

Licence Creative Commons



# Graphes

# Mathématique



Guillaume CONNAN

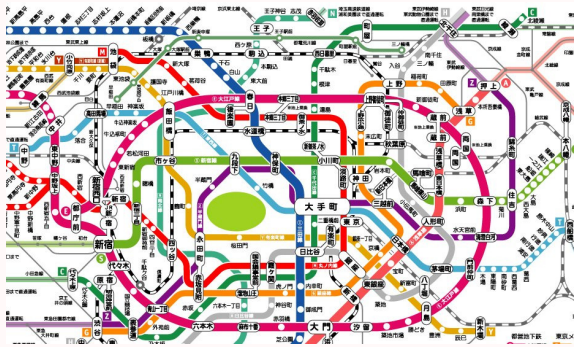
# TABLE DES MATIÈRES

<b>1 Graphes</b>	<b>4</b>
1.1 Comment un mathématicien regarde une carte	5
1.2 Les bases	7
1.2.1 Graphes symétriques	7
1.2.2 Graphes orientés	9
1.2.3 Quelques graphes simples particuliers	10
1.2.4 Sous-graphes et réunion de graphes	11
1.2.5 Matrices d'adjacence	12
1.3 Parcours de graphes	13
1.3.1 Algorithme général	13
1.3.2 Parcours en largeur	14
1.3.3 Parcours en profondeur	15
1.4 Fermeture transitive	17
1.5 Isomorphisme de graphes	18
1.6 Connexité	20
1.6.1 Chaînes et chemins	20
1.6.2 Nombre de chaînes et matrice d'adjacence	21
1.6.3 Connexité des graphes non orientés	22
1.6.4 Point d'articulation	23
1.6.5 Connexité des graphes orientés	25
1.6.6 Petit résumé du vocabulaire	26
1.7 Chaînes, chemins, circuits et cycles eulériens	27
1.7.1 Graphes symétriques	27
1.7.2 Graphes orientés	29
1.8 Chaînes, chemins, circuits et cycles hamiltoniens	29
1.9 Chemins de longueur minimum	30
1.9.1 Graphe pondéré	30
1.9.2 Algorithme de DIJKSTRA	31
1.10 Coloration	31
1.11 Graphe planaire	36
1.12 Arbres de recouvrement de poids extremum	36
1.12.1 Généralités	36
1.12.2 Algorithme de Kruskal	37
1.13 Flots et réseaux de transport	37
1.13.1 Réseau de transport	37
1.13.2 Flot dans un réseau de transport	37
1.13.3 Algorithme de Ford et Fulkerson	38
1.14 Quelques autres applications	39
1.14.1 Ordonnancement	39
1.14.2 Codage de Prüfer	43
1.15 Les graphes avec Python	45
1.15.1 Présentation	45
1.15.2 La classe « graphe »	47
1.15.3 Un peu de travail...	48
1.16 EXERCICES	51
1.16.1 Les bases	51
1.16.2 Isomorphisme	52
1.16.3 Connexité	52
1.16.4 Euler et Hamilton	52
1.16.5 Plus court chemin	53

---

1.16.6 Coloration . . . . .	54
1.16.7 Exercices divers . . . . .	55
1.16.8 Exercices supplémentaires sur les graphes . . . . .	56
1.16.9 Exercices semaine 38 . . . . .	59
1.16.10 Semaine 39 : BELLMAN-FORD et DIJKSTRA . . . . .	60
1.16.11 Semaine 40 : coloration . . . . .	63
1.16.12 Semaine 41 : arbre de recouvrement et flots . . . . .	66

# Graphes



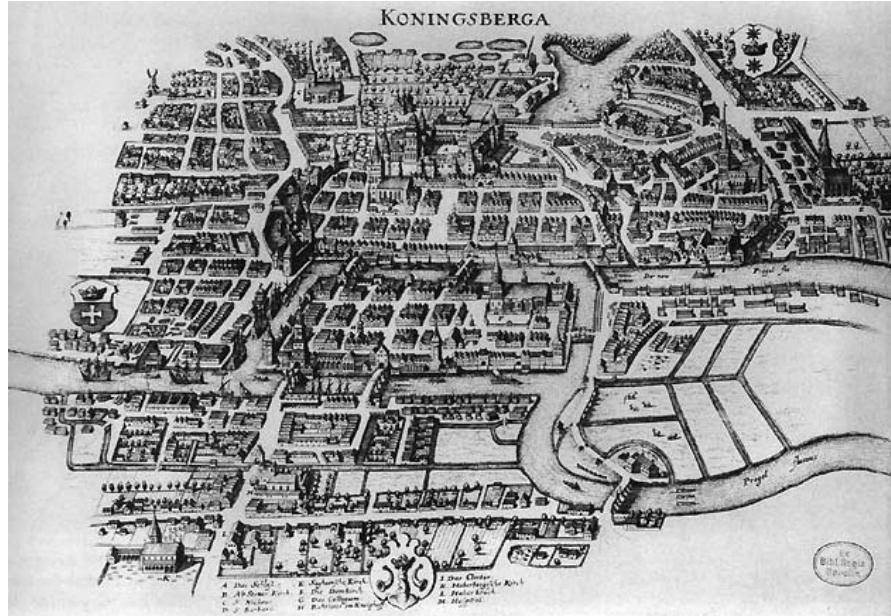
Tout à commencé au XVIII<sup>e</sup> siècle avec une histoire de ponts résolue par EULER puis ce gadget s'est fait oublier jusque dans les années 1960 où on a enfin découvert toutes les richesses qui permettent de résoudre de nombreux problèmes informatiquement : circuits électriques, réseaux de transport,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

Cependant, le célèbre et illustre mathématicien français Alain CONNES estime que les connaissances sur la théorie des graphes ne forment pas une théorie mais « un savoir, une série de faits » .

# 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 :

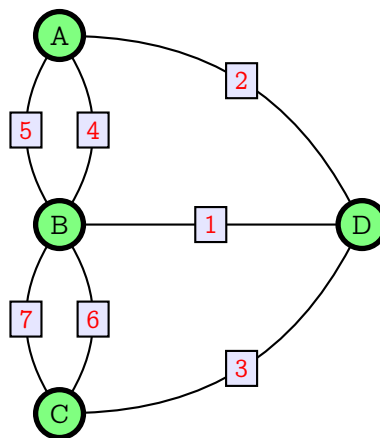


FIGURE 1.1 – Graphe des ponts de Königsberg

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).

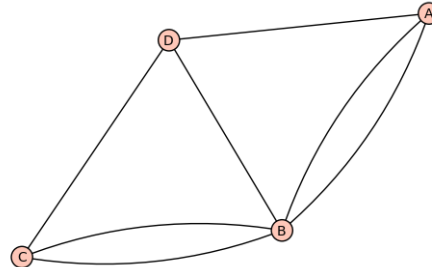
Le logiciel Sage sera notre ami pour vérifier nos raisonnements.

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','D'],'D':['B','C'],'C':['B','B']})
```

On peut obtenir un dessin :

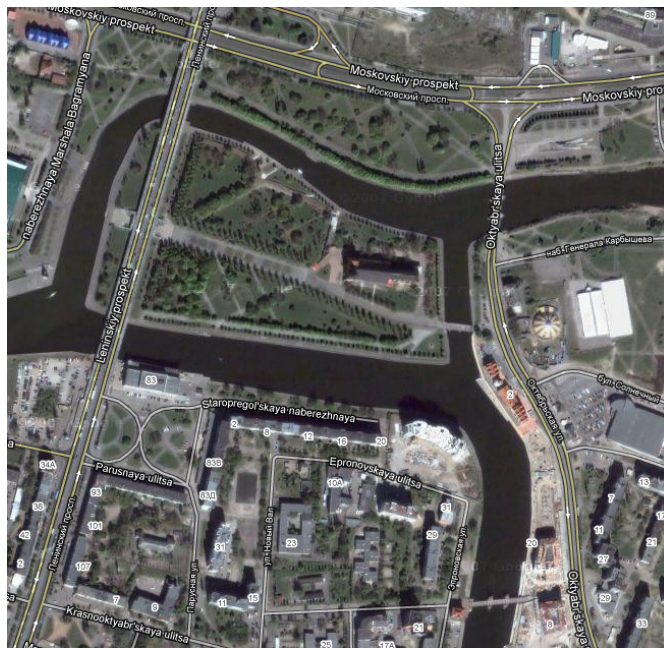
```
sage: g.show()
```



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.

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
```

Recherche

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 :  $A \in X \times X$  ?  $A \subseteq X \times X$  ?  $A \in \mathcal{P}(X \times X)$  ?  
Aucune des trois ?

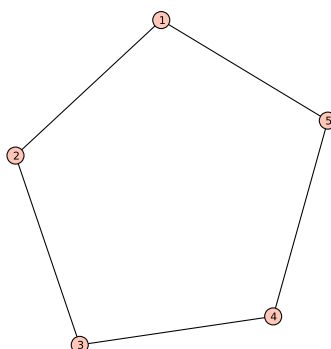
On pourra noter  $\binom{X}{k}$  l'ensemble des sous-ensembles de  $X$  à  $k$  éléments. Ainsi  $A \subseteq \binom{X}{2}$ .

#### Définition 1 - 2

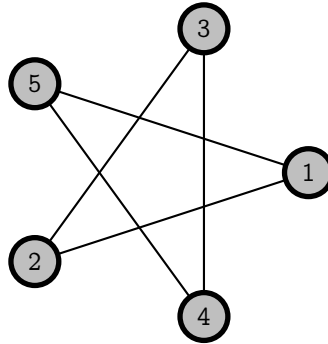
Si  $\{s, t\}$  est une arête d'un graphe  $\mathcal{G}$ , on dit que les sommets  $s$  et  $t$  sont **adjacents** ou que  $s$  est un **voisin** de  $t$  ou que l'arête  $\{s, t\}$  est **incidente** aux sommets  $s$  et  $t$  ou que  $s$  et  $t$  sont les **extrémités** de l'arête.

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

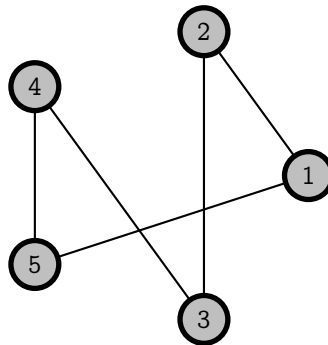
```
sage: g=Graph()
sage: g.add_vertices([1,2,3,4,5])
sage: g.add_edges([(1,2), (2,3), (3,4), (4,5), (5,1)])
sage: g.show()
```



Mais on aurait très bien pu dessiner ça :



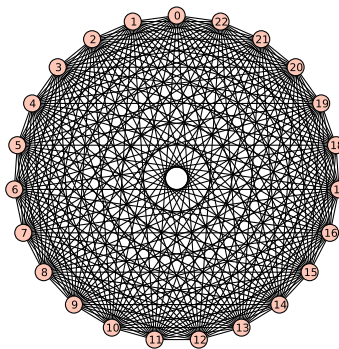
ou ça :



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

$$(\deg_{\mathcal{G}}(s_1), \deg_{\mathcal{G}}(s_2), \dots, \deg_{\mathcal{G}}(s_n))$$

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 :

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

**Théorème 1 - 1**

$$\sum_{s \in X} \deg_{\mathcal{G}}(s) = 2 \times \text{Card}(A)$$

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 Graphes orientés

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

Un **graphe orienté** est un couple  $(X, A)$  où  $X$  est un ensemble quelconque et  $A$  un sous-ensemble de  $X \times X$  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  $t$ .

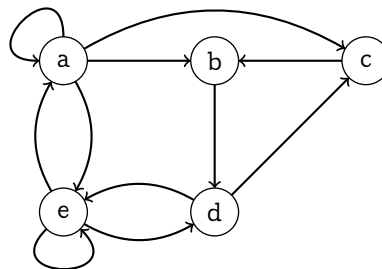
**Définition 1 - 6**

Un **chemin**  $c$  du graphe  $\mathcal{G} = (X, A)$  est une suite finie d'arcs  $a_1, a_2, \dots, a_n$  telle que,  $\forall i, 1 \leq i \leq n, or(a_{i+1}) = ext(a_i)$ . 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 :

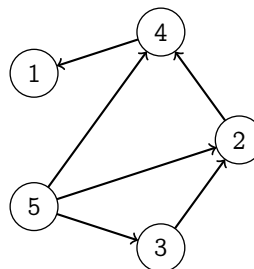
$$\sum_{s \in X} \text{deg}_{\mathcal{G}}^{-}(s) + \sum_{s \in X} \text{deg}_{\mathcal{G}}^{+}(s) = \text{Card}(X)$$

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'extrémité  $x_k$  ; 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 :



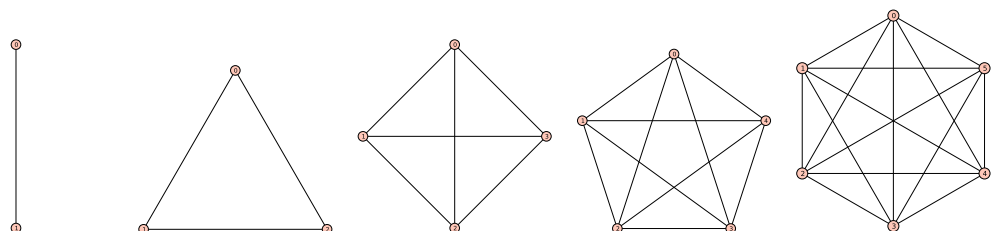
Le sommet 1 n'a pas de descendant,  $\mathcal{G}^+(\{1\}) = \emptyset$ . Ses ascendants sont tous les autres sommets,  $\mathcal{G}^-(\{1\}) = \{2, 3, 4, 5\}$ . Pour ce qui concerne les descendants et les ascendants du sommet 5, on a

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

**2 3 Quelques graphes simples particuliers**

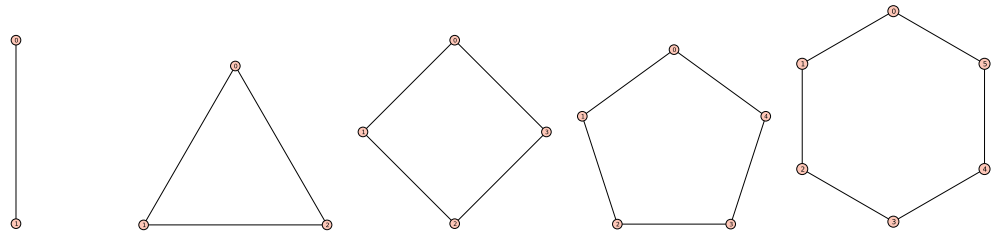
**Définition 1 - 9**

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



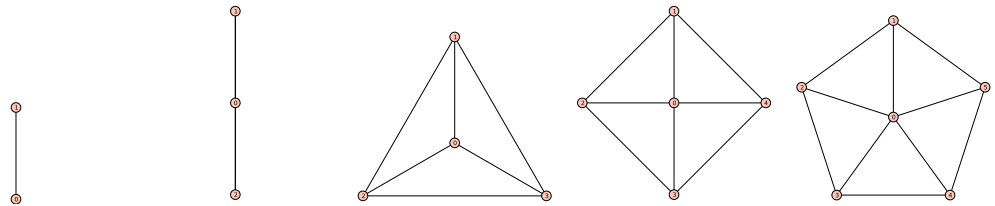
**Définition 1 - 10**

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



Définition 1 - 11

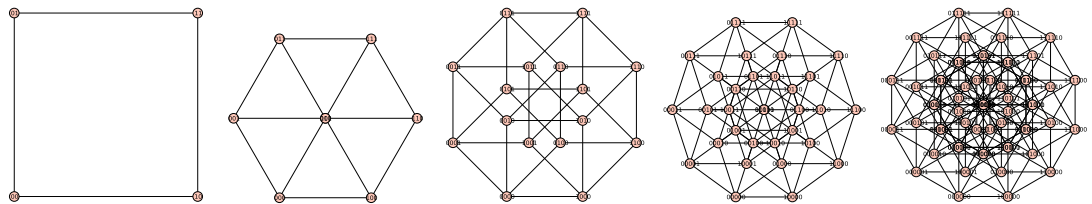
La roue  $W_n$  est telle que  $X = \{0, 1, 2, \dots, n\}$  et  $A = \{\{1, 2\}, \{2, 3\}, \dots, \{n, 1\}\} \cup \{\{0, 1\}, \{0, 2\}, \dots, \{0, n\}\}$  :



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

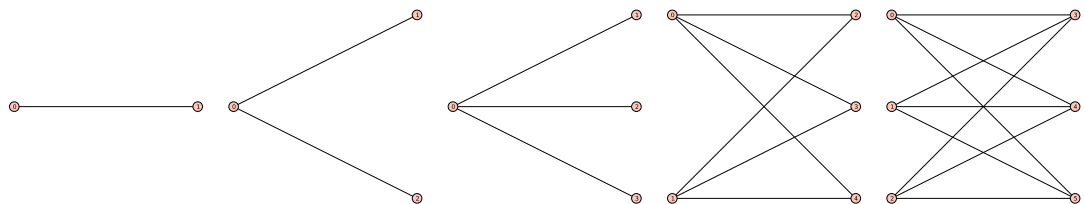
**Graphe biparti**  
 Un graphe  $\mathcal{G} = (X, A)$  est biparti si  $X$  peut être divisé en deux ensembles disjoints  $X_1$  et  $X_2$  de sorte que chaque arête de  $A$  relie un sommet de  $X_1$  et un sommet de  $X_2$ .  
 Ainsi  $A \subseteq \{\{x_1, x_2\}, x_1 \in X_1, x_2 \in X_2\}$ .

Ce qui nous permet d'agrandir notre famille :

Définition 1 - 14

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

Qui est qui ?



**2 4 Sous-graphes et réunion de graphes**

Définition 1 - 15

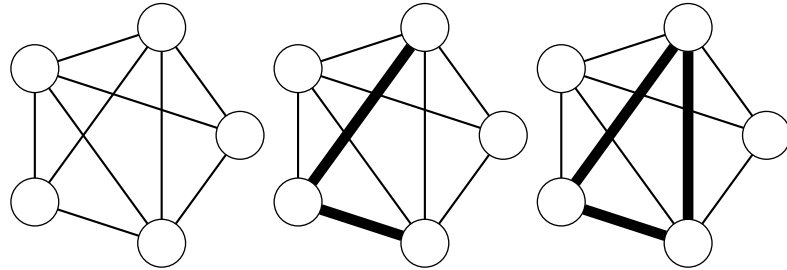
Soit  $\mathcal{G} = (X, A)$  et  $\mathcal{G}' = (X', A')$  deux graphes.  
 On dit que  $\mathcal{G}$  est un **sous-graphe** de  $\mathcal{G}'$  si  $X \subseteq X'$  et  $A = A' \cap \binom{X}{2}$ .  
 On dit que  $\mathcal{G}$  est un **sous-graphe partiel** de  $\mathcal{G}'$  si  $X' \subseteq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

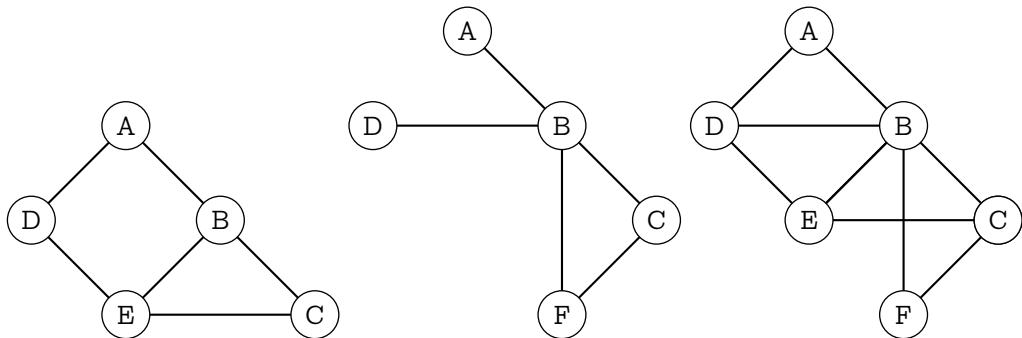


Au lieu d'extraire, on peut combiner des graphes :

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

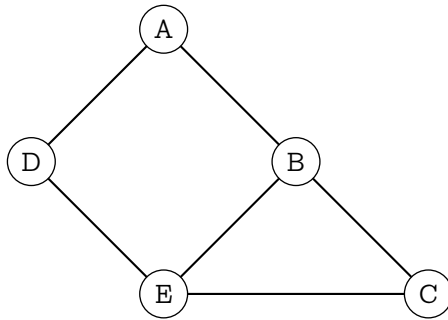


**2 5 Matrices d'adjacence**

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  $s_1, s_2, \dots, s_n$  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  $n \times n$ ,  $M_{\mathcal{G}} = (m_{ij})_{1 \leq i, j \leq n}$ , 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  $\text{Card}(X)^2$  alors qu'un tableau présentant les listes d'adjacence de chaque sommet n'occupe qu'un espace de l'ordre de  $\text{Card}(X) + \text{Card}(A) \dots$

### 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  $x \in X$  Faire
  | etat[x]  $\leftarrow$  non_atteint
FinPour
 $T \leftarrow \emptyset$ 
à_traiter  $\leftarrow \emptyset$ 
Pour tout  $x \in X$  Faire
  | Si etat[x]=non_atteint Alors
  |   | à_traiter  $\leftarrow$  à_traiter  $\cup$  {x}
  |   | etat[x]  $\leftarrow$  atteint
  |   TantQue à_traiter  $\neq \emptyset$  Faire
  |     | y  $\leftarrow$  choix(à_traiter)
  |     | à_traiter  $\leftarrow$  à_traiter  $\setminus$  {y}
  |     Pour tout z voisin de y Faire
  |       | Si etat[z] = non_atteint Alors
  |       |   |  $T \leftarrow T \cup \{(y, z)\}$ 
  |       |   | etat[z]  $\leftarrow$  atteint
  |       |   | à_traiter  $\leftarrow$  à_traiter  $\cup$  {z}
  |       |   FinSi
  |     FinPour
  |     etat[y]  $\leftarrow$  examine
  |   FinTantQue
  | FinSi
FinPour

```

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

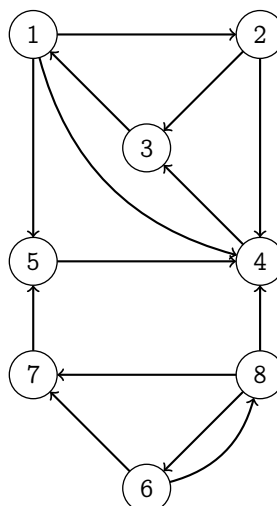
### 3 2 Parcours en largeur

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 la 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  $x \in X$  Faire
  | etat[x] ← non_atteint
FinPour
index ← 1
à_traiter ← ∅
Pour tout  $z \in X$  Faire
  | Si etat[z]=non_atteint Alors
  | | pere[z] ← NIL
  | | ordre[z] ← index
  | | index ++
  | | à_traiter.AjouterEnQueue(z)
  | | TantQue à_traiter ≠ ∅ Faire
  | | |  $x \leftarrow Tete(\text{à\_traiter})$ 
  | | | 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
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 parents des sommets visités.

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

```

Procédure parcours_pro( $x$  : sommet)
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  $x \in X$  Faire
  etat[ $x$ ] ← non_atteint
FinPour
index ← 1
Pour chaque  $x \in X$  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).

