Etude de la percolation 2D

On se propose ici de mesurer numériquement le taux d'impuretés à introduire dans un milieu à deux dimensions pour que le courant traverse l'échantillon. Cette modélisation est analogue à l'introduction de micro-canaux pour l'écoulement d'un fluide par porosité et bien d'autres choses. Deux électrodes sont soudées aux extrémités de l'échantillon et chaque impureté permet la connexion électrique avec ses voisins du haut, du bas, de gauche et de droite.

Le chemin de connexion est inconnu et la grille peut être très grande N >> 1 ; il faut donc s'appuyer sur une structure de données efficace pour répondre à la question « est-ce que les points de départ et d'arrivée sont connectés ». On utilise ici la classe **Quick Find** [qui est déjà rédigée en début de code]. C'est la plus simple mais pas la plus rapide.

Attention : ce TP ne peut pas être fait en 50 minutes …
- Les 3 premières questions sont très générales et n'abordent pas les structures de données. Elles doivent être préparées avant le TP (Révisions Python).
- La suite est plus complexe mais on ne fait qu'utiliser les méthodes de classe à la manière de fonctions. Elle demande également du temps.

Question ouverte :
Nous verrons en cours comment améliorer cette classe : **Quick Union** [cf X-ens 2016] ou mieux : **Weighted Quick Union** qui optimisent les opérations d'union. Elles permettent de surcroît l'implémentation d'un algorithme dit de « Path Compression » rendant les calculs très très rapides.

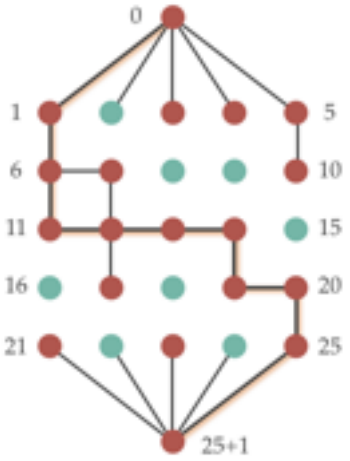
1 - La grille

Créer la liste des cellules de la grille ne contenant aucune impureté : **maGrille**.
On la représente par une simple liste de taille N² à 1-dimension, contenant 0 si la cellule n'a pas d'impureté et 1 si oui. Initialement la liste ne contient que des zéros. On saura donc si la cellule d'indice k contient une impureté ou non avec maGrille[k] :

maGrille :

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

maGrilleUF :



Exemple : maGrille[17] = 0
maGrille[19] = 1

maGrilleUF.isConnected(0, 26) = True
maGrilleUF.isConnected(5, 16) = False

Attention : Notez bien que les cellules de la grille (à gauche) sont indexées de 0 à N² - 1 alors que celles dans les partitions (à droite) le sont de 1 à N² [0 et N² + 1 étant les électrodes].

2 Affichage

Écrire une fonction **afficheMaGrille** qui prend en argument la grille et qui en réalise l'affichage dans la console sous la forme d'une matrice carrée de 0 et de 1 comme ceci :

```
>>> afficheMaGrille(maGrille)
| 1 0 0 0 0 1 1 0 0 0 |
| 0 1 1 1 0 1 0 1 0 0 |
| 0 0 1 1 0 1 1 0 1 1 |
| 1 0 0 1 1 1 0 0 1 0 |
| 0 0 1 1 1 0 1 1 1 1 |
| 0 1 1 1 1 1 1 1 0 1 |
| 1 1 1 0 0 0 0 1 0 1 |
| 1 0 1 1 1 1 0 1 1 1 |
| 0 1 1 1 1 1 1 0 0 1 |
| 1 1 1 0 1 1 0 0 1 0 |
```

Cela permet de visualiser l'état de la grille : très utile ! En particulier au début pour faire des tests sur de petites tailles de grilles.

3 Conversions des coordonnées :

Il y a plusieurs de systèmes de numérotation différents :
la grille de 0 —> N^2 - 1,
la partition de 0 (borne supérieure) —> N^2 + 1 (borne inférieure) les cellule étant de 0 —> N^2,
la position géométrique (x, y) dans la matrice : y va vers le bas et x vers la droite de 1 à N.

Pour manipuler confortablement ces indices et pouvoir « penser notre problème » avec facilité nous allons coder de simples fonctions de conversion d'un système de numérotation à l'autre. Celles-ci seront vite indispensables pour la fonction **jonction** de la question 5.

a - Écrire une fonction **coordCellule** qui prend en argument l'indice n de la cellule de la grille et qui renvoie le tuple (x , y) de ses coordonnées géométriques à la manière d'indices matriciels. Ainsi la cellule n = 0 sera en (1,1) et la cellule n = N^2 - 1 sera en (N, N).

