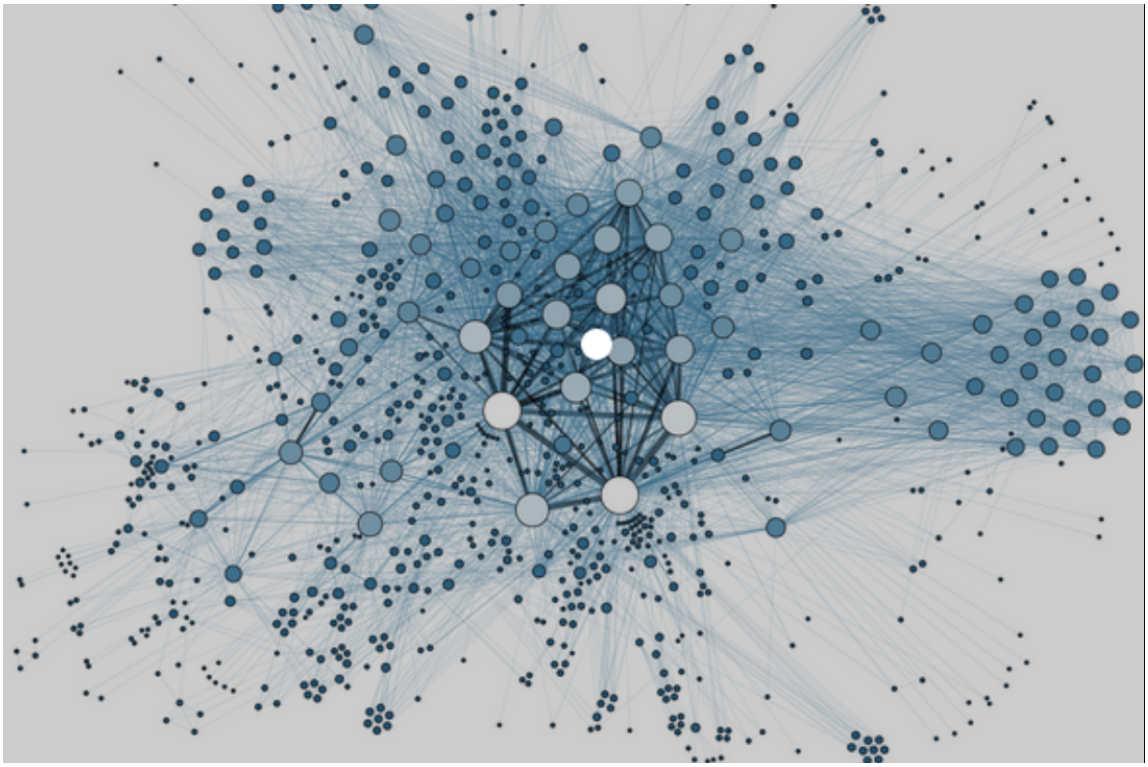
Graphes et algorithmes de graphes



Ces notes ont pour vocation d'assurer une base complète et uniformisée du contenu du cours "algorithmes de graphes". Elles ne contiennent pas la totalité des dessins, exemples, remarques et commentaires donnés au cours des séances : pour compléter, la prise de notes individuelle est donc vivement recommandée !

Table des matières

Chapitre 1

Préambule : le yin et le yang des graphes

Quand on explore le monde des graphes, on peut avoir la sensation de naviguer entre deux pôles : d'un côté, le formalisme théorique avec ses définitions et théorèmes, et de l'autre, les méthodes pratiques avec leurs algorithmes.
À un type de problème de la "vraie vie", on fait correspondre un type de graphe avec éventuellement des structures supplémentaires - une distance sur les arcs par exemple. Les algorithmes de graphes correspondent alors à la recherche des éventuelles solutions au problème. Les théorèmes, au contraire, sont rarement constructifs : ils imposent souvent des limites, expriment des conditions qui excluent ou garantissent l'existence de solutions, sans pour autant les dévoiler.
D'un côté, les algorithmes sont construits par l'intelligence humaine dans un but bien précis, et peuvent être améliorés, ou repensés : un peu comme la technologie. De l'autre côté, les théorèmes sont des vérités universelles et immuables : un peu comme les lois de la physique.
Cette dualité est féconde ; chacun des deux pôles motive et enrichit l'autre. La technologie permet de mesurer, expérimenter, pour mieux connaître les lois de l'univers, et réciproquement la connaissance théorique permet de développer la technologie. Il en va de même dans le cas des graphes : certains théorèmes d'existence découlent de l'exécution d'un algorithme, et certains algorithmes sont améliorés grâce à une connaissance théorique plus fine.

Chapitre 2

Généralités sur les graphes

Remarque

Dans le domaine des graphes, contrairement à la plupart des autres domaines mathématiques, il existe un certain foisonnement de définitions, de dénominations, de notations et de conventions, pas toujours très bien uniformisées... Cette diversité reflète la multitude des pratiques et des utilisations (on peut voir des graphes presque partout !). Il ne s'agit donc pas d'apprendre par coeur des notations rigides, mais plutôt de comprendre de quoi on parle, de faire le lien entre son intuition et le formalisme, et de prendre un peu de hauteur pour être flexible et savoir s'adapter aux différents contextes - tout en restant bien sûr cohérent avec ses propres conventions.

Dans ce cours, tous les graphes sont supposés finis et simples, c'est à dire sans arêtes multiples ni boucles.

I Graphes non orientés

Définition

On appelle graphe non orienté G tout couple $G = (\mathcal{S}, \mathcal{A})$ où :

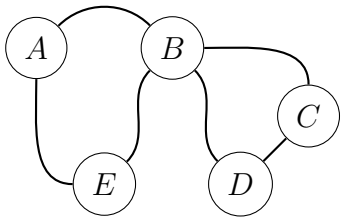
- $\mathcal{S}$ est un ensemble fini d'objets appelés **sommets** ou **noeuds**.
- $\mathcal{A}$ est un ensemble fini d'éléments appelés arêtes tel que chaque arête est associée à une paire $\{x, y\}$ de sommets de $\mathcal{S}$. Les graphes de ce cours étant tous des graphes simples, on pourra confondre un élément de $\mathcal{A}$ avec la paire $\{x, y\}$ de sommets distincts auquel il est associé.

Remarque

Un graphe apparaît donc dès lors qu'on veut représenter une **relation** entre des objets. Il est souvent représenté par un dessin où les sommets sont des points et les arrêtes des lignes reliant deux sommets.

Exemple

Si on prend $\mathcal{S} = \{A, B, C, D, E\}$ et $\mathcal{A} = \{\{A, B\}, \{A, E\}, \{B, C\}, \{B, D\}, \{B, E\}, \{C, D\}\}$, on peut représenter le graph $G = (\mathcal{S}, \mathcal{A})$ par le dessin ci-dessous.



Le réseau Facebook, ayant pour sommets les différents comptes, et pour arêtes les relations d'amitié, est un graphe non orienté.

Remarque

L'ensemble des arrêtes $\mathcal{A}$ est inclus dans l'ensemble des parties à deux éléments de $\mathcal{S}$, donc, pour un graphe ayant n sommets avec $n \in \mathbb{N}^*$, on a :

$$\text{card}(\mathcal{A}) \leqslant \binom{n}{2} \qquad \text{donc} \qquad \text{card}(\mathcal{A}) \leqslant \frac{n(n-1)}{2}$$

Définitions

Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe non orienté.

- On appelle ordre du graphe le nombre de ses sommets.
- Deux sommets x et y sont dit **adjacents** (ou voisin) s'ils sont reliés par une arrête, c'est à dire $\{x, y\} \in \mathcal{A}$
- L'ensemble des voisins d'un sommet x est noté $\Gamma(x)$.
- Une arrête est dite **incidente** aux sommets qu'elle relie. c'est à dire que l'arrête $\{x, y\}$ est incidente au sommet x et au sommet y.
- Le degré d'un sommet noté $\deg(x)$ est le nombre d'arêtes qui lui sont incidentes.

Propriétés

- Dans notre cas des graphes simples, le nombre d'arêtes incidente à un sommet x d'un graphe G est aussi le nombre de ses voisins, d'où

$$\deg(x) = \text{card}\left(\Gamma(x)\right)$$

- Dans tout graphe $G = (\mathcal{S}, \mathcal{A})$, on a la relation

$$\sum_{s \in \mathcal{S}} \deg(s) = 2 \cdot \text{card}(\mathcal{A}),$$

qu'on appelle parfois le **lemme des poignées de main**.

Démonstration

Chaque arête $\{s, s'\}$ de $\mathcal{A}$ étant incidente à deux sommets distincts s et s', elle apporte une contribution de "+2" dans la somme des degrés, d'où :

$$\sum_{s \in \mathcal{S}} \deg(s) = 2 \cdot \text{card}(\mathcal{A})$$

Vous trouverez ci-dessous une démonstration plus formelle.

$\sum_{s\in\mathcal{S}} \deg(s) = \sum_{s\in\mathcal{S}} \text{card}\left(\left\{\{s,s'\}\in\mathcal{A} | s'\in\mathcal{S}\right\}\right)$ Ici, on dénombre les paires "sommet - arête"

$= \sum_{a\in\mathcal{A}} \text{card}\left(\left\{s\in\mathcal{S} | s\in a\right\}\right)$ Ici, on a changé de point de vu, on dénombre les paires "arêtes-sommet".

or $\forall a\in\mathcal{A}, \text{card}\left(\left\{s\in\mathcal{S} | s\in a\right\}\right) = 2$, c'est à dire que chaque arête est incidente à deux sommets, d'où :

$\sum_{s\in\mathcal{S}} \deg(s) = \sum_{a\in\mathcal{A}} 2$

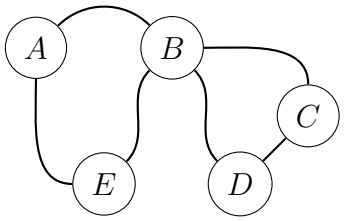
donc :

$$\sum_{s\in\mathcal{S}} \deg(s) = 2\cdot \text{card}(\mathcal{A})$$

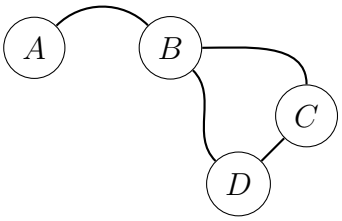
Définition

- On appelle **sous-graphe** d'un graphe G tout graphe obtenu en supprimant certains sommets de G (ainsi que toutes les arêtes incidentes ces sommets).
- On appelle **graphe partiel** d'un graphe G tout graphe obtenu en gardant tous les sommets mais en supprimant certaines arêtes.
- Un graphe partiel d'un sous graphe de G est appelé **sous graphe partiel** de G.