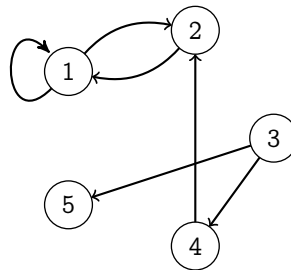
Nous savons aussi que la matrice booléenne de  $\mathcal{G}^*$  est  $\sum_{k=0}^{n-1} M^k$  et que son calcul effectif peut être laborieux (mais on a Sage...).

L'autre méthode est la suivante :

**Théorème 1 - 4** Nous notons  $M_0$  la matrice booléenne de  $\mathcal{G}$  et  $\theta_i$  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  $\Gamma$ , 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  $\theta_1$ , 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  $M_1 = \theta_1(M_0) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ . Appliquons

maintenant  $\theta_2$  à  $M_1$ , nous allons ajouter la ligne numéro 2 à toute ligne possédant un 1 en colonne

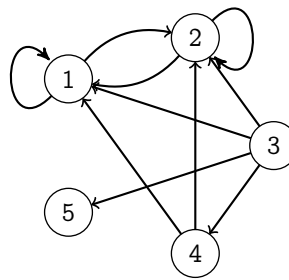
numéro 2, les lignes 1, 2 et 4 sont concernées<sup>a</sup>. Nous obtenons  $\theta_2(M_1) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ .

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

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

$$4, \theta_4(M_3) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} = M_4 \text{ et } \theta_5 \text{ ne changeant rien, la matrice de la fermeture}$$

$$\text{transitive est } M_5 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ qui nous donne le graphe de la fermeture transitive de } \mathcal{G}.$$



## 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.

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

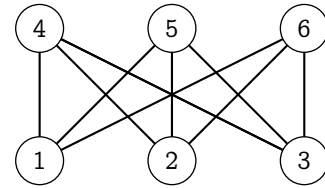
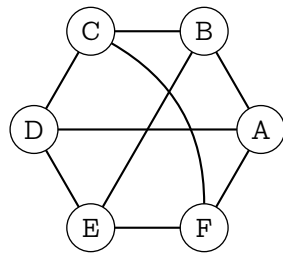
Les graphes simples  $\mathcal{G} = (X, A)$  et  $\mathcal{G}' = (X', A')$  sont dits **isomorphes** s'il existe une bijection  $f : X \rightarrow X'$  telle que nous ayons, pour tout couple de sommets distincts du premier graphe  $(x, y) \in X^2$ ,  $x \neq x'$ ,

l'arête  $\{x, y\} \in A$  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  $\mathcal{G} \cong \mathcal{G}'$ .

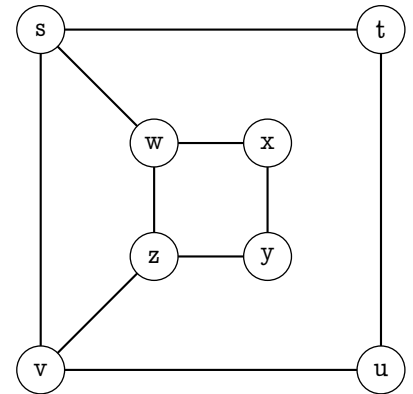
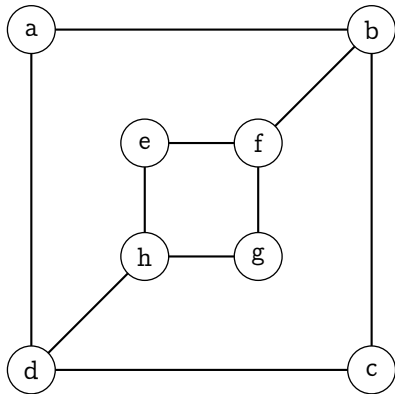
Un isomorphisme conserve donc « l'adjacence » des sommets et conserve donc les degrés.

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.

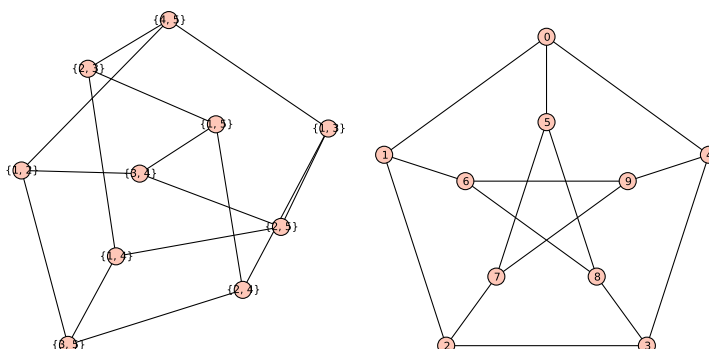
Il est paradoxal de remarquer qu'on ne connaît pas d'algorithme efficace pour reconnaître si deux graphes sont isomorphes (on ne sait même pas si c'est envisageable) alors que le problème paraît facile a priori.

Cependant Sage, par exemple, peut vous aider pour des graphes de taille raisonnable :

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

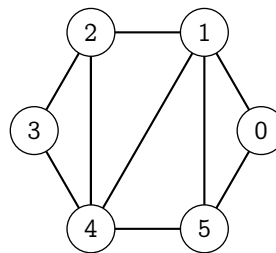
Nous avons déjà parlé de chemins au sujet des graphes orientés. Le pendant pour les graphes symétriques est parfois appelé *chaîne*.

Définition 1 - 20

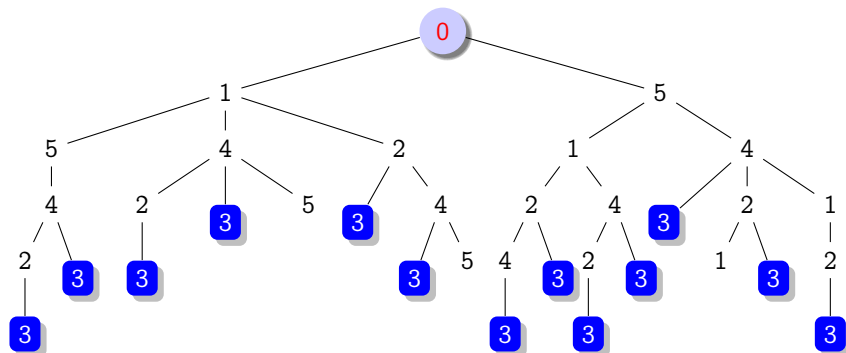
Soit  $\mathcal{G} = (X, A)$  un graphe non orienté. Une **chaîne de longueur n** de  $s$  à  $t$  dans  $\mathcal{G}$  est une suite d'arêtes  $a_1, a_2, \dots, a_n$  de  $\mathcal{G}$  telle que  $a_1 = \{s, x_1\}, a_2 = \{x_1, x_2\}, \dots, a_n = \{x_{n-1}, t\}$ .  
 Si le graphe est simple, on peut juste donner la suite des sommets  $s, x_1, \dots, x_{n-1}, t$ .  
 La chaîne est **fermée** si elle commence là où elle se termine et que sa longueur est non nulle.  
 On dit que la chaîne **passse** 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 Exercice 1 - 13 page 52).

**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  $X = \{x_1, x_2, \dots, x_n\}$ , 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  $m_{ip} \times m_{pj}^{(k-1)}$ .

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, soit  $x_i$  et  $x_j$  deux sommets d'un graphe  $\mathcal{G}$ . Alors :

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

Reprenons par exemple le graphe illustrant la définition 1 - 20 page précédente.

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 Sage :

```

sage: M=matrix([[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]])

sage: M^2
[2 1 1 0 2 1]
[1 4 1 2 2 2]
[1 1 3 1 2 2]
[0 2 1 2 1 1]
[2 2 2 1 4 1]
[1 2 2 1 1 3]

sage: M^3
[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 :

```

sage: def d_G(M,i,j):
    m=M
    k=1
    while m[i][j]==0:
        m*=M
        k+=1
    return k
.....: .....: .....: .....: .....: .....:
sage: d_G(M,0,3)
3

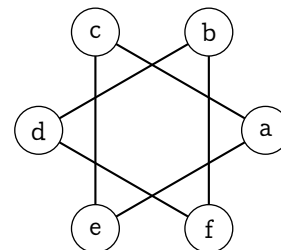
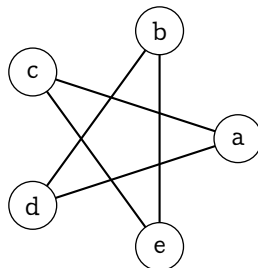
```

### 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'oubliez 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 connectivité qui renvoie la connectivité d'un graphe.

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

```

Variable
  S : liste de sommets
  c : entier
Début
  S ← X
  c ← 0
  TantQue S est non vide Faire
    Choisir s dans S
    Lister tous les sommets joignant s par n'importe quel chaîne
    Supprimer ces sommets et s de S et les arêtes correspondantes
    c ← c+1
  FinTantQue
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.

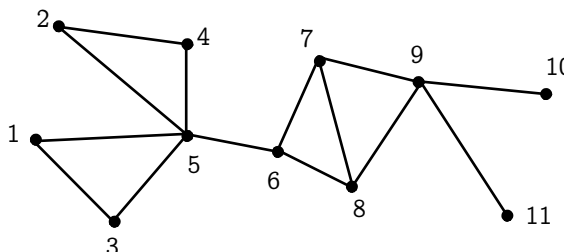
On construit ainsi une **relation d'équivalence**. Les **classes d'équivalence** de cette relation sont appelées **composantes connexes**. Un graphe ne comportant qu'une seule composante connexe est dit **connexe**

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 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[z] }
|     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.

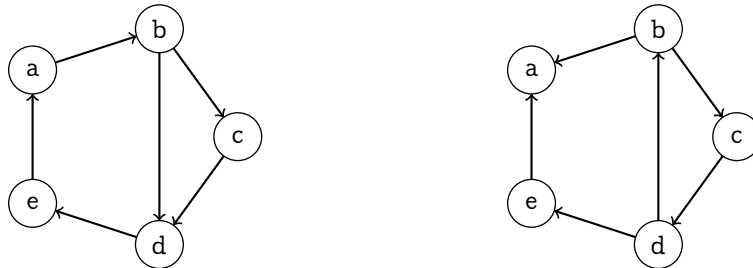
Une partie (non vide)  $Y$  de  $X$  est **fortement connexe** si, et seulement si, le sous graphe défini par  $Y$  ou engendré par  $Y$ , c'est-à-dire la restriction du graphe  $\mathcal{G} = (X, A)$  à  $Y$ ,  $(Y, A \cap (Y \times Y))$ , est fortement connexe ; cela signifie que pour toute paire de 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

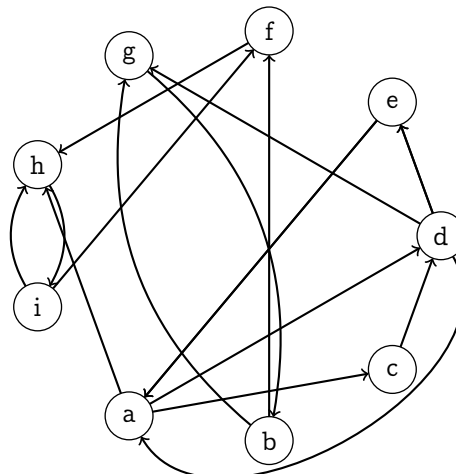
Une partie restreinte à un seul sommet, un singleton de  $X$ , ne contredit pas la définition, c'est une partie fortement connexe.

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 <sup>b</sup> 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 :



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

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

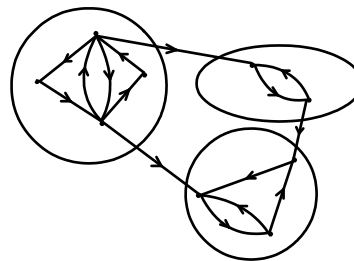
$$\{c, d, e\}$$

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 :



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

**6 6 Petit résumé du vocabulaire**

$\mathcal{G} = (X = \{x_1, x_2, \dots, x_n\}, A)$  est un graphe orienté avec  $\text{Card}(A) \geq 2$  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  $(y_1, y_2, \dots, y_k)$  avec  $y_i \in X$  est une chaîne, qu'est-ce que cela signifie ?  $i \in \{1, 2, \dots, n - 1\}$  on a  $(y_i, y_{i+1})$  ou  $(y_{i+1}, y_i) \in A$ .
3. On vous dit que la suite  $(y_1, y_2, \dots, y_k)$  avec  $y_i \in X$  est un chemin, qu'est-ce que cela signifie ?  $\forall i \in \{1, 2, \dots, n - 1\}$  on a  $(y_i, y_{i+1}) \in A$ .
4. Que signifie  $\mathcal{G}$  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  $(x_i, x_j) \in X \times X$  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  $\mathcal{G}_Y = (Y, A \cap (Y \times Y))$ .
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  $\{x_i\} \cup (\mathcal{G}^+(x_i) \cap \mathcal{G}^-(x_i))$

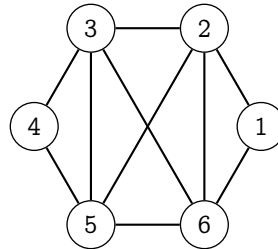
**À retenir**

## 7

## Chaînes, chemins, circuits et cycles eulériens

## 7.1 Graphes symétriques

Vous avez sûrement joué à ceci à la maternelle (ou pendant un cours d'amphi) : dessiner ce graphe sans lever le stylo du papier et en ne traçant chaque arête qu'une seule fois en partant et en revenant au même sommet...



Une telle chaîne est appelée un **cycle eulérien** : c'est un cycle qui contient toutes les arêtes du graphe.

Un graphe qui contient un cycle eulérien est un **graphe eulérien**.

On peut ne pas former de cycle mais seulement une chaîne simple : une **chaîne eulérienne** est une chaîne simple qui contient toutes les arêtes du graphe.

Que pensez-vous du graphe suivant :



## À retenir

Les arêtes n'interviennent qu'une fois mais les sommets peuvent apparaître plusieurs fois.

Ce problème avait déjà été évoqué au début du cours au sujet des ponts de Königsberg.

C'est d'ailleurs en cherchant à résoudre ce problème qu'EULER a émis et démontré le théorème suivant :

## Théorème 1 - 11

Un graphe symétrique  $\mathcal{G} = (X, A)$  est eulérien si, et seulement si, il est connexe et si chaque sommet est de degré pair.

Avant de le démontrer, que pouvez-vous en déduire pour les ponts de Königsberg? Peut-on transformer le plan de la ville pour améliorer la situation?

Occupons-nous d'abord de la *condition nécessaire* : il est évident que le graphe doit être connexe. De plus, si un cycle arrive en un sommet, il doit pouvoir en repartir sans emprunter une arête déjà utilisée : on peut donc mettre en bijection l'ensemble des arêtes entrantes et l'ensemble des arêtes sortantes, ce qui signifie que les sommets sont tous de degré pair.

La *condition* est aussi *suffisante*. Partons d'un sommet quelconque. On avance en parcourant des arêtes. À chaque fois qu'on « entre » en un sommet, on en « sort » par une autre arête car son degré est pair. On continue ainsi à « traverser » des sommets sans être bloqué en empruntant des arêtes différentes. Cependant, le nombre de sommets est fini et il reste un sommet emprunté une seule fois, celui de départ : on y retourne donc forcément et on a ainsi *construit* un cycle avec a priori une partie des arêtes empruntées une et une seule fois.

Considérons maintenant une arête non empruntée, s'il en existe une (sinon, on a trouvé un cycle eulérien). Il y a donc au moins une de ses extrémités qui n'appartient pas au cycle initial. Comme le graphe est connexe, il existe une chaîne vers un des sommets du cycle. Éliminons dans le cycle de départ tous les sommets sauf celui-là (qu'on notera  $s$ ) et toutes les arêtes empruntées. Il nous reste un sous-graphe ayant un sommet en commun avec le cycle initial. Tous ses sommets sont de degré pair, certes, mais il peut ne pas être connexe. On peut cependant construire un cycle

eulérien comme précédemment dans ce sous-graphe et on fusionne ces deux cycles qui n'ont aucune arête en commun.

On réitère ce procédé jusqu'à ne laisser aucune arête non parcourue. Il se termine car il y a un nombre fini d'arêtes.

Comme le procédé est constructif, on peut en déduire un premier algorithme qui renvoie un cycle eulérien dans un graphe connexe dont les sommets sont pairs :

```

Fonction Euler1(G : graphe connexe de sommets pairs) : cycle
Variable
  | C : cycle dans G qui commence en un sommet choisi arbitrairement et dont les arêtes
  | s'ajoutent jusqu'à obtenir un cycle
  | H : le graphe obtenu en éliminant de G les arêtes de C
Début
  | TantQue H contient des arêtes Faire
  |   | sousC ← un cycle dans H qui commence en un sommet commun à H et C
  |   | H ← le sous-graphe obtenu en éliminant les arêtes de sousC et tous les sommets
  |   | isolés
  |   | C ← C fusionné avec sousC
  | FinTantQue
  | Retourner C
Fin
    
```

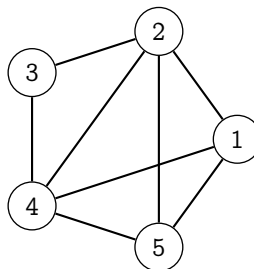
Tout est dans la fusion...

Voici deux autre théorèmes que nous admettrons mais qui nous permettrons de répondre à certains problèmes :

**Théorème 1 - 12**

Un graphe symétrique admet une chaîne eulérienne si deux sommets *s* et *t* sont de degré impair et que tous les autres sont de degré pair, la chaîne ayant pour extrémité *s* et *t*.

Cela nous guide pour résoudre le problème de la maison :



Ces théorèmes sont liés à un problème classique, celui du *postier chinois*, la nationalité du postier étant celle de Meigu Guan qui le proposa en 1962 : est-ce qu'un postier peut faire sa tournée en ne repassant jamais par la même rue ?

Voici un autre problème, qu'on pourrait appeler le problème du musée : si nous partons de l'entrée et que nous parcourons les salles du musée en choisissant la suivante parmi celles qui n'ont pas été encore visitées, on peut toujours revenir à notre point de départ.

Mathématisons un peu :

**Définition 1 - 26**

Un graphe est dit **aléatoirement eulérien** en un sommet *x* si chaque chaîne simple maximal qui commence en *x* est un cycle eulérien dans ce graphe.

Voici la clé du problème :

**Théorème 1 - 13**

Un graphe connexe simple  $\mathcal{G} = (X, A)$  dont tous les sommets sont de degré pair est aléatoirement eulérien en un sommet *x* si, et seulement si, le graphe  $(X \setminus \{x\}, \{a \in A \mid x \notin a\})$  ne contient pas de cycle.

