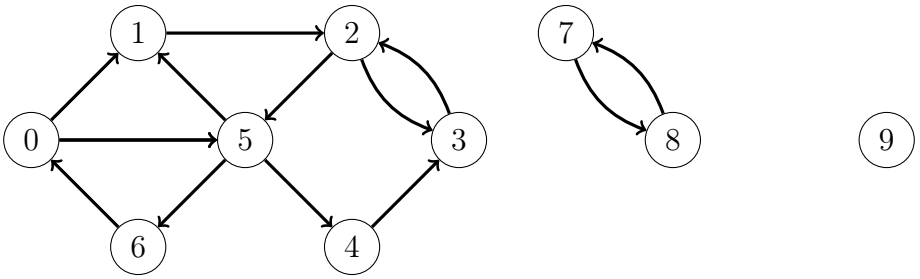
MPSI - ITC - TD 23 : Graphes, représentations

Aujourd'hui on étudiera différentes façons de représenter un graphe en Python.

Numérotation des sommets

Vu qu'un graphe a toujours un nombre fini de sommets, si on note n le nombre de sommets on peut numéroter les sommets du graphe de 0 à $n-1$ et utiliser le numéro d'un sommet pour l'identifier.
Voici ci-dessous un exemple de graphe orienté à 10 sommets :



En identifiant les sommets à leur numéro, représenter l'ensemble de tous les sommets devient trivial : il suffit de préciser combien il y en a (ici 10).
Par la suite on verra 3 façons différentes de représenter l'ensemble des arêtes (pour les graphes non-orientés) et des arcs (pour les graphes orientés).

Arêtes/arcs comme liste des paires

Une option est de représenter chaque arête/arc comme une paire d'entiers : les indices des deux sommets aux extrêmités de l'arête/arc en question. Un ensemble des arêtes/arcs est alors représenté par une liste python de paires d'entiers.
Par exemple, l'ensemble d'arcs du graphe du début de sujet peut être représenté en Python par

$$[(0,1),(0,5),(1,2),(2,3),(2,5),(3,2),(4,3),(5,1),(5,4),(5,6),(6,0),(7,8),(8,7)]$$

Et pour représenter l'intégralité du graphe en Python (et notamment le sommet 9 dans l'exemple ci-dessus), il suffit d'indiquer le nombre total de sommets et son ensemble d'arêtes, par exemple avec une paire :

$$(10,[(0,1),(0,5),(1,2),(2,3),(2,5),(3,2),(4,3),(5,1),(5,4),(5,6),(6,0),(7,8),(8,7)])$$

Arcs/arêtes comme matrice d'adjacence

Si un graphe a n sommets, alors on peut représenter l'ensemble des ses arcs ou arêtes par une matrice M de taille $n \times n$, telle que $M_{i,j}$ indique s'il existe un arc/arête entre le sommet numéro i et le sommet numéro j.
Dans ce TP on représentera les matrice d'adjacence en Python par des listes de listes de booléens. Par exemple, pour le graphe du début d'énoncé :

```
[[False, True, False, False, False, True, False, False, False, False],
 [False, False, True, False, False, False, False, False, False, False],
 [False, False, False, True, False, True, False, False, False, False],
 [False, False, True, False, False, False, False, False, False, False],
 [False, False, False, True, False, False, False, False, False, False],
 [False, True, False, False, True, False, True, False, False, False],
 [True, False, False, False, False, False, False, False, False, False],
 [False, False, False, False, False, False, False, False, True, False],
 [False, False, False, False, False, False, False, True, False, False],
 [False, False, False, False, False, False, False, False, False, False]]
```

Remarquez qu'il n'y a ici pas besoin de préciser dans la représentation combien de nœuds a le graphe : il suffit de regarder la taille de la matrice d'adjacence (c'est à dire le nombre de lignes).

Arcs/arêtes comme listes d'adjacence

Une troisième façon de représenter un graphe est de donner, pour chaque sommet (nœud du graphe) d'indice i, l'ensemble des j tels qu'il existe un arc/une arête entre i et j.
Par exemple, pour le graphe du début de sujet on peut utiliser la liste de listes suivante :

```
adj = [[1, 5], [2], [3, 5], [2], [3], [1, 4, 6], [0], [8], [7], []]
```

En effet, les sommets sont numérotés entre 0 et 9 inclus, et pour chaque numéro de sommet i, `adj[i]` contient les numéros j des nœuds tels que (i, j) est un arc du graphe.
Là encore, pas besoin de préciser le nombre de sommets du graphe dans la représentation : il suffit de compter le nombre de listes d'adjacence dans la représentation du graphe.

Exercice 1 (Conversions entre représentations)

1. Écrire une fonction `liste_paires_vers_matrice` qui prend un graphe dans la première représentation et renvoie le même graphe dans la deuxième représentation.
2. Écrire une fonction `matrice_vers_listes_adjacence` qui prend un graphe dans la deuxième représentation et renvoie le même graphe dans la troisième représentation.
3. Écrire une fonction `listes_adjacence_vers_listes_paires` qui prend un graphe dans la troisième représentation et renvoie le même graphe dans la première représentation.
4. Écrire une fonction `listes_paires_vers_listes_adjacence` qui prend un graphe dans la première représentation et renvoie le même graphe dans la troisième représentation.
5. Écrire une fonction `listes_adjacence_vers_matrice` qui prend un graphe dans la troisième représentation et renvoie le même graphe dans la deuxième représentation.
6. Écrire une fonction `matrice_vers_liste_paires` qui prend un graphe dans la deuxième représentation et renvoie le même graphe dans la première représentation.

Exercice 2 (Graphes non-orientés)

1. Que dire de la matrice d'adjacence d'un graphe non-orienté ?
2. Que dire de la représentation en liste de paires d'un graphe non-orienté ?
3. Que dire de la représentation en listes d'adjacence d'un graphe non-orienté ?

Exercice 3 (Bonus : distance entre deux nœuds)

Écrire une fonction `distance(graphe, i, j)` qui prend en entrée un graphe, et deux indices de sommets i et j, et renvoie en sortie la distance entre les deux sommets. (Par exemple, dans le graphe de l'exercice du cours, la distance entre Harper et Alex est de 2). Nous n'avons pas encore vu le cours sur comment coder cette fonction. Mais essayer de la coder vous permettra de vous poser des questions intéressantes telles que :

1. Quelles sont les principales difficultés auxquelles vous êtes confronté·e ?
2. Vers quelle façon de représenter les graphes vous êtes vous tourné·e pour coder votre fonction ? Quels avantages et désavantages vous semble avoir cette représentation pour résoudre ce problème ?