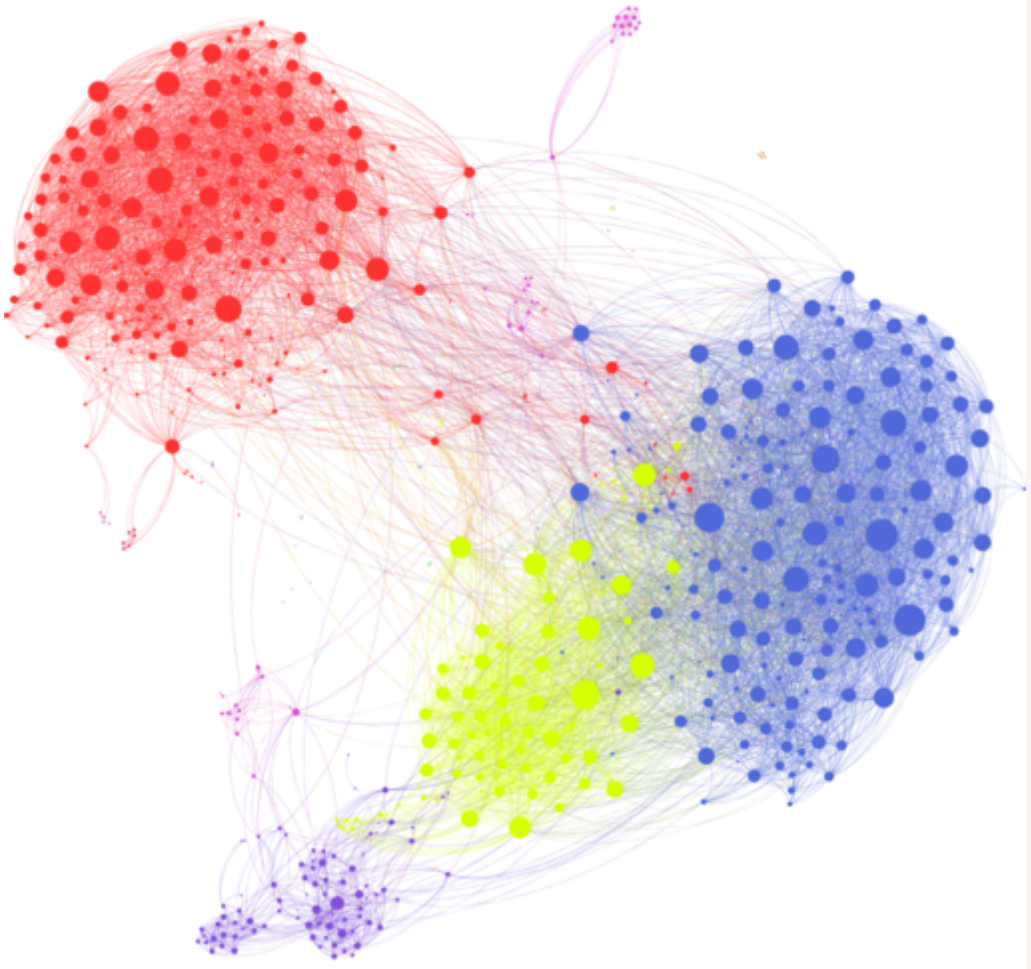
informatique

L2 MIASHS / Semestre 2 - 2019

Mis à jour le 20 janvier 2019 à 21:18

UE42
Algorithmes de graphes



Guillaume CONNAN

UCO
UNIVERSITÉ CATHOLIQUE DE L'OUEST

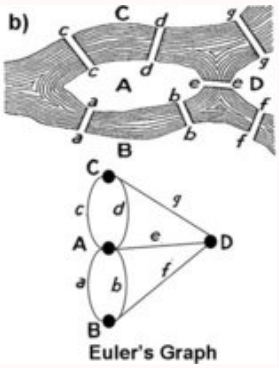
THÈME N° 1

Graphes : généralités

a)

Leonhard Euler [1707-1783]

b)



Euler's Graph

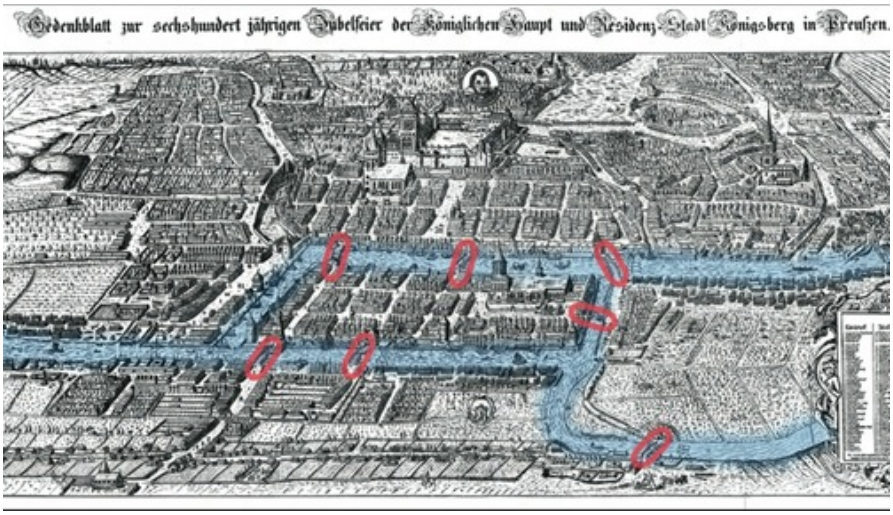
Tout à commencé au XVIII^e siècle avec une histoire de ponts résolue par EULER puis ce gadget s'est fait oublier jusque dans les années 1960 où on a enfin découvert toutes les richesses qui permettent de résoudre de nombreux problèmes informatiquement : circuits électriques, réseaux de transport, réseaux d'ordinateurs, mise au point d'un planning, recherche d'optimum, jeux de stratégie, etc.

Par exemple, vous branchez votre GPS et donnez une destination : paf! Un plus court chemin apparaît...Déterminer si des villes sont « connectables », trouver le plus court chemin d'un point à un autre du réseau sont des exemples de problèmes que nous allons traiter.

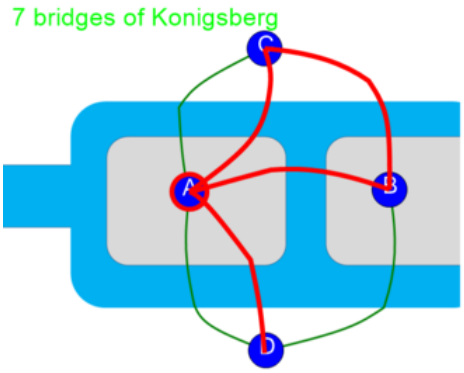
C'est un thème très actuel car chaque jour ou presque naît un nouvel algorithme plus efficace pour résoudre toute sorte de problèmes.

1 Comment un mathématicien regarde une carte

Voici une vue de l'antique ville de Königsberg (aujourd'hui appelée Kaliningrad et située sur l'enclave russe séparant la Lituanie de la Pologne) :



Comme vous pouvez le constater sur cette carte datant de 1651, la rivière Pregel entoure deux îles reliées entre elles et au reste de la ville par sept ponts. Euler, entre deux théorèmes, s'est demandé s'il était possible de se promener en traversant les sept ponts mais sans traverser deux fois le même tout en revenant à son point de départ.
Dans la tête d'Euler, la carte avait plutôt cette allure :



ou plus synthétiquement :

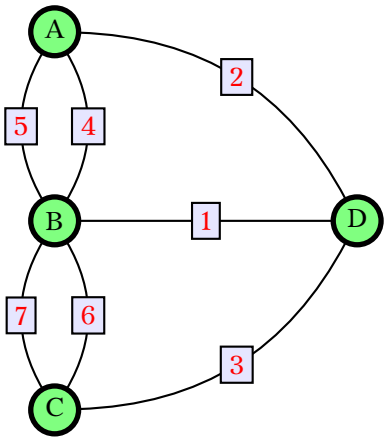


Figure 1.1 – Graphe des ponts de Königsberg

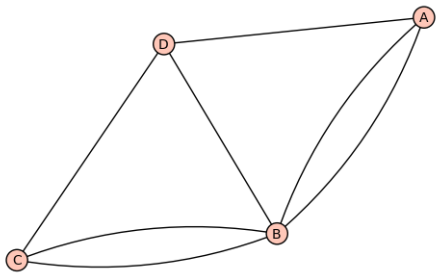
Les étiquettes rectangulaires numérotent les ponts, B correspond à la petite île rectangulaire, A est la rive droite, C la rive gauche et D la seconde île.

Nous aurons besoin d'étudier les graphes un peu plus en profondeur pour répondre au problème (avez-vous une idée ?).
Dans ce graphe, A, B, C et D sont des sommets (*vertices* en anglais) et 1, 2, 3, 4, 5, 6, et 7 sont des arêtes ou arcs (*edges* en anglais).
Un graphe n'est en fait qu'un ensemble de sommets et d'arêtes qu'on dénote par un couple (X,A).
Le logiciel Sage sera notre ami pour vérifier nos raisonnements.
Sage
On commence par déterminer le graphe, par exemple à partir d'un dictionnaire associant à chaque sommet la liste de ses voisins. On peut même oublier certains voisins quand ils ont été évoqués auparavant.

```
sage:  g = Graph({'A':['B','B','D'],'D':['B','C'],'C':['B','B']})
```

On peut obtenir un dessin :

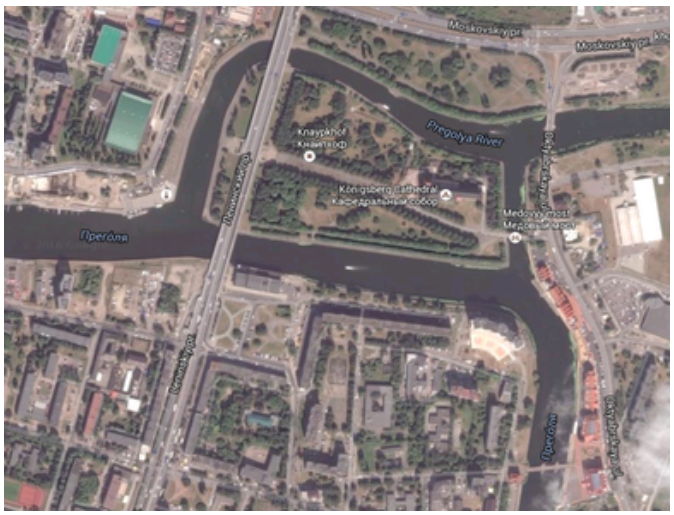
```
sage: g.show()
```



Répondre au problème reviendra, comme nous le verrons bientôt, à déterminer si le graphe est eulérien. Demandons-le à Sage :

```
sage: g.is_eulerian()
False
```

Au fait, voici une vue actuelle de Kaliningrad : que deviendrait ce problème aujourd'hui ?...