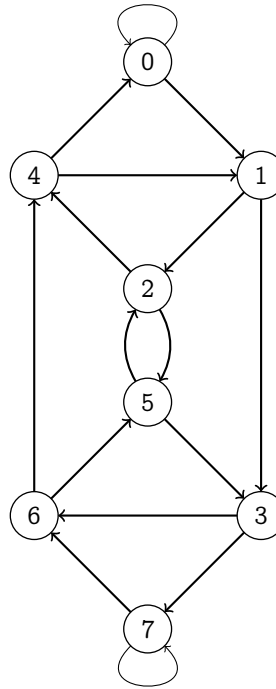
**7 2 Graphes orientés**

Si on met des sens uniques pour traverser les ponts :

**Théorème 1 - 14**

Un graphe *orienté* possède un circuit eulérien si, et seulement si, il est faiblement connexe et que le nombre d'arcs arrivant en chaque sommet est égal au nombre d'arcs sortant de ce sommet.

Voici une méthode simple pour trouver un circuit eulérien dans un graphe orienté. Voyons-le sur un exemple : le graphe de DE BRUIJN d'ordre 3.



Nous vérifions tout d'abord qu'un circuit eulérien existe bien.

Remplissons ensuite le tableau suivant où nous mettons tous les arcs avec sur la première ligne les origines et en bas les extrémités :

or	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
ext	0	1	2		4				0							

On cherche ensuite des circuits en prenant les arcs successivement puis en les éliminant dès qu'un circuit est formé.

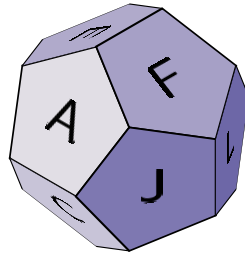
Vérifiez qu'on obtient les circuits (0), (0124), (1364), (25), (3765) et (7).

Il ne reste plus qu'à les fusionner de proche en proche. Quel est le circuit obtenu ?

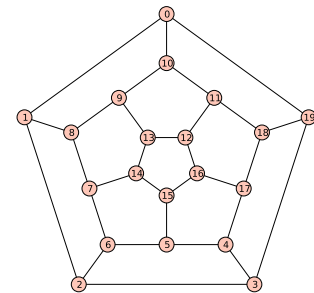
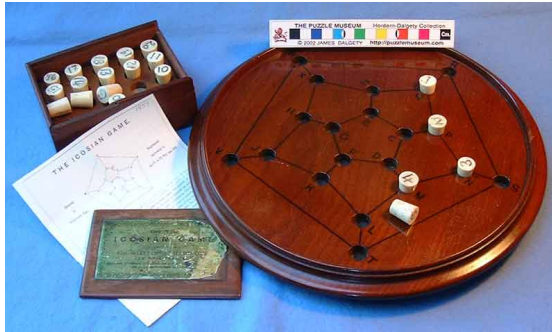
**8 Chaînes, chemins, circuits et cycles hamiltoniens**

William HAMILTON a été un grand mathématicien irlandais à l'origine de la théorie des quaternions. Il a également, pendant ses temps libres, commercialisé un casse-tête en 1857, sous deux formes.

Une sorte de dodécaèdre tout d'abord (*The Travellers Dodecahedron*) où 20 punaises figurent 20 villes qu'il faut visiter en suivant les arêtes mais en ne passant qu'une seule fois par chaque ville :



Une autre version (*The Icosian Game*) consiste à parcourir le graphe représenté en suivant les arêtes et en visitant chaque sommet une, et une seule fois :



C'est pourquoi on appelle cycle hamiltonien d'un graphe symétrique une chaîne qui passe par tous les sommets du graphe une, et une seule fois avant de revenir au sommet de départ.

Un graphe qui contient un cycle hamiltonien est appelé un **graphe hamiltonien**.

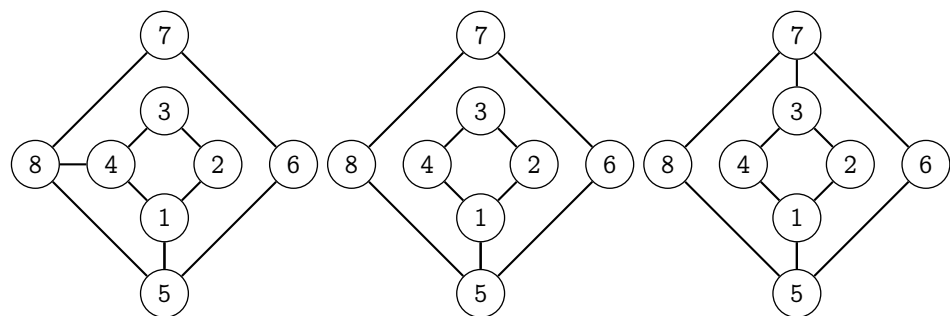
Malheureusement, à la différence des graphes eulériens, il n'existe aucune propriété générale permettant de conclure si un graphe est hamiltonien ou non.

Il y a malgré tout quelques petites propriétés locales qui peuvent nous aider.

**Théorème 1 - 15**

- Si un graphe possède un sommet de degré 1, il n'est pas hamiltonien.
- Si un cycle hamiltonien existe, il traverse obligatoirement les sommets de degré 2.

Pourquoi? Cela rend le passage par certaines arêtes obligatoire et permet parfois de conclure. Par exemple, existe-t-il des cycles hamiltoniens dans les graphes suivant :



**9 Chemins de longueur minimum**

**9 1 Graphe pondéré**

C'est un graphe dont les arêtes sont affectées d'un « poids ». Par exemple, on peut penser à un réseau routier, les sommets étant des villes, les arêtes des routes entre ces villes pondérées par la distance séparant les deux villes sommets de l'arête. On peut alors chercher le plus court chemin d'une ville à l'autre (GPS) ou le plus court chemin passant par tous les sommets (problème du

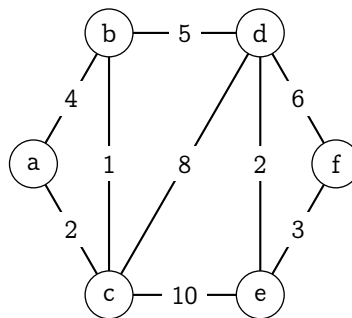
voyageur de commerce).

On appelle **matrice de pondération** d'un graphe  $\mathcal{G} = (X, A)$  la matrice dont les coefficients correspondant aux sommets  $s$  et  $t$  vaut :

Définition 1 - 27

$$\begin{cases} 0 & \text{si } s = t \\ \infty & \text{si } \{s, t\} \text{ n'est pas une arête} \\ p & \text{si } \{s, t\} \text{ est une arête de poids } p \end{cases}$$

Voici un graphe pondéré :



Écrire sa matrice de pondération.

**9 2 Algorithme de Dijkstra**

Edsger DIJKSTRA (prononcer « Deijkstra » en roulant le r...) est un mathématicien néerlandais mort en 2002 à l'âge de 72 ans. Il a mis au point un algorithme permettant d'obtenir le plus court chemin (chaîne) entre deux sommets d'un graphe pondéré simple connexe et dont les poids sont positifs.

Le plus simple est de remplir un tableau. Voici celui correspondant à la recherche de la plus courte chaîne entre a et f dans le graphe précédent :

Sommets	b	c	d	e	f
a	4	2	$\infty$	$\infty$	$\infty$
c	3	-	10	12	$\infty$
b	-	-	8	$\infty$	$\infty$
d	-	-	-	10	14
e	-	-	-	-	13

Sauriez-vous l'expliquer ?

**10 Coloration**

Définition 1 - 28

Un **stable** d'un graphe est un sous-graphe sans arête. Le **nombre de stabilité**  $\alpha(\mathcal{G})$  d'un graphe  $\mathcal{G}$  est la taille d'un stable maximum.

**Colorier**<sup>a</sup> les sommets du graphe simple non orienté  $\mathcal{G} = (X, A)$ , c'est affecter une couleur à chaque sommet de sorte que deux sommets quelconques adjacents aient toujours des couleurs différentes. Le nombre minimal de couleurs nécessaires pour colorier les sommets du graphe est appelé le **nombre chromatique** du graphe.

Définition 1 - 29

**Colorier les arêtes** du graphe c'est affecter une couleur à chaque arête du graphe de sorte que deux arêtes quelconques adjacentes aient des couleurs différentes. Le nombre minimal de couleurs pour colorier les arêtes du graphe est l'**indice chromatique** du graphe.

a. Extrait du dictionnaire Hachette :  
 colorer : donner une couleur ou de la couleur à ...  
 colorier : appliquer des couleurs sur ...  
 Vous pouvez donc remplacer colorier par colorer si vous préférez.

À retenir

Les ensembles monochromes sont stables donc la coloration permet d'obtenir une partition de  $X$  en un nombre minimum de stables.

Théorème 1 - 16

$$\chi(\mathcal{G}) = 1 \Leftrightarrow \mathcal{G} \text{ est stable}$$

$$\chi(\mathcal{G}) = 2 \Leftrightarrow \mathcal{G} \text{ est biparti}$$

Théorème 1 - 17

On a pour tout graphe  $\mathcal{G} = (X, A)$  :

$$\left\lceil \frac{n}{\alpha(\mathcal{G})} \right\rceil \leq \chi(\mathcal{G}) \leq n - \alpha(\mathcal{G}) + 1$$

Conséquence :

Théorème 1 - 18

Le graphe complet  $K_n$  est  $n$ -coloriable.

Définition 1 - 30

Un sous-graphe complet d'un graphe  $\mathcal{G}$  est appelé une **clique**. L'ordre de la plus grande clique dans  $\mathcal{G}$  est noté  $\omega(\mathcal{G})$ .

Théorème 1 - 19

$$\chi(\mathcal{G}) \geq \omega(\mathcal{G})$$

Théorème 1 - 20

**Tutte 1954**

Pour tout entier  $k \geq 2$ , il existe un graphe  $k$ -chromatique sans clique à 3 sommets (triangle).

Théorème 1 - 21

**Nordhaus et Gaddum - 1960**

Soit  $\mathcal{G} = (X, A)$  un graphe et  $\bar{\mathcal{G}} = (X, \bar{A})$  son complémentaire.

1.  $\chi(\mathcal{G}) \cdot \chi(\bar{\mathcal{G}}) \geq n$
2.  $\chi(\mathcal{G}) + \chi(\bar{\mathcal{G}}) \leq n + 1$
3.  $\chi(\mathcal{G}) + \chi(\bar{\mathcal{G}}) \geq 2\sqrt{n}$
4.  $\chi(\mathcal{G}) \cdot \chi(\bar{\mathcal{G}}) \leq \left(\frac{n+1}{2}\right)^2$

On admettra le théorème suivant :

Théorème 1 - 22

**Vizing - 1964**

Pour tout graphe  $\mathcal{G}$ ,

$$\Delta(\mathcal{G}) \leq \chi'(\mathcal{G}) \leq \Delta(\mathcal{G}) + 1$$

avec  $\Delta(\mathcal{G})$  le degré maximum d'un sommet du graphe.



La coloration des sommets d'un graphe est un problème difficile en ce sens que si le nombre de sommets est grand et si le graphe comporte beaucoup d'arêtes, on ne connaît pas d'algorithme performant pour déterminer la solution minimale. Par contre l'algorithme suivant donne une solution satisfaisante même s'il ne donne pas forcément la solution minimale.

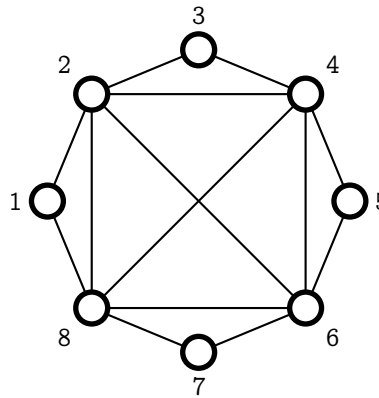
Soit donc le graphe  $\mathcal{G} = (X, A)$  et on cherche à colorier les sommets avec un minimum de couleurs.

La première étape consiste à déterminer sa matrice sommets-sommets (on a donc ordonné les sommets) et à déterminer le degré de chaque sommet.

On note  $\mathcal{G}_k = (X_k, A_k)$  et on affecte la valeur 0 à la variable  $k$ ,  $X_0 = X$  est l'ensemble des sommets non encore coloriés et  $A_0 = A$ .

1. Si  $X_k \neq \emptyset$ , on choisit un sommet  $s_k$  de degré maximal de  $X_k$  et on lui affecte la couleur  $c_k$ ; on considère maintenant  $\mathcal{G}_{k_1}$  le sous graphe de  $\mathcal{G}_k$  obtenu en supprimant  $s_k$  et tous ses voisins et, s'il n'est pas vide, on choisit un sommet  $s_{k_1}$  de degré maximal dans  $\mathcal{G}_{k_1}$  et on lui affecte la couleur  $c_{k_1}$ .  $\mathcal{G}_{k_2}$  est le sous graphe de  $\mathcal{G}_{k_1}$  obtenu en ayant supprimé dans  $\mathcal{G}_{k_1}$  le sommet  $s_{k_1}$  et tous ses voisins. Etc., jusqu'à ce que  $\mathcal{G}_{k_i}$  soit un graphe vide. À ce stade un certain nombre de sommets ont la couleur  $c_k$  et  $k \leftarrow k + 1$ . On note encore  $\mathcal{G}_k$  le sous graphe engendré par les sommets non encore coloriés et on retourne en 1.
2. Si  $X_k = \emptyset$ , on a affecté une couleur à chaque sommet.

Considérons le graphe



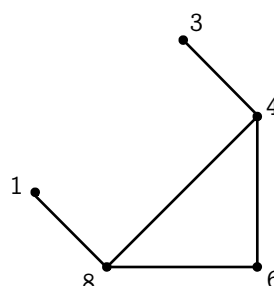
Déterminons le degré de chaque sommet

Sommets	1	2	3	4	5	6	7	8
Degré	2	5	2	5	2	5	2	5

Nous choisissons le sommet 2 et nous lui affectons la couleur  $c_1$ . Supprimons le sommet 2 et tous ses voisins, nous obtenons



et nous affectons la couleur  $c_1$  aux sommets 5 et 7. Maintenant considérons le sous graphe engendré par les sommets non encore coloriés :



Nous avons

Sommets	1	3	4	6	8
Degré	1	1	3	2	3

et nous choisissons le sommet 8 auquel nous affectons la couleur  $c_2$ . Supprimons ce sommet et tous ses voisins



Nous affectons la couleur  $c_2$  au sommet 3. Le sous graphe engendré par les sommets non encore coloriés est



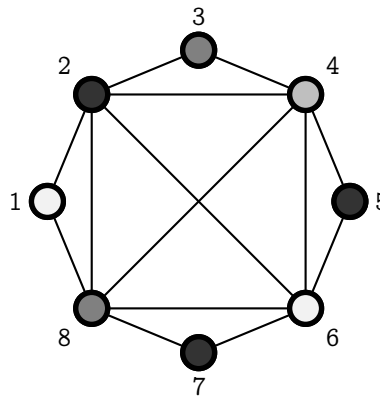
Le sommet 6 est un sommet de plus fort degré, affectons-lui la couleur  $c_3$ , supprimons-le ainsi que tous ses voisins



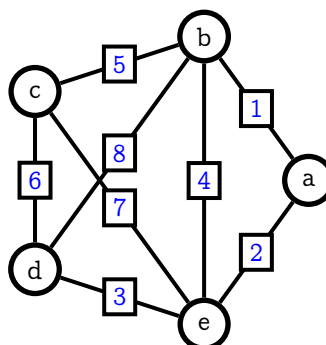
On affecte au sommet 1 la couleur  $c_3$ . Le sous graphe engendré par les sommets non encore coloriés est maintenant



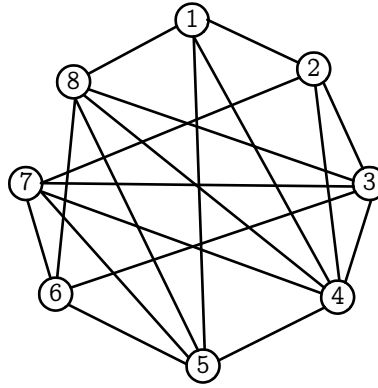
On affecte au sommet 4 la couleur  $c_4$  et tous les sommets sont coloriés. Nous avons réussi à colorier le graphe avec 4 couleurs :



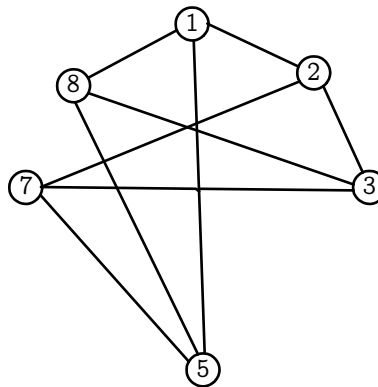
Pour colorier les arêtes d'un graphe on peut utiliser l'algorithme précédent sur le graphes aux arêtes, ce graphe se détermine ainsi, chaque arête est représentée par un sommet et deux sommets sont reliés par une arête si, et seulement si, ils correspondent à des arêtes adjacentes. Considérons le graphe dont nous voulons colorier les arêtes que nous avons numérotées :



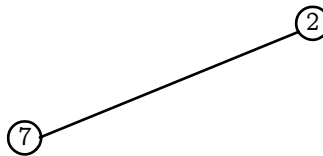
Son graphe aux arêtes est le suivant :



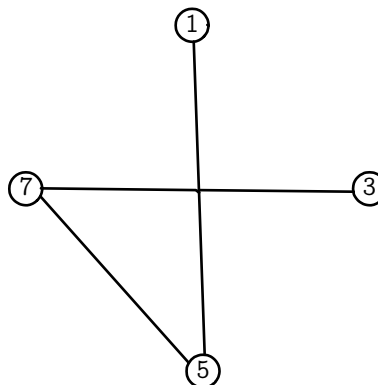
C'est le sommet 4 qui a le plus fort degré, affectons-lui la couleur  $c_1$  et supprimons le sommet 4 ainsi que tous ses voisins, il nous reste le sommet 6 auquel on affecte la couleur  $c_1$ . Le sous graphe engendré par les sommets non encore coloriés est alors



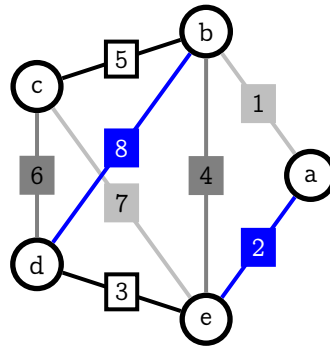
Choisissons le sommet 8 qui a un plus fort degré, affectons-lui la couleur  $c_2$ , supprimons-le ainsi que tous ses voisins.



Nous choisissons le sommet 2 (pourquoi pas) auquel nous affectons la couleur  $c_2$ . Le sous graphe engendré par les sommets non encore coloriés est maintenant



Nous choisissons le sommet 7 et nous lui affectons la couleur  $c_3$  ainsi qu'au sommet 1. Les sommets 5 et 3 sont alors coloriés avec la couleur  $c_4$  : le résultat est



4 couleurs suffisent. Il est inutile de chercher une coloration avec moins de couleurs car il est évident qu'un minimum de 4 couleurs est nécessaire puisque le graphe possède au moins un sommet de degré 4.

Une utilisation de la coloration est la suivante : on désire faire passer  $k$  examens écrits dans un laps de temps minimum sans créer d'impossibilité à des étudiants inscrits à plusieurs examens. On impose, par exemple, que chaque étudiant ait au plus un examen par jour, que tous les étudiants inscrits à un même examen le passent en même temps et on veut rendre minimum le nombre de jours nécessaires pour faire passer tous les examens. On représente chaque examen par un sommet et deux sommets sont reliés par une arête si les examens ne peuvent avoir lieu le même jour. Une coloration minimale des sommets nous donne une solution.

## 11 Graphe planaire

Définition 1 - 31

Un graphe planaire est un graphe que l'on peut *dessiner* sur un plan sans que ses arêtes ne se croisent.

Dans un graphe plan, il y a des faces internes et une face externe.

Théorème 1 - 23

Formule d'Euler

Soit  $f$  le nombre de faces d'un graphe planaire :

$$n - m + f = 2$$

## 12 Arbres de recouvrement de poids extremum

### 12 1 Généralités

On rappelle qu'un arbre est un graphe connexe sans cycle.

Soit  $G = (X, A)$  un graphe avec  $|X| = n$  et  $|A| = m$ .

On a montré en exercice (cf exercice [Exercice 1 - 53 page 60](#)) qu'un arbre de  $n$  sommets a  $n - 1$  arêtes.

Recherche

Montrer que chaque arête d'un arbre est un isthme.

Définition 1 - 32

On appelle **arbre recouvrant** (ou couvrant ou de recouvrement) d'un graphe  $G$ , un graphe partiel qui est un arbre.

Expliquez l'origine du choix du mot « recouvrant ».

Théorème 1 - 24

Un graphe  $G$  admet un arbre de recouvrement si, et seulement si  $G$  est connexe.

Pour la condition suffisante, on peut montrer par l'absurde que le graphe partiel sans cycle contenant un nombre maximal d'arêtes de  $G$  est connexe.

