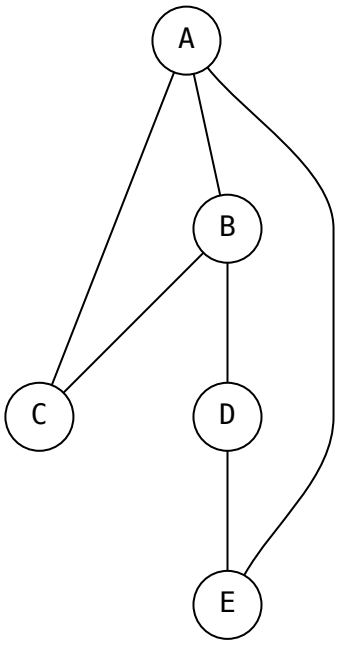
ITC MPSI
Notes de cours sur les graphes

Les *graphes* sont une manière, en informatique, de représenter des données avec des liens.

Exemple 1



I) Présentation générale

I.1) Définitions et vocabulaire

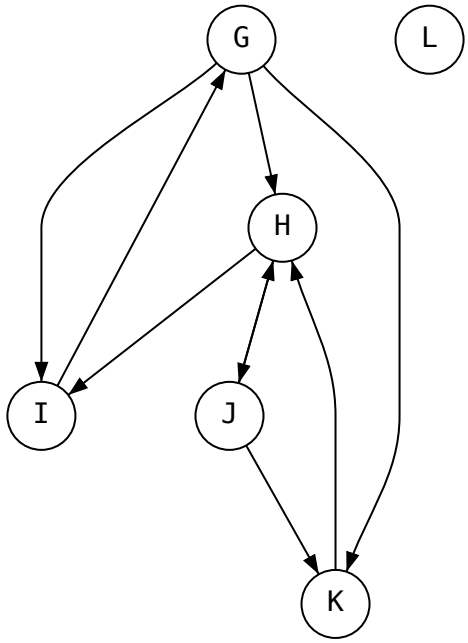
Définition 2 [Graphes]

> Un *graphe* est un couple $G = (S, A)$, où S est un ensemble fini de *sommets* (ou *noeuds*) et $A \in S \times S$ est l'ensemble des *arrêtes*. Chaque arrête relie deux sommets *distincts*.
>
> On dit que le graphe est *orienté* si les arrêtes ont un sens ("que ce sont des flèches"). On les appelles alors souvent *arcs*.

Définition 3 [Voisin]

> Si on a une arrête (s, s'), on dit que s' est un voisin de s.

Exemple 4



Ici, les voisins de `H` sont `J` et `I`.

Définition 5 [Degrés d'un sommet]

> Dans un graphe non orienté, le degrés d'un sommet s, noté $d(s)$, est le nombre d'arrêtes qui le touchent.
>
> Dans un graphe orienté, on note $d_+(s)$ le degrés entrant et $d_-(s)$ le degrés sortant, qui sont respectivement le nombre d'arcs qui arrivent et qui partent de s.

Exemple 6

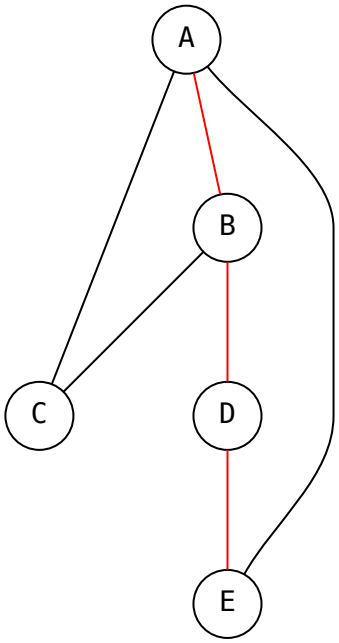
- $d(B) = 3, d(D) = 2$
- $d_+(G) = 1, d_-(G) = 3, d_+(L) = 0$

II) Chemins, cycles et connexité

Définition 7 [Chemin]

> Un *chemin* dans un graphe est une suite finie d'arrêtes consécutives $(s_1, s_2), (s_2, s_3), ..., (s_{n-1}, s_n)$.
>
> On parle de chemin *élémentaire* lorsque les sommets $s_1, ..., s_n$ sont distincts, et de chemin *simple* lorsque les arrêtes sont distinctes (ces mots de vocabulaire ne sont pas au exigibles).