Nous distinguerons toute sorte de graphes mais il existe avant tout deux grandes catégories : les graphes orientés et les autres.
Le graphe précédent n'était pas orienté. Si nous étudions par exemple le graphe du réseau internet : chaque sommet représente une page et on relie le sommet a au sommet b s'il existe un lien de la page a vers la page b : ce graphe est orienté.

Recherche

Imaginer un graphe illustrant le programme suivant :

```
L1 x:=0
L2 x:=x+1
L3 y:=2
L4 z:=y
L5 x:=x+2
L6 y:=x+z
l7 z:=4
```

En fait un graphe non orienté est un graphe orienté : chaque arc non orienté en cache deux orientés de manière opposée. C'est pourquoi on appelle les graphes non orientés des graphes symétriques car c'est la « réunion » d'un graphe orienté et de son « symétrique » relativement aux sens de parcours. Par exemple, dans notre problème de pont, on peut parcourir chaque pont dans n'importe quel sens.

Il y a d'autres distinctions :

- *graphes simples* : graphes symétriques où toutes les arêtes sont simples ;
- *multigraphes* : graphes symétriques pouvant contenir des arêtes multiples ;
- *pseudographes* : multigraphes symétriques pouvant contenir des boucles ;
- *multigraphes orientés* : on devine ce que ça désigne...

2 Les bases

2.1 Graphes symétriques

Donnons une définition qui peut s'adapter aux autres types de graphes :

Définition 1 - 1

Un **graphe simple non orienté** est un couple (X, A) où X est un ensemble quelconque et A un ensemble formé de sous-ensembles de deux points de X.
Les éléments de X sont appelés les **sommets** et ceux de A les **arêtes**.

Vous aurez remarqué qu'un graphe n'est pas un dessin mais un couple d'ensembles.
La notation $\mathcal{G} = (X, A)$ signifiera donc que $\mathcal{G}$ est un graphe dont l'ensemble des sommets est X et l'ensemble des arêtes est A.
On conviendra que A peut être vide mais pas X.

Recherche

Pour rigoler, quelle notation est correcte : $A \in X \otimes X$? $A \subseteq X \otimes X$? $A \in \mathcal{P}(X \otimes X)$? Aucune des trois ?

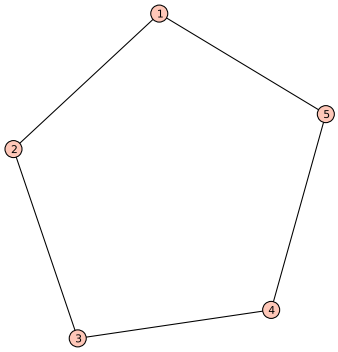
On pourra noter $\binom{X}{k}$ l'ensemble des sous-ensembles de X à k éléments. Ainsi $A \subseteq \binom{X}{2}$.

Définition 1 - 2

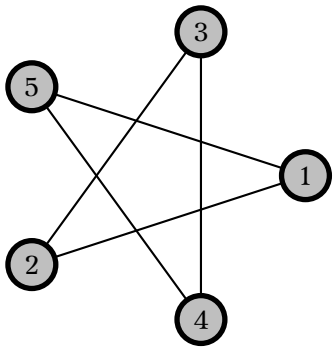
Si $\{s, t\}$ est une arête d'un graphe $\mathcal{G}$, on dit que les sommets s et t sont **adjacents** ou que s est un **voisin** de t ou que l'arête $\{s, t\}$ est **incidente** aux sommets s et t ou que s et t sont les **extrémités** de l'arête.

Par exemple, comment dessiner le graphe $\mathcal{G} = \big(\{1,2,3,4,5\}, \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,1\}\}\big)$?
Demandons à Sage :

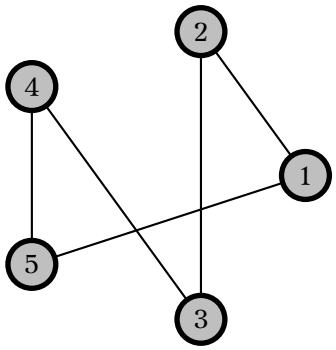
```
1 sage: g=Graph()
2 sage: g.add_vertices([1,2,3,4,5])
3 sage: g.add_edges([(1,2),(2,3),(3,4),(4,5),(5,1)])
4 sage: g.show()
```



Mais on aurait très bien pu dessiner ça :

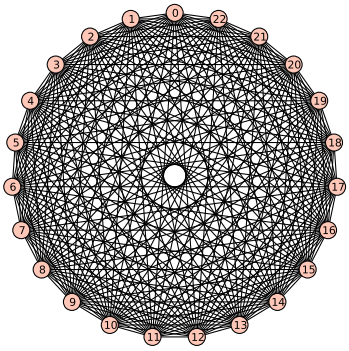


ou ça :



Ce sont les représentations du même graphe $\mathcal{G}$: même ensemble de sommets, même ensemble d'arêtes.
Parfois les dessins deviennent compliqués mais peuvent avoir un intérêt esthétique...Admirez par exemple le graphe complet d'ordre 23 (noté aussi K_{23} en l'honneur de C.Kuratowski) :

```
sage: h=graphs.CompleteGraph(23)
sage: h.show()
```



Définition 1 - 3

Le **degré** d'un sommet d'un graphe symétrique est le nombre d'arêtes qui lui sont incidentes. (En cas de boucle, celles-ci comptent pour deux unités).
On note le degré du sommet s $\deg_{\mathcal{G}}(s)$ ou $\deg(s)$ quand il n'y a pas d'ambiguïté.
Un sommet de degré zéro est dit **isolé**.
Un sommet de degré un est une **feuille** (on dit aussi qu'il est **pendant**).

Par exemple, chaque sommet du graphe précédent a pour degré 22. Dans le graphe des ponts, C, A, et D ont pour degré 3 et B a pour degré 5.

Définition 1 - 4

Notons s_1, s_2,...,s_n les sommets d'un graphe $\mathcal{G}$ dans un ordre quelconque alors la suite :

$$\left(\deg_{\mathcal{G}}(s_1), \deg_{\mathcal{G}}(s_2), ..., \deg_{\mathcal{G}}(s_n)\right)$$

est appelée le **score** du graphe $\mathcal{G}$.

L'ordre n'est pas important. On a cependant l'habitude de donner les scores dans l'ordre croissant.
Bon, maintenant, on peut observer la somme des degrés de chaque sommet. Pour le graphe du pont on obtient $3+3+3+5=14$...et il y a 7 arêtes...
Pour le graphe en pentagone, $2+2+2+2+2=10$ ...et il y a 5 arêtes...
Pour K_{23}...euh...je vous laisse le faire.
C'est assez immédiat : chaque arête contient deux sommets et intervient donc deux fois dans le calcul du degré de chacune de ses extrémités.
On en déduit le théorème suivant :

Théorème 1 - 1

Pour tout graphe $\mathcal{G}=(X,A)$ symétrique, nous avons :

$$\sum_{s\in X} \deg_{\mathcal{G}}(s) = 2\,\mathrm{Card}(X)$$

Ainsi, la somme des degrés d'un graphe symétrique est paire. On en déduit le lemme suivant :

Théorème 1 - 2

Théorème des poignées de main
Tout graphe fini possède un nombre pair de sommets de degré impair.

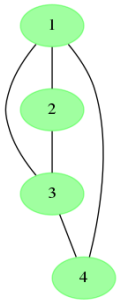
Proposez une démonstration...

Définition 1 - 5

Un graphe simple est dit **régulier** si son score est composé de nombres tous égaux.
Un graphe simple est dit ***n*-régulier** si son score est composé de nombres tous égaux à n.

2 2 Classe Python pour les graphes simples

Parallèlement à notre utilisation de Sage, nous allons fabriquer nos propres outils Python. Nous créons une classe **`Graphe`**. Un graphe sera créé à partir d'un dictionnaire d'adjacence où chaque sommet sera associé aux adjacents qui lui sont supérieurs dans l'ordre lexicographique pour éviter les redondances. Par exemple, pour obtenir le graphe suivant :



on rentrera :

```
g = Graphe({'1': {'2', '4', '3'}, '2': {'3'}, '3': {'4'}})
```

Voici le début de la classe qu'il faudra développer au fur et à mesure du cours :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from graphviz import Graph
from dataclasses import dataclass
from typing import Dict, Set


@dataclass
class Graphe:
        """
        On définit un graphe à l'aide du dictionnaire des adjacents
        Par exemple :
        g = Graphe({{'1': {'2', '4', '3'}, '2': {'3'}, '3': {'4'}}})
        """
        dict_adj: Dict[str, Set[str]]

        def sommets(self) -> Set[str]:
                """
                Donne l'ensemble des sommets
                """
                s = set(self.dict_adj.keys())
                for v in self.dict_adj.values():
                        s |= v
                return s

        def adjacents(self, a):
                """
                Méthode pour avoir l'ensemble des adjacents d'un sommet
                """
                s = set([])
                if a in self.sommets():
                        if a in self.dict_adj:
                                s = set(self.dict_adj[a])
                        for v in self.dict_adj:
                                if a in self.dict_adj[v]:
                                        s |= {v}
                return s

        def __repr__(self):
                """ Affichage dans le terminal"""
                r = ''
                for v in sorted(list(self.sommets())):
                        r += ('\t' + v + ' -> { ')
                        for u in sorted(list(self.adjacents(v))):
                                r += (u + ' ')
                        r += '}\n'
                return r

        def dot(self, name: str = 'graphe') -> None:
                """
                Sortie graphique, ici au format PNG, à l'aide de Graphviz
                """
                d = Graph()
                d.node_attr.update(style='filled', color="#00ff005f")
                for v in self.sommets():
                        d.node(v)
                        for u in self.adjacents(v):
                                if u > v:
                                        d.edge(v, u)
                d.render(name, view=True, format='png')
```

On obtient ainsi par exemple :

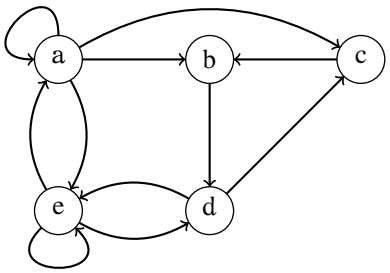
```
In [74]: g = Graphe({'1': {'2', '4', '3'}, '2': {'3'}, '3': {'4'}})