**12 2 Algorithme de Kruskal**

Joseph KRUSKAL, mathématicien et informaticien américain décédé en 2010, a proposé un algorithme permettant de déterminer un arbre de recouvrement de poids extrémal en 1956.

En voici une approche simplifiée. On considère un graphe pondéré  $G = (X, A, P)$ .

- On trie les arêtes par ordre de poids (dé)croissant ;
- tant qu'on n'a pas obtenu  $n - 1$  arêtes (pourquoi ?) on considère la première arête non examinée. Si elle forme un cycle avec les arêtes choisies précédemment, on la rejette, sinon on la garde.

On peut remplacer le test sur le choix de l'arête par « est-ce que l'arête nouvelle formerait un isthme pour le nouveau graphe en formation ? » ou bien « est-ce que les deux sommets de la nouvelle arête sont dans la même composante connexe du nouveau graphe ? »

Une variante consiste à enlever du graphe d'origine les arcs les plus légers (ou les plus lourds) dans la mesure où la connexité est conservée.

**13 Flots et réseaux de transport**

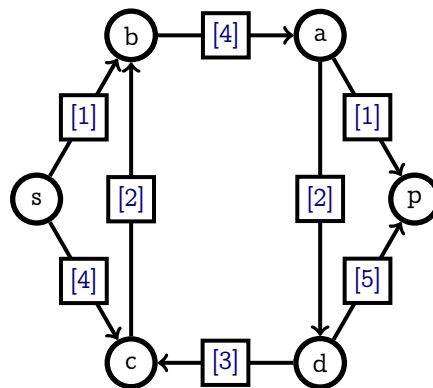
**13 1 Réseau de transport**

Un réseau est défini par :

- un graphe orienté  $G = (X, A)$  ;
- une source  $s \in E$  et un puits  $p \in E - \{s\}$  tels qu'il existe au moins un chemin de  $s$  vers  $p$  ;
- une application  $c$  de  $A$  dans  $\mathbb{R}_+ \cup \{+\infty\}$ . On appelle  $c(u)$  la capacité de l'arc  $u$ .

En fait,  $c(u)$  représente le débit maximal (de voitures, d'électricité, d'informations, d'étudiants...) pouvant transiter par le réseau.

Par exemple :



**13 2 Flot dans un réseau de transport**

Soit  $R = (X, A, s, p, c)$  un réseau de transport. Soit  $\varphi$  une application  $\varphi : A \rightarrow \mathbb{R}^+$ . Si  $u = (x, y)$ , on écrira aussi  $\varphi(u) = \varphi(x, y)$ .

On définit, sur  $X$  cette fois, les deux applications :

$$\varphi^+(x) = \sum_{y \in \Gamma(x)} f(x, y)$$

$$\varphi^-(x) = \sum_{y \in \Gamma^{-1}(x)} f(y, x)$$

On appellera  $\varphi^+(x)$  le flot sortant en  $x$  et  $\varphi^-(x)$  le flot entrant en  $x$ .

On a pour tout graphe et toute application  $\sum_{x \in X} \varphi^+(x) = \sum_{x \in X} \varphi^-(x)$ .

Définition 1 - 33

La fonction  $\varphi$  est un **flot** sur  $G = (X, A)$  si elle vérifie la propriété d'équilibre local :

$$\forall x \in X, \quad \varphi^+(x) = \varphi^-(x)$$

Définition 1 - 34

Le flot  $\varphi$  est compatible avec le réseau  $R = (X, A, s, p, c)$  si :

$$\forall u \in A, \quad \varphi(u) \leq c(u)$$

Définition 1 - 35

Un arc  $u$  est dit saturé si  $\varphi(u) = c(u)$ .

Définition 1 - 36

Un flot est dit complet si tout chemin de  $s$  à  $p$  passe par au moins un arc saturé.

Définition 1 - 37

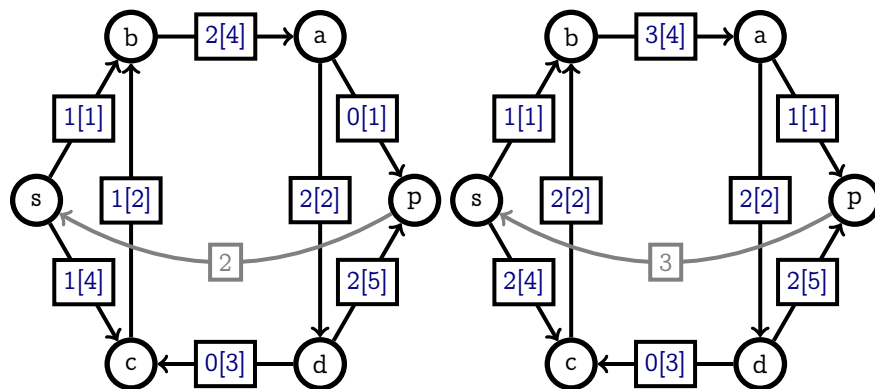
Soit  $R = (X, A, s, p, c)$  un réseau de transport et  $\varphi$  un flot sur  $(X, A \cup \{(p, s)\})$ . L'arc  $(p, s)$  est appelé arc de retour et  $\varphi(p, s)$  est appelée valeur du flot.

Compte tenu de la propriété d'équilibre, la valeur du flot est en fait égale au flux qui transite de  $s$  vers  $p$ .

Définition 1 - 38

On appelle valeur résiduelle d'un chemin  $\gamma$  de  $s$  à  $p$  le nombre :

$$r(\gamma) = \min_{u \in \gamma} \{c(u) - \varphi(u)\}$$



**13 3 Algorithme de Ford et Fulkerson**

Soit  $R = (X, A, s, p, c)$  un réseau de transport et  $\varphi$  un flot sur  $R$ .

Un chemin qui relie la source au puits par des arcs tous non saturés est appelé un chemin augmentant.

Définition 1 - 39

Une chaîne qui relie la source au puits par des arêtes  $u$  vérifiant :

- $f(u) < c(u)$  si l'arête est orientée dans le sens  $s \rightarrow p$ ,
- $f(u) > 0$  si l'arête est orientée dans le sens  $p \rightarrow s$ ,

est appelée chaîne augmentante.

On peut alors augmenter le flot de la valeur résiduelle sur un chemin augmentant et de cette même valeur sur les arcs « positifs » d'une chaîne augmentante et diminuer le flot de la valeur résiduelle sur les arcs « négatifs ».

Théorème 1 - 25

Un flot est alors :

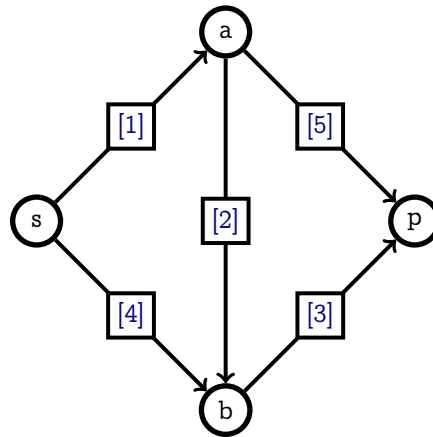
- complet s'il n'admet pas de chemin augmentant ;
- maximal s'il n'admet pas de chaîne augmentante.

L'algorithme des amis FORD et FULKERSON (1956) se résume donc à :

- partir d'un flot quelconque ;
- augmenter le flot tant qu'il existe un chemin augmentant ;

– augmenter le flot tant qu'il existe une chaîne augmentante.

Essayer sur le réseau suivant :



## 14 Quelques autres applications

### 14 1 1 Ordonnancement

La réalisation d'une recette de cuisine, la construction d'un pont ou d'une maison, l'élaboration d'un logiciel, le fonctionnement d'une chaîne de fabrication , etc., demandent une coordination d'un ensemble complexe « d'unités de travail ». Les problèmes d'ordonnancement sont de ce type, ils se présentent sous la forme d'un objectif que l'on souhaite atteindre et dont la réalisation demande l'exécution d'un certains nombre de tâches soumises à de nombreuses contraintes qui peuvent être des contraintes de durée, de matériels ou moyens humains disponibles, d'une météo favorable, etc..

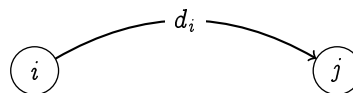
Nous n'allons étudier que le cas particulier de base : les seules contraintes sont des contraintes de succession dans le temps en ce sens que les seules contraintes que nous retenons sont du type :

la tâche  $j$  ne peut commencer que si la tâche  $i$  est terminée ou achevée  
et la durée de chaque tâche est une durée certaine

### 14 1 1 Représentation MPM

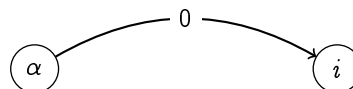
MPM désigne « Méthode des Potentiels Metra<sup>c</sup> », on représente le problème à l'aide d'un graphe :

1. On associe à chaque tâche un sommet du graphe.
2. On définit l'arc  $(i, j)$  de valuation ou de longueur  $d_i$  si la tâche  $i$  a une durée  $d_i$  et si la tâche  $i$  doit être terminée avant que la tâche  $j$  ne puisse commencer.



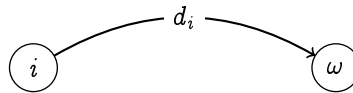
Ce graphe se lisant : la tâche  $i$  a une durée  $d_i$  et la tâche  $j$  ne peut démarrer qu'après la réalisation de la tâche  $i$ .

3. On introduit deux tâches fictives, la tâche début des travaux représentée par le sommet  $\alpha$  et la tâche fin des travaux représentée par le sommet  $\omega$ , ces deux tâches ayant chacune une durée nulle.
4. On relie le sommet  $\alpha$  par un arc de valuation nulle à tout sommet  $i$  représentant une tâche ne possédant pas de contrainte d'antériorité.



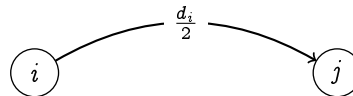
<sup>c</sup>. Metra était, dans les années 1970, une société franco-britannique, associé à la SIA (Société d'Informatique Appliquée) et à la SEMA (Société de Mathématiques Appliquées) où beaucoup de chercheurs français de très haut niveau ont travaillé.

- 5. On relie le sommet  $i$  au sommet  $\omega$  par un arc de valuation égale à la durée de la tâche  $i$  si la tâche  $i$  ne possède pas de contrainte de postériorité.

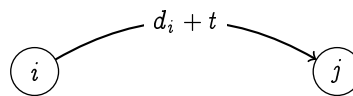


Indiquons tout de suite le traitement de quelques cas particuliers :

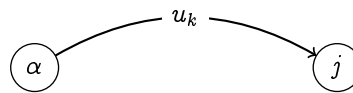
- La contrainte : la tâche  $j$  ne peut commencer avant la moitié (par exemple) de la réalisation de la tâche  $i$  se représente par



- La contrainte : la tâche  $j$  ne peut commencer qu'un temps  $t$  après la fin de la tâche  $i$  se représente par



- La contrainte : la tâche  $j$  ne peut commencer qu'après la date  $u_k$  se représente par



**Remarque** Le graphe obtenu doit être sans circuit, sinon cela impliquerait qu'une tâche devrait de succéder à elle même.

Étudions l'exemple suivant :

La réalisation d'un projet nécessite un certain nombre de tâches dont les durées et les contraintes d'antériorité (ou de postériorité) sont les suivantes :

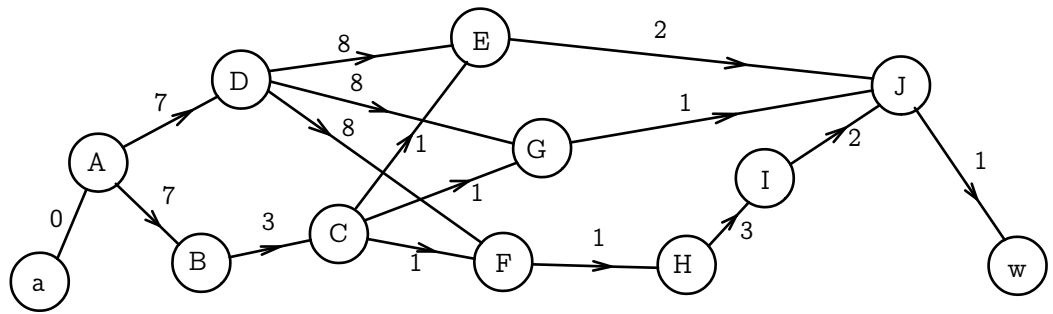
Tâches	Durées	Tâches antérieures
A	7	-
B	3	A
C	1	B
D	8	A
E	2	D, C
F	1	D, C
G	1	D, C
H	3	F
I	2	H
J	1	E, G, I

Pour la représentation du graphe MPM associé à la réalisation de ce projet, il est fortement conseillé de tester que le graphe est sans circuit ; le niveau affecté à chaque sommet va nous permettre d'obtenir une représentation satisfaisante rapidement. Allons-y : on obtient

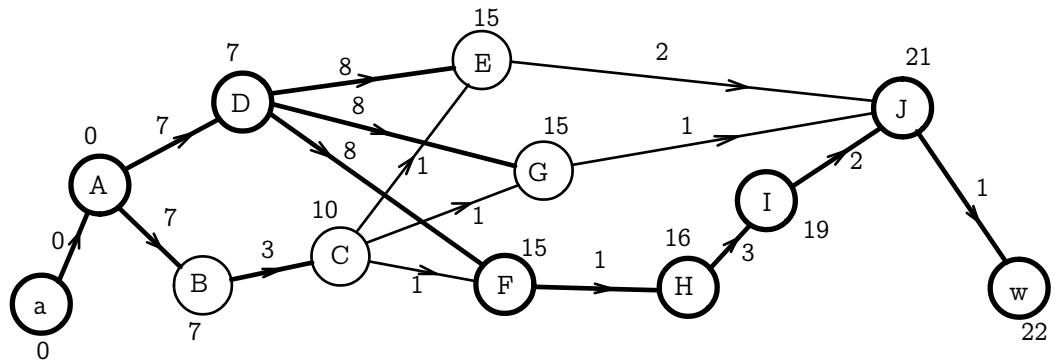
Niveau	0	1	2	1	3	3	3	4	5	6
Sommets	A	B	C	D	E	F	G	H	I	J
Précédents		A	B	A	D, C	D, C	D, C	F	H	E, G, I

Une représentation graphique MPM est alors





Le projet commençant à la date  $t_\alpha = 0$ , la durée minimale de réalisation du projet est donc égale à la longueur des plus longs chemins de  $\alpha$  à  $\omega$ . Appliquons un algorithme de recherche des plus longs chemins, on obtient

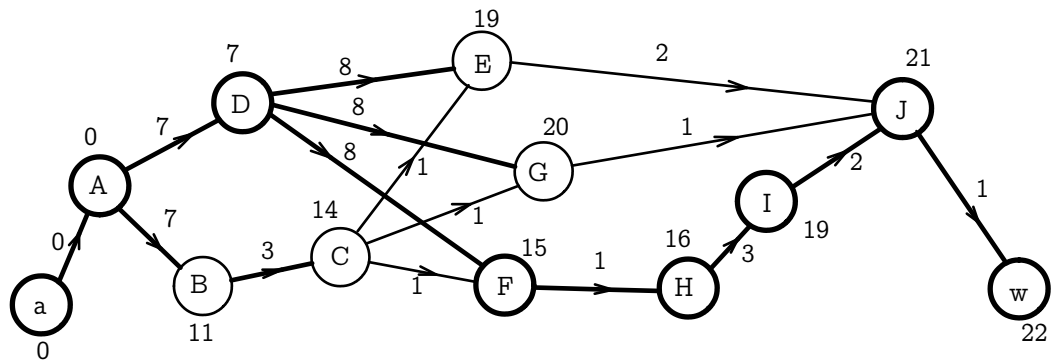


La durée minimale de réalisation du projet est de 22 unités de temps. La marque  $t_i$  de chaque tâche  $i$  nous donne sa date de démarrage au plus tôt en considérant, bien entendu que la tâche  $\alpha$ , début des travaux, débute à la date 0. Par exemple, la date de démarrage au plus tôt de la tâche C est 10, cela signifie que la tâche C ne peut démarrer qu'au minimum 10 unités de temps après la date de démarrage des travaux.

Maintenant se pose la question « quelle est la date de démarrage au plus tard de chaque tâche ? », c'est à dire, « à quelle date maximale peut-on faire démarrer une tâche sans augmenter la durée minimale des travaux », à savoir, ici, 22 unités de temps. Notons  $T_i$  la date de démarrage au plus tard de la tâche  $i$ , en imposant  $t_\omega = 22$ , nous avons de façon évidente

$$T_i = \text{Min} \{T_j - d_i \mid j \in \mathcal{G}(\{i\})\}$$

La marge de la tâche  $i$  étant  $m_i = T_i - t_i$ . Les tâches ayant une marge nulle (celles dont la date de démarrage au plus tôt est égale à la date de démarrage au plus tard) sont appelées les tâches critiques, ces tâches ne souffrent aucun retard. Les tâches critiques sont les tâches correspondant aux sommets des plus longs chemins de  $\alpha$  à  $\omega$ . Dans notre exemple les tâches critiques sont les tâches A, D, F, H, I et J. Les dates de démarrage au plus tard des tâches sont données par

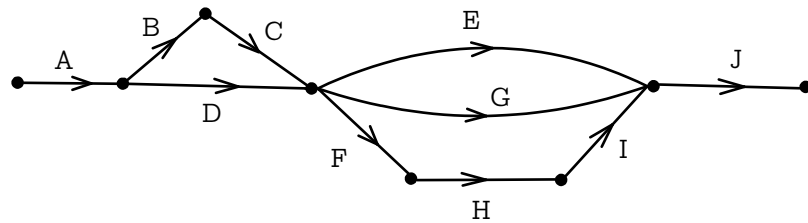


**14 1 2 Représentation PERT**

PERT = Preview Evaluation Research Task, c'est la représentation « américaine ». Ici une tâche n'est pas représentée par un sommet mais par un arc, on donne aux sommets des noms arbitraires, les sommets sont appelés des étapes. L'arc  $U_i$ , qui représente la tâche  $i$ , précède l'arc  $U_j$  si la

tâche  $i$  doit être terminée avant que la tâche  $j$  ne puisse commencer ; chaque arc est valué par la durée de la tâche à laquelle il correspond. Chaque étape (chaque sommet) définit un ensemble de tâches déjà effectuées, on introduit une étape début et une étape fin. On prend comme définition d'une étape les  $\mathcal{G}^{-1}(\{i\})$ .

Une représentation PERT de l'exemple précédent donne



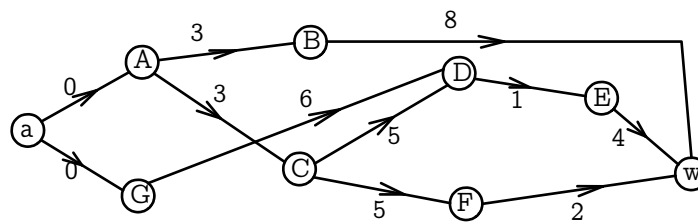
La construction d'un tel graphe est un peu compliquée et ne se prête pas bien aux cas particuliers, néanmoins les algorithmes sont les mêmes.

Étudions l'exemple suivant.

Un contrat vient d'être signé pour un projet informatique. Ce projet a été décomposé en tâches confiées à des ingénieurs et les contraintes de précedence ainsi que les durées des tâches sont présentées dans le tableau ci-dessous :

Rang	Libellé de la tâche	durée en mois	Tâches préalables
0	A	3	
1	B	8	A
1	C	5	A
2	D	1	C, G
3	E	4	D
2	F	2	C
0	G	6	

On décide de représenter ce problème d'ordonnancement à l'aide d'un graphe par la méthode MPM. En faisant abstraction des deux tâches fictives « début du projet » et « fin du projet », nous affectons un rang à chaque sommet dans le tableau précédent. Une représentation MPM de ce projet est :



La durée minimale de réalisation du projet est de 13 mois et, en s'imposant cette durée minimale de réalisation et en choisissant la date 0 comme date de démarrage, la date de démarrage au plus tôt et au plus tard de chaque tâche est donnée par

Tâche	date au plus tôt	date au plus tard
A	0	0
B	3	5
C	3	3
D	8	8
E	9	9
F	8	11
G	0	2

Les tâches critiques sont  $A, C, D, E$ ; tout retard d'une de ces tâches augmentera la durée minimale des travaux.

**14 2 Codage de Prüfer**

Le codage de Prüfer permet de décrire un arbre (graphe non orienté connexe sans cycle simple ou acyclique) de façon condensée. Soit donc  $\mathcal{G} = (X, A)$  un arbre,  $X$  est l'ensemble des sommets et  $A$  est l'ensemble des arêtes. Pour appliquer le codage, il nous faut ordonner, de façon quelconque, les sommets et pour cette raison on s'impose souvent de noter les  $n$  sommets par  $1, 2, 3, \dots, n$ , mais on peut tout aussi bien les numéroter de  $1$  à  $n$ . Bien comprendre que cet ordre n'intervient que dans le codage, il n'est en rien associé à l'organisation des sommets avec les arêtes et on ne crée aucune hiérarchie entre les sommets.