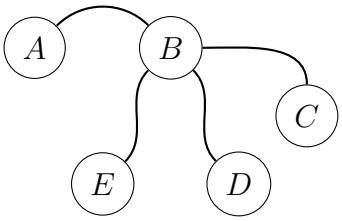
Exemples



Graphe G



Sous graphe de G



Graphe partiel de G

Si l'on considére le graphe G des liaisons aériennes mondiales, où les sommets représentent les aéroports. Alors le graphe des liaisons intra-européennes est un sous-graphe de G. Le graphe des liaisons de moins de trois heures est un graphe partiel de G.

Définitions

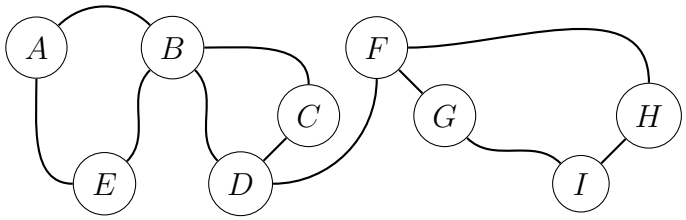
Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe non orienté.

- On appelle **chaîne** de **longueur** p toute suite de $p+1$ sommets $s_0, s_1, s_2, ..., s_p$ telle que deux sommets consécutifs sont adjacents.
  C'est à dire que pour tout $i\in\mathbb{N}$ tel que $0\leqslant i\leqslant p-1$, on a $\{s_i, s_{i+1}\}\in\mathcal{A}$.
  Cela correspond à un parcours dans le graphe, d'un sommet s_0 à un sommet s_p, en suivant les p arêtes.
  Cette définition n'est valable que pour les graphes simples.
- Une chaîne est dite **simple** si elle ne passe pas deux fois par la même arête.
- Une chaîne est dite **élémentaire** si elle ne passe pas deux fois par le même sommet.

La condition "élémentaire" est la plus forte : toute chaîne élémentaire est nécessairement simple. La réciproque étant fausse, en effet, il existe des chaînes simples qui ne sont pas élémentaires.

- Un **cycle** est une chaîne dont le sommet d'arrivée est égal au sommet de départ. En général, quand on parle de cycle, on sous-entend "cycle simple" ou même "cycle élémentaire". En effet, les cycles non simples n'ont pas grand intérêt pour caractériser un graphe (pour chaque arête $\{s,t\}$, la suite s,t,s est un exemple de cycle non simple...), et les cycles simples non élémentaires peuvent se décomposer en cycles élémentaires.
- On appelle distance entre deux sommets x et y la longueur de la plus courte chaîne entre ces deux sommets. On la note $d(x,y)$.

Exemple



Dans le graphe représenté ci-dessus, $\{D,F,G,I,H,F\}$ est une chaîne simple de longueur 5 composée des arêtes $\{\{D,F\},\{F,G\},\{G,I\},\{I,H\},\{H,F\}\}$, Ce n'est cependant pas une chaîne élémentaire puisqu'on retrouve deux fois le sommet F.
$\{F,G,I,H,F\}$ est un cycle de longueur 4 composée des arêtes $\{\{F,G\},\{G,I\},\{I,H\},\{H,F\}\}$ et c'est un cycle élémentaire.

Définition

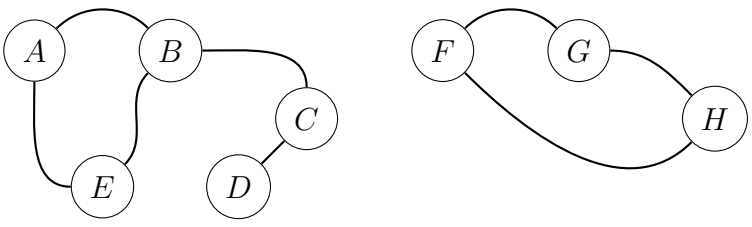
Étant donné un sommet s dans un graphe $G=(\mathcal{S},\mathcal{A})$, on peut se demander quels sont les sommets qui peuvent être reliés à s par une chaîne. Formellement, il s'agit de l'ensemble

$$\widehat{\Gamma}(s)=\bigcup_{k\geq 0}\Gamma^k(s)=\{s\}\cup\Gamma(s)\cup\Gamma(\Gamma(s))\cup\ldots$$

On inclut s dans l'ensemble car on considère qu'il est relié à lui-même par une chaîne de longueur 0. Puis on rajoute tous les $\Gamma^k(s)=\Gamma(\Gamma\ldots(s))$ qui sont les voisins de voisins de voisins ...(k fois) de s, autrement dit les sommets reliés à s par une chaîne de longueur k.

Le sous graphe de G associé à $\widehat{\Gamma}(s)$ s'appelle la **composante connexe** de s.

Exemple



Le graphe ci-dessus est composé de deux composantes connexes :

$$\widehat{\Gamma}(A) = \{A, B, C, D, E\} \quad \text{et} \quad \widehat{\Gamma}(F) = \{f, G, H\}$$

Propriété

Un graphe se décompose de manière unique comme l'union de composantes connexes deux à deux disjointes, deux sommets pouvant être reliés s'ils appartiennent à la même composante connexe.

Démonstration

Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe fini et simple.
On note $\widehat{G}(s)$ la composante connexe associé à $s \in \mathcal{S}$.
On a $G = \bigcup_{s\in\mathcal{S}} \widehat{G}(s)$, $\bigcup_{s\in\mathcal{S}} \widehat{G}(s)$ étant le sous graphe de G engendré par l'ensemble les sommets $\bigcup_{s\in\mathcal{S}} \widehat{\Gamma}(s)$, d'où l'existence d'une décomposition de G en composantes connexes.
D'autre part, s'il existe deux décompositions distinctes de G, cela signifie qu'il existe $s \in \mathcal{S}$ et $s' \in \widehat{\Gamma}(s)$ tel que $\widehat{G}(s') \neq \widehat{G}(s)$, d'où $\widehat{\Gamma}(s') \neq \widehat{\Gamma}(s)$ et par conséquent $s' \notin \widehat{\Gamma}(s)$, d'où l'incohérence.

Définition

Un graphe est dit connexe s'il est formé d'une seule composante connexe : c'est le cas où l'on peut relier chaque paire de sommets par une chaîne.

II Graphes orientés

Définition

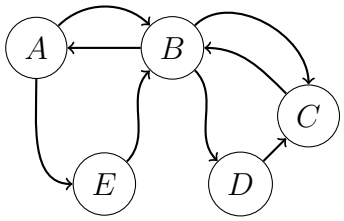
On appelle graphe orienté G tout couple $G = (\mathcal{S}, \mathcal{A})$ où :

- $\mathcal{S}$ est un ensemble d'objets appelés **sommets** ou **noeuds**.
- $\mathcal{A}$ est un ensemble fini d'éléments appelés **arcs** tel que chaque arc est associé à un couple (x, y) de sommets de $\mathcal{S}$. Les graphes considérés dans ce cours étant simples, chaque arc sera confondu avec le couple de sommet auquel il est associé. Les **arcs** du graphe et sont représentés par des flèches.

Remarque

Dans le cas d'un graphe orienté, le couple (x, y) est ordonné : il traduite une relation du sommet x vers le sommet y, mais il n'implique pas de relation de y vers x, c'est à dire l'existence de l'arc (y, x).

Exemple



le graphe représenté ci-dessus est composé de 5 sommets $\mathcal{S} = \{A, B, C, D, E\}$
Il est orienté et possède 8 arcs :

$$\mathcal{A} = \{(A,B), (A,E), (B,A), (B,C), (B,D), (C,B), (D,C), (E,B)\}$$

Le réseau Instagram contrairement à Facebook est un graphe orienté : un arc (s,t) indique que "s est un follower de t", ce qui est différent de "t est un follower de s".

Remarque

Les graphes non orientés peuvent être vus comme des graphes orientés en dédoublant chaque arête $\{s,t\}$ pour créer un couple d'arcs (s,t) et (t,s).
Ainsi les graphes orientés forment une classe plus générale et toute proposition concernant les graphes orientés sera en particulier valable pour les graphes non orientés, via cette transformation de "dédoublement".

Définition

Dans le cas d'un graphe orienté, on peut distinguer les voisins selon le sens des flèches. En considérant un arc (s,t) d'un graphe G, on dit que :

- t est un **successeur** de s et on note $\Gamma^+(s)$ l'ensemble des successeurs de s
- Le **degré sortant** ou **extérieur** d'un sommet s, noté $\deg^+(s)$, est le nombre d'arcs sortants de s, ou de manière équivalente le nombre de successeurs de s :

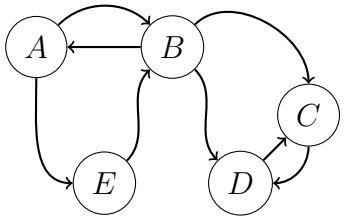
$$\deg^+(s) = \text{card}\left(\Gamma^+(s)\right).$$

- s est un **prédécesseur** de t et on note $\Gamma^-(s)$ l'ensemble des prédécesseurs de s.
- Le **degré entrant** ou **intérieur** d'un sommet s, noté $\deg^-(s)$, est le nombre d'arcs entrants en s, ou de manière équivalente le nombre de prédécesseurs de s :

$$\deg^-(s) = \text{card}\left(\Gamma^-(s)\right).$$

Exemple

Soit $G = (\mathcal{S}, \mathcal{A})$ le graphe représenté ci-dessous.



