \cap S_2 = \emptyset$ et `q1 < q2` pour l'ordre lexicographique, $r = -1$ si $S_1 \cap S_2 = \emptyset$ et `q2 < q1`, $r = 2$ si $S_1 \subset S_2$, et $r = -2$ si $S_2 \subset S_1$.

Question 1.10. Intersection efficace d'ensembles compactés Montrer comment adapter l'algorithme `intersection_lebesgue` au cas des ensembles compactés, avec la même complexité. *Indication. Dans l'algorithme de fusion évoqué dans la question 1.5, il faut maintenant traiter le cas $S_i \subset T_j$ ou inversement.*