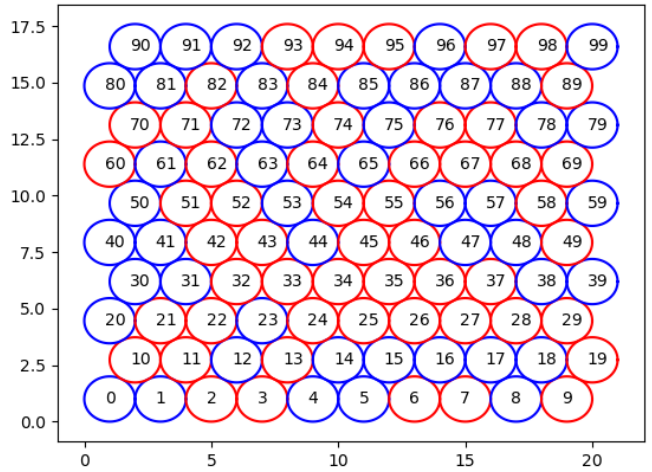
Percolation

1 - Introduction

La **percolation** (du latin percolare, "passer à travers") désigne dans le langage commun le passage des fluides à travers les matériaux poreux. En physique, la théorie de la percolation montre que dans un ensemble de points répartis de manière aléatoire selon leur porosité il existe une probabilité critique p au delà de laquelle les points se connectent. On doit cette théorie à John Hammersley (1957) qui cherchait à comprendre comment les masques à gaz des soldats devenaient inefficaces.

Pour modéliser ce phénomène on imagine des billes numérotées de 0 à n^2-1 que l'on emplie dans une boîte à raison de n billes par lignes. On suppose que certaines billes sont en métal donc conductrices et d'autres en plastique (et donc non conductrices). On cherche à savoir si il existe un chemin conducteur du bas de la boîte jusqu'au haut.



2 - Installation des billes

1. Ecrire une fonction `conducteur(p)` qui prend comme paramètre un réel $p \in]0,1[$ et qui renvoie `True` avec la probabilité p et `False` avec la probabilité $1-p$.
2. Pour configurer notre boîte on souhaite numéroter les billes de 1 à n^2 de gauche à droite et de bas en haut. Nos billes sont de rayon 1 et chaque bille est conductrice avec la probabilité p.
   Ecrire une fonction `config(n,p)` qui renvoie une liste de listes `L` dont les éléments sont de la forme `[i,x,y,cond]`, i étant le numéro de la boule, x et y les coordonnées de sont centre et *cond* étant un booléen désignant le caractère conducteur de la bille.
   On pourra écrire deux boucles imbriquées par ligne ℓ et par colonne c en remarquant que :
   - le numéro de la bille placé en ℓ-ème ligne - c-ème colonne est $n \times \ell + c$;
   - son abscisse est $1+2c$ si ℓ est pair et $2+2c$ si ℓ est impair ;
   - son ordonnée est $1+\sqrt{3}\ell$.

3 - Construction d'un graphe associé

On décide de représenter notre système par un graphe dont les sommets sont les billes conductrices et tel que les arêtes relient deux billes conductrices et en contact.

3. Ecrire une fonction `distance(P,Q)` qui à deux points du plan P : (x_1,y_1) et Q : (x_2,y_2), associe leur distance euclidienne.

AM, JB, NG, SR

4. Ecrire une fonction `voisins(L,k)` qui prend comme paramètres la liste des billes telle qu'elle est définie en question 2 et un numéro de bille k et qui renvoie la liste des numéros des billes qui sont en contact avec elle et qui sont conductrices.
5. Ecrire une fonction `sommets(L)` qui associe à une liste `L` les seuls éléments de `L` représentant des billes conductrices.
6. Ecrire une fonction `graphe(L)` qui associe à une liste `L` le dictionnaire du graphe composé des sommets de la question précédente, les arcs reliant de façon non-orienté les différents sommets voisins.

4 - Percolation

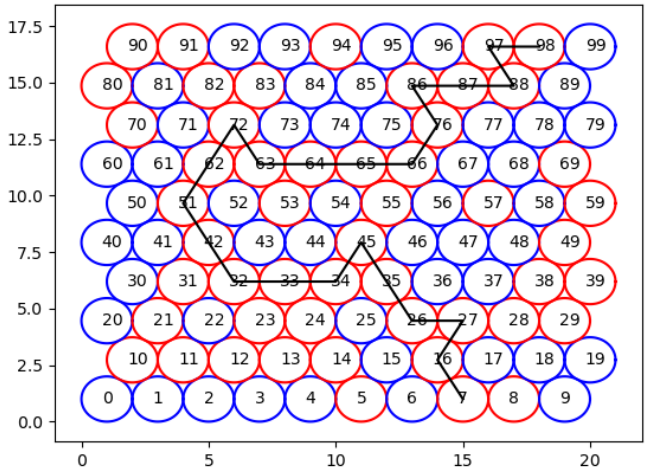
Il y a percolation s'il existe un chemin conducteur allant du bas de la boite jusqu'au haut. C'est à dire qu'il existe un chemin dont la première bille a un numéro strictement inférieur à n et la dernière bille un numéro supérieur ou égal à $n^2 - n$.