A, C et D sont les successeurs de B, d'où

$$\Gamma^+(B) = \{A, C, D\} \quad \text{et} \quad \deg^+(B) = 3$$

A et E sont les prédécesseurs de B d'où

$$\Gamma^-(B) = \{A, E\} \quad \text{et} \quad \deg^-(B) = 2$$

Propriété

Le lemme des poignées de main peut se reformuler ainsi dans le cas orienté :

$$\sum_{s\in\mathcal{S}} \deg^+(s) = \sum_{s\in\mathcal{S}} \deg^-(s) = \text{card}(\mathcal{A}).$$

Démonstration

Dans un graphe orienté $G = (\mathcal{S}, \mathcal{A})$, puisque chaque arc apporte une contribution de "+1" dans la somme totale des degrés sortants (resp. entrants), on a bien :

$$\sum_{s\in\mathcal{S}} \deg^+(s) = \sum_{s\in\mathcal{S}} \deg^-(s) = \text{card}(\mathcal{A}).$$

Une démonstration plus formelle similaire à celle faite dans le cas des graphes non orientés section I , page 8, est donnée ci-dessous.

$$\begin{aligned}\sum_{s\in\mathcal{S}} \deg^+(s) &= \sum_{s\in\mathcal{S}} \text{card}\left(\left\{(s,s') \in \mathcal{A} | s' \in \mathcal{S}\right\}\right)\\ &= \sum_{a\in\mathcal{A}} \text{card}\left(\left\{s \in \mathcal{S} | (s,s') = a\right\}\right)\\ &= \sum_{a\in\mathcal{A}} 1\\ &= \text{card}\left(\mathcal{A}\right)\end{aligned}$$

de même, on a :

$$\begin{aligned}\sum_{s\in\mathcal{S}} \deg^-(s) &= \sum_{s\in\mathcal{S}} \text{card}\left(\left\{(s',s) \in \mathcal{A} | s' \in \mathcal{S}\right\}\right)\\ &= \sum_{a\in\mathcal{A}} \text{card}\left(\left\{s \in \mathcal{S} | (s',s) = a\right\}\right)\\ &= \sum_{a\in\mathcal{A}} 1\\ &= \text{card}\left(\mathcal{A}\right)\end{aligned}$$

Remarque
Dans le cas des graphes orientés, on a des définitions similaires aux graphes non orienté, mais il faut faire attention au sens de parcours des arcs...

Définition
Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe orienté.
- On appelle **chemin** de **longueur** p toute suite de $p+1$ sommets $s_0, s_1, s_2, ..., s_p$ telle que chaque sommet $s_0, s_1, s_2, ..., s_{p-1}$ est prédécesseur du sommet suivant.
  C'est à dire que pour tout $i \in \mathbb{N}$ tel que $0 \leqslant i \leqslant p-1$, on a $(s_i, s_{i+1}) \in \mathcal{A}$.
  Cela correspond à un parcours dans le graphe composé de p arcs, d'un sommet s_0 à un sommet s_p.
- Un chemin est dit **simple** si il ne passe pas deux fois par le même arc.
- Un chemin est dit **élémentaire** si il ne passe pas deux fois par le même sommet.
  La condition "élémentaire" est la plus forte : tout chemin élémentaire est nécessairement simple.
- Un **circuit** est un chemin dont le sommet d'arrivée est égal au sommet de départ.
- On appelle distance entre deux sommets x et y la longueur d'un plus court chemin entre ces deux sommets. On la note $d(x, y)$.

Remarque
Dans un graphe orienté $G = (\mathcal{S}, \mathcal{A})$, les trois expressions suivantes sont équivalentes à dire qu'il existe un chemin d'origine s et d'extrémité t :

1. t est **accessible** depuis s,
2. t est un **descendant** de s,
3. s est un **ascendant** (ou **ancêtre**) de t.

Les notions d'ancêtre et de descendant sont transitives : l'ancêtre d'un ancêtre est encore un ancêtre, **etc**.
Comme précédemment, l'ensemble $\widehat{\Gamma}^+(s)$ des descendants de s et l'ensemble $\widehat{\Gamma}^-(s)$ des ancêtres de s peuvent s'écrire (remplacer $\pm$ par $+$ ou $-$) :

$$\widehat{\Gamma}^{\pm}(s) = \bigcup_{k \geq 0} (\Gamma^{\pm})^k(s) = \{s\} \cup \Gamma^{\pm}(s) \cup \Gamma^{\pm}(\Gamma^{\pm}(s)) \cup ...$$

Définition
Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe orienté.
Le sous graphe de G engendré par $\widehat{\Gamma}^+(s) \cap \widehat{\Gamma}^-(s)$ des sommets qui sont à la fois ancêtres et descendants de $s \in \mathcal{S}$ s'appelle la **composante fortement connexe** de s. On la note $CFC(s)$.

Remarque
Il est facile de vérifier que si deux sommets u, v sont dans $CFC(s)$, alors ils peuvent être reliés entre eux par un chemin dans le sens $u \rightsquigarrow v$ comme dans le sens $v \rightsquigarrow u$.

Propriété

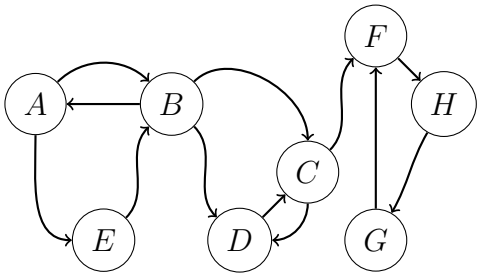
L'ensemble des sommets d'un graphe orienté se décompose de manière unique comme union de CFC deux à deux disjointes, deux sommets pouvant être reliés dans les deux sens si et seulement s'ils appartiennent à la même CFC.

Démonstration

Cette démonstration peut être faite suivant le même schéma que celle sur les graphes orientés. (Voir section I , page 11.)

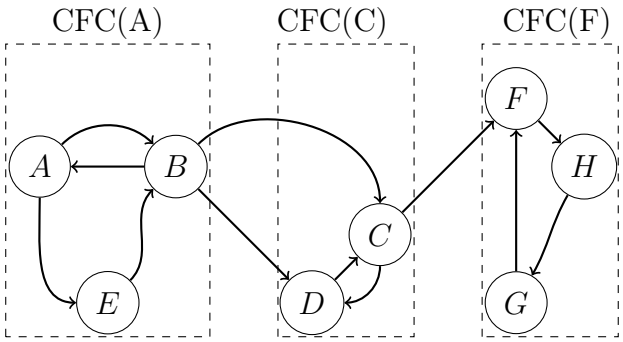
Exemple

Soit $G = (\mathcal{S}, \mathcal{A})$ le graphe représenté ci-dessous.



$\widehat{\Gamma}^+(A) = \{A, B, C, D, E, F, G, H\}$ et $\widehat{\Gamma}^-(A) = \{A, B, E\}$ donc $CFC(A) = \{A, B, E\}$

On remarque que $CFA(B) = CFA(E) = CFA(A)$
La décomposition en CFC du graphe ci-dessus donne 3 composantes fortement connexes représentée ci-dessous :



Définition

Un graphe est dit **fortement connexe** s'il est formé d'une seule CFC.

III Des graphes spéciaux

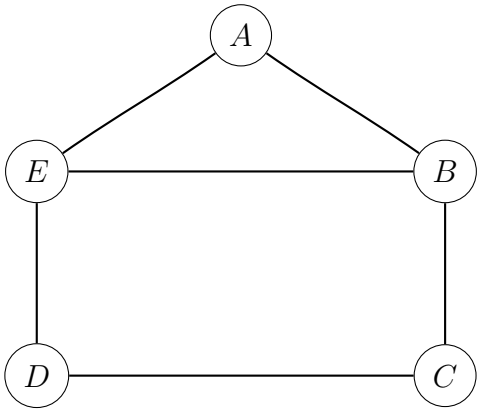
Dans cette partie, tous les graphes seront considérés non orientés sauf si cela est précisé explicitement.

1 Graphe Eulérien

Définition

- On appelle chaîne eulérienne d'un graphe non orienté toute chaîne qui passe par toutes les arêtes, une fois et une seule fois par arête.
- Si une chaîne eurélienne revient au sommet de départ, on parle de cycle eulérien.
- Un graphe qui admet un cycle eulérien est dit eulérien.
- S'il admet une chaîne eulérienne, il est dit semi-eulérien.

Exemple



Propriété

- Un graphe connexe est semi-eurélien si et seulement si le nombre de ses sommets de degré impair est 0 ou 2.
- Un graphe connexe est eurélien si et seulement si tous ses sommets sont de degré pair.

Démonstration

2 Graphe Hamiltonien

Remarque

Au IXe siècle, le poète indien Rudrata a été le premier à trouver le problème des cycles/chemins hamiltoniens sur un échiquier mais avec un cavalier (c'est-à-dire de parcourir chaque case une et seule fois en n'utilisant que les mouvements d'un cavalier). Il s'agit du problème du cavalier.

Définition

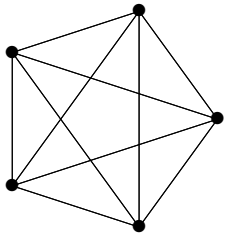
- Un chemin (resp. une chaîne) hamiltonien(e) d'un graphe orienté (resp. non orienté) est un chemin (resp. une chaîne) qui passe par tous les sommets une et une seule fois.
- Un circuit (resp. cycle) hamiltonien est un chemin (resp. chaîne) hamiltonien(e) qui est un circuit (resp. cycle).
- Un graphe hamiltonien est un graphe qui possède un circuit ou un cycle hamiltonien.

3 Graphes complets

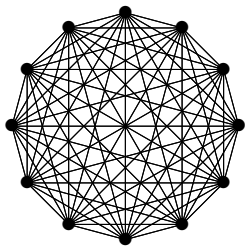
Définition

Un graphe G non orienté est **complet** si toute paire de sommets distincts est reliée par une arête.
On appelle **clique** tout sous graphe complet d'un graphe G.

Exemples



graphe complet d'ordre 5



graphe complet d'ordre 12

Propriété

Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe non orienté complet tel que $n = \text{card}\,(\mathcal{S}) \in \mathbb{N}^*$.

$$\text{card}(\mathcal{A}) = \frac{n(n-1)}{2}$$

Démonstration

Le nombre d'arêtes de G est égal au nombre de combinaisons à deux sommets (non ordonnés) parmi les n sommets du graphe, donc :

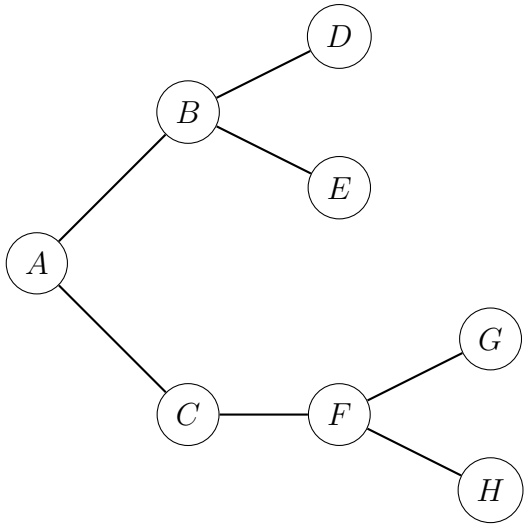
$$\text{card}(\mathcal{A}) = \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$$

4 Arbres

Définition

Un **arbre** est un graphe non orienté, connexe et sans cycle simple ni élémentaire.
Les arbres sont très souvent utilisés pour représenter des données (arbre généalogique, **etc**).

Exemple



Définition

Dans un graphe G quelconque, un **arbre couvrant** est un graphe partiel de G qui est un arbre.
Autrement dit, c'est un arbre inclus dans G et qui relie tous les sommets de G.

