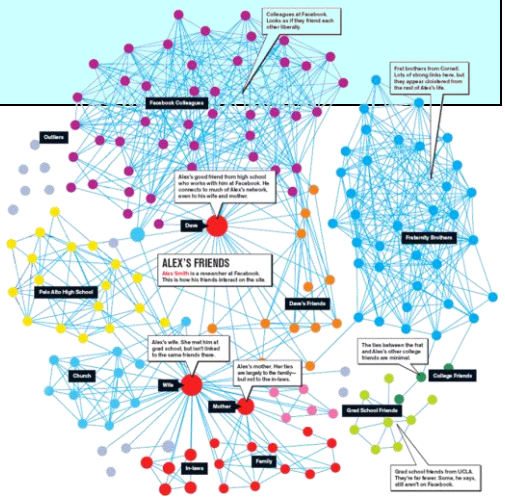
Cours – Graphes



Objectifs :

- ⇨ Découvrir le concept de graphe et le vocabulaire associé
- ⇨ Voir plusieurs méthodes pour représenter un graphe en mémoire

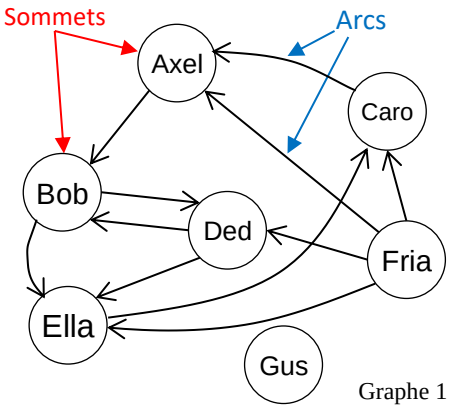
I – Qu'est-ce qu'un graphe ?

1) Définitions

Un **graphe** est un ensemble de **sommets** reliés entre eux par des **arcs**.

Les **sommets**, parfois appelés nœuds (*nodes* ou *vertices* en anglais), ont généralement une **étiquette** associée, porteuse d'information. Ils sont représentés sur un schéma par des ronds ou des rectangles.

Les **arcs** (*edges* en anglais) représentent les liens entre les sommets. Ils sont représentés sur les schémas par de traits droits ou courbes entre les sommets.



Graphe 1

On parle de **graphe orienté** lorsque l'on distingue un sens pour les arcs. Dans ce cas, les arcs sont représentés sur les schémas avec une flèche indiquant le sens de la relation entre les sommets.

Deux sommets A et B peuvent ainsi être reliés par un arc allant de A vers B uniquement. On peut également trouver des sommets qui soient reliés par deux arcs : un de A vers B et un de B vers A.

En revanche les **graphes non-orientés** ne possèdent que des arcs non-orientées. Dans ce cas on parle généralement d'**arêtes** pour désigner les arcs.

Remarques :

- Un sommet peut être relié par un arc à lui-même. On parle dans ce cas de **boucle**. Non ne traiterons pas des graphes ayant des boucles cette année.
- Deux sommets peuvent être reliés par plusieurs arcs de même sens. On parle alors d'**arcs multiples**. On se limitera cette année à des **graphes simples**, c'est-à-dire ne contenant pas d'arcs multiples.

2) Vocabulaire des graphes

L'**ordre d'un graphe** est égal au nombre de ses sommets.

Ex : L'ordre du graphe 1 est ..... , celui du graphe 2 est .....

Lorsqu'il y a un arc entre un sommet A et un sommet B, on dit que A et B sont **adjacents**.

L'ensemble des sommets adjacents à un sommet A constitue les **voisins** de A.

Ex : Les voisins de Bob sont ..... , et ceux de Ded sont .....

Le **degré** d'un sommet est égal au nombre de ses voisins.

Ex : Le degré de Ded est ....., celui de Ella ..... et celui de Fria ..... .

Un sommet qui n'a pas de voisins (de degré 0) est dit **isolé**.