Nous appelons feuille d'un arbre un sommet dont le degré est 1 ou, c'est pareil, un sommet qui n'a qu'un seul voisin ou qui n'a qu'un seul sommet adjacent.

On rappelle qu'un graphe partiel d'un graphe  $\mathcal{G}$  est le graphe obtenu en supprimant des arêtes mais en aucun cas des sommets et qu'un sous graphe de  $\mathcal{G}$  est un graphe obtenu en supprimant des sommets et uniquement les arêtes de  $\mathcal{G}$  utilisant ces sommets, si  $Y$  est une partie de  $X$ , le sous graphe de  $\mathcal{G}$  ayant pour ensemble de sommets  $Y$  est le sous graphe engendré par  $Y$ .

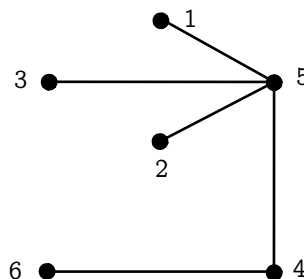
Le codage de Prüfer d'un arbre de  $n$  sommets numérotés de  $1$  à  $n$  est une suite de  $n - 2$  entiers de l'ensemble  $\{1, 2, \dots, n\}$ , suite construite par l'algorithme suivant :

```

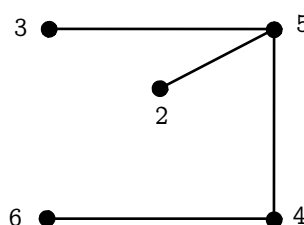
Variable
|
| T : arbre courant
| S : ensemble
| C : suite
|
Début
|
| T ← (X, A)
| S ← ∅
| C ← liste vide
|
| TantQue T contient plus de 2 sommets Faire
|   | identifier la feuille f de T ayant le plus petit numéro
|   | S ← S ∪ {j}
|   | ajouter à C le seul sommet k adjacent à j dans T
|   | T est maintenant le sous-graphe engendré par X - S
|
| FinTantQue
Fin
    
```

Par exemple :

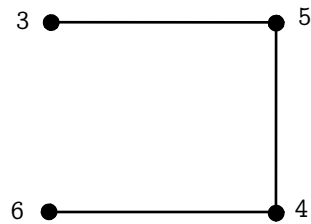
On considère l'arbre  $\mathcal{G} = (X, A)$  donné par



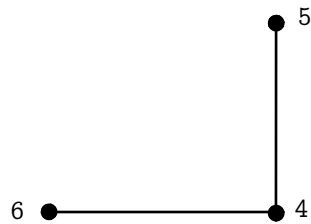
la première itération nous donne  $C = (5)$  et  $T$  donné par



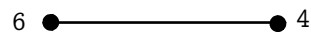
la deuxième itération donne  $C = (5, 5)$  et  $T$  donné par



la troisième itération donne  $C = (5, 5, 5)$  et  $T$  donné par



la troisième itération donne  $C = (5, 5, 5, 4)$  et  $T$  donné par



l'algorithme s'arrête puisque l'arbre courant  $T$  ne comporte que deux sommets. L'arbre  $\mathcal{G} = (X, A)$  est codé en  $(5, 5, 5, 4)$ .

Tout ceci n'a d'intérêt que si « on peut revenir en arrière ». Voici l'algorithme qui le permet à partir de la suite  $C$  et on sait qu'on a affaire à un arbre de  $n$  sommets numérotés de 1 à  $n$ .

<p><b>Variable</b>  <math>I</math> : ensemble</p> <p><b>Début</b>  <math>I \leftarrow \{1, 2, \dots, n\}</math>  <b>TantQue</b> il reste des éléments dans la suite <math>C</math> (le codage) <b>Faire</b>            déterminer le plus petit <math>i</math> de <math>I</math> n'apparaissant pas dans <math>C</math>            relier par une arête <math>i</math> avec le sommet <math>j</math> en tête de <math>C</math>            enlever <math>i</math> de <math>I</math> et <math>j</math> de <math>C</math>  <b>FinTantQue</b>            Les deux éléments qui restent dans <math>I</math> nous donnent la dernière arête</p> <p><b>Fin</b></p>
---

Appliquons au codage  $C = (5, 5, 5, 4)$  codage d'un arbre de 6 sommets notés (ou numérotés) de 1 à 6.

Le plus petit numéro n'apparaissant pas dans la suite est 1, on crée donc l'arête  $[1, 5]$ ,  $I$  devient  $I = \{2, 3, 4, 5, 6\}$  et  $C$  devient  $C = (5, 5, 4)$ . Le plus petit numéro n'apparaissant pas dans  $C$  est 2, on crée l'arête  $[5, 2]$ ,  $I$  devient  $I = \{3, 4, 5, 6\}$  et  $C$  devient  $C = (5, 4)$ . Le plus petit numéro de  $I$  n'apparaissant pas dans la suite est 3, on crée l'arête  $[5, 3]$ .  $I$  devient  $I = \{4, 5, 6\}$  et  $C$  devient  $C = (4)$ . Le plus petit numéro de  $I$  n'apparaissant pas dans la suite  $C$  est 5, on crée l'arête  $[4, 5]$ ,  $I$  devient  $I = \{4, 6\}$  et  $C$  devient  $C = ()$ . On crée en dernier l'arête  $[4, 6]$ .

## 15 Les graphes avec Python

La plupart des résultats présentés dans cette section sont issus d'un merveilleux ouvrage qui paraîtra en janvier aux éditions Dunod <sup>d</sup>.


### 15 1 Présentation

Python est disponible sur toute distribution GNU-Linux. C'est un langage de très haut niveau dont la syntaxe encourage à écrire du code clair et de qualité. Dans le domaine de la gestion de la mémoire, nombres de détails de bas niveau de langages comme le C disparaissent. De plus l'apprentissage de Python est facilité par l'existence d'une interface interactive. Ceci dit, son intérêt ne se réduit pas à l'apprentissage de la programmation ou de l'algorithmique; en témoigne sa popularité croissante. Il a été choisi par des acteurs majeurs : Google, YouTube, la NASA, etc. Il favorise la programmation impérative structurée et la programmation orientée objet; dans une moindre mesure, il permet de programmer dans un style fonctionnel. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions. C'est un langage multi-plateforme, polyvalent (jusque dans les domaines comme le web, les graphiques, le réseau), open source et gratuit.

### 15 1 1 fonctions

Il est tout à fait possible de définir une fonction (au sens du langage Python) qui ressemble à une fonction mathématique. La syntaxe est alors la suivante :

```
>>> def f(x):
...     return x**2
...
>>> f(2)
4
```

Tout d'abord, la déclaration d'une nouvelle fonction commence par le mot-clé **def**. Ensuite, toujours sur la même ligne, vient le nom de la fonction (ici **f**) suivi du paramètre formel de la fonction, placé entre parenthèses (le paramètre formel **x** de la fonction Python correspond ici à la variable de la fonction mathématique), le tout ponctué de deux points. Une fois la première ligne saisie, on appuie sur la touche  et on constate que les chevrons de l'invite de commande ont été remplacés par des points; cela signifie que l'interpréteur attend obligatoirement une suite à la ligne précédente avant de lire ce qui sera un bloc d'instruction. Ensuite, il faut saisir quatre espaces pour indenter la suite des instructions qui définira la fonction. Enfin, une ligne vide signifiera à l'interpréteur que l'on a terminé de définir notre fonction, et qu'il peut désormais lire la totalité du bloc d'instructions qui définit notre fonction.

### 15 1 2 La structure conditionnelle

Sans surprise mais on n'oublie pas l'indentation. On a droit également au classique **elif**.

```
def max2(a, b):
    if a <= b:
        return b
    else:
        return a
```

### 15 1 3 Les boucles « tant que »

Par exemple :

```
def pgcd(a, b):
    while b > 0:
        a, b = b, a % b
    return a
```

On remarque au passage que le calcul modulaire s'effectue à l'aide de l'opérateur binaire **%**.

<sup>d</sup>. « Mathématiques et programmation en Python- du lycée à la licence en passant par la prépa » par Alexandre CASAMAYOU, Pascal CHAUVIN, Guillaume CONNAN...

**15 1 4 Les listes**

C'est la structure ordonnée de base :

```
>>> liste = [12,11,18,7,15,3]
>>> liste[0]
12
>>> liste[:2]
[12, 11]
>>> liste[2:]
[18, 7, 15, 3]
>>> liste[2:4]
[18, 7]
>>> liste[-1]
3
>>> tete=liste.pop(0)
>>> tete
12
>>> liste
[11, 18, 7, 15, 3]
>>> len(liste)
5
```

Quelques méthodes :

la méthode	son effet
<code>list.append(x)</code>	ajoute l'élément <b>x</b> en fin de liste
<code>list.extend(L)</code>	ajoute en fin de liste les éléments de <b>L</b>
<code>list.insert(i, x)</code>	insère un élément <b>x</b> en position <b>i</b>
<code>list.remove(x)</code>	supprime la première occurrence de <b>x</b>
<code>list.pop(i)</code>	supprime l'élément d'indice <b>i</b> et le renvoie
<code>list.index(x)</code>	renvoie l'indice de la première occurrence de <b>x</b>
<code>list.count(x)</code>	renvoie le nombre d'occurrences de <b>x</b>
<code>list.sort()</code>	modifie la liste en la triant
<code>list.reverse()</code>	modifie la liste en inversant l'ordre des éléments

**15 1 5 Les dictionnaires**

Nous utiliserons des dictionnaires pour définir nos graphes. Il s'agit de listes indexées par des objets de n'importe quel type.

Par exemple :

```
>>> tel={'Roger' : '02 40 00 00 00', 'Marcelle' : '02 40 00 00 01'}
>>> tel['Marcelle']
'02 40 00 00 01'
```

Une méthode importante est de pouvoir vérifier si le dictionnaire contient une entrée donnée grâce à `has_key()` et de créer la liste des entrées avec `keys()` :

```
>>> tel.has_key('Roger')
True
>>> tel.keys()
['Marcelle', 'Roger']
```

**15 1 6 Les boucles « pour »**

D'abord, un mot sur `range()` :

```
>>> r=range(5)
>>> r
[0, 1, 2, 3, 4]
>>> s=range(3,9)
>>> s
```

```
[3, 4, 5, 6, 7, 8]
>>> t=range(3,9,2)
>>> t
[3, 5, 7]
```

Puis la boucle elle-même :

```
>>> def f(n):
    for i in range(n):
        print('Je me répète '+str(n)+' fois')
    ... ..
>>> f(3)
Je me répète 3 fois
Je me répète 3 fois
Je me répète 3 fois
```

## 15 2 La classe « graphe »

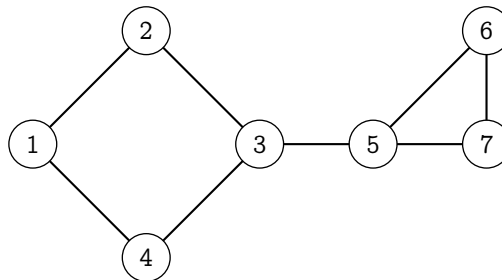
Avec Python, on va définir une classe d'objets qu'on appellera **graphe**. Le constructeur de classe de Python s'appelle toujours `__init__` et prend au moins un argument **self**.

Nous choisissons de construire un graphe à partir d'un dictionnaire de d'adjacence de chaque sommet, les sommets seront entrés comme des chaînes de caractères (de type **string**).

```
class graphe(object):
    def __init__(self, liste_adj =None):
        self.__liste_adj = liste_adj

    def __str__(self):
        return str(self.__liste_adj)
```

Par exemple, pour entrer le graphe suivant :



on tapera :

```
>>> g=graphe( { '1':['2','4'],
                '2':['1','3'],
                '3':['2','4','5'],
                '4':['1','3'],
                '5':['3','6','7'],
                '6':['5','7'],
                '7':['5','6']
              } )
```

Voici maintenant les outils de base :

```
## méthode pour avoir les adjacents d'un sommet
def adjacents(self, a):
    if self.__liste_adj is not None:
        if self.__liste_adj.has_key(a):
            return self.__liste_adj[a]
    return []

## teste si a et b sont adjacents
def est_adjacent(self, a, b):
    return (b in self.adjacents(a))
```

```

## liste les adjacents de a
def liste_adjacence(self):
    return self.__liste_adj

## donne la liste des sommets
def sommets(self):
    if self.__liste_adj is None:
        return []
    else:
        t = self.__liste_adj.keys()
        t.sort()
        return t

## donne le nombre de sommets d'un graphe
def nombre_sommets(self):
    return len(self.sommets())

```

### 15 3 Un peu de travail...

1. Définissez une fonction `degre_sommet(self, a)` qui renvoie le degré d'un sommet `a` du graphe. Par exemple, avec le graphe précédent :

```

>>> g.degre_sommet('5')
3

```

2. Définissez une fonction `nombre_aretes(self)` qui calcule le nombre d'arêtes du graphe.
3. On suppose à présent qu'on dispose d'une méthode donnant la composante connexe d'un sommet.  
Écrivez une méthode donnant la liste de toutes les composantes connexes d'un graphe et une autre testant si un graphe est connexe.  
Écrivez également une méthode testant si deux sommets d'un graphe sont connectés.  
Écrivez enfin une méthode testant si une arête est un isthme.
4. On suppose qu'on dispose d'une méthode listant toutes les chaînes reliant deux sommets donnés. Écrivez une méthode qui recherche tous les cycles passant par un sommet donné puis une autre qui teste si un graphe est un arbre.
5. On aura besoin de savoir si un sommet est en queue des sous-listes d'une liste. Définissez une fonction `arret(liste,s)` qui effectue ce test.

On devra par exemple obtenir :

```

>>> ll = [[1,2,3],[4,5,3]]
>>> arret(ll,3)
True
>>> LL = [[1,2,3],[4,5,5]]
>>> arret(LL,3)
False

```

6. Voici une méthode très importante : que fait-elle ?...

```

def methode_mystérieuse_1(self, s):
    p = []
    if s in self.sommets():
        q = []
        q.append(s)
        visite = []
        visite.append(s)
        while len(q) > 0:
            x = q.pop(0)
            p.append(x)
            for t in self.adjacents(x):
                if t not in visite:
                    q.append(t)
                    visite.append(t)
        p.sort()
    return p

```



Transformez-vous en serpent et appliquez cette fonction au graphe page 47

7. Voici une nouvelle méthode tout aussi mystérieuse : que fait-elle ?...

```
def methode_mystérieuse_2(self, a, b):
    ll = []
    ll.append([a])
    while not arret(ll, b):
        q = []
        for p in ll:
            u = p[-1]
            if u == b:
                q.append(p)
            else:
                s = self.adjacents(u)
                for t in s:
                    if not (t in p):
                        v = [i for i in p]
                        v.append(t)
                        q.append(v)
        ll = q
    return ll
```

Appliquez cette méthode toujours au graphe page 47 pour les sommets 1 et 7 (i.e. g.méthode\_mystérieuse\_2(1, 7)).

8. À l'aide de tous les outils précédents, écrivez une méthode qui teste l'existence d'une chaîne eulérienne dans un graphe puis une autre qui teste si un graphe est eulérien.
9. Analyser la méthode suivante :

```
def graphe_reduit(self, a, b):
    h = dict()
    for s in self.sommets():
        l = []
        for t in self.adjacents(s):
            if (s != a or t != b) and (s != b or t != a):
                l.append(t)
        if len(l) != 0:
            h[s] = l
    return graphe(h)
```

10. Analyser la méthode suivante :

```
def fleury(self):
    g = self
    fin = (g.nombre_arettes() == 1)
    if fin:
        return g.sommets()
    p = []
    choix = []
    for s in g.sommets():
        if g.degre_sommet(s) % 2 == 1:
            choix.append(s)
    if len(choix) == 0:
        choix = g.sommets()
    a = choix[random.randrange(0, len(choix))]
    while not fin:
        u = g.adjacents(a)
        v = []
        for s in u:
            if not g.est_un_pont(a, s):
                v.append(s)
        b = v[random.randrange(0, len(v))]
        p.append(a)
        g = g.graphe_reduit(a, b)
        a = b
        fin = (g.nombre_arettes() == 1)
    p.append(a)
```

```
p.append(g.adjacents(a)[0])  
return p
```

On a utilisé `random.randrange(0,n)` qui renvoie un entier aléatoirement choisi entre 0 et  $n - 1$ .

Appliquer cette méthode au graphe symétrique sous-jacent du graphe de DE BRUIJN dessiné page 29 en supprimant une des deux arêtes entre 2 et 5.

# EXERCICES

## Les bases

### Exercice 1 - 1

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

### Exercice 1 - 2

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

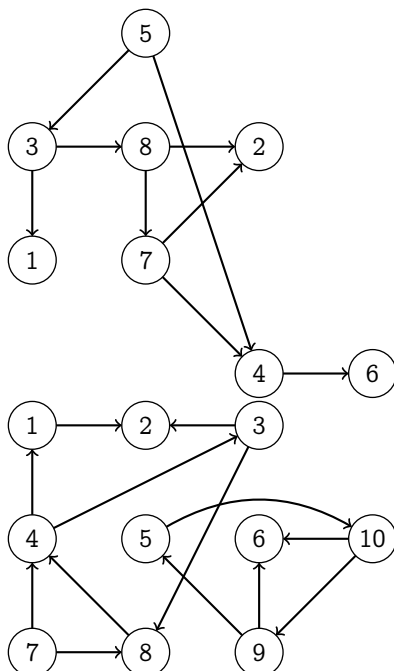
### Exercice 1 - 3

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 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 :

```

S ← ∅
on affecte le niveau ou le rang 0 aux sommets n'ayant
pas de précédent
Si cet ensemble de sommets est vide Alors
| Quitter { au moins un circuit }
FinSi
On met dans S les sommets de degré 0
i ← 1
TantQue S ≠ X Faire
| G_i est le sous graphe engendré par X - S, on
affecte le niveau i aux sommets de G_i n'ayant pas de
précédent
| Si cet ensemble de sommets est vide Alors
| | Quitter { au moins un circuit }
| FinSi
| On met dans S les sommets de degré i
| i ++
FinTantQue
    
```

Comparer-le au précédent. Utiliser enfin ce troisième algorithme appliqué aux matrices d'adjacences des graphes précédents : on supprime à chaque itération tous les sommets qui n'ont pas de précédent, c'est à dire tous ceux qui correspondent à une colonne ne comportant que des zéros.

### Exercice 1 - 4

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

### Exercice 1 - 5

- Combien de sommets a un graphe régulier de degré 4 contenant 10 arêtes ?
- Pour quelles valeurs de  $n$  les graphes suivant sont-ils réguliers ?
  - a.  $K_n$
  - b.  $C_n$
  - c.  $W_n$
  - d.  $Q_n$

### Exercice 1 - 6

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

### Exercice 1 - 7

Donner les matrices d'adjacence des graphes suivants :

- $K_4$
- $K_{1,4}$
- $K_{2,3}$
- $C_4$
- $W_4$
- $Q_3$

### Exercice 1 - 8

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

**Exercice 1 - 9**

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.

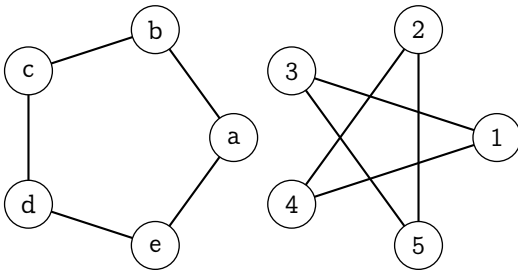
Déterminer un mini programme qui détermine le degré de chaque sommet et le nombre d'arêtes.

**Isomorphisme**

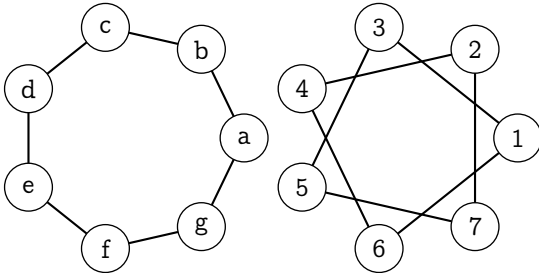
**Exercice 1 - 10**

Les graphes suivant sont-ils isomorphes ?

1. cinq sommets :



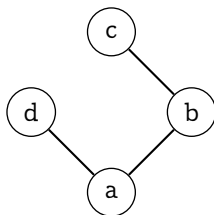
2. sept sommets :



**Exercice 1 - 11**

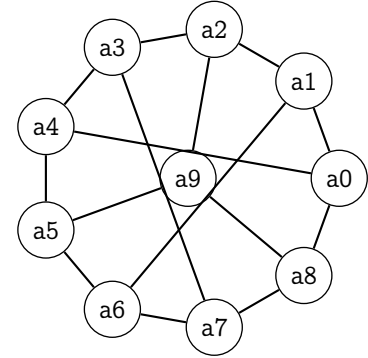
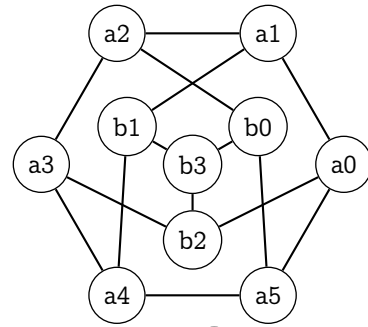
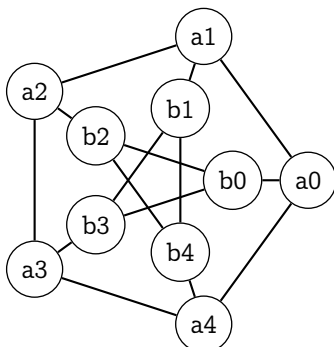
On dit qu'un graphe simple est **autocomplémentaire** s'il est isomorphe à son complémentaire.