In [75]: g
Out[75]:
        1 -> { 2 3 4 }
        2 -> { 1 3 }
        3 -> { 1 2 4 }
        4 -> { 1 3 }

In [76]: g.dot('graphe1')

In [77]: g.sommets()
Out[77]: {'1', '2', '3', '4'}

In [78]: g.adjacents('4')
Out[78]: {'1', '3'}
```

2 3 Graphes orientés

Quelques menus changements sont à noter : cette fois, il faut tenir compte de l'ordre.

Définition 1 - 6

Un **graphe orienté** est un couple (X, A) où X est un ensemble quelconque et A un sous-ensemble de XX appelé ensembles des **arcs**.
Les éléments de X sont toujours appelés les **sommets** et ceux de A sont maintenant des **arcs** (pensez flèches).
Un arc $a = (s, t)$ a pour **origine** le sommet s et pour **extrémité** le sommet t.
On note $or(a) = s$ et $ext(a) = t$.
Le sommet t est un **successeur** de s qui est lui un **prédécesseur** de t.
Un **chemin** c du graphe $\mathcal{G} = (X, A)$ est une suite finie d'arcs a_1, a_2,...,a_n telle que, $\forall i,\ 1 \leqslant i \leqslant n,\ or(a_{i+1}) = ext(a_i)$. L'origine d'un chemin est l'origine du premier arc et son extrémité est l'extrémité du dernier arc.
Un chemin est **simple** s'il ne comporte pas plusieurs fois le même arc et il est **élémentaire** s'il ne rencontre pas plusieurs fois le même sommet.
La **longueur** d'un chemin est son nombre d'arcs.
Un chemin tel que $or(c) = ext(c)$ est appelé un **circuit**.



Définition 1 - 7

Soit $\mathcal{G} = (X, A)$ un graphe orienté. Le **degré d'entrée** (ou **demi-degré intérieur**) d'un sommet s est noté $\deg^-_{\mathcal{G}}(s)$ et correspond au nombre d'arêtes dont s est l'extrémité.
Le **degré de sortie** (ou **demi-degré extérieur**) d'un sommet s est noté $\deg^+_{\mathcal{G}}(s)$ et correspond au nombre d'arêtes dont s est l'origine.

Amusez-vous avec le graphe représenté précédemment.
À partir du théorème 1 - 4 page 8, on obtient le corollaire suivant :

Théorème 1 - 3

Pour tout graphe $\mathcal{G} = (X, A)$ orienté, nous avons :

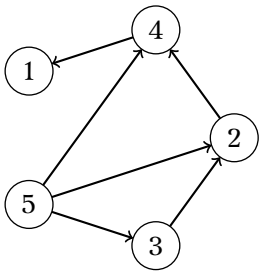
$$\sum_{s \in X} \deg^-_{\mathcal{G}}(s) + \sum_{s \in X} \deg^+_{\mathcal{G}}(s) = \text{Card}(X)$$

Parfois, le fait qu'un graphe soit orienté ou non n'est pas important : on travaille alors avec le **graphe symétrique sous-jacent** obtenu en remplaçant les arcs par des arêtes...

Définition 1 - 8

On dit que x_k est un **descendant** du sommet x_i si, et seulement si, il existe un chemin d'origine x_i et d'extémité x_k ; on dit aussi que x_k est **accessible** à partir de x_i. Le sommet x_i est alors un **ascendant** ou un ancêtre du sommet x_k.

Comme le graphe $\mathcal{G}$ n'est rien d'autre qu'une relation binaire et, comme un descendant de x_i est un suivant d'un suivant ... d'un suivant de x_i, les suivants de x_i sont les éléments de $\bigcup_{k=1}^{+\infty} \mathcal{G}^k(\{x_i\}) = \mathcal{G}^+(\{x_i\}) = \bigcup_{k=1}^{n} \mathcal{G}^k(\{x_i\})$ puisque nous savons que $\mathcal{G}^+ = \bigcup_{i=1}^{n} \mathcal{G}^i$ pour toute relation binaire sur un ensemble possédant n éléments. De même, un **ascendant** de x_i est un précédent d'un précédent ... d'un précédent de x_i, l'ensemble des précédents de x_i est donc $\bigcup_{k=1}^{+\infty} \mathcal{G}^{-k}(\{x_i\}) = \bigcup_{k=1}^{n} \mathcal{G}^{-k}(\{x_i\})$, ensemble que nous notons $\mathcal{G}^-(\{x_i\})$.
Par exemple, considérons ce graphe :



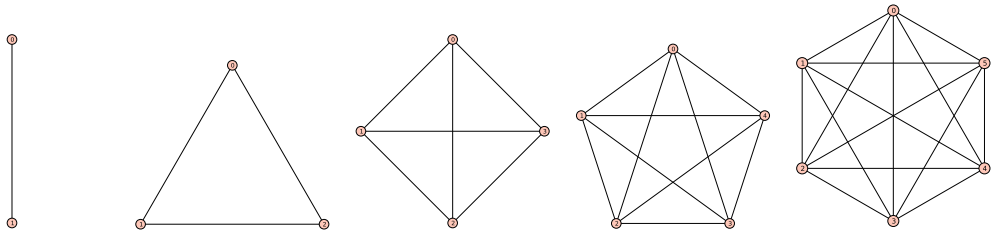
Le sommet 1 n'a pas de descendant, $\mathcal{G}^+(\{1\}) = \emptyset$. Ses ascendants sont tous les autres sommets, $\mathcal{G}^-(\{1\}) = \{2,3,4,5\}$. Pour ce qui concerne les descendants et les ascendants du sommet 5, on a

$$\mathcal{G}^+(\{5\}) = \{1,2,3,4\} \text{ et } \mathcal{G}^-(\{5\}) = \emptyset$$

2 4 Quelques graphes simples particuliers

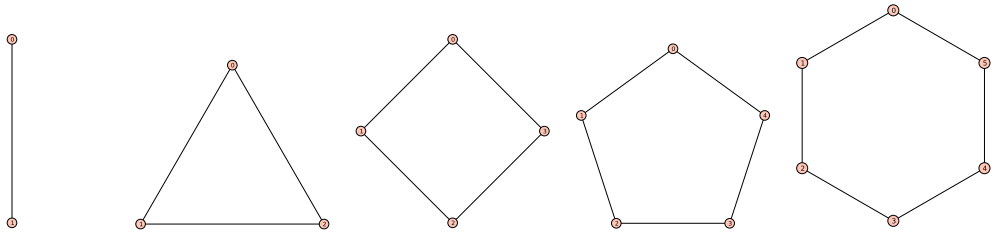
Définition 1 - 9

Le **graphe complet** K_n est tel que $X = \{1,2,...,n\}$ et $A = \binom{X}{2}$:



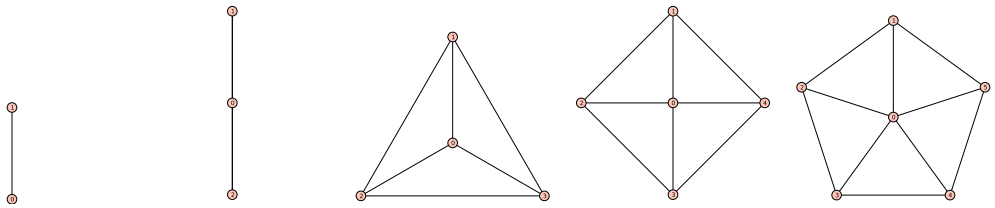
Définition 1 - 10

Le **cycle élémentaire** C_n est tel que $X = \{1,2,...,n\}$ et $A = \{\{1,2\},\{2,3\},...,\{n,1\}\}$:



Définition 1 - 11

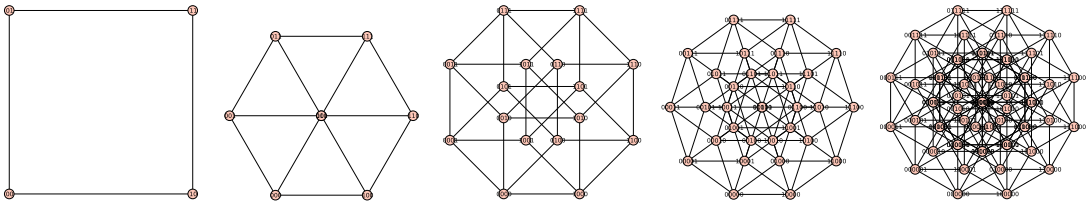
La **roue** W_n est telle que $X = \{0,1,2,...,n\}$ et $A = \{\{1,2\},\{2,3\},...,\{n,1\}\} \cup \{\{0,1\},\{0,2\},...,\{0,n\}\}$:



Définition 1 - 12

Un ***n*-cube** Q_n est le graphe dont les sommets représentent les 2^n chaînes de bits de longueur n et tel que deux sommets sont adjacents si, et seulement si, les chaînes de bits qu'ils représentent diffèrent d'un bit.

Attention ! Il y a un piège dans Q_3...



Avant d'introduire notre prochain candidat, un peu de vocabulaire :

Définition 1 - 13

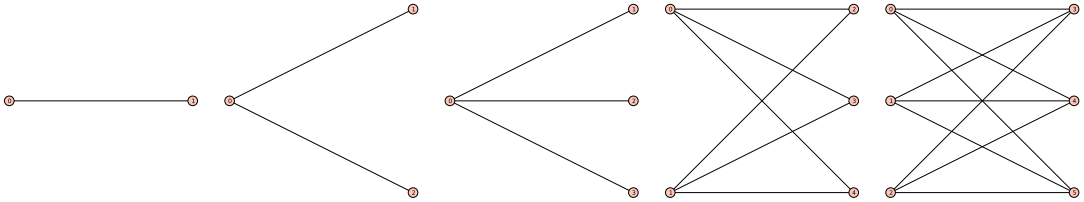
Graphe biparti
Un graphe $\mathcal{G} = (X, A)$ est biparti si X peut être divisé en deux ensembles disjoints X_1 et X_2 de sorte que chaque arête de A relie un sommet de X_1 et un sommet de X_2.
Ainsi $A \subseteq \{\{x_1, x_2\}, x_1 \in X_1, x_2 \in X_2\}$.

Ce qui nous permet d'agrandir notre famille :

Définition 1 - 14

Le **graphe biparti complet** $K_{m,n}$ est tel que $X = \{x_1, x_2, ..., x_n\} \cup \{y_1, y_2, ..., y_m\}$ et $A = \{\{x_i, y_j\} \mid i \in \{1, 2, .., n\}, j \in \{1, 2, .., m\}\}$

Qui est qui ?

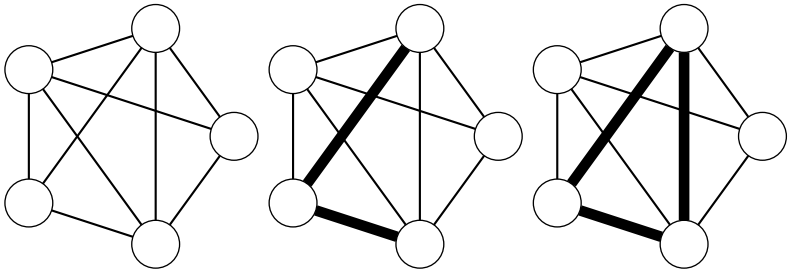


2 5 Sous-graphes et réunion de graphes

Définition 1 - 15

Soit $\mathcal{G} = (X, A)$ et $\mathcal{G}' = (X', A')$ deux graphes.
On dit que $\mathcal{G}$ est un **sous-graphe** de $\mathcal{G}'$ si $X \subseteq X'$ et $A = A' \cap \binom{X}{2}$.
On dit que $\mathcal{G}$ est un **sous-graphe partiel** de $\mathcal{G}'$ si $X' \subseteq X$ et $A \subseteq A'$.
On dit que $\mathcal{G}$ est un **graphe partiel** de $\mathcal{G}'$ si $A \subseteq A'$.

Ah, ça devient subtil. En fait non ! Concrètement, on obtient un sous-graphe à partir d'un graphe donné en supprimant certains sommets et les arêtes qui les contenaient.
On fait de même avec un sous-graphe partiel, mais on peut enlever quelques arêtes en plus.
Un graphe partiel contient les mêmes sommets que le graphe initial mais a des arêtes en moins.
Un sous-graphe et deux sous-graphes partiels en gras se cachent dans les dessins suivant :

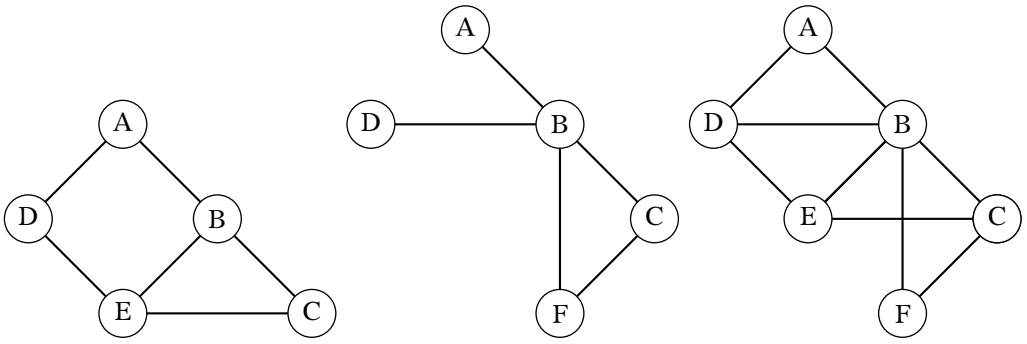


Au lieu d'extraire, on peut combiner des graphes :

Définition 1 - 16

La réunion de deux graphes simples $\mathcal{G}_1 = (X_1, A_1)$ et $\mathcal{G}_2 = (X_2, A_2)$ est le graphe simple $\mathcal{G}_1 \cup \mathcal{G}_2 = (X_1 \cup X_2, A_1 \cup A_2)$

Voici deux graphes et leur réunion :

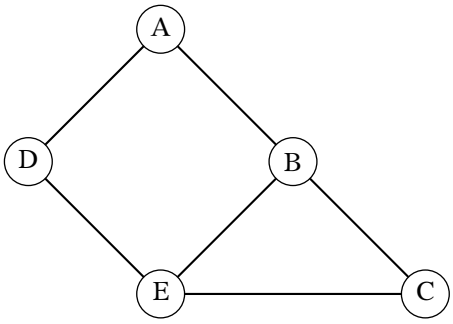


Définition 1 - 17

Le complémentaire $\overline{\mathcal{G}}$ d'un graphe simple $\mathcal{G}$ a les mêmes sommets que $\mathcal{G}$ mais deux sommets sont adjacents dans $\overline{\mathcal{G}}$ si, et seulement si, ils ne le sont pas dans $\mathcal{G}$.

2 6 Matrices d'adjacence

Nous avons vu que nous pouvons représenter un graphe par un dessin mais ça ne dit rien à un ordinateur. Sur Sage, on peut entrer un graphe comme un dictionnaire ou en entrant la liste de ses sommets et la liste de ses arêtes.
Pour simplifier les calculs, il va s'avérer efficace d'utiliser des matrices en s'inspirant de ce qui avait été fait pour les relations binaires.
Reprenons par exemple un des graphes vus précédemment.



Sommets	Sommets adjacents
A	B,D
B	A,C,E
C	B,E
D	A,E
E	B,C,D

Définition 1 - 18

Soit $\mathcal{G} = (X, A)$ un graphe à n sommets. Notons s_1, s_2,...s_n les sommets dans un ordre arbitraire. La **matrice d'adjacence** de $\mathcal{G}$ relativement à la numérotation choisie sur les sommets est la matrice de taille nn, $M_{\mathcal{G}} = (m_{ij})_{1 \leqslant i,j \leqslant n}$, définie par :

$$m_{ij} = \begin{cases} 1 & \text{si } \{s_i, s_j\} \in A \\ 0 & \text{sinon} \end{cases}$$

Dans l'exemple précédent, avec l'ordre alphabétique, cela donne :

$$M = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Dans le cas d'un graphe simple, la matrice est symétrique et n'a que des zéros sur la diagonale.

À retenir

Une matrice occupe un espace mémoire de l'ordre de $\mathrm{Card}(X)^2$ alors qu'un tableau présentant les listes d'adjacence de chaque sommet n'occupe qu'un espace de l'ordre de $\mathrm{Card}(X) + \mathrm{Card}(A)$...

3 Parcours de graphes

On peut être amené à déterminer les plus courts chemins, des composantes connexes, etc. On aura donc souvent besoin de parcourir un graphe.

3.1 Algorithme général

Le principe est le suivant : on sélectionne un sommet, on le « traite » : on détermine ses voisins.
Par exemple :