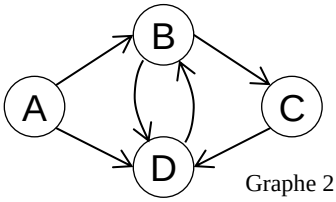
Ex : Dans le graphe 1, les sommets isolés sont : .....

Dans un graphe orienté, pour un sommet A, les sommets vers lesquels il existe un arc de A vers ce sommet sont les **descendants** du sommet A. Les sommets pour lesquels il existe un arc qui va vers A sont les **ascendants** de A.

Ex : Les descendants de Ded sont ..... , les ascendants de Ella sont .....

Un **chemin** entre deux sommets A et B est une suite finie de sommets, reliés deux à deux, et menant de A vers B. La *longueur* du chemin est égale au nombre d'arc (ou d'arêtes) dans le chemin.

Ex : Chemin A → B → C → D ou Axel → Bob → Ded → Ella → Caro.



Graphe 2

Un chemin est dit **simple** s'il n'emprunte pas deux fois le même arc et **élémentaire** s'il ne passe pas deux fois par le même sommet.

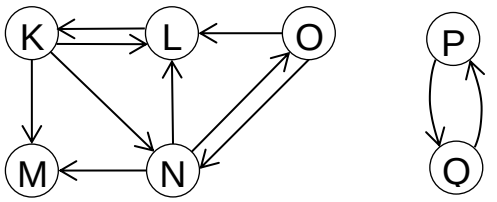
Ex : Chemins simples reliant A à C : .....

Chemins élémentaires reliant A à C : .....

Un chemin simple reliant un sommet à lui-même et contenant au moins un arc est appelé un **cycle**.

Ex : Les cycles pour Bob sont .....

Un graphe non orienté est **connexe** si pour toute paire (A, B) de sommets, il existe un chemin de A à B. Il est non-connexe dans le cas contraire.
Un graphe orienté est connexe si le graphe non orienté obtenu en ne tenant pas compte du sens des arêtes est connexe.



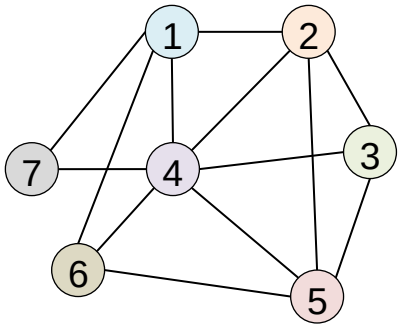
Graphe 3

Ex : Graphe(s) connexe(s) : .....

Graphe(s) non-connexe(s) : .....

Un graphe orienté est **fortement connexe** si pour toute paire (A, B) de sommets, il existe un chemin de A à B et un chemin de B à A.

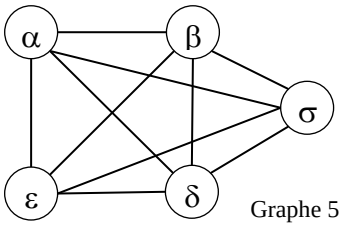
Ex : Graphe(s) fortement connexe(s) : .....



Graphe 4

Un graphe est dit **complet** si deux sommets quelconques distincts sont toujours adjacents.
Autrement dit, tous les sommets sont reliés deux à deux par une arête.

Ex : Graphe(s) complet(s) : .....



Graphe 5

Sous-graphe : graphe obtenu en enlevant quelques nœuds et/ou arêtes du graphe de départ

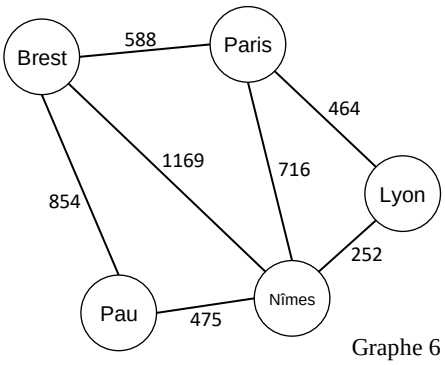
Ex : Dans le graphe 3, (K, L, M) forment un sous-graphe. De même pour (P, Q), ou même pour (N, O, Q)

Une **clique** est un sous-graphe complet à l'intérieur d'un graphe.