Exemple 8

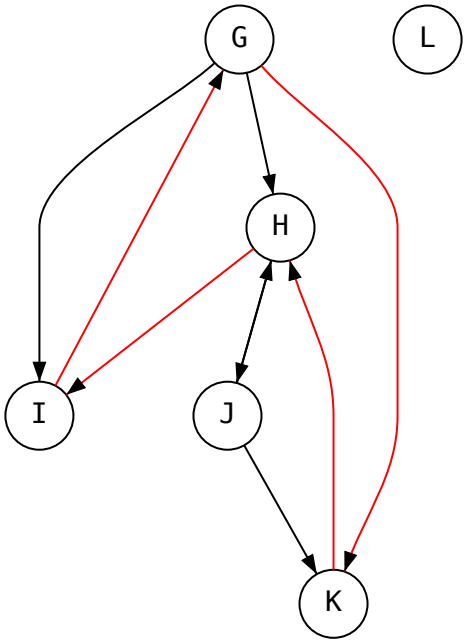


AB, BD, DE est un chemin.

Définition 9 [Cycle]

> Un *cycle* est un chemin simple dont le premier et le dernier sommet sont les mêmes.

Exemple 10



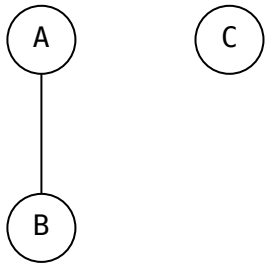
HI, IG, GK, KH est un cycle.

Définition 11 [Connexité]

> Un graphe non orienté est *connexe* si, pour tout sommets s_1, s_2, il existe un chemin de s_1 à s_2.

Exemple 12

L'exemple 1 est connexe.



Ce graphe n'est pas connexe.

Remarque 13

La notion de connexité est hors-programme pour les graphes orientés. Pour donner une idée, on parlera de graphe *fortement connexe* si il existe un chemin de s_1 à s_2 et de s_2 à s_1.

III) Représentation informatique

Il y a plusieurs manières de représenter les graphes en Python. On va en voir deux principales.

III.1) Représentation par liste d'adjacence

L'idée ici, est de stocker, pour chaque sommet, l'ensemble de ses voisins. Dans le cas des graphes non-orienté, on stockera en général les connections dans les deux sens.

Par exemple, pour le graphe de l'ex 1, la liste d'adjacence du sommet `A` est :

```
['B', 'C', 'E']
```

Ensuite, on va regrouper toutes ces listes d'adjacence dans un dictionnaire, pour avoir la représentation suivante :

```
g = {
    'A': ['B', 'C', 'E'],
    'B': ['A', 'C', 'D'],
    'C': ['A', 'B'],
    'D': ['B', 'E'],
    'E': ['A', 'D']
}
```

Exercice : faire la même chose avec l'exemple 2.

Remarque 14

Dans le cas où les sommets sont numérotés de 0 à n-1, on peut utiliser une liste à la place d'un dictionnaire.

Remarque 15

La représentation par liste d'adjacence est très adaptée lorsqu'on veut souvent accéder aux voisins d'un sommet particulier, par exemple lors d'une exploration du graphe.

III.2) Matrice d'adjacence

Dans le cas de la matrice d'adjacence, on va toujours numéroter les sommets de 0 à $n-1$.

On va alors se donner une matrice de taille $n \times n$, et à la position (i, j), on va placer 1 si l'arrête $i \to j$ si elle existe, et 0 sinon.

Dans le cas de l'exemple 1 :

```
g = [[0, 1, 1, 0, 1],
     [1, 0, 1, 1, 0],
     [1, 1, 0, 0, 0],
     [0, 1, 0, 0, 1],
     [1, 0, 0, 1, 0]]
```

Remarque 16

La matrice d'adjacence d'un graphe non-orienté est symétrique.

Exercice : faire la même chose avec l'exemple 2.

Remarque 17

La représentation par matrice d'adjacence est adaptée lorsqu'on a souvent besoin de tester si deux sommets sont adjacents.