Théorème

Tout graphe connexe admet au moins un arbre couvrant.

Démonstration

On peut démontrer par récurrence que tout graphe connexe d'ordre $n \in \mathbb{N}^*$ admet au moins un arbre couvrant.

- **Initialisation**
  Si $n = 1$, alors G est un arbre.
- **Hérédité**
  Soit $n \in \mathbb{N}^*$ tel que tout graphe connexe d'ordre n admet au moins un arbre couvrant.
  Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe connexe d'ordre $n+1$ et s un sommet de $\mathcal{S}$.
  D'après l'hypothèse de récurrence, le sous graphe de G associé aux n sommets $\mathcal{S} \setminus \{s\}$ admet au moins un arbre couvrant $A_n = \big(\mathcal{S} \setminus \{s\}, E_n\big)$.
  G étant connexe, il existe $\{s, s'\} \in \mathcal{A}$ tel que $s' \in \mathcal{S} \setminus \{s\}$, et le graphe $A_{n+1} = \big(\mathcal{S}, E_n \cup \big\{\{s, s'\}\big\}\big)$ est un arbre couvrant de G.
  En effet, d'après la construction de A_{n+1}, il n'existe qu'une seule arête incidente à s, donc si A_{n+1} admet un cycle simple, ce dernier ne contient pas le sommet s, c'est donc un cycle de A_n, et par conséquent A_n n'est pas un arbre, ce qui contredit l'hypothèse de récurrence. Donc A_{n+1} est un arbre.
- **Conclusion**

. Tout graphe connexe d'ordre $n \in \mathbb{N}^*$ admet au moins un arbre couvrant.

Remarque

On peut déterminer l'arbre couvrant d'un graphe G, par exemple, grâce aux algorithmes d'exploration de graphes (voir Section IV , page 33).

Théorème

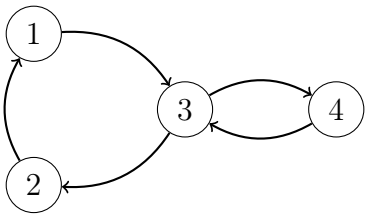
Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe connexe non orienté.
Les 4 affirmations suivantes sont équivalentes :

1. G est un arbre,
2. $\text{card}(\mathcal{S}) - \text{card}(\mathcal{A}) = 1$,
3. Entre deux sommets quelconques du graphe, il existe une unique chaîne simple,
4. La suppression de n'importe quelle arête de $\mathcal{A}$ rend le graphe non connexe.

Démonstration

On souhaite démontrer l'implication logique $(i \implies j)$ pour chaque arc (i, j) du graphe orienté P ci-dessous, dont les sommets représentent les 4 affirmations.

Le graphe P



- Démontrons par contraposition que $1 \Rightarrow 3$:
  Soient G un graphe connexe.
  Soit P l'ensemble des paires de sommets de G reliés par au moins deux chaînes simples distinctes.
  Supposons que P est un ensemble fini et non vide.
  Soit $\{x, y\}$ une paire de sommets de P dont la distance est minimale, c'est à dire :

  $$\forall \{s, t\} \in P, d(x, y) \leqslant d(s, t)$$

  Soit $\mathcal{C}$ l'ensemble des chaînes élémentaires reliant x et y. $\mathcal{C}$ contient au moins deux éléments distincts $c_n = (x, s_1, ..., s_n, y)$ et $c_m = (x, s'_1, ..., s'_m, y)$ de longueurs minimales avec $0 \leqslant n \leqslant m$.
  c_n et c_m n'ont que les sommets x et y en commun, sinon, $\{x, y\}$ ne serait pas la paire de sommet de P de distance minimale, et par conséquent la chaîne $(x, s_1, ..., s_n, y, s'_m, ..., s'_1, x)$ est un cycle simple, donc G n'est pas un arbre.
- Démontrons $3 \Rightarrow 4$
  Soit G un graphe connexe tel qu'entre deux sommet quelconque de G il existe une unique chaîne simple.
  En considérant l'arrête $\{x, y\}$ de G, x et y n'étant reliés que par une unique chaîne simple, le seul moyen d'aller de x à y est la chaîne composé de l'arête $\{x, y\}$. Donc si

l'on supprime cette arête, il n'y a plus aucune chaîne permettant de relié x et y et par conséquent le graphe ne sera plus connexe.

- On démontre $4 \Rightarrow 3$ par contraposition, c'est à dire que s'il existe deux sommets d'un graphe connexe $G(\mathcal{S}, \mathcal{A})$ reliés par au moins deux chaînes distinctes, alors il existe une arête de $\mathcal{A}$ telle que sa suppression ne rendra pas le graphe non connexe.
  Soit D l'ensemble des paires de sommets de G reliés par au moins deux chaînes simples distinctes et supposons que D soit non vide.
  D étant non vide et fini, il existe une paire de sommets $\{x, y\}$ de D de distance minimale.
  Comme dans la démonstration $1 \Rightarrow 3$, on peut trouver deux chaînes simples c_1 et c_2 reliant x et y n'ayant aucun sommet en commun à part x et y.
  Soient v_1 et v_2 les deux voisins de y tels que les arêtes $\{v_1, y\} \in c_1$ et $\{v_2, y\} \in c_2$.
  G étant connexe, on peut relié x à tout autre sommet de G. Si l'on supprime l'arête $\{v_1, y\}$ alors on obtient un graphe partiel G' dans lequel x reste relié a tous les sommets de G', en effet il reste relié à y avec la chaîne c_2 et à v_1 via la chaîne c_1 donc G' est connexe. d'où la conclusion.
- Démontrons que $3 \Rightarrow 2$. c'est à dire que s'il existe une unique chaîne simple entre deux sommets quelconques d'un graphe $G = (\mathcal{S}, \mathcal{A})$, alors $\text{card}(\mathcal{S}) - \text{card}(\mathcal{A}) = 1$
  On fait une démonstration par récurrence sur l'ordre n du graphe $G_n = (\mathcal{S}_n, \mathcal{A}_n)$.
  - **initialisation**
    Si $\text{card}(G) = 1$, alors G a un unique sommet et aucune arête, et par conséquent $\text{card}(\mathcal{S}) = \text{card}(\mathcal{A}) + 1$.
  - **Hérédité**
    Soit $n \in \mathbb{N}$ avec $n \geqslant 1$.
    Soit $k \in [\![1; n]\!]$ et supposons que tout graphe $G_k = (\mathcal{S}_k, \mathcal{A}_k)$ connexe, d'ordre k inférieur ou égale à n et tel que toutes paires de sommet est reliée par une unique chaîne simple vérifie l'affirmation suivante :
    $\text{card}(\mathcal{S}_k) = \text{card}(\mathcal{A}_k) + 1$.
    Soit G_{n+1} un graphe connexe d'ordre $n+1$ tel qu'entre deux sommets quelconques, il existe une unique chaîne simple.
    Si l'on supprime une arrête de G_{n+1}, l'on obtient d'après 4 un graphe partiel composé de deux graphes connexes $G' = (\mathcal{S}', \mathcal{A}')$ et $G'' = (\mathcal{S}'', \mathcal{A}'')$ disjoints et d'ordre strictement inférieur ou égale à n. Donc d'après l'hypothèse de récurrence, on a :

    $$\text{card}(\mathcal{S}') = \text{card}(\mathcal{A}') + 1 \quad \text{et} \quad \text{card}(\mathcal{S}'') = \text{card}(\mathcal{A}'') + 1$$

    or $\text{card}(\mathcal{S}_{n+1}) = \text{card}(\mathcal{S}') + \text{card}(\mathcal{S}'')$ et $\text{card}(\mathcal{A}_{n+1}) = \text{card}(\mathcal{A}') + \text{card}(\mathcal{A}'') + 1$
    donc $\boxed{\text{card}(\mathcal{S}_{n+1}) = \text{card}(\mathcal{A}_{n+1}) + 1}$.
    d'où l'hérédité.
  - **conclusion**
    Pour tout graphe connexe d'ordre $n \geqslant 1$ tel que chaque paire de sommet est reliée par une unique chaîne simple, on a :

    $$\text{card}(\mathcal{S}_n) = \text{card}(\mathcal{A}_n) + 1$$

- Démontrons $2 \Rightarrow 1$ par contraposition.
  Supposons que $G = (\mathcal{S}, \mathcal{A})$ est un graphe connexe qui n'est pas un arbre. D'après le théorème de la section 4 , page 18, G admet au moins un arbre couvrant, c'est à dire un graphe partiel $G' = (\mathcal{S}, \mathcal{A}')$ qui est un arbre, on a donc $\text{card}(\mathcal{S}) = \text{card}(\mathcal{A}') + 1$ et

$\text{card}(\mathcal{A}) > \text{card}(\mathcal{A}')$, d'où $\text{card}(\mathcal{S}) > \text{card}(\mathcal{A})+1$, et par conséquent $\text{card}(\mathcal{S}) \neq \text{card}(\mathcal{A})+1$.
Donc si $\text{card}(\mathcal{S}) = \text{card}(\mathcal{A})+1$, alors G est un arbre

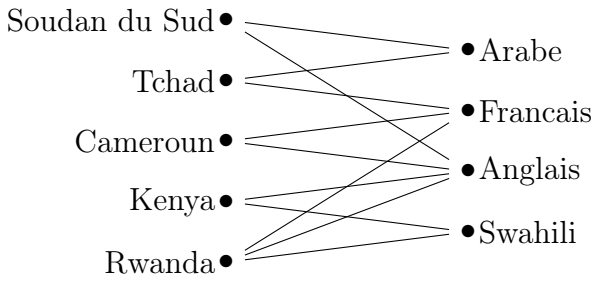
5 Graphes bipartis

Définition

Un graphe $G = (\mathcal{S}, \mathcal{A})$ est **biparti** s'il est non orienté et que l'on peut diviser l'ensemble des sommets $\mathcal{S}$ en deux parties disjointes, $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1$, de telle sorte que toute arête du graphe relie un sommet de $\mathcal{S}_0$ à un sommet de $\mathcal{S}_1$.

Exemple

Une relation entre deux ensembles $\mathcal{S}_0$ et $\mathcal{S}_1$ peut être représentée par un graphe biparti. Dans l'exemple ci-dessous, on obtient un graphe biparti en reliant un ensemble de pays ($\mathcal{S}_0$) à leurs langues officielles ($\mathcal{S}_1$) :



Théorème

Un graphe est biparti si et seulement s'il ne contient pas de cycle de longueur impaire.