Ex : .....

Un graphe peut être **valué** (ou **pondéré**). Dans ce cas on associe à chaque arc ou arête une valeur numérique.

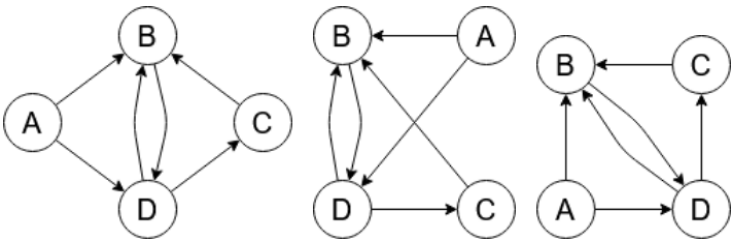
Voir l'exemple ci-contre. Ce type de graphe est utilisé lorsqu'on veut par exemple minimiser une grandeur (parcours le plus court en km ou le plus rapide en temps)



Graphe 6

Remarques :

- Un arbre est un donc un graphe ....., ... et ..... .
- Un graphe est entièrement défini par la liste de ses sommets et de ses arcs. Un même graphe peut ainsi avoir de nombreuses représentations graphiques. Ainsi les 3 graphes ci-dessous sont équivalents.



II – Représentation des graphes

Voyons maintenant comment représenter les graphes en mémoire.

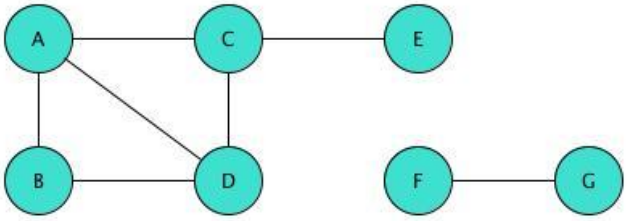
1) Matrices d'adjacence

On considère un graphe d'ordre n dont les sommets sont numérotés de 0 à n-1.

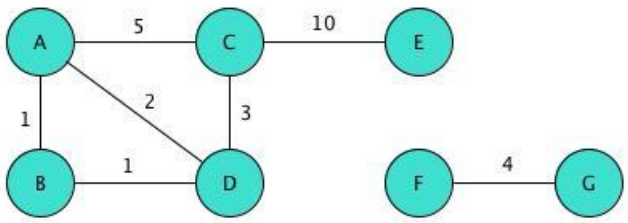
On appelle **matrice d'adjacence** associée à ce graphe la matrice M dont le terme $m_{i,j}$ vaut Vrai (ou 1) si les sommets sont reliés par une arête et Faux (ou 0) sinon. *i* et *j* variant de 0 à n-1.

Dans le cas d'un graphe valué, on mettra la valeur de l'arc à la place du 1 quand un arc est présent ou None (ou une valeur interdite comme 0) sinon.

Exemples :

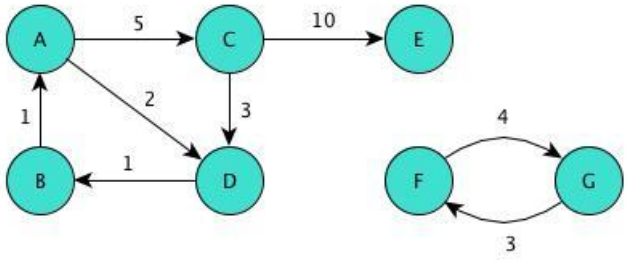


$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$



$M =$

	A	B	C	D	E	F	G
A							
B							
C							
D							
E							
F							
G							



$M =$

	A	B	C	D	E	F	G
A							
B							
C							
D							
E							
F							
G							

Le sens de lecture est important

Remarque : Pour les graphes non-orientés, la matrice d'adjacence est symétrique.