Attention : y va vers le bas et x vers la droite [comme les pixels dans une image].

b - Écrire à l'inverse une fonction **indexCellule(x, y)** qui prend en argument le tuple (x, y) et retourne l'indice n entre 0 et N^2 - 1 de la cellule dans la grille.

c - Écrire enfin une fonction **indexPartition(x, y)** qui prend en argument le tuple (x, y) et retourne l'indice k entre 0 et N^2 + 1 de la cellule mais dans la partition.

Rq : notez bien que les cellules de la grille (cf schéma de gauche) sont indexées de n = 0 à N^2 - 1 alors que celles dans les partitions (cf schéma de droite) le sont de k = 1 à N^2.
[0 et N^2 + 1 étant les électrodes].

4 - Les partitions associées aux cellules de la grille

On doit au préalable créer une instance de partition de taille $N^2 + 2$ représentant les N^2 cases de la grille plus les deux points de contact en haut en et bas. On stockera cette instance dans la variable **maGrilleUF** pour signifier que cela ne représente pas la grille elle-même, mais les partitions des cellules de la grille obtenues après différentes opérations d'union de de recherche : Union-Find. (voir les deux schémas au début).

On utilise pour cela la classe QuickUFind fournie en début de fichier à l'aide la commande : **QuickUFind(n).** Cette commande construit et renvoie l'objet python de la partition (on parle d'instance de partition) qui a tous les attributs & méthodes nécessaires concernant les partitions.

Structure de données pour la classe de partition QuickUFind

attributs :

- maGrilleUF.Np_part : nombre de partitions.
- maGrilleUF.id : liste des représentants (maGrilleUF.id[p] —> représentant de la cellule p)

méthodes :

- maGrilleUF.Find(p) : trouve le numéro de partition de l'élément p.
- maGrilleUF.isConnected(p, q) : dit si p et q sont dans la même partition.
- maGrilleUF.Union(p, q) : réalise l'union de 2 partitions.

5 - Insertion d'une impureté

On veut introduire une impureté sur la cellule d'indice n de la grille.

- Il faut bien sûr fixer l'état de la cellule n à 1 => mettre la jour la grille (question suivante).
- mais il faut surtout mettre à jour les partitions :

Si des cellules voisines possèdent une impureté [dessus dessous gauche droite] alors il faut faire l'union de ces cellules avec la nouvelle impureté, dans l'instance de partition. Soient p et q ces deux cellules voisines : **maGrilleUF.Union(p, q)** réalise ce travail.

On sera amené à utiliser les coordonnées géométriques (x, y) de la cellule, l'indice n de grille et aussi l'indice k dans les partitions : d'où les fonctions de conversion à la question 3.

Écrire une fonction **jonction(n)** qui prend en argument l'indice n de la cellule et met à jour la partition **maGrilleUF**. Cette fonction ne retourne rien mais réalise la mise à jour des partitions.

Attention : Il faut tenir compte du fait que les bords de gauche et droite ne sont pas en contact. De plus les cellules de la première ligne sont déjà reliées à la même borne supérieure (question 6) (idem pour la dernière ligne).

6 - Déroulement de la partie

On va maintenant introduire aléatoirement les impuretés dans la grille initialement vide.
Écrire une fonction **playTheGame(maGrille, maGrilleUF)** qui prend en argument la grille et l'instance de partition et qui retourne le nombre d'impuretés introduites au moment ou la connexion sera réalisée à travers la matrice. Pour cela :

- Faire l'union de toutes les cellules de la première ligne avec la borne supérieure (indice 0 de la partition) et idem pour la dernière ligne avec la borne du bas (indice $N^2 + 1$).

- Écrire une boucle qui, tant la connexion n'est pas réalisée :
    - Génère un indice de cellule aléatoire.
    - Si la case est sans impureté :
        - Introduit l'impureté.
        - Réalise les jonctions nécessaires.
- On comptera le nombre d'impuretés introduites.

En déduire le taux d'impureté : nombre de cases impures sur le nombre total de cases.

7 - Finalisation

Pour obtenir un chiffrage précis de ce taux, on réalisera l'expérience un grand nombre de fois, de façon à obtenir un taux moyen plus fiable (ex : 100 fois).

Écrire une portion de code qui lance ~100 expériences à la suite et calcul le taux moyen.
On peut ensuite ajouter des chronométrages avec le module time :

```
import time
start = time.clock()  #stock le temps en seconde
*** Expérience numérique ***
stop = time.clock()
```