Démonstration

Soit $G(\mathcal{S}, \mathcal{A})$ un graphe biparti d'ordre supérieur ou égale à deux.
Il existe donc deux sous ensembles $\mathcal{S}_0$ et $\mathcal{S}_1$ de $\mathcal{S}$ tel que $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1$ et que chaque arête relie un sommet de $\mathcal{S}_0$ à un sommet de $\mathcal{S}_1$
Soit $(x_0, x_1, ..., x_n)$ une chaîne et Φ un application de $\mathcal{S}$ dans $\{0, 1\}$ telle que :

$$\begin{cases} \Phi(x) = 0 & \text{si } x \in \mathcal{S}_0 \\ \Phi(x) = 1 & \text{si } x \in \mathcal{S}_1 \end{cases}$$

Donc pour tout $k \in [\![0, n-1]\!]$, on a $\{x_k, x_{k+1}\} \in \mathcal{A}$, d'où $\Phi(x_k) \neq \Phi(x_{k+1})$ et par conséquent pour tout $i \in \left[\!\left[0\ ,\ \left\lfloor \frac{n}{2} \right\rfloor\right]\!\right]$, $\Phi(x_{2i}) = \Phi(x_0)$ et $\Phi(x_{2i+1}) = \Phi(x_1)$.
Donc si $(x_0, x_1, ..., x_n)$ est un cycle, alors $x_n = x_0$ et il existe $k \in \mathbb{N}$ tel que $2k = n$ et par conséquent, c'est un cycle de longueur paire.
Réciproquement
Soit $G(\mathcal{S}, \mathcal{A})$ un graphe non orienté ne possédant pas de cycle de longueur impaire. Un graphe étant décomposable en réunion deux à deux disjointes de sous graphes connexes, on peut supposer que G est connexe.
Soit x_0 un sommet de G.

On note $d(x, y)$ la distance de x à y.
Soit Φ l'application de $\mathcal{S}$ dans $\{0, 1\}$ qui à tout élément x de $\mathcal{S}$ associe 0 si $d(x_0, x)$ est paire et 1 sinon.
Soit $\{x, y\} \in \mathcal{A}$. L'objectif est de montrer que $\Phi(x) \neq \Phi(y)$.
$d(x_0, y) \leqslant d(x_0, x) + d(x, y)$
$\leqslant d(x_0, x) + 1$ car $\{x, y\} \in \mathcal{A}$
Donc $d(x_0, y) - d(x_0, x) \leqslant 1$
de même, $d(x_0, x) - d(x_0, y) \leqslant 1$, d'où $|d(x_0, x) - d(x_0, y)| \leqslant 1$
D'autre part, si $d(x_0, x) = d(x_0, y)$, alors il existe un cycle de longueur $2d(x_0, x)+1$ contenant x_0 et l'arête $\{x, y\}$. Or G ne possède pas de cycle impair, donc $d(x_0, x) \neq d(x_0, y)$ et par conséquent $|d(x_0, x) - d(x_0, y)| = 1$, donc $d(x_0, x)$ et $d(x_0, y)$ ne sont pas de même parité ce qui implique que $\Phi(x) \neq \Phi(y)$.
En posant $\mathcal{S}_0 = \Phi^{-1}(\{0\})$ et $\mathcal{S}_1 = \Phi^{-1}(\{1\})$, on a bien obtenu une partition de $\mathcal{S}$ telle que pour chaque arête de $\mathcal{A}$ les deux extrémités ne soient pas dans le même ensemble.

Remarque

Si un graphe est un arbre, alors il n'a aucun cycle et par conséquent aucun cycle de longueur impaire, c'est donc un graphe biparti.
La réciproque est fausse. En effet un graphe biparti ayant un cycle de longueur paire ne sera pas un arbre.

IV Exercices

Exercice 1

Montrer que tout graphe simple à un nombre pair de sommets de degré impair.

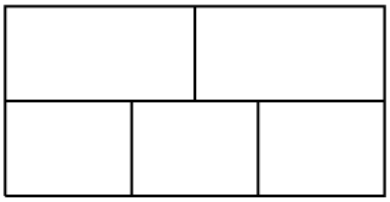
Exercice 2

Un programme prend en entrée une liste (x_1, x_2, x_3) de trois valeurs $x_i \in \{0, 1\}$, et renvoie en sortie une nouvelle liste $(x_1 + x_2, x_2 + x_3, x_3 + x_1)$ avec la convention que $1 + 1 = 0$ (calcul modulo 2). Par exemple, $(0, 0, 1)$ renvoie $(0, 1, 1)$. Représenter l'action du programme par un graphe, afin de répondre aux questions suivantes.

1. En partant d'une liste choisie aléatoirement, quelle est la probabilité d'obtenir le résultat $(1, 1, 0)$? Et le résultat $(0, 1, 0)$?
2. On part de $(0, 0, 1)$ et on fait tourner le programme en boucle un million de fois. Quel résultat obtient-on ?
3. On a fait tourner le programme un million de fois, et le résultat est $(1, 1, 0)$. De quelle suite est-on parti ?

Exercice 3

Est-il possible de tracer une courbe, sans lever le crayon, qui coupe chacun des 16 segments de la figure suivante une et une seule fois ?



Exercice 4

Montrer qu'un graphe simple non orienté ayant au moins deux sommets contient deux sommets de même degré.

Exercice 5

Montrer que dans un arbre (non réduit à un seul sommet), il existe toujours des sommets de degré 1.

Exercice 6

Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe connexe non orienté.
Montrer que entre deux sommets quelconques du graphe, il existe une unique chaîne simple si et seulement si la suppression de n'importe quelle arête de $\mathcal{A}$ rend le graphe non connexe.

Exercice 7

Montrer que dans un graphe, si la moyenne des degrés des sommets est supérieure ou égale à 2, alors il existe au moins un cycle.

Exercice 8

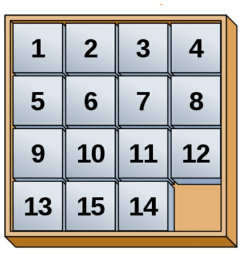
Soit G un arbre à n sommets et m arêtes avec $n \geqslant 2$.

1. Dessiner un exemple d'arbre avec $n = 6$ sommets.
2. Rappeler la relation qui existe entre n et m.
3. On suppose de plus que tous les sommets de G sont de degré inférieur ou égal à trois. Pour $k \in \{1; 2; 3\}$, on note alors n_k le nombre de sommets de degré k. En utilisant la réponse à la question précédente ainsi qu'une autre relation vue en cours, démontrer que $n_1 = n_3 + 2$.

Exercice 9

[Un problème à 25 dollars]

Le taquin est un jeu solitaire en forme de damier créé vers 1870 aux États-Unis. Il est composé de 15 petits carreaux numérotés de 1 à 15 qui glissent dans un cadre prévu pour 16. Peut-on remettre dans l'ordre les carreaux à partir de la configuration suivante, ou les numéros 14 et 15 ont été inversés ?



1. Montrer que le graphe du taquin, où chaque sommet représente une configuration possible, et chaque arête représente un mouvement élémentaire (glissage d'un carreau voisin sur la case vide), est biparti.
2. Montrer que le graphe des permutations, où chaque sommet représente une configuration possible, et chaque arête représente un échange entre deux cases quelconques, est lui aussi biparti.
3. Conclure.

Exercice 10

Montrer que dans un groupe formé de 6 personnes, il y a nécessairement trois qui se connaissent mutuellement ou trois qui ne se connaissent pas. (On suppose que si A connaît B, alors B connaît également A)
Montrer que cela n'est plus nécessairement vrai dans un groupe de 5 personnes.

Chapitre 3

Représentation des graphes

Remarque

Contrairement à un être humain, un ordinateur ne peut pas (encore) si facilement interpréter le schéma d'un graphe griffoné à la main sur un coin de table. Pour lui transmettre, et pour transformer les algorithmes en véritables programmes informatiques, il faut représenter la structure du graphe dans son langage...

On considère ici un graphe $G = (\mathcal{S}, \mathcal{A})$ dont les sommets sont numérotés de 1 à n et les arcs sont numérotés de 1 à m.

Il existe plusieurs manières classiques d'encoder G en machine. On en présente ici trois : par matrice d'incidence, par matrice d'adjacence et par listes d'adjacence.

I Matrice d'incidence

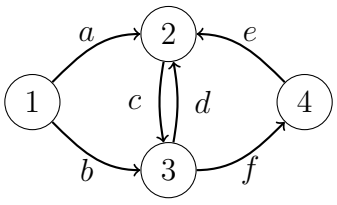
Définition

La matrice d'incidence d'un graphe G est la matrice de taille $n \times m$ dont les lignes sont indexées par les sommets, les colonnes sont indexées par les arcs, et l'entrée en ligne s et en colonne a vaut

- 1 si l'arc a sort de s,
- -1 si l'arc a entre en s,
- 0 sinon.

Dans le cas d'un graphe non orienté, on affiche simplement 1 si l'arête a est incidente à s.

Exemple



$$\begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & -1 & 0 \\ 0 & -1 & -1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \end{array}$$

Remarque

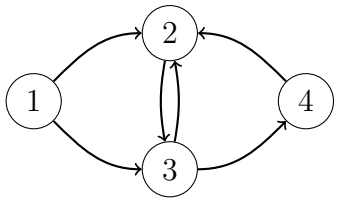
En termes d'efficacité et d'espace de stockage mémoire :
seulement $2m$ entrées de la matrice sont non nuls parmi les nm emplacements, ce qui est loin d'être optimal.
Cette représentation est donc assez peu utilisée, en général peu adaptée aux algorithmes. Un avantage néanmoins : elle peut se généraliser naturellement au cas des "multigraphes" ayant plusieurs arcs entre deux sommets.

II Matrice d'adjacence

Définition

La matrice d'adjacence d'un graphe G est la matrice carrée de taille $n \times n$ dont les lignes et les colonnes sont indexées par les sommets, et l'entrée en ligne s et en colonne t vaut
- 1 si (s, t) est un arc de G,
- 0 sinon.

Exemple



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

En python, cela donne :

```
1 G=[[0 ,1 ,1 ,0] ,[0 ,0 ,1 ,0] ,[0 ,1 ,0 ,1] ,[0 ,1 ,0 ,0]]
```

Remarque

Dans le cas des graphes non orientés, la matrice est symétrique, et on peut ne garder en mémoire que sa composante triangulaire supérieure.
Avec cette représentation, le test de l'existence d'un arc (s, t) est immédiat : il suffit de consulter l'entrée (s, t) de la matrice.
En revanche, pour obtenir la liste des successeurs (ou prédécesseurs) du sommet s, il faut parcourir toute la ligne s (ou toute la colonne s) de la matrice.
En terme d'espace mémoire, l'efficacité de la représentation par matrice d'adjacence dépend de la "densité" du graphe, c'est à dire du rapport entre le nombre d'arcs m et le nombre n^2 d'entrées de la matrice : en effet, seulement m entrées de la matrice sont non nulles parmi les n^2 entrées. Plus G est dense, plus cette représentation est efficace.