Avantages et inconvénients :

- Cette représentation est **simple** à comprendre et à mettre en œuvre.
- Le calcul matriciel, à l'aide d'outils (comme les produits et puissances de matrices) permet de déterminer facilement diverses propriétés du graphe (nombre de chemins de longueur donnée, présence de cycles etc.). Cependant le calcul matriciel est coûteux en temps (complexité en $O(n^2)$ pour une matrice de taille $n \times n$).
- De même la complexité pour parcourir les voisins est en O(n) puisqu'on doit parcourir toute la ligne de la mattrice.
- En général, la matrice est presque entièrement « vide », c'est-à-dire que la matrice est principalement composée de 0. Ainsi cette représentation n'est **pas du tout efficace en termes d'espace mémoire**. Si on souhaite par exemple faire le graphe de tous les élèves du lycée en mettant une arête entre tous ceux qui se fréquentent, on aurait une matrice de 1400 x 1400 soit environ 2 millions d'éléments.

On peut utiliser la programmation objet pour représenter un graphe avec une matrice d'adjacence :

```
class Graphe_m():
    """Classe représentant un graphe orienté par une matrice d'adjacence.
       les sommets sont numérotés de 0 à n-1."""
    def __init__(self, n:int) -> None:
        """Crée un graphe de n sommets sans aucun arc"""
        self.n = n                                  # Ordre du graphe
        self.m = [[False]*n for _ in range(n)]   # Matrice d'adjacence

    def ajouter_arc(self, s1:int, s2:int) -> None:
        """Ajoute un arc entre le sommet s1 et s2. Attention : l'arc entre
           s2 et s1 n'est pas crée."""
        self.m[s1][s2] = True

    def arc_existe(self, s1: int, s2: int) -> bool:
        """Renvoie True si un arc existe entre les sommets s1 et s2."""
        return self.m[s1][s2]
```

Application 1 :

1) En utilisant la classe précédente, quelle doit être la série d'instructions permettant de créer le graphe 3 du cours ?
2) Ecrire une méthode `afficher()` qui affiche le graphe sous la forme d'une ligne par sommet avec pour chaque sommet, la liste de ses successeurs.

Par exemple pour le graphe 3, cela donnerait :

```
0 -> 1 3
1 -> 2 3
2 -> 3
3 -> 1
```

3) Ecrire une méthode `voisins(s)` qui renvoie la liste de tous les voisins du sommet numéro `s`.

2) Liste de successeurs

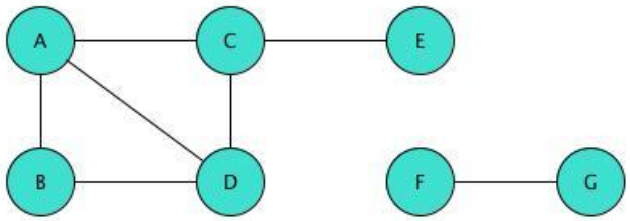
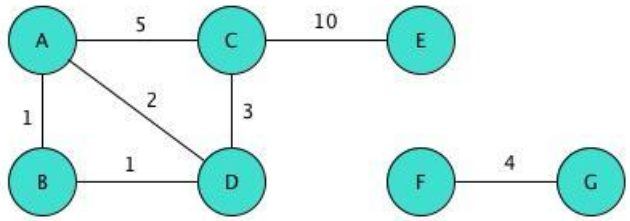
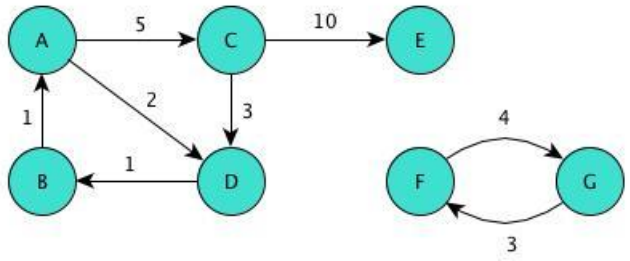
Dans cette nouvelle représentation, on va utiliser une liste ou mieux, un dictionnaire python qui à chaque sommet associe une liste de tous ses successeurs.