```
Pour tout x ∈ X Faire
    etat[x] ← non_atteint
FinPour
T ← ∅
à_traiter ← ∅
Pour tout x ∈ X Faire
    Si etat[x]=non_atteint Alors
        à_traiter ← à_traiter ∪{x}
        etat[x] ← atteint
        TantQue à_traiter ≠ ∅ Faire
            y ← choix(à_traiter)
            à_traiter ← à_traiter\{y}
            Pour tout z voisin de y Faire
                Si etat[z] = non_atteint Alors
                    T ← T ∪ {(y,z)}
                    etat[z] ← atteint
                    à_traiter ← à_traiter∪{z}
                FinSi
            FinPour
            etat[y] ← examine
        FinTantQue
    FinSi
FinPour
```

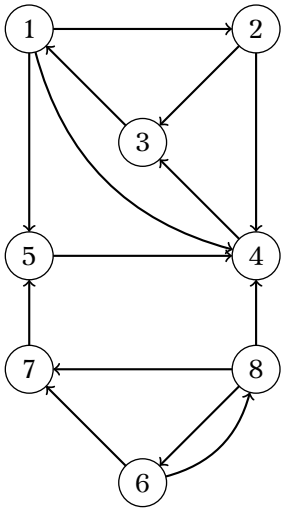
Cet algorithme demeure cependant un peu trop général, surtout à un moment précis : où ?....

3.2 Parcours en largeur

Le critère de choix sera le fait qu'un sommet atteint sera toujours un sommet examiné, c'est-à-dire que la variable `à_traiter` sera une file (vous savez, « First In First Out », comme au RU).
Ensuite, on numérotera les sommets et on les parcourra dans l'ordre croissant.
Lorsque l'on fait un parcours en largeur à partir d'un sommet voisins de x, on atteint d'abord les voisins, ensuite les voisins des voisins (sauf ceux qui sont déjà atteints) et ainsi de

suite. C'est pourquoi le parcours en largeur est aussi appelé parcours concentrique. Il sert notamment à le recherche des plus courts chemins dans un graphe.

Voyons d'abord un EXEMPLE :



- on commence donc par regarder 1 et ses voisins 2, 4 et 5 ;
- on regarde le plus petit voisin de 1 qui est 2 ;
- on regarde les voisins de 2 qui sont 3 et 4 mais 4 a déjà été regardé donc on ne tient pas compte de l'arc (2,4) ;
- le sommet suivant est 4. Son seul voisin est 3 qui a déjà été vu ;
- le sommet suivant est 5. Son seul voisin est 4, qui a déjà été vu ;
- le sommet suivant est 3. Le seul voisin est 1 qui a déjà été traité ;
- la file est maintenant vide. On remonte donc maintenant à un sommet non encore atteint dans l'ordre des numéros croissant ;
- il s'agit donc du sommet 6 dont les voisins sont 7 et 8 ;
- on traite 7 dont le seul voisin est 5 qui a déjà été traité ;
- on traite 8 dont les voisins 4, 6 et 7 ont tous été déjà traités ;
- la file est maintenant vide et tous les sommets ont été traités.

Dessinez l'arbre « d'exploration » avec à côté la file et une patate contenant les sommets déjà traités.

Passons à l'algorithme. On a besoin de traiter une file qu'on nommera `à_traiter` et qui permet d'effectuer deux actions : `AjouterEnQueue` et `EnleverTete`.