Montrez que le graphe suivant est autocomplémentaire :



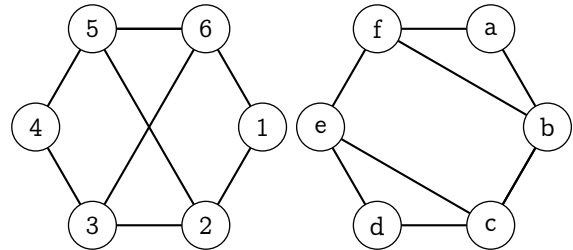
**Exercice 1 - 12**

Montrer que les trois graphes suivant sont isomorphes :



**Exercice 1 - 13**

En comparant certains cycles des graphes suivants, montrer qu'ils ne sont pas isomorphes :



**Connexité**

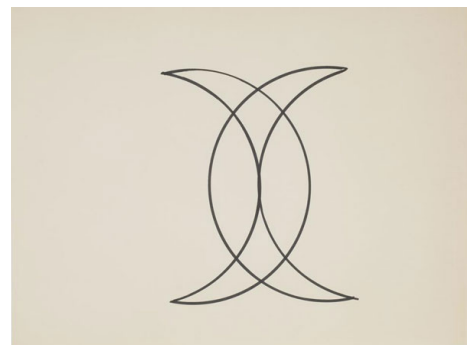
**Exercice 1 - 14**

Un graphe est connexe. Supposons qu'il contienne deux sous-graphes connexes : que peut-on dire au sujet de ces deux sous-graphes ?

**Euler et Hamilton**

**Exercice 1 - 15**

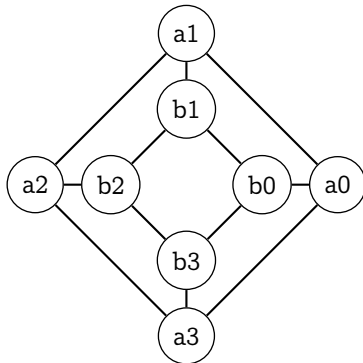
Le mathématicien français Édouard LUCAS en 1891 rapporta dans ses *récréations mathématiques* (disponibles à cette adresse : <http://gallica.bnf.fr/ark:/12148/bpt6k3943s/f61.table>) : « je me suis laissé dire que Mahomet dessinait d'un seul coup, avec la pointe de son cimeterre, sa signature formée de deux croissants opposés »



Est-ce possible ?

**Exercice 1 - 16**

Combien y a-t-il de cycles hamiltoniens dans le cube :



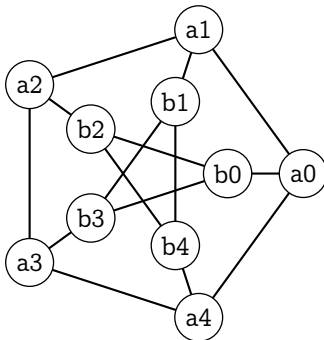
Imaginer un algorithme qui les compte. Combien y a-t-il de « formes » ?

**Exercice 1 - 17**

Les graphes complets  $K_n$  ont-ils des cycles hamiltoniens ? Combien ?

**Exercice 1 - 18 Graphe de Petersen**

Le voici :



Il s'agit de montrer que ce graphe n'est pas hamiltonien. Voici un début de raisonnement.

Supposons qu'il existe un cycle  $C$  hamiltonien.

Il faut absolument pouvoir relier le pentagone extérieur et l'étoile intérieure. On doit donc emprunter par exemple l'arête  $\{a_0, b_0\}$  (à une symétrie près, on ne perd pas de généralité).

Pour inclure le sommet  $a_0$  dans notre cycle, il faut donc choisir une des arêtes  $\{a_0, a_1\}$  ou  $\{a_0, a_4\}$ . Prenons  $\{a_0, a_1\}$  sans perdre de généralité par symétrie.

L'arête  $\{a_0, a_4\}$  n'est donc pas utilisée : pour inclure  $a_4$  dans le cycle, il faut donc utiliser les deux arêtes  $\{a_4, a_3\}$  et  $\{a_4, b_4\}$

Redessiner alors le graphe en effaçant les arêtes éliminées, en dessinant en trait plein les arêtes incluses et en pointillées celles qui sont encore candidates.

Terminer alors le raisonnement.

**Plus court chemin**

**Exercice 1 - 19**

On considère un graphe pondéré  $(X, A)$ , a un sommet de ce graphe et  $p(u, v)$  le poids de l'arête  $\{u, v\}$ .

Que fait cet algorithme ?

```

Début
  Pour tout s de X Faire
    d(s) ← p(a, s)
    chaîne ← a
  FinPour
  Marquer a
  TantQue ∃ des sommets non marqués Faire
    u ← sommet libre de distance à a mini
    Marquer u
    Pour v non marqué et {u, v} ∈ A Faire
      dd ← d(u)+p(u, v)
      Si dd < d(v) Alors
        d(v) ← dd
        chaîne(v) ← chaîne(u), v
      FinSi
    FinPour
  FinTantQue
Fin
    
```

**Exercice 1 - 20 Algorithme de Bellman-End-Ford**

Cette fois, il s'agit d'un algorithme dû à Richard BELLMAN, Samuel END et Lester FORD, mathématiciens américains, qui permet de trouver le plus court chaîne dans un arbre orienté pondéré connexe ayant des poids éventuellement négatifs.

En voici une présentation qui donne la longueur des plus courts chaînes du sommet 1 à tous les autres dans un graphe orienté  $\mathcal{G} = (X, A)$  pouvant avoir des valuations négatives ou qui détecte s'il existe un circuit de longueur négative. On note  $X = \{1, 2, \dots, i, \dots\}$  et  $\mathcal{G}_i^{-1}$  l'ensemble des précédents du sommet  $i$ .

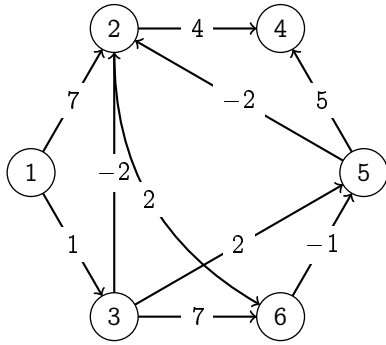
1. On pose  $\pi^0(1) = 0$ ,  $\pi^0(i) = +\infty$  pour les autres sommets et  $k = 1$ ;
2. à l'itération  $k$ , faire  $\pi^k(1) = 0$  et pour tous les autres sommets poser

$$\pi^k(i) = \min \left( \pi^{k-1}(i), \min_{j \in \mathcal{G}_i^{-1}} (\pi^{k-1}(j) + p_{j,i}) \right)$$

- a. si  $\pi^k(i) = \pi^{k-1}(i)$  pour tout  $i$  alors FIN ;
- b. si  $k \leq n - 1$  aller en 2 avec  $k \leftarrow k + 1$  ;
- c. si  $k = n$ , il existe un circuit de longueur négative.

On considère le graphe orienté pondéré  $\mathcal{G} = (X, A)$  défini

par la représentation graphique suivante :



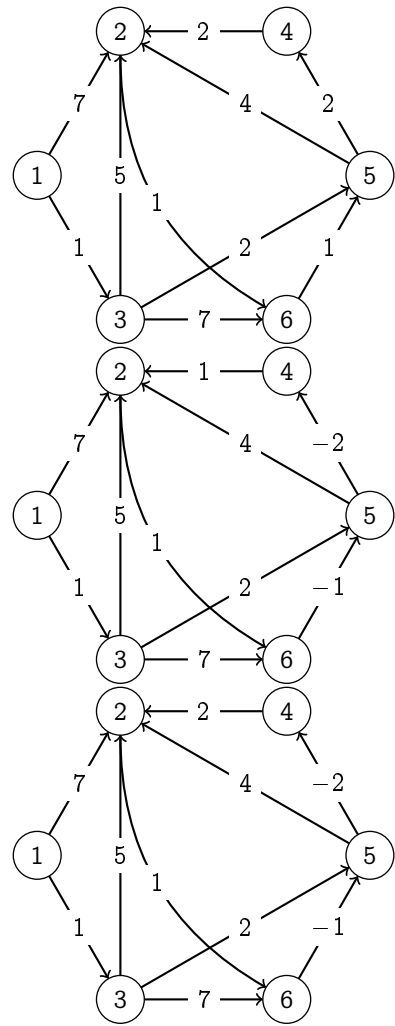
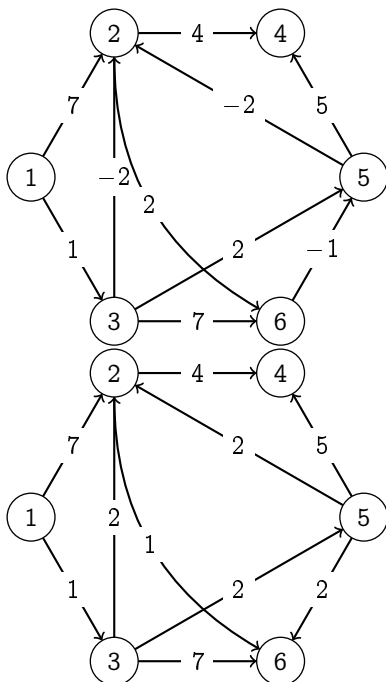
Rechercher les plus courts chaînes du sommet 1 aux autres sommets en utilisant **obligatoirement** l'algorithme de Bellman. Il vous est demandé de donner votre résultat en remplissant le tableau suivant :

$k$	$i$	1	2	3	4	5	6
0	$\pi^0(i)$	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	$\pi^1(i)$	0					
2	$\pi^2(i)$	0					
3	$\pi^3(i)$	0					
4	$\pi^4(i)$	0					
5	$\pi^5(i)$	0					
6	$\pi^6(i)$	0					

Donner votre conclusion.

**Exercice 1 - 21**

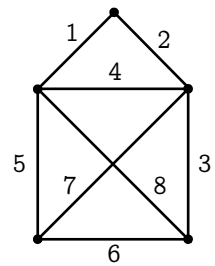
Déterminer les longueurs des plus courts chemins (s'ils existent) du sommet 1 aux autres sommets en utilisant un algorithme du cours (Bellman ou Dijkstra). Il vous est demandé de remplir des tableaux. Il est possible que le nombre de lignes du tableau soit plus grand que nécessaire.



**Coloration**

**Exercice 1 - 22**

1. Déterminer le graphe aux arêtes du graphe suivant (on a donné un nom à chaque arête) :



2. Quatre professeurs P1, P2, P3, P4 doivent donner des heures (1 heure par classe) de cours à 3 classes C1, C2, C3, les affectations sont données par le tableau :

	P1	P2	P3	P4
C1	oui		oui	oui
C2	oui	oui		oui
C3		oui		oui

On se pose la question : « Quel nombre minimum d'heures faut-il pour dispenser tous les cours ? ». Il est évident qu'un professeur ne peut pas faire cours à plus d'une classe dans un même créneau horaire

et qu'une classe ne peut pas avoir cours avec plus d'un prof au même moment. Il vous est demandé de transformer ce problème en un problème de coloration d'un graphe, vous indiquerez donc sur quel graphe le travail va se faire et vous représenterez ce graphe en expliquant clairement la notation que vous utiliserez pour nommer les sommets et le pourquoi des arêtes; vous préciserez aussi si c'est une recherche de coloration des sommets ou des arêtes (une coloration doit être ici un créneau horaire, par exemple rouge = 8h-9h, etc.)

**Exercices divers**

**Exercice 1 - 23 plus court chemin**

La distance  $d(x, y)$  est la longueur d'une plus courte chaîne entre les sommets  $x$  et  $y$  si une telle chaîne existe (sinon on la note  $\infty$ ).

Étant donné un sommet  $s$  d'un graphe, à l'aide d'un parcours en largeur, écrivez un algorithme qui donne  $d(s, x)$  pour tout sommet  $x$  du graphe.

Voici un début possible :

```

Pour tout  $x \in X$  Faire
  |  $d[x] \leftarrow \infty$ 
FinPour
 $d[s] \leftarrow 0$ 
...
    
```

**Exercice 1 - 24 fermeture transitive**

Montrer que la méthode vue en cours revient à utiliser l'algorithme suivant, si on note  $(m_{ij})$  la matrice booléenne d'incidence d'un graphe donné :

```

Pour k variantDe 1 Jusque n Faire
  Pour j variantDe 1 Jusque n Faire
    Pour i variantDe 1 Jusque n Faire
      |  $m_{ij} \leftarrow m_{ij} + m_{ik} \times m_{kj}$ 
    FinPour
  FinPour
FinPour
    
```

**Exercice 1 - 25 graphe biparti**

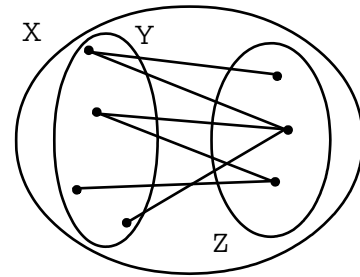
Une bipartition du graphe  $\mathcal{G} = (X, A)$  est une paire  $(Y, Z)$  où

$$\emptyset \neq Y \subseteq X, \emptyset \neq Z \subseteq X, Y \cap Z = \emptyset$$

$$\mathcal{P}_2(Y) \cap A = \emptyset, \mathcal{P}_2(Z) \cap A = \emptyset$$

Cela signifie que  $\{Y, Z\}$  est une partition de  $X$  et toute arête de  $A$  a une de ses extrémités dans  $Y$  et l'autre dans

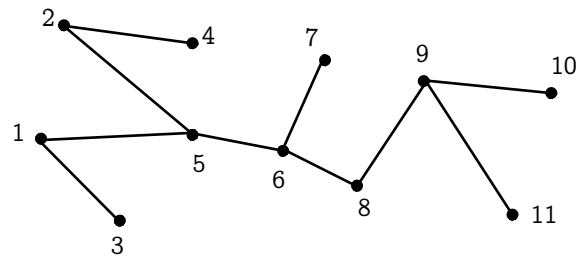
$Z$ . Un graphe est dit biparti s'il admet une bipartition. Le graphe suivant est biparti :



On le note  $\mathcal{G} = (Y, Z, A)$ .

Si  $Y$  a  $p$  éléments et  $Z$  a  $q$  éléments, le graphe biparti  $\mathcal{G} = (Y, Z, A)$  est souvent noté  $K_{pq} = K_{qp}$

1. Démontrer qu'un graphe biparti n'admet pas de cycle de longueur impaire.
2. Donner une bipartition du graphe suivant :



3. Donner un graphe biparti qui n'est pas un arbre.
4. En admettant qu'un arbre (non orienté, c'est sous entendu ici) est forcément un graphe biparti, donner un algorithme **CLAIR** qui donne une bipartition :

en entrée : l'arbre  $\mathcal{G} = (X, A)$

en sortie : une bipartition  $(Y, Z)$

**Exercice 1 - 26 séquence de De Bruijn**

Nicolaas DE BRUIJN (« de breuynne »), encore un hollandais, a réfléchi en 1946 au problème suivant même s'il avait déjà été résolu cinquante ans auparavant par Camille FLYE SAINTE-MARIE.

Le problème est simple : vous revenez d'un tonus et vous avez oublié le code de la porte de votre appartement que vous aviez pourtant vous-même mis au point à partir. Il s'agit en fait d'un code de quatre chiffres à choisir parmi 0, 1 et 2.

Combien y a-t-il de codes possibles ?

Vérifier qu'il faut a priori appuyer 324 fois sur les touches du digicode.

Notre ami hollandais a en fait montré qu'il suffisait d'appuyer sur les 84 touches suivantes :

00001002101112002201022110101212212012222000200  
1202221001102011220202121121021111000

Pourquoi à votre avis ?

Pour faire plus simple et plus habituel, considérons à partir de maintenant que nous travaillons sur des codes binaires de longueur  $n$ .

Combien y a-t-il de codes possibles ?

Explorez la situation pour  $n = 2$  en essayant de donner une séquence de 5 chiffres qui réponde au problème.

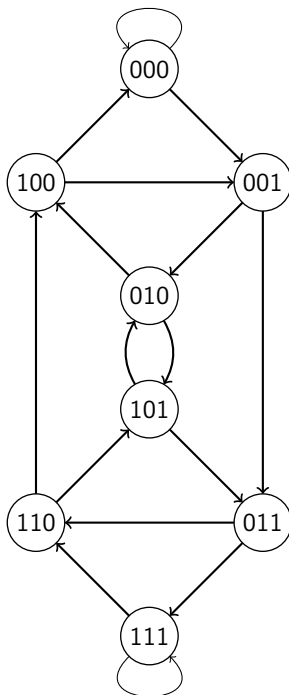
L'idée, qui vient de FLYE SAINTE-MARIE est de travailler dans  $\mathbb{Z}/2^n\mathbb{Z}$ .

On part de 0 et on construit récursivement un arbre en passant au choix, si la feuille actuelle vaut  $x$ , à  $2x$  ou à  $2x + 1$  modulo  $2^n$ . Il s'avère qu'on parcourt ainsi tout  $\mathbb{Z}/2^n\mathbb{Z}$ .

Essayez dans les cas simples  $n = 2$  puis  $n = 3$ . Vous construirez un arbre de racine 0, chaque nœud admettant deux successeurs. Si on retombe sur un nombre déjà présent sur la branche, on « coupe » cette branche.

Si nous considérons le graphe orienté dont les sommets sont numérotés de 0 à  $2^{n-1}$  et tel qu'à partir de chaque sommet  $x$  partent deux arêtes vers les sommets  $2x$  et  $2x + 1$  (modulo  $2^n$  !), il s'agit de savoir s'il existe un cycle hamiltonien.

Vérifier que l'arbre que vous avez tracé précédemment pour  $n = 3$  correspond au graphe suivant (que nous noterons  $G_3$ ) :



Il reste à trouver un chemin hamiltonien...

En fait, une propriété fort plaisante que nous admettrons nous donne une méthode fort pratique :

À chaque circuit eulérien de  $G_n$  correspond un cycle hamiltonien de  $G_{n+1}$

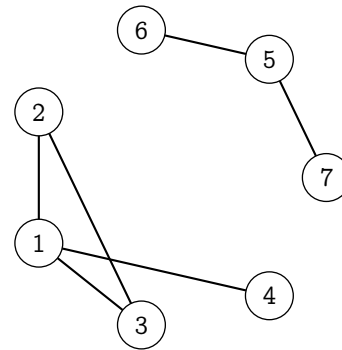
Comparez tout d'abord le nombre d'arêtes de  $G_n$  et le nombre de sommets de  $G_{n+1}$ .

Dessinez  $G_2$ . Numérotez l'arête  $(ab, bc)$  par  $abc$ . Remarques ? Conclusion ?

**Exercices supplémentaires sur les graphes**

**Exercice 1 - 27 Vocabulaire**

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



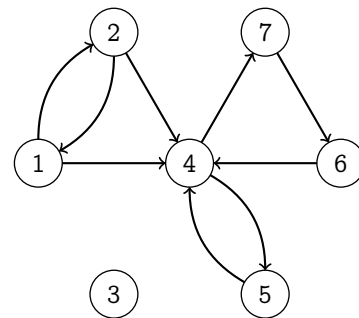
1. Déterminer  $X$  et  $A$ .
2. Déterminer le degré de chaque sommet.
3. Déterminer sa matrice d'adjacence (on a ordonné les sommets par numéros croissants).
4. Donner un graphe partiel de  $\mathcal{G}$ .
5. Donner un sous graphe de  $\mathcal{G}$ .
6. Déterminer le sous graphe engendré par la partie  $Y = \{1, 2, 3, 5\}$ .
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  $\mathcal{G}$ .

**Exercice 1 - 28 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.

**Exercice 1 - 29 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 sont-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-graphe 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.  $\mathcal{G}' = (X' = X - \{3\}, A \cap (X' \times X'))$  admet-il des circuits eulériens ? Des circuits hamiltoniens ?

**Exercice 1 - 30**

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 ?

**Exercice 1 - 31**

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 ?

**Exercice 1 - 32 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  $Q_3$ .

**Exercice 1 - 33 reconnaître un graphe connexe**

Soit  $\mathcal{G} = (X, E)$  un graphe symétrique ayant  $n$  sommets et tel que chaque sommet ait un degré au moins supérieur à  $d_0 = \frac{n-1}{2}$ .

1. Considérer un sommet quelconque  $x$  et noter  $V_x = \Gamma(x) \cup \{x\}$  montrer que  $\text{Card}(V_x) \geq \frac{n+1}{2}$ .
2. Considérer  $Y = X \setminus V_x$ . Montrer que son cardinal est au plus  $\frac{n-1}{2}$ . Que représente « concrètement »  $Y$  ?
3. En déduire que  $\mathcal{G}$  est connexe.