On appelle **liste de successeurs** ou **dictionnaire d'adjacence** la liste L ou le dictionnaire D qui à chaque sommet du graphe associe la liste de tous ses successeurs.

Nous privilégierons ici l'utilisation de dictionnaires qui permettent d'avoir des sommets nommés (et pas uniquement numérotés de 0 à n-1. A chaque sommet on associera un ensemble python plutôt qu'une liste car les opérations sur les ensembles se font en temps constant.

Dans le cas des graphes valués, on associera à chaque arc sa valeur en utilisant un tuple (arc, valeur).

Exemples :

<table>
<tr>
<td> 
</td>
<td><pre>graphe = { 'A' :{'B', 'C', 'D'},
           'B' :{'A', 'D'},
           'C' :{'A', 'D', 'E'},
           'D' :{'A', 'B', 'C'},
           'E' :{'C'},
           'F' :{'G'},
           'G' :{'F'}}</pre></td>
</tr>
<tr>
<td> 
</td>
<td><pre>graphe = {</pre></td>
</tr>
<tr>
<td> 
</td>
<td><pre>graphe = {</pre></td>
</tr>
</table>

Avantages et inconvénients :

- Cette représentation est **un peu plus complexe** à comprendre et à mettre en œuvre.
- Les calculs sont de complexité optimale car par exemple pour parcourir tous les voisins, on effectue une opération par voisin seulement.
- La **complexité en espace mémoire est bonne si le graphe contient peu d'arcs ou d'arêtes**, mais dans le cas contraire, la taille de la liste de successeurs pourra être bien plus importante que celle d'une matrice d'adjacence (car stocker la référence vers un sommet coûte plus de mémoire que de stocker la simple présence ou absence d'arête (qui peut se stocker sur 1 bit)).

Application 2 :

1) Ecrire une classe `Graphe_s` implémentant les listes de successeurs. Pour cette première question, on écrira juste le constructeur de la classe qui crée un attribut `succ` initialisé au dictionnaire vide (`{}`).
2) Ecrire une méthode `ajouter_sommet(s)` qui vérifie si le sommet `s` (qui peut être n'importe quelle valeur non mutable (nombre, chaîne, …)) est déjà dans le dictionnaire `self.succ` et si ce n'est pas le cas l'ajoute avec une valeur initialisée à l'ensemble vide (obtenu en python avec l'instruction `set()`).
3) Implémenter une méthode `ajouter_arc(s1, s2)` qui ajoute les sommets `s1` et `s2` s'ils n'existent pas (en appelant la méthode `ajouter_sommet` précédente) puis ajoute un arc entre `s1` et `s2` en ajoutant une valeur `s2` à l'ensemble de `s1`. On rappelle que l'ajout dans un ensemble se fait avec la méthode `add(`*`valeur`*`)`.
4) La série d'instructions permettant de créer le graphe 3 du cours est-elle différente de celle utilisée pour l'application 1 (avec les matrices d'adjacence) ?
5) Ecrire une méthode `voisins(s)` qui renvoie la liste de tous les voisins du sommet `s`.
6) Ecrire une méthode `afficher()` qui affiche le graphe sous la forme d'une ligne par sommet avec pour chaque sommet, la liste de ses successeurs.

Par exemple pour le graphe 3, cela donnerait :

```
A -> {'B', 'D'}
B -> {'D', 'C'}
D -> {'C'}
C -> {'D'}
```

7) D'après-vous dans l'affichage de la méthode précédente, pourquoi le sommet `'D'` est-il affiché avant le sommet `'C'` ?