```
Pour tout x ∈ X Faire
    etat[x] ← non_atteint
FinPour
index ← 1
à_traiter ← ∅
Pour tout z ∈ X Faire
    Si etat[z]=non_atteint Alors
        pere[z] ← NIL
        ordre[z] ← index
        index ++
        à_traiter.AjouterEnQueue(z)
        TantQue à_traiter≠ ∅ Faire
            x ← Tete(à_traiter)
            Pour y voisin de x Faire
                Si etat[y]=non_atteint Alors
                    etat[y] ← atteint
                    ordre[y] ← index
                    index ++
                    pere[y] ← x
                    à_traiter.AjouterEnQueue(y)
                FinSi
            FinPour
            à_traiter.EnleverTete()
            etat[x] ← traite
        FinTantQue
    FinSi
FinPour
```

Comment peut-on utiliser cet algorithme pour déterminer les composantes connexes du graphe ? Pour chercher un plus court chemin entre deux sommets ?

3 3 Parcours en profondeur

Contrairement au parcours en largeur, lorsque l'on fait un parcours en profondeur à partir d'un sommet x, on tente d'avancer le plus loin possible dans le graphe, et ce n'est que lorsque toutes les possibilités de progression sont bloquées que l'on revient (étape de backtrack) pour explorer un nouveau chemin ou une nouvelle chaîne.

Le parcours en profondeur correspond aussi à l'exploration d'un labyrinthe.

Les applications de ce parcours sont peut-être moins évidentes que pour le parcours en largeur, mais le parcours en profondeur permet de résoudre efficacement des problèmes plus difficiles comme la recherche de composantes fortement connexes dans un graphe orienté.

Concrètement, on part d'un sommet, on va voir un de ses voisins puis un voisin du voisin, etc., jusqu'à être bloqué, alors on va en arrière.

Ici, le critère de choix sera que le dernier sommet atteint sera examiné. On utilisera cette fois une pile (Last In First Out).

Reprenons l'exemple précédent :

- on commence par 1 ;
- on visite ensuite un voisin de 1 : 2 ;
- on visite ensuite un voisin de 2 : 3 ;
- on est alors bloqué. On revient à 2 et on visite son autre voisin 4 ;
- on est bloqué et on revient à 1 et on visite le seul voisin restant, 5 ;
- on est bloqué. Il faut prendre un nouveau sommet : 6 ;
- on visite 7, voisin de 6 et on est bloqué ;
- on retourne à 6 et on visite le seul voisin restant, 8 et c'est terminé.

Tracer « l'arbre des visites », la pile correspondant et les patate des sommets visités.

Voici un exemple d'algorithme qui est composé de deux parties.

```
Fonction parcours_pro( x : sommet ) : rien
Début
    etat[x] ← atteint
    ordre[x] ← index
    index ++
    Pour y voisin de x Faire
        Si etat[y]=non_atteint Alors
            pere[y] ← x
            parcours_pro(y)
        FinSi
    FinPour
    etat[x] ← traite
Fin
```

L'algorithme principal :

```
Pour tout x ∈ X Faire
    etat[x] ← non_atteint
FinPour
index ← 1
Pour chaque x ∈ X Faire
    Si etat[x]=non_atteint Alors
        pere[x] ← NIL
        parcours_pro(x)
    FinSi
FinPour
```

4 Fermeture transitive

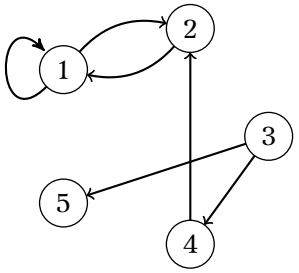
$\mathcal{G} = (X, A)$ étant une relation binaire, nous savons que la fermeture transitive de $\mathcal{G}$ est la relation $\mathcal{G}^+ = \bigcup_{i=1}^{n} \mathcal{G}^i$ (la fermeture transitive et réflexive de $\mathcal{G}$ est donnée par $\mathcal{G}^* = \bigcup_{i=0}^{n-1} \mathcal{G}^i$) c'est la plus petite relation transitive contenant $\mathcal{G}$ (rappelons-nous encore qu'une relation binaire est assimilée à un ensemble).

Nous savons aussi que la matrice booléenne de $\mathcal{G}^*$ est $\sum_{k=0}^{n-1} M^k$ et que son calcul effectif peut être laborieux (mais on a Sage...).
L'autre méthode est la suivante :

Théorème 1 - 4

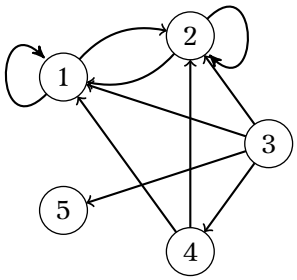
Nous notons M_0 la matrice booléenne de $\mathcal{G}$ et θ_i l'opérateur qui consiste à ajouter, au sens de l'addition booléenne, la $i^{ème}$ ligne de M_{i-1} à toute ligne de M_{i-1} possédant un 1 en colonne numéro i ; le résultat se notant $M_i = \theta_i(M_{i-1})$. La matrice M_n donne alors la matrice de la fermeture transitive de $\mathcal{G}$.

Si nous notons $M = (m_{i,j})$ la matrice carrée booléenne d'ordre n de Γ, l'algo se traduit par :
La démonstration est un peu compliquée...
Considérons par exemple le graphe $\mathcal{G}$ défini par :



Sa matrice d'incidence est $M = M_0 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$, appliquons θ_1, nous allons ajouter (au sens booléen) la ligne numéro 1 sur toute ligne possédant un 1 en colonne numéro 1, seule la deuxième ligne est concernée, on obtient $M_1 = \theta_1(M_0) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$.

Appliquons maintenant θ_2 à M_1, nous allons ajouter la ligne numéro 2 à toute ligne possédant un 1 en colonne numéro 2, les lignes 1, 2 et 4 sont concernées[a]. Nous obtenons $\theta_2(M_1) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ \mathbf{1} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$. Aucune ligne n'a de 1 en colonne 3, donc $\theta_3(M_2) = M_3 = M_2$. La ligne 3 a un 1 en colonne 4, $\theta_4(M_3) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} = M_4$ et θ_5 ne changeant rien, la matrice de la fermeture transitive est $M_5 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ qui nous donne le graphe de la fermeture transitive de $\mathcal{G}$.



Remarque Si on s'interdit les boucles, il suffit, après avoir déterminé la fermeture transitive, de les supprimer.

5 Isomorphisme de graphes

Nous avons observé que l'ordre des sommets, leurs noms, pouvaient varier au moment de représenter un graphe.

a. Ajouter (en calcul booléen) la ligne numéro i à la ligne numéro i ne la modifie pas.

Imaginez un graphe représenté à l'aide de fils en pâte à modeler dont les sommets ne sont pas nommés. Vous placez des étiquettes sur les sommets, vous prenez une photo, vous enlevez les étiquettes puis vous déformez les arêtes sans les casser et vous renommez aléatoirement les sommets.
Les deux graphes obtenus auront beaucoup de propriétés communes. Comme ils ont été construits avec la même structure (le même « corps »), on dit qu'ils sont *isomorphes* (puisque vous avez étudié le Grec ancien, vous avez reconnu les racines ισος qui signifie « le même » et μορφος qui signifie « le corps »...).
Donnons une définition plus formelle :

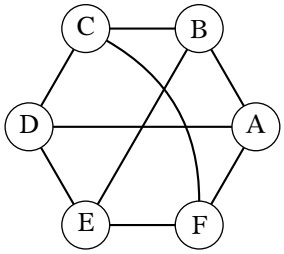
Définition 1 - 19

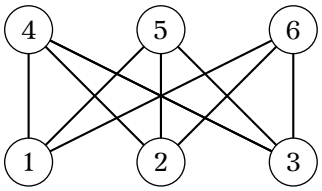
Les graphes simples $\mathcal{G} = (X, A)$ et $\mathcal{G}' = (X', A')$ sont dits **isomorphes** s'il existe une bijection $f : X \to X'$ telle que nous ayons, pour tout couple de sommets distincts du premier graphe $(x, y) \in X^2$, $x \neq x'$,

$$\text{l'arête } \{x, y\} \in A \text{ si, et seulement si, l'arête } \{f(x), f(y)\} \in A'$$

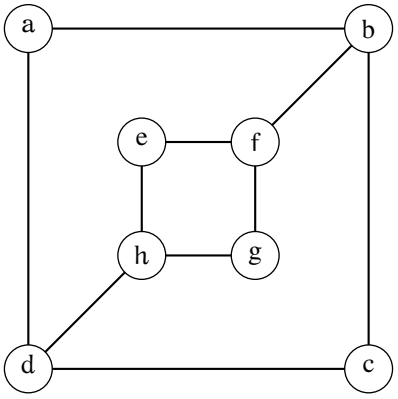
Une telle fonction f est appelée **isomorphisme** entre les graphes $\mathcal{G}$ et $\mathcal{G}'$ et nous notons $\mathcal{G} \cong \mathcal{G}'$.

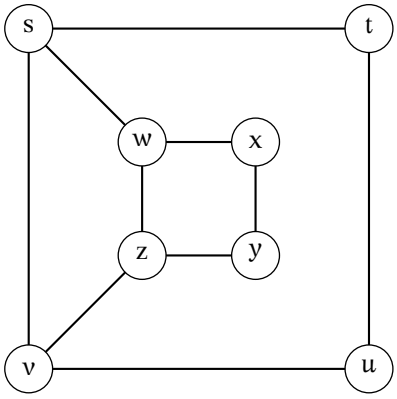
Un isomorphisme conserve donc « l'adjacence » des sommets et conserve donc les degrés.
Montrez par exemple que les graphes suivants sont isomorphes :





Cela peut parfois devenir plus subtil.
Comparez le nombre de sommets, le nombre d'arêtes, le nombre de sommets selon leur degré : tout semble se conserver, et pourtant....





Quel est le degré de a ? Vers quel(s) sommet(s) peut-il être envoyé ? Observez les degrés des sommets adjacents des candidats et concluez.

Danger

Il ne faut pas confondre bijection et isomorphisme !
Trouver une bijection entre les sommets, ce n'est pas forcément trouver un isomorphisme. Si on a une bijection « candidate », on peut vérifier qu'elle convient en comparant les matrices d'adjacence.

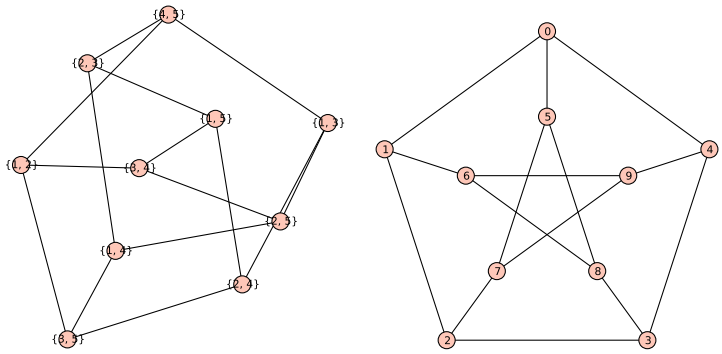
Il est paradoxal de remarquer qu'on ne connaît pas d'algorithme efficace pour reconnaître si deux graphes sont isomorphes (on ne sait même pas si c'est envisageable) alors que le problème paraît facile a priori.
Cependant Sage, par exemple, peut vous aider pour des graphes de taille raisonnable :

```
1 sage: K = graphs.KneserGraph(5,2)
2 sage: P = graphs.PetersenGraph()
```

```
sage: K.is_isomorphic(P)
True
```

Et voici leurs têtes :

