Feuille de TP n° 9 - Graphes

Représentation des graphes Rappelons qu'un graphe est constitué d'un ensemble de *sommets* et d'un ensemble d'*arêtes* reliant ces sommets.
Dans ce TP, les sommets des graphes seront numérotés de 0 à $n-1$ (où n est le nombre de sommets du graphe).

Exercice 1.

1. Tracer à la main les graphes représentés par les listes de sommets et d'arêtes suivantes.

```
sommets=[k for k in range(4)]
arcs1=[(0,1),(1,2),(2,3),(3,0)]
arcs2=[(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)]
G1=[sommets,arcs1]
G2=[sommets,arcs2]
```

2. Ecrire en Python les graphes $G1$ et $G2$ sous forme de liste d'adjacence (liste de listes des arêtes).
3. Ecrire une fonction `matrice_adjacence` qui prend en entrée un graphe G sous forme de liste d'adjacence, et qui renvoie la matrice d'adjacence de G.
   On écrira cette matrice sous forme de liste de listes.
4. Tester la fonction `matrice_adjacence` avec les graphes $G1$ et $G2$.
5. Ecrire une fonction `liste_adjacence` qui prend en entrée un graphe G sous forme de matrice d'adjacence, et qui renvoie la liste d'adjacence de G.
6. Ecrire une fonction `non_oriente` qui prend en entrée un graphe `G` orienté, sous forme de liste d'adjacence, et qui renvoie en sortie le graphe non-orienté obtenu à partir de `G`.
7. Ecrire une fonction `test_oriente` qui prend en entrée un graphe `G` et qui renvoie `False` si le graphe est non-orienté et `True` s'il est orienté.

On étudie les deux manières naturelles de chercher un chemin dans un graphe en partant d'un sommet initial `S_i` vers un sommet final `S_c` : la recherche en profondeur et la recherche en largeur.

La convention vue en cours consiste à ranger les sommets en trois listes :

- La liste `N` des sommets visités et coloriés en noir.
- La liste `G` des sommets à visiter et coloriés en gris. Ce sont les sommets voisins d'un sommet de la liste `N`, mais qui n'ont pas encore été visités.
- La liste `B` des sommets qui ne sont pas dans les deux listes précédentes.

L'algorithme général pour une recherche de sommet avec mémoire est le suivant :

```
On dispose d'un sommet initial S_i, et un sommet cible S_c.
Colorier S_i en gris (à explorer), et les autres sommets en blanc.
Tant qu'il reste des sommets gris :
    Choisir un sommet gris S
    Si S = S_c : l'algorithme se termine par un succès.
    Sinon : le colorier en noir (exploré).
        Colorier tous ses voisins blancs en gris (à explorer).
L'algorithme se termine par un échec (S_c n'a pas été trouvé).
```

L'instruction `Colorier tous ses voisins blancs en gris` correspond à mettre à la suite de la liste `G` tous les nouveaux voisins qui étaient coloriés en blanc.

La différence entre la recherche d'un chemin en largeur et en profondeur se situe sur la ligne `Choisir un sommet gris S`. Si l'on choisit le dernier élément de la liste `G`, on a un parcours en profondeur et `G` est utilisée comme une pile (dernier arrivé, premier sorti). Si l'on choisit le premier élément de la liste `G`, on a un parcours en largeur et `G` est utilisée comme une file (premier arrivé, premier sorti).

Exercice 2.

1. Écrire une fonction `parcours_profondeur(G,S_i,S_c)` qui prend en argument un graphe `G`, un sommet initial `S_i` et un sommet cible `S_c`, et renvoie la liste des sommets coloriés en noir après un parcours en profondeur (ou False si `S_i` et `S_c` ne sont pas reliés).
   Le graphe G sera écrit comme une liste d'adjacence.

2. Écrire une fonction `parcours_largeur(G,S_i,S_c)` qui prend en argument un graphe `G`, un sommet initial `S_i` et un sommet cible `S_c`, et renvoie la liste des sommets coloriés en noir après un parcours en largeur (ou False si `S_i` et `S_c` ne sont pas reliés).
3. Ecrire une fonction `degre_e` qui prend en entrée un graphe `G` et qui renvoie la liste des degrés entrants de chacun de ses sommets.
   Le graphe G sera écrit comme une liste d'adjacence.
4. Ecrire une fonction `degre_s` qui prend en entrée un graphe `G` et qui renvoie la liste des degrés sortants de chacun de ses sommets.

Pour utiliser davantage les deux idées de parcours de graphe (largeur et profondeur), il faut ajouter davantage d'information à celles-ci. Au lieu de juste griser un sommet à visiter ou de noircir un sommet visité, on crée une liste supplémentaire qui stockera une information de plus pour chaque sommet (longueur parcourue, sommet d'où l'on vient, degré entrant, etc). Cette information dépendra du contexte.

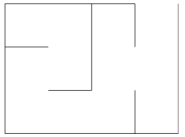
Exercice 3. Ecrire une fonction `chemin` qui prend en entrée un graphe `G` et deux sommets `S_i` et `S_c`, et qui renvoie un chemin c de longueur minimale reliant ces deux sommets.
Le graphe `G` sera sous forme de liste d'adjacence, les sommets `S_i` et `S_c` sont supposés reliés, et le chemin c sera une liste contenant le nom de tous les sommets à parcourir de `S_i` à `S_c`.
On ajoutera pour cela à la fonction `parcours_largeur` une liste `O` qui contiendra pour chaque sommet visité son sommet d'origine. Cela servira à construire le chemin `c`.

Exercice 4 (**Sortir d'un labyrinthe**)**.** On modélise un labyrinthe rectangulaire de taille $m \times n$ à l'aide d'un graphe, de la façon suivante :

- Les cases du labyrinthe sont numérotées de 0 à $mn-1$ (de gauche à droite puis de haut en bas) et constituent les sommets du graphe,
- Si l'on peut se déplacer d'une case à une autre (si elles sont voisines et ne sont pas séparées par un mur), on met une arête entre les sommets correspondants.

On suppose que la sortie du labyrinthe est la case en haut à droite, c'est-à-dire de numéro $n-1$.

i. Ecrire en Python le graphe `G_L` correspondant au labyrinthe suivant :



ii. Ecrire une fonction `sortie` qui prend en entrée le graphe `G` correspondant à un labyrinthe, un sommet de départ `k`, et qui renvoie un chemin `c` reliant le sommet `k` à la sortie.
   Le graphe `G` sera sous forme de liste d'adjacence, on supposera que `k` est relié à la sortie, et le chemin `c` sera sous forme de liste.

iii. Afficher les chemins donnés par la fonction `sortie` pour le graphe `G_L`.

Exercice 5.

1. Ecrire une fonction `boucle` qui prend en entrée un chemin `c`, sous forme de liste de sommets, et qui renvoie `True` si le chemin contient une boucle, et `False` sinon.
   On considèrera que si le chemin `c` part et revient au même sommet, il ne contient pas de boucle.
2. Ecrire une fonction `raccourci` qui prend en entrée un chemin `c`, et qui renvoie en sortie un chemin `d` qui correspond au chemin `c` auquel on a retiré toutes les boucles.
   On pourra utiliser une boucle while et la fonction `boucle`.