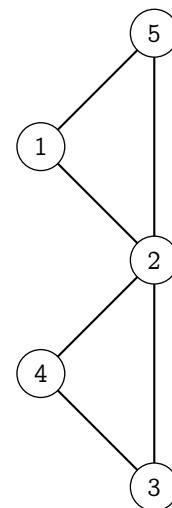
**Exercice 1 - 34**

Soit  $\mathcal{G}_n = (X_n, A_n)$  un graphe simple avec  $X_n = \{1, 2, \dots, n\}$ . 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.

**Exercice 1 - 35 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 ?

**Exercice 1 - 36 cycles**



Trouvez tous les cycles de ce graphe.  
 Peut-on avoir un cycle dans un graphe ayant deux arêtes ? Une arête ?  
 Et si l'on précise que le graphe est simple ?

**Exercice 1 - 37 graphes réguliers**

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

**Exercice 1 - 38**

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.

**Exercice 1 - 39**

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

**Exercice 1 - 40 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

**Exercice 1 - 41 algorithme mystérieux**

Que fait l'algorithme suivant sur un graphe simple symétrique  $\mathcal{G} = (X, A)$  :

```

Y ← X
Choisir un sommet x et le marquer d'un +
TantQue Y non vide Faire
    Choisir un sommet y marqué dans Y
    Marquer tous ses voisins avec le signe opposé
    Si un sommet reçoit deux signes opposés Alors
        Quitter { sortie de la boucle Tant Que :
le graphe n'est pas... }
    FinSi
    Y ← Y - {y}
    
```

Si chaque sommet a reçu un seul signe alors  $\mathcal{G}$  est...

**Exercice 1 - 42 algorithme mystérieux bis**

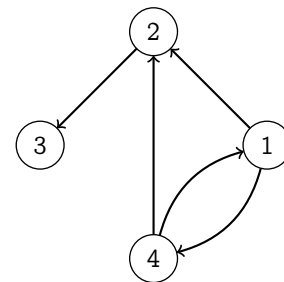
On considère un graphe  $\mathcal{G} = (X, A)$  avec  $X = \{1, 2, \dots, n\}$  et sa matrice d'adjacence  $(m_{ij})$  en prenant les sommets par ordre croissant.

On considère l'algorithme suivant :

```

Pour i variantDe 1 Jusque n Faire
    aii ← 1
    Pour j variantDe 1 Jusque n Faire
        Pour k variantDe 1 Jusque n Faire
            ajk ← max{ajk, ajiaik}
        FinPour
    FinPour
FinPour
    
```

Appliquer-le au graphe suivant :



Que peut-on déduire du fait que  $a_{ij} = 1$  dans la matrice finale ?

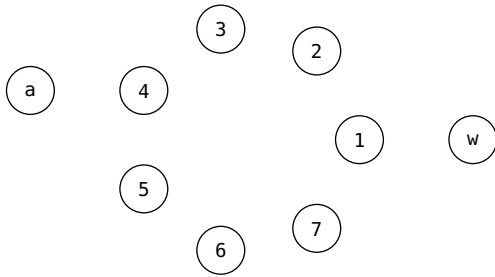
**Exercice 1 - 43 ordonnancement**

Un contrat vient d'être signé pour un projet informatique. Ce projet a été décomposé en tâches confiées à des ingénieurs et les contraintes de précedence ainsi que les durées des tâches sont présentées dans le tableau ci-dessous :

Rang	Libellé de la tâche	durée en mois	Tâches préalables
	A	3	
	B	8	A
	C	5	A
	D	1	C, G
	E	4	D
	F	2	C
	G	6	

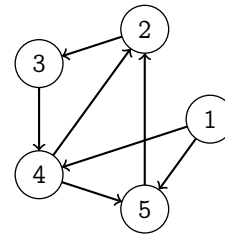
- (1) On décide de représenter ce problème d'ordonnancement à l'aide d'un graphe par la méthode MPM. En faisant abstraction des deux tâches fictives "début du projet" et "fin du projet", affecter un rang à chaque sommet en complétant le tableau précédent en utilisant l'algorithme qui permet de tester qu'un graphe est sans circuit.
- (1) Représenter ce problème à l'aide d'un graphe

MPM en complétant le dessin ci-dessous :



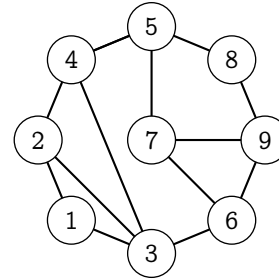
3. (4) Déterminer la durée minimale de réalisation du projet ainsi que la date de démarrage au plus tôt et au plus tard de chaque tâche en supposant que le projet démarre à la date 0.

Libellé de la tâche	date au plus tôt	date au plus tard
A		
B		
C		
D		
E		
F		
G		



**Exercice 1 - 46**

Voici un réseau de ce que vous voulez :



Comment obtenir le nombre de chaînes de longueurs 5 entre les sommets 1 et 9 ?

**Exercice 1 - 47**

1. Un graphe orienté modélise le réseau routier d'une ville : est-il connexe et (ou) fortement connexe ?
2. Même question avec un arbre généalogique.

**Exercice 1 - 48**

Soit le graphe  $G$  défini par son dictionnaire :  
 $G(a) = \{c, f\}$ ,  $G(b) = \{a, c, d, f\}$ ,  $G(e) = \{a, b, c, d, f\}$   
 $G(c) = \{d, f\}$ ,  $G(d) = \{a, f\}$ ,  $G(f) = \emptyset$  Déterminer les composantes fortement connexes du graphe  $G$ .

**Exercice 1 - 49**

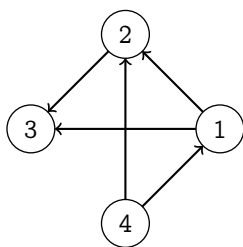
1. La relation de congruence modulo 3 est, comme vous le savez tous, une relation d'équivalence. Soit  $\mathcal{G}_3$  le graphe orienté de sommets  $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$  tel qu'il existe un arc entre les sommets  $a$  et  $b$  si, et seulement si,  $a \equiv b [3]$ .  
 Quelles sont ses composantes connexes ?
2. Soit  $\mathcal{G}(X, A)$  un graphe orienté. On définit la relation suivante :

$$\forall(x, y) \in X, x\mathcal{R}y \Leftrightarrow \begin{cases} x = y \\ \text{ou} \\ \text{il existe une chaîne entre } x \text{ et } y \end{cases}$$

Montrez qu'il s'agit d'une relation d'équivalence. Que peut-on dire des classes d'équivalences ?

**Exercice 1 - 50**

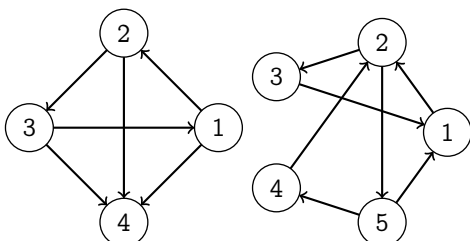
1. Soit  $\mathcal{G} = (X, A)$  un graphe simple, symétrique, sans boucles, ayant  $2n$  sommets de degrés au moins égal à  $n$ . Montrer que  $\mathcal{G}$  est alors connexe. (Vous pourrez par exemple raisonner par l'absurde.)
2. Soit  $\mathcal{G}$  un graphe simple, symétrique. Montrer que parmi  $\mathcal{G}$  et son complémentaire, l'un des deux est connexe.



Donner sa matrice d'adjacence  $M$ , puis  $M^2$ , puis  $M^3$ . Comment interpréter les coefficients de ces matrices ? Démontrer ce résultat par récurrence.

**Exercice 1 - 45**

Les graphes suivants sont-ils fortement connexes ? Dans le cas contraire, donner leurs composantes fortement connexes.

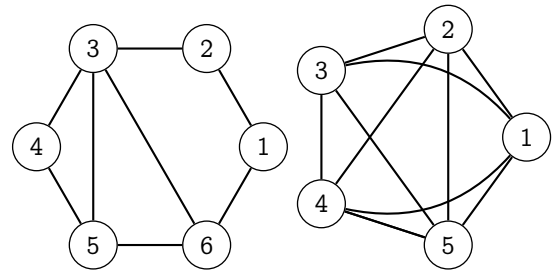


**Exercice 1 - 51**

Un graphe est  $k$ -connexe si on peut le déconnecter en retirant  $k$  sommets et qu'on ne peut pas en en supprimant  $k-1$ ; son degré de connexité est alors  $k$ . Cela permet, par exemple, de mesurer la résistance aux pannes d'un réseau informatique.

Dessinez la représentation d'un graphe 1-connexe; 2-connexe.

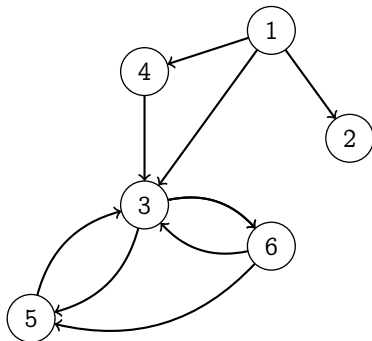
On définit de la même manière la  $k$ -connexité au sens des arêtes. Dessinez la représentation d'un graphe 1-connexe au sens des arêtes; 2-connexe au sens des arêtes.



Montrer que si un graphe simple symétrique est eulérien, alors tous les sommets sont de degré pair. Quelle est la taille d'un cycle eulérien ?

**Exercice 1 - 52**

Appliquer aux graphes suivant l'algorithme de parcours en largeur en faisant figurer l'arbre (ou la forêt<sup>e</sup>) de parcours résultant dans lequel les sommets de profondeur égale seront mis à la même hauteur :



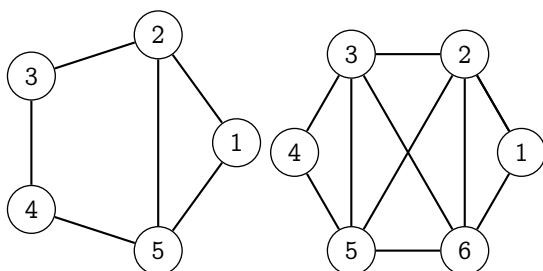
Faites de même avec le parcours en profondeur.

**Exercice 1 - 53**

1. Montrer qu'un arbre ayant  $n$  sommets avec  $n > 1$  a au moins une feuille (considérer une chaîne simple de longueur maximale d'extrémités  $s$  et  $t...$ ).
2. Montrer alors par récurrence qu'un arbre de  $n$  sommets avec  $n > 0$  a  $n - 1$  arêtes.
3. Proposer, à partir de ces résultats, un algorithme qui vérifie si un graphe non orienté et connexe possède un cycle. On suppose qu'un graphe est connu par son dictionnaire d'adjacence. Quel est l'inconvénient de cet algorithme ?

**Exercice 1 - 54**

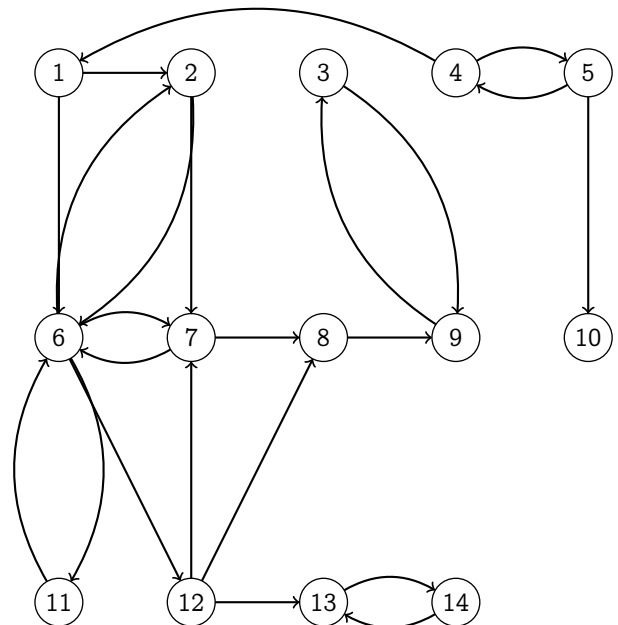
Est-ce que les graphes suivants sont eulériens ? Si oui, exhibez un cycle eulérien. Si non, existe-t-il au moins une chaîne eulérienne ?



e. Une forêt est un graphe constitué de plusieurs arbres...

**Exercice 1 - 55**

Effectuer un parcours en profondeur du graphe suivant et dresser la forêt du parcours puis déterminer ses composantes fortement connexes (il y en a 7...).



**Semaine 39 : Bellman-Ford et Dijkstra**

**Exercice 1 - 56 Bellman-Ford**

On considère un graphe valué sans circuit de valeur négative (ou circuit absorbant) mais ayant éventuellement des arcs pondérés négativement.

On s'intéresse à la distance d'un sommet particulier, la source, aux différents sommets du graphe.

On note  $\pi(v)$  la distance de la source  $s$  à un sommet  $v$ . Au départ, on pose  $\pi(s) = 0$  et pour tous les autres sommets du graphe,  $\pi(v) = +\infty$ .

L'idée est que, pour tout sommet  $v$ , si passer par un prédécesseur  $u$  de  $v$  raccourcit la distance, alors on remplace  $\pi(v)$  par  $\pi(u) + p(u, v)$  avec  $p(u, v)$  la pondération de l'arc  $(u, v)$ .

La propriété de BELLMAN est donc la suivante :

$$\pi(v) = \min_{u \in \Gamma^{-1}(v)} \{ \pi(u) + p(u, v) \}$$

**Algorithme Bellman version 1**

Un graphe valué  $G$  orienté est défini par un dictionnaire

Un sommet  $s$  de  $G$  est la source

$\pi$ : un dictionnaire indexé par les sommets

**Début**

$\pi(s) = 0$

**Pour** tous les sommets du graphe  $\neq s$  **Faire**

$\pi(s) \leftarrow +\infty$

**FinPour**

**TantQue**  $\pi$  change **Faire**

**Pour** chaque arc  $(u, v)$  de  $G$  **Faire**

**Si**  $\pi(v) > \pi(u) + p(u, v)$  **Alors**

$\pi(v) \leftarrow \pi(u) + p(u, v)$

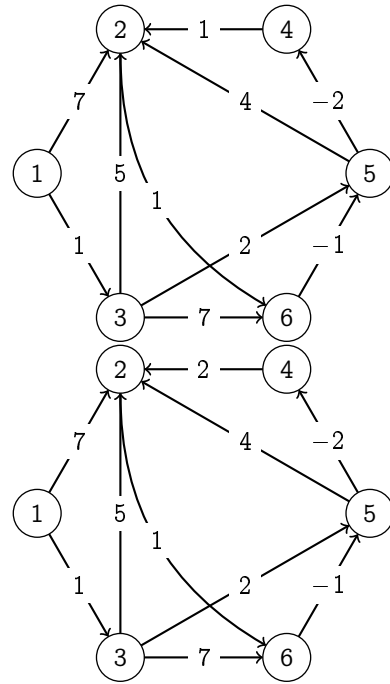
**FinSi**

**FinPour**

**FinTantQue**

**Retourner**  $\pi$

**Fin**

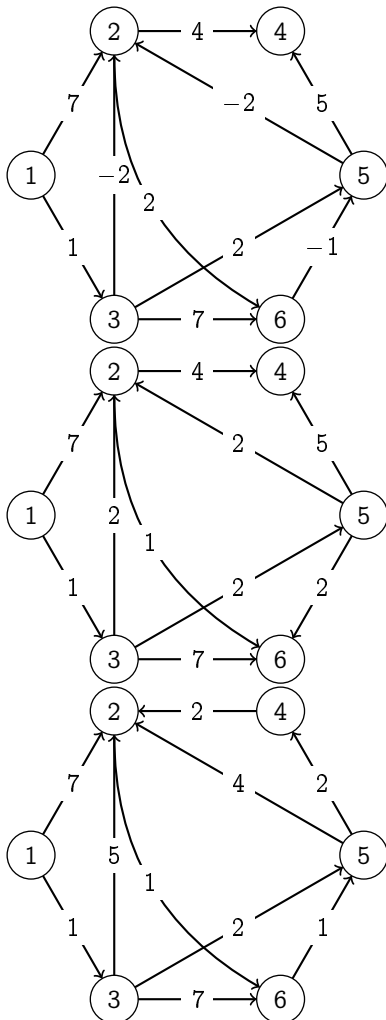


Une amélioration consisterait à mémoriser les plus courts chemins et pas seulement leur longueur.

À chaque fois que le traitement d'un arc  $(u, v)$  modifie  $\pi(v)$ , le nouveau prédécesseur de  $v$  devient  $u$  (on dit qu'on *relâche*  $(u, v)$ ).

On peut donc stocker ces prédécesseurs dans un nouveau dictionnaire.

Appliquer aux graphes suivants :



**Algorithme Bellman version 2**

Un graphe valué  $G$  orienté est défini par un dictionnaire

Un sommet  $s$  de  $G$  est la source

$\pi$ : un dictionnaire indexé par les sommets

**Début**

$\pi(s) = 0$

**Pour** tous les sommets du graphe  $\neq s$  **Faire**

$\pi(s) \leftarrow +\infty$

$pred \leftarrow$  un dictionnaire vide

**FinPour**

**TantQue**  $\pi$  change **Faire**

**Pour** chaque arc  $(u, v)$  de  $G$  **Faire**

**Si**  $\pi(v) > \pi(u) + p(u, v)$  **Alors**

$\pi(v) \leftarrow \pi(u) + p(u, v)$

$pred[v] \leftarrow u$

**FinSi**

**FinPour**

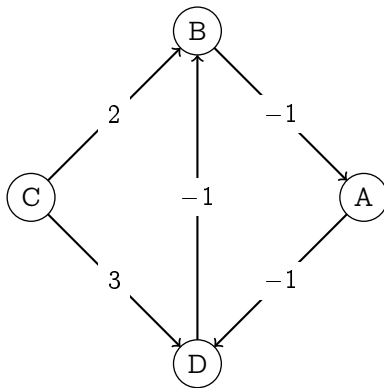
**FinTantQue**

**Retourner**  $\pi, pred$

**Fin**

Que se passe-t-il si le graphe contient un circuit né-

gatif ?



Il faudrait pouvoir détecter de tels circuits dans des cas plus compliqués.

En fait, on peut montrer que la boucle **Tant que**, s'il n'existe pas de tels circuits **absorbants**, se termine au maximum en  $n - 1$  itérations.

Imaginez un circuit de  $n$  sommets dont le plus court chemin donné par l'algorithme soit exactement  $n - 1$ .

On peut donc être amené à effectuer  $n - 1$  boucles. Montrons qu'on ne peut pas dépasser cette valeur en cas d'absence de circuit absorbant.

Tout d'abord, on va montrer l'invariant de boucle :

À l'étape  $i$ , pour tout sommet  $v$ ,  $\pi(v)$  est le plus court chemin de  $s$  à  $v$  contenant au plus  $i$  arcs.

La preuve d'un invariant de boucle s'effectue le plus souvent par récurrence.

Démontrer l'invariant de boucle. Pour l'hérédité, on distinguera deux cas.

Bon, maintenant, un chemin non élémentaire est forcément plus long qu'un chemin élémentaire : pourquoi ?

Il suffit donc d'étudier les chemins élémentaires : combien contiennent-ils d'arcs au maximum ? Conclusion ?

Comment peut-on alors changer la boucle principale ?

Comment alors tester si un graphe contient un circuit absorbant ?

Il peut s'avérer que l'on recherche les plus longs chemins d'un graphe : comment adapter le travail précédent pour trouver les plus longs chemins partant d'une source ?

Rappel de l'algorithme :

#### Algorithme Dijkstra

##### Début

$p_i \leftarrow$  dictionnaire vide

$\pi[\text{début}] = 0$

Visités  $\leftarrow$  liste vide

Chaîne  $\leftarrow$  [début]

$x \leftarrow$  début

Pour tout  $s$  de  $X \setminus \{\text{début}\}$  Faire

$\pi(s) \leftarrow +\infty$

    chaîne  $\leftarrow$  a

FinPour

TantQue  $x \neq$  fin Faire

    Pour  $v \notin$  Visités et voisin de  $x$  Faire

$\pi[v] \leftarrow \min \{ \pi[v], \pi[x] + p(x, v) \}$

    FinPour

    Extraire un sommet  $u \notin$  Visités tel que

$\pi[u] = \min \{ \pi[y], y \notin$  Visités  $\}$

$x \leftarrow$  u

    Visités  $\leftarrow$  Visités  $\cup \{u\}$

FinTantQue

Retourner  $\pi$

Fin

Appliquer l'algorithme de DIJKSTRA aux exemples précédents avec la méthode vue en cours. Que remarquez-vous ?

Python permet de gagner un peu de temps grâce à des petites particularités de traitement des dictionnaires.

Voici même une version récursive de l'algorithme. Décrypter la et écrire un programme avec une boucle au lieu de la récursion selon l'algorithme présenté précédemment.