```
sage: K.show()
sage: P.show()
```



6 Connexité

6 1 Chaînes et chemins

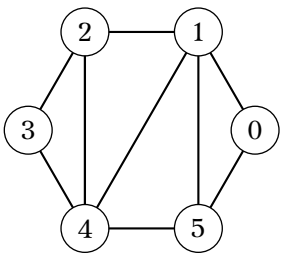
Nous avons déjà parlé de chemins au sujet des graphes orientés. Le pendant pour les graphes symétriques est parfois appelé *chaîne*.

Définition 1 - 20

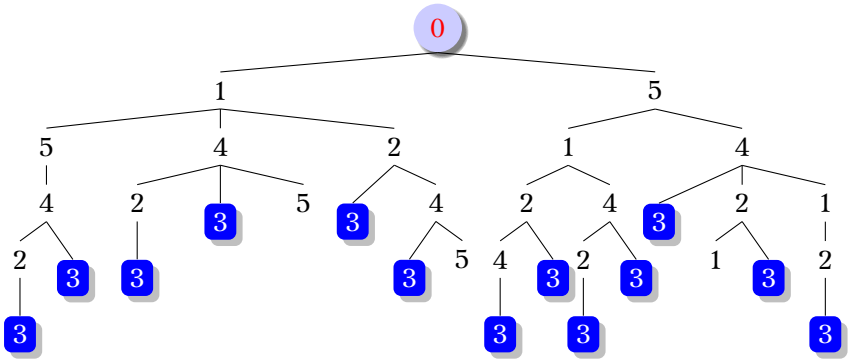
Soit $\mathcal{G} = (X, A)$ un graphe non orienté. Une **chaîne de longueur *n*** de s à t dans $\mathcal{G}$ est une suite d'arêtes a_1, a_2,...,a_n de $\mathcal{G}$ telle que $a_1 = \{s, x_1\}$, $a_2 = \{x_1, x_2\}$,...,$a_n = \{x_{n-1}, t\}$.
Si le graphe est simple, on peut juste donner la suite des sommets s, x_1,...,x_{n-1}, t.
La chaîne est **fermée** si elle commence là où elle se termine et que sa longueur est non nulle.
On dit que le chaîne **passe** par les sommets x_i.
Un chaîne est **élémentaire** si elle ne passe qu'une fois par un même sommet et **simple** si elle n'utilise pas deux fois la même arête.
La chaîne est un **cycle** si elle est fermée et simple.

On peut comptabiliser toutes les chaînes élémentaires allant d'un sommet à un autre à l'aide d'un arbre.

Sur le graphe suivant, nous voulons aller de 0 jusqu'à 3.



Décomposons les chaînes simples :



Il y a donc treize chaînes simples qui vont de 0 jusqu'à 3. Les plus courtes chaînes ont pour longueur 3 et sont au nombre de 3.

Recherche

Cela permet d'avoir une première idée d'algorithme pour trouver la plus courte chaîne d'un point à un autre : lequel ? Avantages ? Inconvénients ?

À retenir

On peut montrer que l'existence d'un cycle de longueur donné est invariant à isomorphisme près : comparer les cycles de deux graphes peut permettre de conclure que deux graphes ne sont pas isomorphes (cf exercice **??** page ??).

6 2 Nombre de chaînes et matrice d'adjacence

Voici un théorème important qui montre tout l'avantage que nous pouvons tirer de l'utilisation de matrices :

Théorème 1 - 5

Soit $\mathcal{G} = (X, A)$, avec $X = \{x_1, x_2, ..., x_n\}$, un graphe de nature quelconque de matrice d'adjacence M selon un ordre fixé des sommets. Soit $m_{ij}^{(k)}$ l'élément de position (i, j) de la matrice M^k avec k un entier naturel.
Alors $m_{ij}^{(k)}$ est égal au nombre de chaînes de longueur k entre le sommet x_i et le sommet x_j dans le graphe $\mathcal{G}$.

Pour le plaisir, voici une petite démonstration par récurrence sur k de ce théorème.
Pour $k = 1$, la proposition correspond à la définition de la matrice d'adjacence : on met un 1 s'il existe une chaîne entre x_i et x_j.
Soit $k > 1$: on suppose la proposition vraie pour $k - 1$.
Soit x_i et x_j deux sommets quelconques et une chaîne de longueur k entre eux : on peut le décomposer en une arête entre x_i et un voisin, disons x_p et une chaîne entre x_p et x_j de longueur $k - 1$.
On peut donc appliquer la relation de récurrence à ce dernier chaîne : le nombre de chaînes entre x_p et x_j est donc égal au coefficient $m_{pj}^{(k-1)}$.
Il se peut qu'il y ait plusieurs arêtes entre x_i et x_p donc le nombre de chaînes de longueur k passant par x_p est $m_{ip} m_{pj}^{(k-1)}$.
Au passage, on peut remarquer que cette relation fonctionne aussi si x_p n'est pas un voisin de x_i car à ce moment-là, m_{ip} est nul.
Finalement, pour avoir toutes les chaînes de longueur k, on calcule :

$$\sum_{p=1}^{n} m_{ip} m_{pj}^{k-1}$$

Or vous qui connaissez bien votre cours sur le produit de matrices, vous savez que le calcul précédent donne exactement le terme m_{ij}^{k} ce qui achève notre démonstration.

Ce théorème a une conséquence très utile pour nous. Il nous faut tout d'abord introduire une nouvelle définition :

Définition 1 - 21

Soit $\mathcal{G} = (X, A)$ un graphe *connexe*. On définit la **distance** entre deux sommets $(x, x') \in X^2$, qu'on note $d_{\mathcal{G}}(x, x')$, comme la longueur de la chaîne simple la plus courte existant entre x et x′ dans $\mathcal{G}$.

Un problème très important à résoudre est de trouver la plus courte chaîne entre deux sommets d'un graphe. Le théorème suivant nous y aidera :

Théorème 1 - 6

Avec les notations du théorème 1 - 5 page précédente, soit x_i et x_j deux sommets d'un graphe $\mathcal{G}$. Alors :

$$d_{\mathcal{G}}(x_i, x_j) = \min\{k \geqslant 0 \mid m_{ij}^{(k)} \neq 0\}$$

Reprenons par exemple le graphe illustrant la définition 1 - 20 page 20.
Sa matrice d'adjacence en suivant l'ordre croissant des numéros de sommet est :

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Pour ne pas se fatiguer à calculer les produits de matrices, on peut s'aider de Python :

```
In[1]: g2 = Graphe({'0':{'1','5'},
                    '1':{'2','4','5'},
                    '2':{'3','4'},
                    '3':{'4'},
                    '4':{'5'} })

In [2]: g2
Out[2]:
        0 -> { 1 5 }
        1 -> { 0 2 4 5 }
        2 -> { 1 3 4 }
        3 -> { 2 4 }
        4 -> { 1 2 3 5 }
        5 -> { 0 1 4 }

In [3]: M = g2.matrice()

In [4]: M
Out[4]:
array([[0, 1, 0, 0, 0, 1],
       [1, 0, 1, 0, 1, 1],
       [0, 1, 0, 1, 1, 0],
       [0, 0, 1, 0, 1, 0],
       [0, 1, 1, 1, 0, 1],
       [1, 1, 0, 0, 1, 0]])


In [4]: M.dot(M.dot(M))
Out[4]:
array([[2, 6, 3, 3, 3, 5],
       [6, 6, 8, 3, 9, 7],
       [3, 8, 4, 5, 7, 4],
       [3, 3, 5, 2, 6, 3],
       [3, 9, 7, 6, 6, 8],
       [5, 7, 4, 3, 8, 4]])
```

Hop : le plus petit $m_{03}^{(k)}$ non nul est bien obtenu pour $k = 3$ et il y a trois chaînes de longueur minimale 3 entre les sommets x_0 et x_3.
On aurait pu s'aider d'un petit programme maison. Comment expliquer le code suivant :

```
def distance(self, s1:str, s2:str) -> float:
                if s2 not in self.composante_connexe(s1):
                        return float('inf')
```

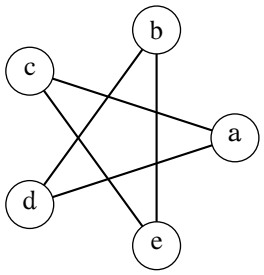
```
        m = self.matrice()
        M = m[:]
        k = 1
        ss = sorted(list(self.sommets()))
        i, j = ss.index(s1), ss.index(s2)
        while m[i,j] == 0:
                m = m.dot(M)
                k += 1
        return float(k)
```

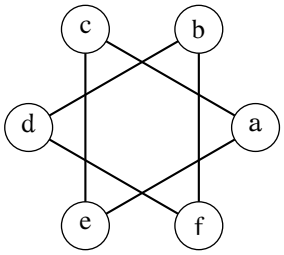
6 3 Connexité des graphes non orientés

Définition 1 - 22

On dit qu'un graphe $\mathcal{G} = (X, A)$ est **connexe** si, pour tous sommets x et y de X, $\mathcal{G}$ contient un chaîne entre x et y

Parmi les deux graphes suivant, qui est connexe et qui ne l'est pas ?





Certains auteurs parlent de graphe connecté au lieu de graphe connexe. C'est plus parlant mais la notion de connexité se retrouve dans de nombreux domaines en mathématiques et désigne quelque chose qui est en un seul morceau.

Théorème 1 - 7

Il existe une chaîne simple entre chaque paire de sommets (distincts) d'un graphe simple connexe.

Démontrez ce résultat. Vous n'oublierez pas la notion de plus petit élément vue l'an dernier.

Théorème 1 - 8

Si un graphe n'est pas connexe, c'est la réunion de sous-graphes connexes qui n'ont pas de sommets en commun deux à deux.
On appelle ces sous-graphes les **composantes connexes** du graphe initial.
On appelle **connectivité** le nombre de composantes connexes d'un graphe.

Nous n'aborderons pas la démonstration mais « ça se voit bien ».

Par exemple, cherchez les composantes connexes du graphe non connexe étudié précédemment.

On peut définir assez simplement un algorithme de connexité qui renvoie la connectivité d'un graphe.

Notons c cette connectivité et $\mathcal{G} = (X, A)$ le graphe.