IV) Pondération

IV.1) Applications

Les graphes se retrouvent un peu partout en informatique :

- Pour tout ce qui est carte. Une carte de transports en commun est un graphe par exemple.
- Internet, à deux niveaux :
  - l'architecture physique d'internet est un grand graphe, avec des serveurs, des routeurs et des terminaux reliés par de la fibre optique, des ondes, etc
  - le web : les pages webs, et les liens entre elles, sont bien représentées par des graphes.

Dans tous ces exemples, des algorithmes sur les graphes (on en étudiera quelques uns) sont utilisés. Par exemple, la recherche d'itinéraire de votre appli de carte préférée, le programme qui décide comment acheminer votre message whatsapp, ou encore les moteurs de recherche (Google).

Dans beaucoup de ces applications, toutes les arrêtes ne sont pas égales. Par exemple, un trajet en train peut prendre deux fois plus longtemps qu'un autre.

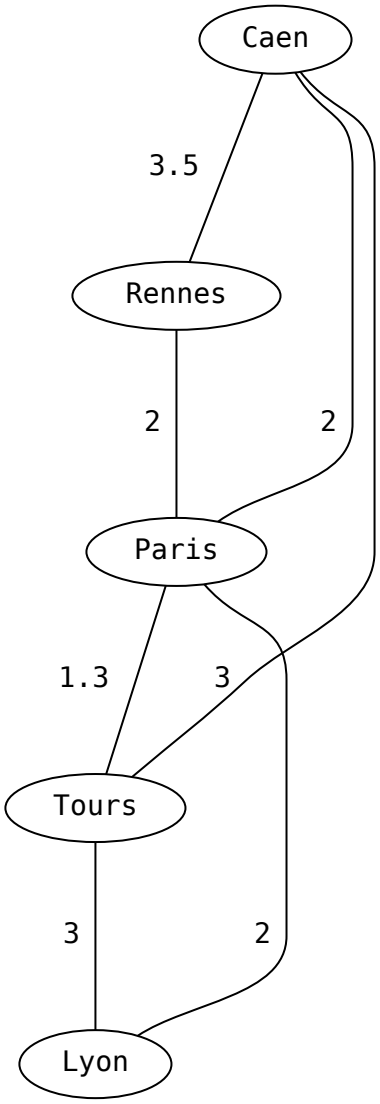
IV.2) Graphes pondérés

Pour représenter ces différences, on va introduire la *pondération*.

Définition 18 [Graphe pondéré]

> Un graphe pondéré est un graphe $G = (S, A)$, orienté ou non, auquel on associe une fonction de pondération $\omega : A \to E$ qui associe un *poids* (ou étiquette) à chaque arrête.
>
> E est ici un ensemble quelconque, mais en pratique, ce sera souvent $\mathbb{N}$ ou $\mathbb{R}^+$.

Exemple 19



Exemple de quelques villes et le temps de trajet en train entre elles. Quels sont les chemins possibles entre Caen et Lyon ? Quel est leur poids ?

Remarque 20

On peut alors parler du poids d'un chemin, d'un cycle, ...

IV.3) Implémentation des graphes pondérés

IV.3.1) Liste d'adjacence

On peut stocker les poids directement dans la liste d'adjacence, en mettant des couples (voisin, poids) au lieu des simples voisins :

Exemple 21

```
g = {
    'Caen': [('Rennes', 3.5), ('Tours', 3.0), ('Paris', 2.0)],
    'Rennes': [('Caen', 3.5), ('Paris', 2.0)],
    'Paris': [('Caen', 2.0), ('Rennes', 2.0), ('Tours', 1.3), ('Lyon', 2.0)],
    'Tours': [('Caen', 3.0), ('Paris', 1.3), ('Lyon', 3.0)],
    'Lyon': [('Paris', 2.0), ('Tours', 3.0)]
}
```

IV.3.2) Matrice d'adjacence

On peut mettre le poids de l'arrête à la place de "1", et utiliser une valeur qui n'est pas dans les poids à la place de "0" (par exemple $+\infty$, où, si ce n'est pas une option, -1).

Exemple 22

```
g = [[ 0.0, 3.5, 2.0, 3.0,-1.0],
     [ 3.5, 0.0, 2.0,-1.0,-1.0],
     [ 2.0, 2.0, 0.0, 1.3, 2.0],
     [ 3.0,-1.0, 1.3, 0.0, 3.0],
     [-1.0,-1.0, 2.0, 3.0, 0.0]]
```