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

```
"""
```

Quelques fonctions python utiles ici :

```
- on va rentrer un graphe comme un dictionnaire de
  dictionnaires pour pouvoir pondérer les arêtes:
g = {'a': {'b': 4, 'c': 2},
     'b': {'a': 4, 'd': 5, 'c': 1},
     'c': {'a': 2, 'b': 1, 'd': 8, 'e': 10},
     'd': {'b': 5, 'c': 8, 'e': 2},
     'e': {'c': 10, 'd': 2, 'f': 3},
     'f': {'e': 3, 'd': 6}}
```

Ainsi,  $g['a']$  est le dictionnaire des voisins avec leurs poids :

```
>>> g['a']
{'c': 2, 'b': 4}
```

### Exercice 1 - 57 Algorithme de Dijkstra

```
>>> g['a']['b']
4
>>> 'c' in g['a']
True

- pour trouver la clé correspondant à la valeur mini
  d'un dictionnaire, on utilise une syntaxe très
  pythonique. Par exemple, pour avoir le voisin le
  plus proche de 'a' :

>>> min(g['a'],key=g['a'].get)
'c'

- dict.get(cle,defaut) : retourne la valeur de cle si
  elle se trouve dans le dictionnaire et défaut sinon
;

- float('inf') représente l'infini
"""
```

```
def
plus_court(graphe,debut,fin,visites,pi,chaîne,espion):
    """trouve la plus courte chaîne entre debut et fin
    avec l'algo de Dijkstra
    visites et chaîne sont des listes et pi un
    dictionnaire
    version récursive
    """
    # si on arrive à la fin, on affiche la distance et
    la chaîne en y rajoutant fin
    if debut==fin:
        chaîne.append(fin)
        return pi[fin], chaîne, espion
    # si c'est la première visite, on met pi(debut) à 0
    if len(visites) == 0 : pi[debut]=0
    # on commence à tester les voisins
    for voisin in graphe[debut]:
        if voisin not in visites:
            dist_voisin = pi.get(voisin,float('inf'))
            candidat_dist = pi[debut] +
                graphe[debut][voisin]
            if candidat_dist < dist_voisin:
                pi[voisin] = candidat_dist
                if debut not in chaîne:
                    chaîne.append(debut)
    # on a regardé tous les voisins : le noeud entier
    est visité
    visites.append(debut)
    espion+=1
    # on cherche le sommet non visité le plus proche
    non_visites = dict((s, pi.get(s,float('inf')))) for
        s in graphe if s not in visites)
    noeud_plus_proche = min(non_visites,
        key=non_visites.get)
    # on applique récursivement en prenant comme début
    le plus proche
    return
    plus_court(graphe,noeud_plus_proche,fin,visites,pi,chaîne,espion)

def dij_rec(graphe,debut,fin):
    return plus_court(graphe,debut,fin,[],{},{},[],1)

if __name__ == "__main__":
```

```
g = {'a': {'b': 4, 'c': 2},
     'b': {'a' : 4,'d': 5, 'c': 1},
     'c': {'a' : 2, 'b' : 1, 'd' : 8,'e': 10},
     'd': {'b': 5, 'c': 8, 'e': 2},
     'e': {'c': 10, 'd' : 2, 'f': 3},
     'f': {'e' : 3,'d' : 6}}
print 'Version récursive : ',dij_rec(g,'a','f')
```

**Semaine 40 : coloration**

**Exercice 1 - 58 Théorème de Tutte**

1. Construire un graphe sans clique de degré 3 qui soit 3-chromatique.
2. Comment obtenir alors un graphe 4-chromatique sans clique de degré 4 ?
3. Faites de même pour des graphes 5-chromatique et 6-chromatique.
4. Peut-on généraliser ?

**Exercice 1 - 59**

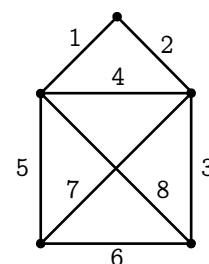
Cinq routes A, B, C, D et E se rejoignent en un seul carrefour (ordre anti-trigonométrique). Voici les franchissements possibles : AC, AE, BA, BD, BE, CA, CD, DA, DC, EC, ED. Certains franchissements ne peuvent être simultanés comme AC et BE, EC et BD, etc. Modélisez la situation à l'aide d'un graphe dont les sommets représentent les franchissements possibles et les arêtes joignent les franchissements incompatibles. Que peut-on dire des ensembles monochromes ? Comment interpréter le nombre chromatique ?

**Exercice 1 - 60**

Sept étudiants A, B, C, D, E, F et G se sont rendus à la cafétéria aujourd'hui. Voici la liste des rencontres, AD, AE, BD, BE, BF, BG, CE, CG, DA, DB, DE, EA, EB, EC, ED, EF, EG, FB, FE, GB, GC, GE, GF. En supposant que tous les étudiants sont assis à la cafétéria, de combien de chaises (denrée devenue rare à l'IUT) doit disposer la cafétéria ?

**Exercice 1 - 61**

1. Déterminer le graphe aux arêtes du graphe suivant (on a donné un nom à chaque arête) :



2. Quatre professeurs P1, P2, P3, P4 doivent donner des heures (1 heure par classe) de cours à 3 classes C1, C2, C3, les affectations sont données par le ta-

bleau :

	P1	P2	P3	P4
C1	oui		oui	oui
C2	oui	oui		oui
C3		oui		oui

On se pose la question : « Quel nombre minimum d'heures faut-il pour dispenser tous les cours ? ». Il est évident qu'un professeur ne peut pas faire cours à plus d'une classe dans un même créneau horaire et qu'une classe ne peut pas avoir cours avec plus d'un prof au même moment. Il vous est demandé de transformer ce problème en un problème de coloration d'un graphe, vous indiquerez donc sur quel graphe le travail va se faire et vous représenterez ce graphe en expliquant clairement la notation que vous utiliserez pour nommer les sommets et le pourquoi des arêtes ; vous préciserez aussi si c'est une recherche de coloration des sommets ou des arêtes (une coloration doit être ici un créneau horaire, par exemple rouge = 8h-9h, etc.)

### Exercice 1 - 62

Un peu de mécanique pour faire plaisir à M. MOTTU : une boîte de vitesse est un mécanisme qui permet de changer la fréquence de rotation de l'arbre intermédiaire pour une fréquence constante de rotation de l'arbre principal. Cela est dû à la présence de pignons (roues dentées) dans la boîte de vitesse. Un des principaux problèmes pour un concepteur de boîte de vitesse est de réduire sa taille ce qui revient souvent à réduire la quantité d'arbres reliés aux pignons d'engrenage. Certains pignons ne peuvent cependant pas se trouver sur le même arbre car ils pourraient créer des blocages ou leur poids dépasseraient les capacités de l'arbre par exemple. Ces paires impossibles sont les suivantes, sachant qu'il y a sept pignons a, b, c, d, e, f et g : ab, ad, ae, ag, bc, be, bg, ce, cf, df, eg, fg. Déterminer le nombre minimum d'arbres qui permet de fixer les pignons.

### Exercice 1 - 63

M. REMM a pris un congé pour assister aux marchés de Noël en Alsace et voilà le département pris au dépourvu au moment de mettre au point les emplois du temps du second semestre. Le jeudi, par exemple, il faut placer sept cours : Maths 1, Maths 2, Maths 3, Algo 1, Algo 2, Tamoul, Couture.

Cependant, certains cours ne peuvent avoir lieu en parallèle car ils sont dispensés par le même professeur ou certains étudiants doivent assister à chacun. C'est le cas de m1/m2, m1/a1, m1/c, m2/m3, m2/a2, m3/a2, m3/t, m3/c, a1/a2, a1/t et a2/t.

Tous les cours pourront-ils être dispensés le jeudi matin ?

### Exercice 1 - 64 Fonction chromatique

La fonction chromatique  $f$  associée à un graphe  $G$  et à un entier  $t$  le nombre de façons de colorier un graphe  $G$  avec  $t$  couleurs différentes.

On remarquera bien sûr que la quantité minimum  $t$  de couleurs pour laquelle  $f(G, t) > 0$  est  $\chi(G)$ .

1. Montrer que  $f(K_n, t) = t(t-1)(t-2)\dots(t-n+1)$  si  $t \geq n$  et 0 sinon.

2. Dans le cas général, le calcul de  $f(G, t)$  est compliqué. Nous allons cependant voir un théorème intéressant. Pour cela, on considérera l'opération de fusion de deux sommets : deux sommets  $u_1$  et  $u_2$  sont fusionnés pour former un unique sommet  $u_0$  qui sera adjacent à tous les sommets auxquels étaient adjacents  $u_1$  et  $u_2$ .

Voici le théorème :

Soit  $u$  et  $v$  deux sommets non adjacents d'un graphe  $G$ . Si le graphe  $G_1$  est obtenu à partir de  $G$  en ajoutant l'arête  $(u, v)$  et le graphe  $G_2$  est obtenu à partir de la fusion des sommets  $u$  et  $v$  alors :

$$f(G, t) = f(G_1, t) + f(G_2, t)$$

Démontrer ce théorème en distinguant les colorations possibles :

- soit  $u$  et  $v$  ont la même couleur ;
- soit  $u$  et  $v$  portent des couleurs différentes.

L'idée est alors de découper un graphe en plusieurs graphes selon la méthode précédente jusqu'à obtenir des graphes complets afin d'utiliser le résultat sur les  $K_n$ .

Par exemple, toujours à cause du congé de M. REMM, le département est en émoi pour organiser l'emploi du temps. Le mardi, cinq cours doivent être donnés avec toujours quelques incompatibilités. On décide de créer un quatrième créneau pour l'après-midi de 18h à 19h20 et de garder les trois créneaux du matin.

Combien y a-t-il de combinaisons possibles le matin et l'après-midi pour placer ces cinq cours en une seule demi-journée ?

Les incompatibilités sont  $C_1/C_2$ ,  $C_1/C_3$ ,  $C_1/C_4$ ,  $C_2/C_3$ ,  $C_2/C_4$ ,  $C_2/C_5$ ,  $C_3/C_5$ ,  $C_4/C_5$

### Exercice 1 - 65 Algorithme glouton de coloration

Il est temps d'utiliser Python pour déterminer un algorithme qui nous donnera une valeur possible de coloration d'un graphe à l'aide d'un algorithme glouton.

Soit  $G$  un graphe défini par un dictionnaire. On codera les couleurs par des entiers dans l'ordre croissant. Considérons un sommet associé à une couleur. Cherchons le prochain sommet non coloré. On le colorie avec le plus petit entier non utilisé par ses voisins.

On affiche ensuite le nombre de couleurs utilisées. À vous de jouer...

L'ordre dans lequel les sommets sont choisis a-t-il une influence ? Essayer avec un autre ordre en créant une permutation aléatoire des sommets.



On pourra utiliser `max(g.values())` qui donne la valeur maximale des valeurs associées aux labels du dictionnaire `g`.

**Semaine 41 : arbre de recouvrement et flots**

**Exercice 1 - 66 Algorithme de Prim**

On considère un arbre pondéré. On choisit un sommet. On a donc un arbre avec un sommet et zéro arête. À l'étape  $k$ , on a un arbre de  $k$  sommets et  $k - 1$  arêtes. On choisit ensuite l'arête de poids extrémal telle que son origine appartienne à l'arbre mais pas son extrémité. Appliquez l'algorithme de KRUSKAL puis celui de PRIM à ces graphes, dans le cas d'une recherche de recouvrement minimal puis maximal :

```
g = {'a':{'b':10,'c':1,'d':2,'a':5},
     'b':{'a':10,'e':3,'d':4,'c':6},
     'c':{'b':6,'a':1,'e':9,'d':3},
     'd':{'e':7,'a':2,'b':4,'c':3},
     'e':{'a':5,'b':3,'c':9,'d':7}
}

h = {'a': {'b': 1, 'f': 3, 'e': 5},
     'b': {'a' : 1, 'g': 2, 'c': 3},
     'c': {'b' : 8, 'h' : 3, 'd': 2},
     'd': {'c' : 6, 'i' : 4, 'e': 4},
     'e': {'d': 3, 'j' : 2, 'a': 5},
     'f': {'a': 4, 'i': 3, 'h': 1},
     'g': {'b' : 1, 'j': 2, 'h': 1},
     'h': {'c' : 8, 'f' : 1, 'j': 3},
     'i': {'d' : 6, 'g' : 4, 'f': 1},
     'j': {'e': 2, 'h' : 1, 'g': 4}
}
```

Quand vous serez en salle machine, déterminez un programme python qui pourra avoir ce squelette :

```
def prim(g,extr):
    T = []
    A = set([choice(list(g))])
    G = set(list(g))-A
    P = 0
    while ...
    ...
    return T,P
```

puis vérifiez les solutions trouvées sur papier.

**Exercice 1 - 67**

Un réseau de fibre optique va être installé pour équiper les maisons représentées par les sommets du graphe ci-dessous. Par mesure d'économie, les fibres doivent suivre les routes reliant ces maisons représentées par les arêtes du graphe et pondérées par leur longueur. Comment procéder pour utiliser une longueur minimale de fibre tout en équipant chaque maison ?

```
r = {'1':{'4':100,'2':200},
     '2':{'1':200,'5':170,'6':180,'3':150},
     '3':{'2':150,'6':100},
     '4':{'1':100,'5':240,'7':380},
     '5':{'2':170,'4':240,'7':210},
     '6':{'3':100,'2':180,'7':260},
```

```
'7':{'4':380,'5':210,'6':260}
}
```

**Exercice 1 - 68**

À Klow, capitale de la Syldavie, on peut se déplacer de n'importe quelle station de métro vers n'importe quelle autre.

Le beau-frère du Guide Suprême de la Syldavie dirigeant une entreprise de travaux d'entretien, il exige qu'un maximum de stations de métro soient fermées pour réparation mais le Guide Suprême demande de les choisir afin que dans le réseau restant, de toute station on continue à pouvoir se rendre vers n'importe quelle autre. Il est à noter que la réparation d'une station empêche les rames de la traverser. Le conseiller technique du maire, grassement payé car ex-étudiant de l'IUT de Nantes, a la solution : quelle est-elle ?

*(Il est à noter que le premier algorithme donnant un arbre couvrant de poids minimal a été donné par le Tchecoslovaque Otakar BORŮVKA en 1926 afin de concevoir le réseau de distribution électrique en Moravie du Sud et ce au moindre coût...)*

**Exercice 1 - 69**

Suite à des manifestations causées par le trop grand nombre de travaux dans la capitale, le Guide Suprême de la Syldavie décide de créer une nouvelle capitale dans le désert. Son conseiller technique, grassement payé car ex-étudiant de l'IUT de Nantes, lui conseille de donner à la ville l'aspect d'un carré de dimension 10 km × 10 km avec 101 rues rectilignes parallèles à un des côtés du carré et 101 autres perpendiculaires aux premières.

La distance entre deux rues parallèles voisines est de 100 m et la longueur de chaque rue est bien sûr de 10 km.

Le Guide Suprême a cependant un budget serré, lourdement grevé par le salaire du conseiller technique. Il exige donc de recouvrir d'asphalte les rues mais en utilisant une quantité minimale d'asphalte tout en permettant de rendre toute intersection accessible.

Combien de mètres linéaires d'asphalte doit-il acheter à son cousin, propriétaire de la société « Syldavian Macadam & Co » ?

**Exercice 1 - 70 Générateur de labyrinthe**

On considère une grille dont les traits horizontaux et verticaux représentent des murs. Un labyrinthe est alors une grille dont certains murs ont été détruits. Il est parfait lorsque toute case a un chemin simple et un seul vers toute autre case.

Comment associer un labyrinthe parfait à un graphe ? Quelle est la nature de ce graphe ?

Comment construire un labyrinthe en modifiant légèrement l'algorithme de Prim ?

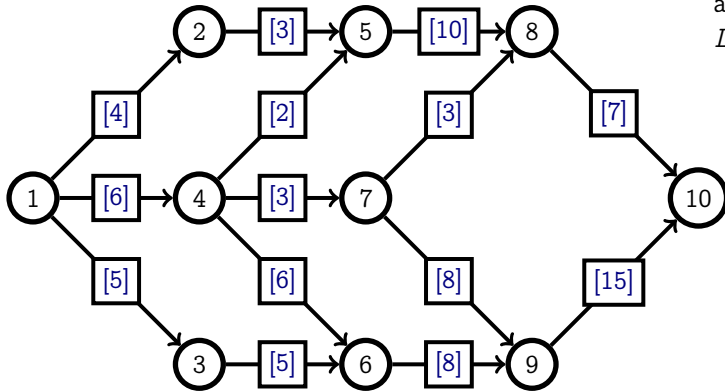
Un projet pourrait être de créer un générateur de labyrinthe et qui le dessine...

**Exercice 1 - 71**

Est-ce que tout flot complet est saturé ?

**Exercice 1 - 72**

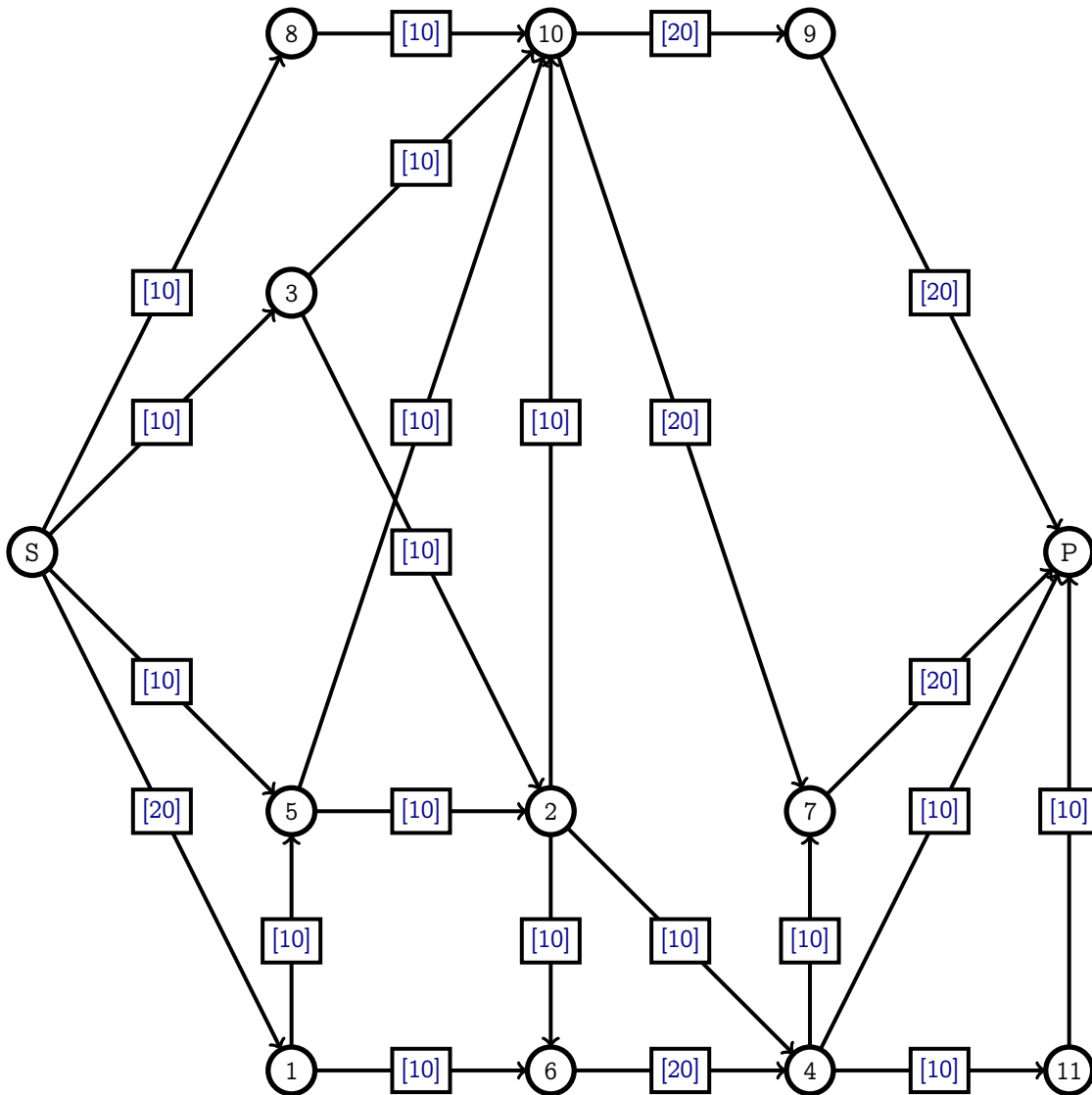
Déterminer un flot de capacité maximum dans le réseau de transport suivant :



La réponse est 14...

**Exercice 1 - 73**

Un serveur S souhaite envoyer des données à un client P. On suppose que les deux postes sont connectés par le réseau Télécom décrit dans le graphe suivant :



Les valuations des arcs représentent un débit en Mo/s. Les sommets sont des routeurs.

On suppose qu'un routeur est capable de recevoir plusieurs parties d'un même message par des voisins différents et de les redécouper pour l'envoyer à plusieurs voisins à la fois. Quelle quantité d'information peut-on alors faire passer de S à P en une seconde ?  
La réponse est 50 Mo...