Au-delà de ces considérations, la représentation par matrice d'adjacence possède certains avantages :

1. Elle peut très bien se généraliser au cas des graphes "valués" (i.e. si certains coefficients appelés valuations sont affectés aux arcs du graphe), en substituant le 1 en position (s, t)

signalant la présence d'un arc, par la valuation de cet arc.

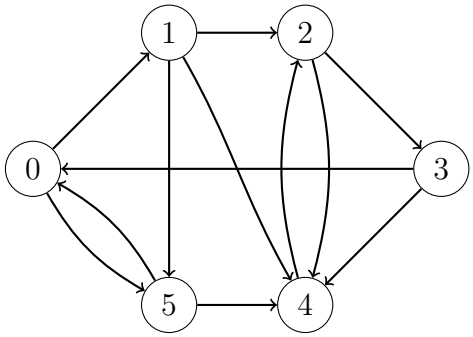
2. Elle possède en outre une certaine pertinence mathématique en tant que matrice (opérateur linéaire). En effet, la puissance L de la matrice donne le nombre de chemins de longueur L entre chaque paire de sommets.

III Listes d'adjacence

Remarque

Un graphe peut être représenté par des listes. Nous proposons ici une possibilité, mais de nombreuses autres peuvent convenir. On définit tout d'abord une liste des sommets, et à chaque sommet s on associe une liste chaînée : la liste $\Gamma^+(s)$ des successeurs de s la liste $\Gamma^-(s)$ des prédécesseurs de s.

Exemple



$\Gamma^-(s)$		s		$\Gamma^+(s)$
$3,5$	$\leftarrow$	0	$\rightarrow$	$1,5$
0	$\leftarrow$	1	$\rightarrow$	$2,4,5$
$1,4$	$\leftarrow$	2	$\rightarrow$	$3,4$
2	$\leftarrow$	3	$\rightarrow$	$0,4$
$1,2,3,5$	$\leftarrow$	4	$\rightarrow$	2
$0,1$	$\leftarrow$	5	$\rightarrow$	$0,4$

```
1 Gplus = [[1 ,5] ,[2 ,4 ,5] ,[3 ,4] ,[0 ,4] ,[2] ,[0 ,4]]
2 Gmoins = [[3 ,5] ,[0] ,[1 ,4] ,[2] ,[1 ,2 ,3 ,5] ,[0 ,1]]
```

Remarque

La seule donnée des successeurs de chaque sommet suffirait à reconstituer la structure du graphe. Mais on préfère parfois ajouter dans cette représentation la liste des prédécesseurs, afin de pouvoir facilement parcourir le graphe dans le sens inverse...
Cette représentation est nettement plus efficace que les précédentes au niveau de l'espace mémoire. Elle est très bien adaptée au parcours du graphe, aussi bien dans le sens direct qu'indirect. En revanche le test de l'existence d'un arc (s,t) n'est pas immédiat, puisqu'il faut pour cela parcourir toute la liste des successeurs de s. (Remarquons que ces caractéristiques sont symétriquement opposées à celles de la matrice d'adjacence.)
Notons aussi que cette structure de listes est "souple" : l'ajout ou la suppression d'arcs ou de sommets est plus aisée qu'avec les représentations matricielles.
Enfin, si le graphe est valué, on peut stocker dans les listes d'adjacence, en plus du numéro de sommet, la valuation de l'arc correspondant.
On peut aussi utiliser un dictionnaire comme dans l'exemple ci-dessous où les clés sont les sommets et les valeurs correspondantes aux clés sont les listes des successeurs. Le test de l'existence d'un arc (s,t) sera alors facilité et il ne sera pas nécessaire de parcourir toute la liste.
le dictionnaire associé au graphe précédent est :

```
1 Gplus={0 : [1,5], 1 : [2,4,5], 2 : [3,4], 3 : [0,4], 4 : [2], 5 : [0,4]}
```

Vous trouverez dans les annexes un programme permettant d'obtenir la liste d'adjacence des successeurs à partir de la liste d'adjacence des prédécesseurs et inversement.
(voir section **??**, page **??**.)

IV Exercices

Exercice 11

Chapitre 4

Exploration de graphes

I Principe général

Définition

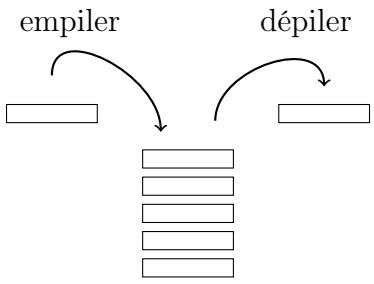
Explorer un graphe, c'est visiter et marquer tous les sommets d'un graphe qui sont accessibles à partir d'un sommet source s. Dans le cas d'un graphe non orienté, on obtient la composante connexe de s. Dans le cas orienté, on obtient l'ensemble $\widehat{\Gamma}^+(s)$ des descendants de s.

Remarque

L'exploration des sommets d'un graphe se fait comme on explore un nouveau territoire. Il y a la zone déjà explorée, la zone inexplorée, et, au confins de la zone explorée, une liste de sommets aux "avant-postes" : c'est là que se déroule l'action... À chaque étape, on sélectionne un des sommets x de la liste, on le retire de la liste, on cherche ses successeurs pas encore marqués, on les marque, et on les ajoute à la liste. Cette procédure se décline en deux types d'algorithmes, suivant que l'on gère la liste des sommets aux avant-postes comme une **file** ou comme une **pile**.

II Exploration en profondeur

Dans cet algorithme, les sommets aux avant-postes sont stockés dans une pile P. Comme dans le cas d'une pile d'assiettes, le dernier sommet empilé sera le seul immédiatement accessible, et donc le premier à être dépilé.



Cette exploration consiste donc à choisir un sommet source, s_0 depuis lequel on souhaite explorer le graphe. On passe ensuite à un de ses voisins, puis à un voisin de ce voisin et ainsi de suite. S'il

n'y a pas de voisin, on revient au sommet précédent et passe à un autre de ses voisins non encore visité.

L'algorithme de cette exploration est le suivant :

Algorithme 1 : Exploration en profondeur.

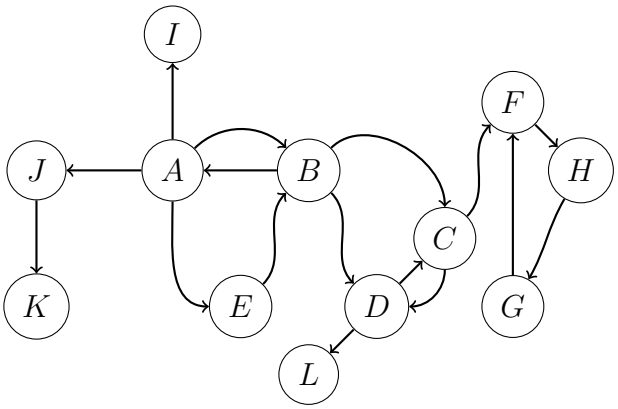
$P \leftarrow \{s\}$;
tant que $P \neq \emptyset$ **faire**
  dépiler de P le dernier sommet x;
  marquer x;
  pour chaque y *successeur de* x *non marqué et non dans* Q **faire**
    empiler y dans P;
  fin
fin

Remarque

Dans le contexte de recherches généalogiques, une exploration en profondeur reviendrait à remonter le plus directement possible dans le temps en sélectionnant un seul parent à chaque génération, puis à reculer vers la branche la plus proche lorsqu'on se retrouve dans une impasse.

Exemple

On considère le graphe suivant :



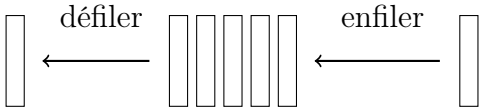
Lorsque l'on applique le parcours en profondeur à partir du sommet A au graphe précédent, on obtient :

P	marqué	x	action
[]	[]		Le sommet A est mis dans P
$[A]$	[]		On dépile A et on le marque.
[]	$[A]$	A	On empile les successeurs de A non marqué, c'est à dire B, E I et J
$[B, E, I, J]$	$[A]$	A	On dépile le dernier sommet de P et on le marque.
$[B, E, I]$	$[A, J]$	J	On empile les successeurs de J non marqué, c'est à dire K
$[B, E, I, K]$	$[A, J]$	J	On dépile le dernier sommet de P et on le marque.
$[B, E, I]$	$[A, J, K]$	K	K n'a aucun successeur, donc on dépile le dernier sommet de P et on le marque.
$[B, E]$	$[A, J, K, I]$	I	I n'a aucun successeur, donc on dépile le dernier sommet de P et on le marque.
$[B]$	$[A, J, K, I, E]$	E	E n'a aucun successeur n'appartenant pas a Q, donc on dépile B et on le marque.
[]	$[A, J, K, I, E, B]$	B	On empile les successeurs de B non marqué.
$[C, D]$	$[A, J, K, I, E, B]$	B	On dépile le dernier sommet de P et on le marque.
$[C]$	$[A, J, K, I, E, B, D]$	D	On empile les successeurs de D non marqué.
$[C, L]$	$[A, J, K, I, E, B, D]$	D	On dépile le dernier sommet de P et on le marque.
$[C]$	$[A, J, K, I, E, B, D, L]$	L	...

Via cet algorithme, on retrouve $\widehat{\Gamma}^+(A) = \{A, J, K, I, E, B, D, L, C, F, H, G\}$
En appliquant cet algorithme au sommet C, on obtient $\widehat{\Gamma}^+(C) = \{C, F, H, G, D, L\}$.

III Exploration en largeur

Remarque
Le parcours en profondeur permet de déterminer l'existence d'un chemin entre deux sommets, mais ce chemin n'est pas nécessairement le plus court. il ne donne donc pas toujours la distance entre deux sommets, c'est à dire la longueur minimale d'un chemin entre deux sommets. si l'on veut déterminer cette distance, il faut utiliser un autre algorithme, à savoir le parcours en largeur.
Dans cet algorithme, les sommets aux avant-postes sont stockés dans une file Q. Comme dans le cas d'une file d'attente, le premier sommet à en sortir sera le plus ancien, c'est à dire le premier à être entré dans la file.