Pour savoir s'il y a percolation, il faut faire un parcours en profondeur à partir de chacune des billes conductrices ayant un numéro strictement inférieur à n. On renvoie `True` dès qu'un chemin possible se termine par une bille de numéro supérieur ou égal à $n^2 - n$.

On reprend la fonction `ajoute_sommet(dico,chemin)` vue dans le TP précédent. Elle prend comme paramètre un dictionnaire représentant un graphe non orienté et un chemin possible dans le graphe. Elle renvoie une liste contenant autant de chemins qu'il y a de voisins du dernier sommet de `chemin` non encore présents dans la liste `chemin`. Elle renvoie la liste vide s'il n'y a aucun chemin possible.

```
def ajoute_sommet(dico,chemin):
    sommet = chemin[-1]
    L = []
    for i in dico[sommet]:
        if not i in chemin:
            L.append(chemin+[i])
    return L
```

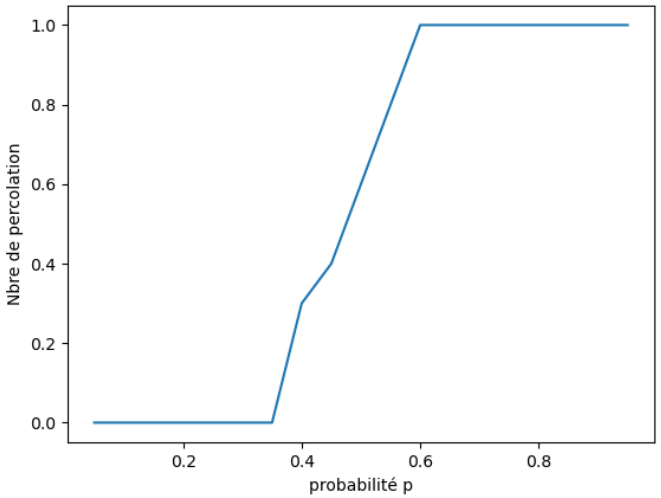
7. Ecrire une fonction `percolation(L,n)` qui renvoie `True` s'il y a percolation en partant de la bille `n` et `False` sinon. On pourra procéder de la manière suivante :
   - on pose `som = sommets(L)`, `dico = graphe(L)`
   - définir une pile `chemins_temp` contenant la liste `[som[i]]` et une liste `chemins` vide ;
   - tant que la pile n'est pas vide :
     - dépiler la pile, soit `c` le chemin qui se trouve sur le dessus de la pile ;
     - si le dernier élément de `c` est un numéro inférieur ou égal à n^2-n, renvoyé `True` ;
     - sinon ajouter à la pile les chemins possibles issus de `c` à l'aide de la fonction `ajoute_sommet`.
   - si on n'a pas renvoyé `True` pendant l'algorithme, c'est qu'il n'y a pas percolation. On renvoie donc `False`
8. Proposer une modification de la fonction précédente pour renvoyer le chemin trouvé lors de la percolation.
9. Proposer enfin une fonction `dessin_chemin(c)` qui à partir du chemin c composé des numéros de billes parcourues affiche le tracé du chemin. (voir figure ci-dessous).



10. Ecrire une fonction `percolation_moyenne(n,p)` qui simule la création de 100 boites de billes à l'aide de la fonction `config` et qui renvoie le nombre moyen de percolation.

Le script suivant permet de calculer la percolation moyenne pour différente valeur de p et de tracer l'évolution de celle-ci en fonction de p.

```
import matplotlib.pyplot as plt
Prob = [i/40 for i in range(1,40)]
moy = [percolation_moyenne(10,p) for p in Prob]
plt.plot(Prob,moy)
plt.show()
```



Remarque 1 :

On parle de bifurcation lorsqu'il y a un changement radical de comportement pour un système. En physique, ce type de comportement est voisin d'une transition de phase.

5 - Recherche des composantes connexes du graphe

9. On considère la fonction suivante :

```
def comp_connexes(L):
    dico = graphe(L)
    s = sommets(L)
    N = len(dico)
    T = [s[i] for i in range(N)]
    modifications = True
    while modifications:
        modifications = False
        for i in range(N):
            for j in range(i+1,N):
                if s[j] in dico[s[i]] and T[i] != T[j]:
                    T[i] = min(T[i],T[j])
                    T[j] = T[i]
                    modifications = True
    return T
```

Justifier que la liste renvoyée est une liste `T` dont la longueur est le nombre de billes conductrices et telle que pour tout i la valeur de `T[i]` est le plus petit numéro des billes appartenant à la même composante connexe que la i-ème bille conductrice.

10. Ecrire une fonction `comp_connexes_dic(L)` qui utilise la fonction `comp_connexes` et qui associe à la liste `L` un dictionnaire `dic` dont les clés sont les plus petits numéros des billes de chaque composante connexe et tel que pour tout i, `dic[i]` soit la liste des billes appartenant à la composante connexe de la bille numéro i.

11. Il y a percolation si il existe une composante connexe contenant une bille de numéro strictement inférieur à n et une bille de numéro supérieur ou égal à $n^2 - n$. Ecrire une fonction `percolation_2(L,n)`, utilisant la fonction `comp_connexes_dic(L)` qui renvoie `True` s'il y a percolation et `False` sinon.