```
Fonction composante_connexe(G : Graphe, s : sommet) : Ensemble de sommets
Début
    CC ← ∅
    a_visiter ← [s]
    visites ← [s]
    TantQue a_visiter est non vide Faire
        x ← dépiler a_visiter
        Ajouter x à CC
        PourChaque t De l'ensemble des adjacents de x Faire
            Si t n'a pas été visité Alors
                Rajouter t aux sommets à visiter
                Rajouter t aux sommets visités
            FinSi
        FinPourChaque
    FinTantQue
    Retourner CC
Fin
```

À retenir

Pour rechercher les composantes connexes maximales en nombre de sommets, on utilise les algos de recherche en largeur ou en profondeur puisque pour trouver une composante connexe maximale contenant un sommet donné, il suffit de trouver tous les descendants et les ascendants de ce sommet.

Une définition équivalente et qui permet d'obtenir plus rapidement certains résultats est la suivante :

Théorème 1 - 9

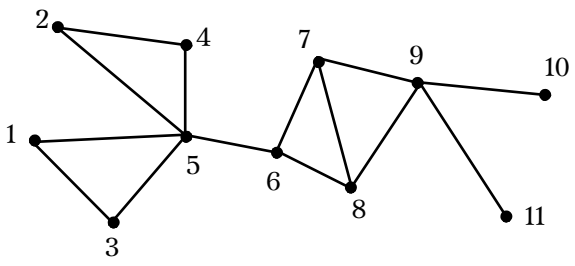
Sommets connectés et relation d'équivalence
Deux sommets sont **connectés** s'il existe une chaîne ayant ces deux sommets pour extrémités. On construit ainsi une **relation d'équivalence**. Les **classes d'équivalence** de cette relation sont appelées **composantes connexes**. Un graphe ne comportant qu'une seule composante connexe est dit **connexe**

C'est plus mathématique mais tellement plus élégant...

6 4 Point d'articulation

Définition 1 - 23

Un **point d'articulation** de $\mathcal{G}$ est un sommet dont la suppression augmente le nombre de composantes connexes. Un **isthme** est une arête dont la suppression a le même effet.



Les sommets 5, 6 et et 9 sont les points d'articulation et l'arête $[5,6]$ est un isthme.

Il doit être clair qu'un point d'articulation ne peut appartenir à un cycle du graphe et que si x est un point d'articulation c'est qu'il existe au moins u et v deux sommets distincts et distincts de x de sorte que toute chaîne de u à v passe par x.

Voici un algorithme qui nous donne les points d'articulation d'un graphe connexe ainsi qu'une arborescence des sommets visités :

```
Variable
    U : ensemble
    prévisite, hauteur : entiers
Début
    prévisite ← 1
    U ← ∅
    empiler s
    prévisite[s] ← prévisite
    hauteur[s] ← 1
    prévisite ← prévisite+1
    TantQue pile non vide Faire
        x ← sommet de la pile
        Si ∃ y non sélectionné et voisin de x Alors
            empiler y
            U ← U ∪ {(x,y)}
            prévisite[y] ← prévisite
            prévisite ← prévisite+1
            hauteur ← prévisite[y]
        Sinon
            dépiler x
            Pour tout sommet z voisin de x avec (z,x) ∉ U Faire
                hauteur[x] ← min{ hauteur[x], hauteur[y]}
            FinPour
        FinSi
    FinTantQue
```

```
    Si s a au moins 2 successeurs dans U Alors
        s est un sommet d'articulation
    FinSi
    Si x ≠ s et (x,y) ∈ U et hauteur[y]⩾prévisite[x] Alors
        x est un sommet d'articulation
    FinSi
Fin
```

6 5 Connexité des graphes orientés

Comme le sens intervient, on distingue deux notions de connexité à présent.

Définition 1 - 24

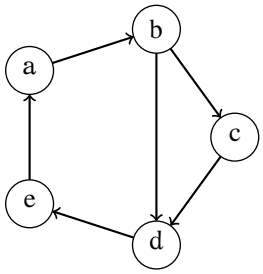
Un graphe orienté est **fortement connexe** s'il existe un chemin de a à b et un chemin de b à a pour toute paire $\{a,b\}$ de sommets du graphe.
Une partie (non vide) Y de X est fortement connexe si, et seulement si, le sous graphe défini par Y ou engendré par Y, c'est-à-dire la restriction du graphe $\mathcal{G} = (X,A)$ à Y, $(Y, A \cap (YY))$, est fortement connexe ; cela signifie que pour toute paire de sommets de Y, il existe un chemin de l'un à l'autre n'utilisant que des sommets de Y. La partie Y est fortement connexe maximale s'il n'existe pas de partie fortement connexe Z de X contenant strictement Y.

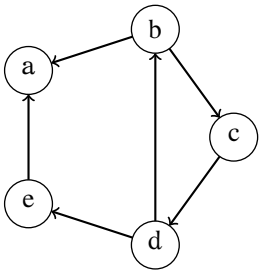
Un graphe orienté peut cependant ne pas être fortement connexe tout en étant « en un seul morceau » :

Définition 1 - 25

Un graphe orienté est **faiblement connexe** si son graphe non orienté sous-jacent est connexe.

Que dire alors des deux graphes suivants :





Pour le graphe qui n'est pas fortement connexe, pouvez-vous trouver ses trois composantes fortement connexes ?

Remarque

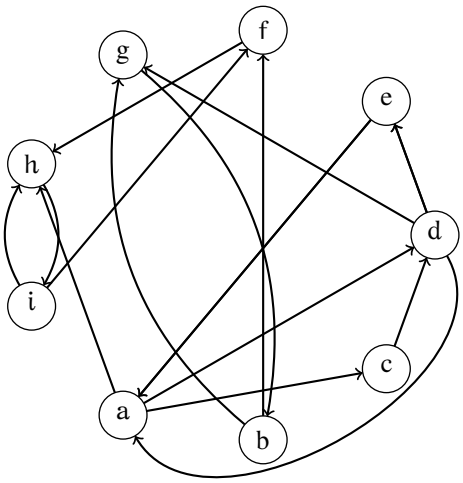
Une partie restreinte à un seul sommet, un singleton de X, ne contredit pas la définition, c'est une partie fortement connexe.

Voici un théorème qui va nous donner un premier algorithme pour déterminer les parties fortement connexes maximales.

Théorème 1 - 10

$Y = (\mathcal{G}^+(\{x_i\}) \cap \mathcal{G}^-(\{x_i\})) \cup \{x_i\}$ est une partie fortement connexe maximale du graphe $\mathcal{G}$.

La scène se passe dans une administration[b] qui veut améliorer le cadre de vie de ses 9 employés a, b, c, d, e, f, g, h et i et qui travaillent dans une même salle. Ils échangent entre eux des documents, le graphe suivant indique de quelle façon :



On voudrait leur permettre de travailler dans de meilleures conditions et, pour cela, les répartir dans plusieurs bureaux en séparant le moins possible les employés entre lesquels les documents circulent. Une solution est donnée par l'obtention des parties fortement connexes maximales, la voici. Choisissons le sommet a par exemple ; l'ensemble de ses descendants est :

$$\{b, c, d, e, f, g, h, i\}$$

et l'ensemble de ses ascendants est

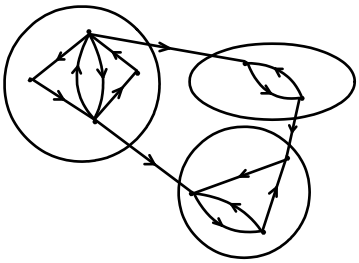
$$\{c, d, e\}$$

La partie

$$\{a, c, d, e\} = (\{b, c, d, e, f, g, h, i\} \cap \{c, d, e\}) \cup \{a\}$$

est fortement connexe maximale.
En choisissant maintenant le sommet b, on prouve que la partie $\{b, g\}$ est fortement connexe maximale et le choix de f donne la dernière partie fortement connexe maximale $\{f, h, i\}$. Utilisons ces résultats pour déterminer une autre représentation du graphe :

b. Exemple extrait d'un (très vieux) polycopié de Mr. Ganachaud professeur à l'université de Nantes.



Cette représentation du graphe (que vous compléterez en étiquetant les sommets) nous donne une solution pour une répartition possible des employés dans trois bureaux.

6 6 Petit résumé du vocabulaire

À retenir

$\mathcal{G} = (X = \{x_1, x_2, ..., x_n\}, A)$ est un graphe orienté avec $\mathbf{Card}(A) \geq 2$ et on note $\mathcal{G}_{NO}$ le graphe non orienté associé à $\mathcal{G}$

1. *Que signifie « x_i est un sommet isolé » ?* x_i n'a pas de voisin.
2. *On vous dit que la suite $(y_1, y_2, ..., y_k)$ avec $y_i \in X$ est une chaîne, qu'est-ce que cela signifie ?* $i \in \{1, 2, ..., n-1\}$ on a (y_i, y_{i+1}) ou $(y_{i+1}, y_1) \in A$.
3. *On vous dit que la suite $(y_1, y_2, ..., y_k)$ avec $y_i \in X$ est un chemin, qu'est-ce que cela signifie ?* $\forall i \in \{1, 2, ..., n-1\}$ on a $(y_i, y_{i+1}) \in A$.
4. *Que sigifie $\mathcal{G}$ est connexe ?* Que pour toute paire $\{x_i, x_j\}$ de sommets de $\mathcal{G}$ il existe une chaîne entre x_i et x_j.
5. *Que signifie $\mathcal{G}$ est fortement connexe ?* Que pour tout couple de sommets distincts $(x_i, x_j) \in XX$ il existe un chemin de x_i à x_j.
6. *Définir parfaitement le sous graphe de $\mathcal{G}$ engendré par $Y \subseteq X$.* C'est le graphe $\mathcal{G}_Y = (Y, A \cap (YY))$.
7. *On vous dit que $Y \subseteq X$ est une partie connexe du graphe $\mathcal{G}$, qu'est-ce que cela signifie ?* Que $\mathcal{G}_Y$ est connexe.
8. *On vous dit que $Y \subseteq X$ est une partie fortement connexe du graphe $\mathcal{G}$, qu'est-ce que cela signifie ?* Que $\mathcal{G}_Y$ est fortement connexe.
9. *Donner une méthode ou une formule ou un algorithme qui permet de déterminer la partie fortement connexe maximale de $\mathcal{G}$ qui contient le sommet x_i.*
   C'est $\{x_i\} \cup (\mathcal{G}^+(x_i) \cap \mathcal{G}^-(x_i))$

EXERCICES

Les bases

Python 1 - 1

À l'aide de la classe **Graphe**, construisez le 3-cube (Q_3) avec comme nom d'arêtes les entiers en binaire de 000 à 111.

Python 1 - 2

Construisez une fonction `degre_sommet(self, a)` qui calcule le degré d'un sommet d'un graphe.

Python 1 - 3

Finissez de mettre les signatures aux fonctions construites.

Python 1 - 4

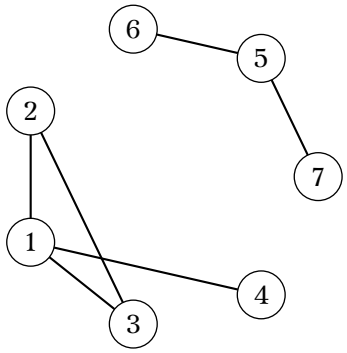
Étudiez la doc de Graphviz. Comment modifier notre classe pour travailler sur des graphes orientés ?

Python 1 - 5

Déterminez une fonction `matrice(self)` qui renvoie la matrice d'adjacence d'un graphe.
Comment créer un graphe à partir d'une matrice d'adjacence ?

Recherche 1 - 1 Vocabulaire

On considère le graphe non orienté suivant : $\mathcal{G} = (X, A)$:



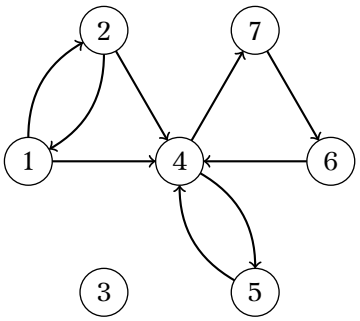
1. Déterminer X et A.
2. Déterminer le degré de chaque sommet.
3. Déterminer sa matice d'adjacence (on a ordonné les sommets par numéros croissants).
4. Donner un graphe partiel de $\mathcal{G}$.
5. Donner un sous graphe de $\mathcal{G}$.
6. Déterminer le sous graphe engendré par la partie $Y = \{1, 2, 3, 5\}$.
7. Déterminer une chaîne simple de longueur 3 qui n'est pas un cycle, est-elle élémentaire ?
8. Déterminer les cycles.
9. Donner des parties connexes et non connexes
10. Donner les composantes connexes de $\mathcal{G}$.

Recherche 1 - 2 Vocabulaire

1. Donner un graphe non orienté qui admet des cycles non élémentaires.
2. Donner un graphe non orienté qui n'admet pas de chemin fermé simple.

Recherche 1 - 3 Vocabulaire

Soit $\mathcal{G} = (X, A)$ le graphe orienté dont voici un diagramme sagittal :



1. Préciser X et A.
2. Donner les suivants ou les successeurs du sommet 2.
3. Donner les suivants ou les successeurs du sommet 4.
4. Donner les voisins du sommet 4.
5. Les sommets 5 et 6 sont-ils adjacents ?
6. Les sommets 6 et 7 sonts-ils adjacents ?
7. Donner les prédécesseurs du sommet 4.
8. Donner la matrice d'adjacence de ce graphe (on a décidé d'ordonner les sommets par numéros croissants).
9. Donner le dictionnaire des prédécesseurs, et celui des successeurs.
10. Donner le sous-gaphe de $\mathcal{G}$ engendré par la partie $\{4, 5, 7\}$.
11. Donner un sous-graphe partiel de $\mathcal{G}$ engendré par la partie $\{4, 5, 7\}$.
12. Donner le sous-graphe engendré par la partie $\{2, 4, 5, 6\}$.
13. Donner le sous graphe engendré par la partie $\{3\}$.
14. Donner un sous-graphe partiel engendré par la partie $\{1, 4, 6\}$.
15. $\mathcal{G}$ admet-il des sommets isolés ?
16. $\mathcal{G}$ est-il un graphe complet ?
17. Déterminer $\mathcal{G}(\{4, 6\})$, $\mathcal{G}^{-1}(\{4, 6\})$, $\mathcal{G}^3(\{4\})$, $\mathcal{G}(\{3\})$.
18. Déterminer le degré d'entrée puis le degré de sortie de chaque sommet.
19. Donner un graphe isomorphe à $\mathcal{G}$.

20. Donner des chaînes et des chemins quelconques, des chaînes et des chemins simples, des chaînes et des chemins non simples, des chaînes et des chemins élémentaires, non élémentaires, des cycles élémentaires, non élémentaires, des circuits.

21. Donner tous les descendants du sommet 4.

22. Donner tous les ascendants du sommet 2.

23. On note

$$\mathcal{G}^+(\{2\}) = \bigcup_{k=1}^{+\infty} \mathcal{G}^k(\{2\}) \text{ et } \mathcal{G}^*(\{2\}) = \bigcup_{k=0}^{+\infty} \mathcal{G}^k(\{2\})$$

Déterminer ces deux ensembles.

24. $\mathcal{G}$ est-il connexe ? Fortement connexe ? Préciser les parties fortement connexes maximales de $\mathcal{G}$.

25. $\mathcal{G}' = (X' = X - \{3\}, A \cap (X'X'))$ admet-il des circuits eulériens ? Des circuits hamiltoniens ?

Recherche 1 - 4

Dans un réseau d'ordinateur, chaque ordinateur est directement relié à au plus trois ordinateurs. Depuis chaque ordinateur, il existe un chemin vers tout autre ordinateur comportant au maximum un relais. Combien d'ordinateurs contient au maximum le réseau ?

Recherche 1 - 5

Dans un réseau il y a 9 ordinateurs, chacun étant relié par un câble à au moins 3 autres ordinateurs. Combien faut-il au minimum prévoir de câbles ?

Recherche 1 - 6 cubes

Un ***n*-cube** Q_n est le graphe dont les sommets représentent les 2^n chaînes de bits de longueur n et tel que deux sommets sont adjacents si, et seulement si, les chaînes de bits qu'ils représentent diffèrent d'un bit. Dessinez Q_2 et Q_3.

Recherche 1 - 7

Soit $\mathcal{G} = (X, A)$ un graphe simple symétrique. Montrer que $\text{Card}(A) \leqslant \frac{1}{2}\text{Card}(X)(\text{Card}(X) - 1)$.

Recherche 1 - 8

Soit $\mathcal{G} = (X, A)$ un graphe simple symétrque.

Peut-on avoir un sommet de degré 0 et un sommet de degré $\text{Card}(X) - 1$?

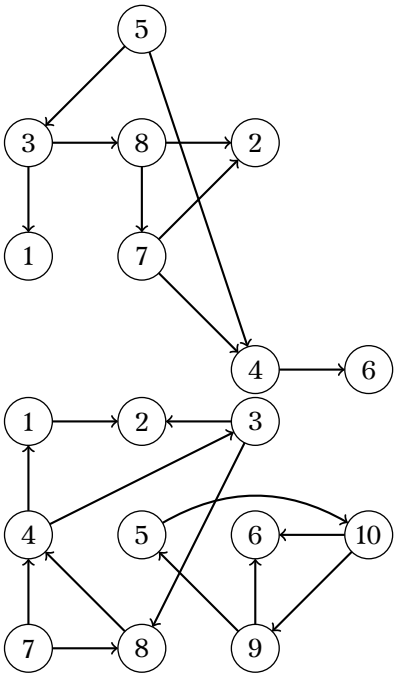
Montrer qu'il existe deux sommets différents ayant le même degré.

Recherche 1 - 9

Analyser l'algorithme suivant appliqué à un graphe orienté $\mathcal{G} = (X, A)$:

```
Variable
  i : entier positif
Début
  i ← 1
  TantQue i <= nombre de sommets Faire
    Si les arcs du sommet i sont tous aboutissants ou tous incidents Alors
      Effacer le sommet i et ses arcs adjacents
      i ← i+1
    Sinon
      i ← i+1
    FinSi
  Fin
```

Appliquer le aux graphes suivant :



Appliquer maintenant l'algorithme suivant aux graphes précédents :
Utiliser cet autre algorithme appliqué aux matrices d'adjacences des graphes précédents : on supprime à chaque itération tous les sommets qui n'ont pas de précédentDans quel but ? Comment reconnaître un sommet sans prédécesseur ? Comment faire fonctionner l'algorithme ?

Recherche 1 - 10

Que pensez-vous de la réunion d'un graphe de n sommets et de son complémentaire ?

Recherche 1 - 11

1. Combien de sommets a un graphe régulier de degré 4 contenant 10 arêtes ?

2. Pour quelles valeurs de n les graphes suivant sont-ils réguliers ?

i. K_n **ii.** C_n **iii.** W_n **iv.** Q_n

Recherche 1 - 12

Si un graphe a x sommets et a arêtes, combien d'arêtes a son complémentaire ?

Recherche 1 - 13
Donner les matrices d'adjacence des graphes suivants :

1. K_4
2. $K_{1,4}$
3. $K_{2,3}$
4. C_4
5. W_4
6. Q_3

Recherche 1 - 14
Dessiner un graphe correspondant aux matrices d'adjacence suivantes :

1. $\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$
2. $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

Recherche 1 - 15
En Python par exemple, on peut rentrer naïvement un graphe sous forme de sa matrice d'adjacence à l'aide d'une liste de listes.
Déterminer un mini programme qui détermine le degré de chaque sommet et le nombre d'arêtes.

Recherche 1 - 16
Soit $\mathcal{G}_n = (X_n, A_n)$ un graphe simple avec $X_n = \{1, 2, ..., n\}$.
Les sommets i et j sont adjacents si, et seulement si, les entiers i et j sont premiers entre eux.
Donnez les matrices d'adjacences (dans l'ordre croissant des sommets) de $\mathcal{G}_4$ et $\mathcal{G}_6$ ainsi que des représentations de ces graphes.

Recherche 1 - 17 nombre d'arêtes

Combien d'arêtes au maximum contient un graphe simple ayant n sommets ?
Combien y a-t-il de graphes non isomorphes (on dit aussi « à isomorphisme près ») ayant 10 sommets et 44 arêtes ? 10 sommets et 43 arêtes ?
Combien y a-t-il de graphes de n sommets étiquetés ? Combien d'entre eux ont m arêtes ?

Recherche 1 - 18 graphes réguliers

On s'intéresse aux graphes dont tous les sommets sont de degré trois. Construisez de tels graphes ayant 4, 5, 6 ou 7sommets. Remarque ? Preuve ?
Et pour les graphes 4-réguliers ?

Recherche 1 - 19
Montrez que dans un groupe de six personnes, il y en a nécessairement trois qui se connaissent mutuellement ou trois qui ne se connaissent pas (on suppose que si A connaît B, B connaît également A).
Considérez une personne qui en connaît trois autres puis le cas contraire.

Recherche 1 - 20
Montrez que dans un groupe de personnes, il y a toujours deux personnes ayant le même nombre d'amis présents.
Cet exercice a déjà été traité auparavant...sous une autre forme

Recherche 1 - 21 modélisation et graphe biparti

Une chèvre, un chou et un loup se trouvent sur la rive d'un fleuve ; un passeur souhaite les transporter sur l'autre rive mais, sa barque étant trop petite, il ne peut transporter qu'un seul d'entre eux à la fois. Comment doit-il procéder afin de ne jamais laisser ensemble et sans surveillance le loup et la chèvre, ainsi que la chèvre et le chou ?
Deux rives, un couple (RG,RD) à chaque sommet d'un graphe

xs