L'exploration en largeur est donné par l'algorithme suivant :

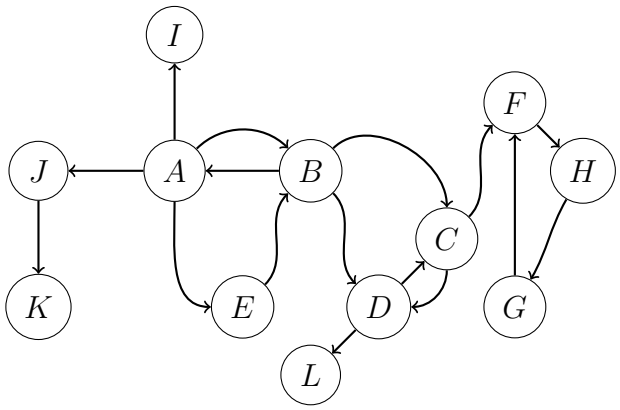
Remarque
Ce qui diffère du parcours en profondeur, c'est l'ordre dans lequel les sommets sont visités.

Algorithme 2 : Exploration en largeur.

$Q \leftarrow \{s\}$;
marquer s;
tant que $Q \neq \emptyset$ **faire**
    défiler de Q le plus ancien sommet x;
    pour chaque y *successeur de* x *non marqué* **faire**
        marquer y;
        enfiler y dans Q;
    fin
fin

Exemple

On considère le graphe suivant :



Lorsque l'on applique le parcours en largeur à partir du sommet A au graphe précédent, on obtient :

Q	marqué	x	action
[]	[]		Le sommet A est mis dans Q
$[A]$	[]		On marque le sommet A
$[A]$	$[A]$		On défile A
[]	$[A]$	A	On marque le premier successeur de A qui est B et on l'en dans Q
$[B]$	$[A, B]$	A	On marque le second successeur de A qui est E et on l'en dans Q
$[B, E]$	$[A, B, E]$	A	On marque le troisième successeur de A qui est I et on l'en dans Q
$[B, E, I]$	$[A, B, E, I]$	A	On marque le dernier successeur de A qui est J et on l'en dans Q
$[B, E, I, J]$	$[A, B, E, I, J]$	A	On défile le premier élément de Q, c'est à dire B.
$[E, I, J]$	$[A, B, E, I, J]$	B	On marque le premier successeur de B non marqué qui est et on l'enfile dans Q
$[E, I, J, C]$	$[A, B, E, I, J, C]$	B	On marque le second successeur de B non marqué qui est et on l'enfile dans Q
$[E, I, J, C, D]$	$[A, B, E, I, J, C, D]$	B	On défile le premier élément de Q, c'est à dire E
$[I, J, C, D]$	$[A, B, E, I, J, C, D]$	E	Tous les successeurs de E sont déjà marqué, donc on défile premier élément de Q, c'est à dire I.
$[J, C, D]$	$[A, B, E, I, J, C, D]$	I	I n'a aucun successeur, donc on défile le premier élément Q, c'est à dire J.
$[C, D]$	$[A, B, E, I, J, C, D]$	J	On marque le seul successeur de J non marqué qui est K on l'enfile dans Q
$[C, D, K]$	$[A, B, E, I, J, C, D, K]$	J	...

Via cet algorithme, on retrouve $\widehat{\Gamma}^+(A) = \{A, B, E, I, J, C, D, K, F, L, H, G\}$
En appliquant cet algorithme au sommet C, on obtient $\widehat{\Gamma}^+(C) = \{C, D, F, L, H, G\}$.

Exemple

Dans le contexte de recherches généalogiques, une exploration en largeur reviendrait à trouver tous les ancêtres d'une génération avant de remonter à la génération précédente.

IV Arborescence couvrante associée à une exploration

Définition

Une **arborescence** est un graphe orienté comportant un sommet particulier s, nommé **racine** de l'arborescence, à partir duquel il existe un chemin unique vers chacun des autres sommets. Autrement dit, il s'agit d'un arbre muni du choix d'un sommet particulier s, et d'une orientation obtenue en parcourant l'arbre à partir de s.

Théorème

Dans une arborescence, tout sommet y (différent de la racine s) admet un unique prédécesseur x.

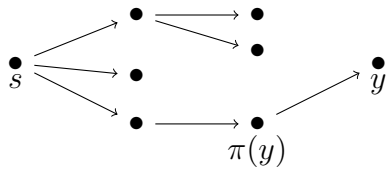
Démonstration

Puisqu'il existe un chemin de s à y, le sommet y admet au moins un prédécesseur. Supposons que y possède plusieurs prédécesseurs x_1 et x_2. Alors, puisqu'il existe un chemin de s à x_1 et un chemin de s à x_2, il existerait plusieurs chemins de s à y (l'un passant par x_1 et l'autre par x_2), ce qui est contradictoire avec la définition d'une arborescence. Donc y admet un unique prédécesseur.

Une arborescence $G = (\mathcal{S}, \mathcal{A})$ peut donc être encodée par une fonction "prédécesseur"

$$\pi : X \to X,$$

telle que $\pi(y)$ désigne l'unique prédécesseur de y. Par convention, puisque la racine n'a pas de prédécesseur, on peut poser $\pi(s) = s$.



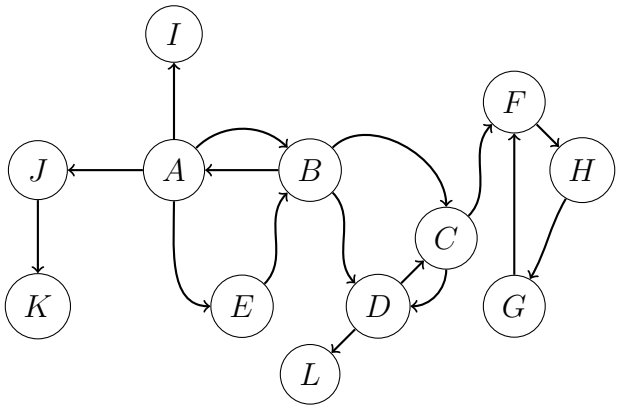
Définition

Dans un graphe G, une **arborescence couvrante** est un graphe partiel de G qui est une arborescence.

Remarque

L'exécution d'un algorithme d'exploration à partir de la source s donne une arborescence couvrante de $\widehat{\Gamma}^+(s)$ encodée par la fonction π. Il suffit de compléter l'algorithme de base avec $\pi(y) \leftarrow x$ pour chaque sommet y exploré à partir de x, comme ci-dessous.

Soit G le graphe représenté ci-dessous.



Algorithme 3 : Fonction π associée à une exploration.

```
P ← {s};
marquer s;
tant que P ≠ ∅ faire
    défiler de P le premier sommet x;
    pour chaque y ∈ Γ⁺(x) non marqué faire
        marquer y;
        π(y) ← x;
        empiler y dans P;
    fin
fin
```

Remarque

- Si l'on oublie l'orientation, on obtient un arbre couvrant.
- L'avantage d'avoir utiliser l'exploration en largeur pour cette arborescence, est qu'en plus de donner une arborescence couvrante du sommet x, on obtient les plus courts chemins entre x et ses descendants.

V Recherche de la composante fortement connexe

On rappelle que la composante fortement connexe d'un sommet s est

$$CFC(s) := \widehat{\Gamma}^+(s) \cap \widehat{\Gamma}^-(s).$$

Pour marquer tous les sommets de $CFC(s)$, il suffit donc de faire deux explorations à partir de s, une dans le sens direct, où l'on marque tous les sommets explorés avec un signe $+$, et l'autre avec l'orientation opposée (on remplace la liste des successeurs par la liste des prédécesseurs), où l'on marque tous les sommets explorés avec un signe $-$. Les sommets de $CFC(s)$ sont alors ceux qui sont marqués à la fois par $+$ et par $-$.
On pourra aussi faire l'intersection des deux listes des sommets marqués $\widehat{\Gamma}^+(s) \cap \widehat{\Gamma}^-(s)$.

Pour la deuxième exploration, on peut se restreindre au sous-graphe induit par les sommets marqués $\widehat{\Gamma}^+(s)$.
En effet, $CFC(s) \subset \widehat{\Gamma}^+(s)$.

Remarque

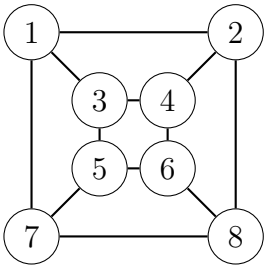
Pour obtenir la représentation d'un graphe avec l'orientation opposée,

- si le graphe est représenté par des listes d'adjacence, on échange les listes $\Gamma^+(s)$ avec $\Gamma^-(s)$ pour tout sommets s du graphe. Ceci peut être fait par un programme comme on l'a vu dans la partie sur les listes d'adjacence.
- si le graphe est représenté par une matrice d'incidence, on échange les coefficients $+1$ et -1,
- si le graphe est représenté par une matrice d'adjacence, on prend la transposée de la matrice (on échange le rôle des lignes et des colonnes).

VI Exercices

Exercice 12

Montrer que le graphe suivant est biparti.



Chapitre 5

Coloration de graphes

I Coloration, cliques et stables

Les problèmes de coloration se réfèrent généralement à la coloration des sommets d'un graphe et ils répondent généralement à des questions de gestion d'**attributions** en tenant compte de certaines **incompatibilités**. On retrouve donc des algorithmes de coloration dans la confection d'emploi du temps par exemples.

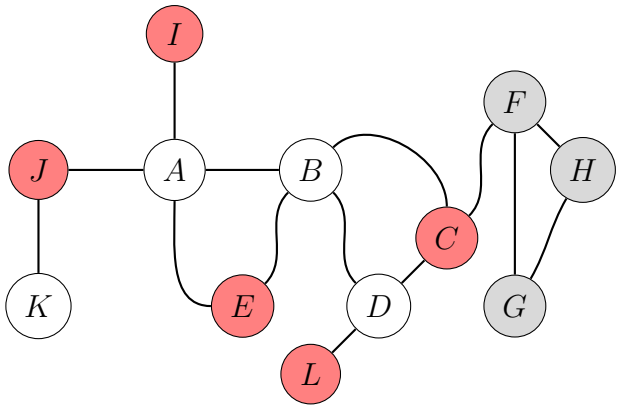
Exemples

- On cherche à transporter des produits chimiques en train. Certaines paires de produits risquent de provoquer des mélanges explosifs s'ils sont entreposés dans le même wagon. On cherche à minimiser le nombre de wagons.
  On peut associer un graphe à la situation : chaque type de produit est représenté par un sommet, et chaque mélange explosif par une arête joignant les deux produits concernés. La distribution des produits dans les wagons revient à attribuer une couleur à chaque sommet, en satisfaisant la contrainte formulée dans la définition ci-dessous :
- Considérons un lycée avec des cours de durée identique. Certain cours ne peuvent avoir lieu en même temps à cause d'incompatibilités (même enseignant, même classe, même salle de TP ...). On suppose que la semaine est divisée en plusieurs créneaux horaires correspondant au horaires d'ouvertures du lycée et on souhaite trouver un emploi du temps minimisant le nombre de créneaux horaires utilisés.
  Ce problème ce modélise par un graphe d'incompatibilité $G = (\mathcal{S}, \mathcal{A})$ où $\mathcal{S}$ désigne l'ensemble des cours à assurer et l'arrête $\{x, y\}$ relie les cours x et y s'ils sont incompatibles. Chaque créneaux utilisé sera représenté par une couleur.
  Si les cours sont de durée différentes, il existe des extensions du principe de coloration permettant de modéliser le problème.