Références :

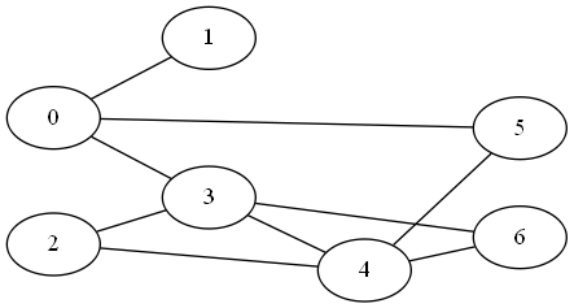
Ensemble de vidéos sur les graphes et leurs algorithmes : https://www.youtube.com/watch?v=YYv2R1cCTa0
Autre vidéo présentant les graphes pour NSI : https://www.lumni.fr/video/une-introduction-aux-graphes
Cours avec exercices : https://kxs.fr/cours/graphes/graphes

Exercices – **Graphes**

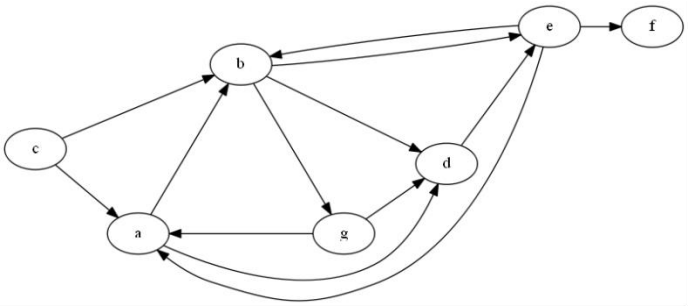
Exercice 1 :

Pour chacun des graphes suivants donner :

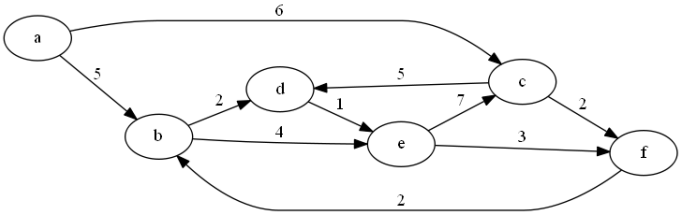
- son ordre
- le degré de chaque sommet
- un exemple de chemin et sa longueur
- un exemple de cycle s'il en existe



Graphe 1



Graphe 2



Graphe 3

Exercice 2 : multiples possibilités

Dessiner tous les graphes non-orientés possédant exactement 3 sommets.

Exercice 3 : Ordre des sommets et nombre d'arêtes

Trouver la relation mathématique existant entre le nombre d'arêtes dans un graphe non-orienté et la somme de l'ordre des sommets dans ce graphe.

Exercice 4 : Les graphes et leurs représentations

1) Donner la représentation matricielle et la représentation par liste de successeurs des graphes de l'exercice 1.
2) Tracer les graphes dont les implémentations sont les suivantes. On tiendra compte des remarques du cours pour trouver le type du graphe dont la représentation est donnée.

Graphe 1 : $\begin{bmatrix} 0 & 0 & 3 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 2 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 9 \\ 3 & 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}$ Graphe 2 : $\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$

Graphe 3 :

```
{A : {('B', 3), ('C', 8), ('E', 1)},
 B : {('A', 3), ('C', 5), ('D', 1)},
 C : {('A', 8), ('B', 5)},
 D : {('A', 1), ('E', 2)},
 E : {('A', 1), ('D', 2)}}
```

Exercice 5 : Graphes non-orientés

Que doit-on modifier dans classes `Graphe_m` et `Graphe_s` des applications du cours pour implémenter des graphes non-orientés ? Créer les classes `Graphe_no_m` et `Graphe_no_s` et tester avec quelques exemples.

Exercice 6 : Qui a tué le duc de Densmore?

Il y'a quelques années, le Duc de Densmore périt dans l'explosion qui détruisit sa chambre. Son testament fut détruit ; or celui-ci avait tout pour déplaire à l'une de ses huit ex-femmes : Ann, Betty, Cynthia, Diana, Emily, Felicia, Georgia et Helen.
Peu avant le crime, elles étaient toutes venues au château mais elles jurèrent que ce fut la seule fois où elles s'y étaient rendues.
Elles peuvent donc être toutes coupables, mais la pose de la bombe a forcément nécessité plus d'une visite. Donc la coupable a menti : elle est venue plus d'une fois.
Aucune des femmes ne peut dire avec précision quand elle est venue au château mais elles se souviennent toutes qui elles ont rencontrées :
– Ann dit avoir rencontré Betty, Cynthia, Emily, Felicia et Georgia ;
– Betty dit avoir rencontré Ann, Cynthia et Helen ;
– Cynthia dit avoir rencontré Ann, Betty, Diana, Emily et Helen ;
– Diana dit avoir rencontré Cynthia et Emily ;
– Emily dit avoir rencontré Ann, Cynthia, Diana et Felicia ;
– Felicia dit avoir rencontré Ann et Emily ;
– Georgia dit avoir rencontré Ann et Helen ;
– Helen dit avoir rencontré Betty, Cynthia et Georgia.

Qui a tué le Duc ? Il se peut qu'un graphe de la situation puisse vous aider…

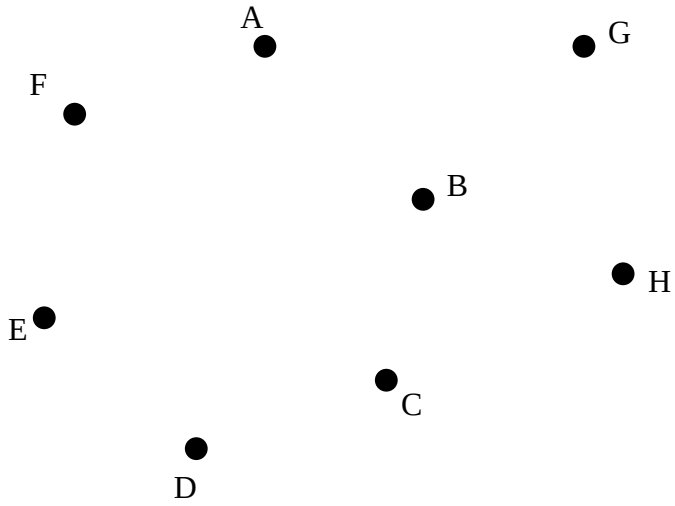
NB : On considère qu'il n'y a qu'une seule coupable, donc une seule menteuse.

Si vous avez du mal à trouver, vous pouvez suivre les indications qui suivent, mais essayez d'abord de trouver sans aide et cherchez bien entre chaque indication !

1) Compléter le tableau des rencontres ci-dessous.

	Ann	Betty	Cynthia	Diana	Emily	Felicia	Georgia	Helen
Ann								
Betty								
Cynthia								
Diana								
Emily								
Felicia								
Georgia								
Helen								

2) Vérifier que les témoignages concordent : si X a vu Y alors Y a bien vu X.
3) Dresser le graphe des rencontres en ajoutant les arêtes au graphe ci-dessous.



Un tel graphe est appelé graphe d'intervalle. Les graphes d'intervalles ont des propriétés dont certaines vont nous être utiles. C'est en regardant la structure de ce graphe et l'impossibilité que certains sous-graphes existent qu'on va pouvoir déterminer la coupable.

4) Est-il possible qu'un carré comme ABHG existe ? Que peut-on en conclure sur Ann, Betty, Helen et Georgia ?
5) Existe-t-il d'autres carrés ? En déduire une liste des menteuses potentielles.
6) Le sous-graphe ABCDEF est-il possible (considérer les intervalles disjoints de B, D et F puis essayer de mettre ceux de E, C et A) ? Que peut-on en conclure ?
7) En reprenant les réponses des deux questions précédentes, en déduire qui est l'assassin.