Définition

- La **coloration** d'un graphe consiste à attribuer une couleur à chacun de ses sommets de telle manière que deux sommets reliés par une arête soient de couleurs différentes.
- Un **stable** est un ensemble de sommets dont le sous-graphe induit ne comporte pas d'arête.
- Une **clique** d'ordre n, au contraire, est un ensemble de n sommets dont le sous-graphe induit est un graphe complet.

Exemple



Dans le graphe ci-dessus, $\{C, E, I, J, L\}$ est un stable d'ordre 5 et $\{A, B, E\}$, $\{B, C, D\}$ et $\{F, G, H\}$ sont trois cliques d'ordre 3.

Voyons le lien entre la coloration d'un graphe, les cliques et les stables :
Si un graphe est colorié, alors l'ensemble des sommets d'une même couleur forme nécessairement un stable. Colorier un graphe avec k couleurs revient donc à partitionner l'ensemble de ses sommets $\mathcal{S}$ en une union disjointe de k stables $X = X_1 \cup X_2 \cup ... \cup X_k$.
Dans une clique, chaque sommet doit être colorié par une couleur différente des autres sommets. Le nombre de couleurs nécessaires pour colorier une clique est donc égal à l'ordre de la clique. Dans un graphe G, on note $\omega(G)$ l'ordre de la plus grosse clique.

Définition

Le **nombre chromatique** d'un graphe G est le nombre minimal de couleurs nécessaires pour colorier G. On le note $\chi(G)$.

D'après la remarque précédente, on a

Théorème

$\omega(G) \leqslant \chi(G)$.

Exemple

Dans le graphe de l'exemple, on a des cliques d'ordre 3, donc le nombre de couleurs nécessaires pour la coloration de ce graphe est supérieur ou égale à 3.

II Algorithmes de coloration

Le principe général des algorithmes de coloration de graphes est le suivant : on commence par ordonner les sommets du graphe (soit arbitrairement, soit en suivant une certaine règle), puis on colore chaque sommet dans l'ordre, avec la plus petite couleur possible dans $\{1, 2, 3, ...\}$.
Le nombre de couleurs utilisées au fil de l'exécution de l'algorithme dépendra de l'ordre des sommets. Il peut être minimal, c'est à dire égal à $\chi(G)$, si l'ordre est très bien choisi...

Notons que cet algorithme nous donne aussi une borne supérieure pour $\chi(G)$. En effet, à chaque étape, on doit colorier un sommet s en évitant les couleurs des sommets voisins déjà coloriés, c'est à dire, dans le pire des cas, en évitant $deg(s)$ couleurs. En posant

$$\Delta(G) := \max_{s \in X} \big(deg(s)\big),$$

on peut en déduire :

Théorème
$\chi(G) \leqslant \Delta(G) + 1$.

Toute la difficulté est donc d'ordonner les sommets de manière à minimiser le nombre de couleurs utilisées.
L'algorithme de Welsh-Powell propose d'ordonner les sommets par degré décroissant. L'algorithme DSATUR est encore plus efficace, car **adaptatif** : il considère, en plus de degré des sommets, leur "degré de saturation",

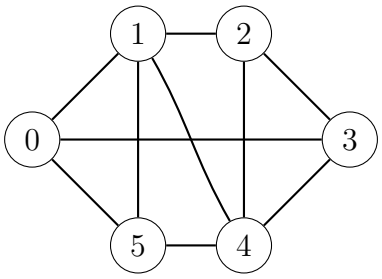
$$DSAT(x) := \text{ nombre de couleurs différentes dans les sommets voisins de } x,$$

qui évolue donc au fil de l'exécution de l'algorithme.

Algorithme 4 : Algorithme DSATUR.

pour chaque *sommet* s **faire**
  $DSAT(s) := 0$
fin
tant que *il existe un sommet non colorié* **faire**
  trouver un sommet non colorié s avec $DSAT(s)$ maximal (en cas d'égalité, choisir un sommet avec $deg(s)$ maximal);
  colorier s avec la plus petite couleur possible k;
  pour chaque x *voisin non colorié de* s **faire**
    actualiser $DSAT(x)$;
  fin
fin

On considère le graphe suivant :



Lorsque l'on applique l'algorithme de coloration au graphe précédent, on obtient :

DSAT	Couleur	x	action
$[0,0,0,0,0,0]$	$[0,0,0,0,0,0]$		Tous les sommet ont le même DSAT, on commence donc par le sommet ayant le plus grand degré, c'est à dire le sommet 1, 3 ou 4. On prend le sommet 1 et on lui attribut la couleur 1.
$[0,0,0,0,0,0]$	$[0,1,0,0,0,0]$	1	On actualise DSAT pour tout les voisins de 1 non coloriés.
$[1,0,1,0,1,1]$	$[0,1,0,0,0,0]$	1	Dans les sommets non coloriés, les sommets 0, 2, 4 et 5 ont la même valeur maximal de DSAT. On prend donc celui avec le plus grand degré, c'est à dire le sommet 4 et on lui attribut la couleur 2.
$[1,0,1,0,1,1]$	$[0,1,0,0,2,0]$	4	On actualise DSAT pour les voisins de 4 non coloriés, c'est à dire pour les sommets 2, 3 et 5.
$[1,0,2,1,1,2]$	$[0,1,0,0,2,0]$	4	Dans les sommets non coloriés ayant la plus grande valeur de DSAT, les sommets 2 et 5 ont le même degré, on choisi le sommet 2 et on le colorie avec la plus petite couleur possible, c'est à dire 3 (En effet, il a déjà des voisins avec les couleurs 1 et 2.
$[1,0,2,1,1,2]$	$[0,1,3,0,2,0]$	2	On actualise DSAT pour les voisins non coloriés du sommet 2.
$[1,0,2,2,1,2]$	$[0,1,3,0,2,0]$	2	Les sommets 3 et 5 ont la plus grande valeur de DSAT et ils ont le même degré. On choisit le sommet 3 et on le colorie avec la couleur 1 car ses voisins n'ont pas été colorié avec cette couleur.
$[1,0,2,2,1,2]$	$[0,1,3,1,2,0]$	3	On actualise DSAT pour les voisins de 3 non coloriés, c'est à dire le sommet 0.
$[2,0,2,2,1,2]$	$[0,1,3,1,2,0]$	3	Les sommets 0 et 5 ont la plus grande valeur de DSAT et ils ont le même degré. On choisit le sommet 0 et on le colorie avec la couleur 2 car au moins un voisin a été colorié en 1 mais aucun de ses voisins n'ont été colorié en 2.
$[2,0,2,2,1,2]$	$[2,1,3,1,2,0]$	0	On actualise DSAT pour le seul voisins de 0 non colorié, c'est à dire le sommet 5.
$[2,0,2,2,1,3]$	$[2,1,3,1,2,0]$	0	On récupère le sommet 5 et le colorie en 3, ses voisins ayant déjà été colorié en 1 et 2
$[2,0,2,2,1,3]$	$[2,1,3,1,2,3]$	5	

III Coloration des arêtes

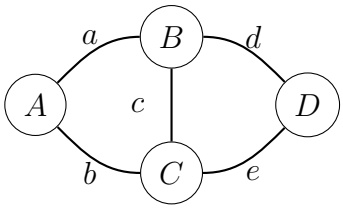
La coloration des arêtes d'un graphe consiste à affecter à chaque arête une couleur de telle sorte que deux arêtes de même couleur ne soient pas incidentes à un même sommet. Un exemple classique de problème de coloration des arêtes est celui de l'exercice 3 où l'on doit donner un calendrier des matches d'un tournois d'échecs où 6 joueurs s'affrontent, chaque sommet du graphe représentant un joueur et les arêtes représentant les matches.
Pour colorer les arêtes d'un graphe, on peut se ramener au problème de coloration des sommets. Il suffit pour cela de travailler sur le graphe adjoint définit comme suit :

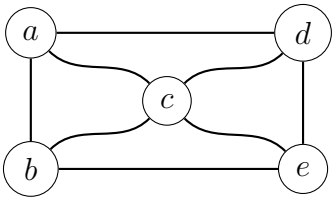
Définition

Soit $G = \{\mathcal{S}, \mathcal{A}\}$ un graphe non orienté.
On appelle graphe adjoint de G le graphe G' défini comme suit :

- Chaque sommet de G' correspond à une arête de G.
- Deux sommets de G' sont relié par une arête si les deux arêtes de G qu'ils représentent sont incidente à un même sommet de G.

Exemple





IV Graphes d'intervalles

Imaginons qu'on ait un ensemble de réunions, dont les heures de début et de fin sont fixés. Chaque réunion correspond donc à un intervalle de temps, et certains intervalles peuvent s'intersecter. Combien de salles seront nécessaires pour la tenue de toutes ces réunions ? La réponse est relativement simple : le nombre minimal de salles nécessaires et égal au nombre maximal de réunions qui ont lieu simultanément pendant un instant t. Appelons ce nombre k. Alors on peut attribuer à chaque réunion une des k salles, en "suivant le cours du temps", c'est à dire en classant les réunions par heure de début croissante. Dès qu'une nouvelle réunion commence, il y aura toujours au moins une salle libre...
On peut relier cette situation à une coloration de graphe.

Définition

Etant donnés un ensemble d'intervalles dans $\mathbb{R}$, le **graphe d'intervalles** associé est le graphe dont chaque sommet représente un des intervalles, et où deux sommets x et y sont reliés par une arête si et seulement si les intervalles correspondants I_x et I_y ont une intersection non vide.

Attribuer une salle pour chaque réunion revient à colorier le graphe d'intervalles.
Un ensemble de réunions qui ont lieu à un même instant t forme une clique dans le graphe.
En résumé, si un graphe G est un graphe d'intervalles, alors $\chi(G) = \omega(G)$, et on peut colorier G de manière optimale en "suivant le cours du temps"...
Deux problèmes se posent alors :

1. Comment reconnaître si un graphe G quelconque est un graphe d'intervalles ?
2. Et si G est bien un graphe d'intervalles, comment retrouver un ordre des sommets qui soit optimal pour la procédure de coloration (l'équivalent du cours du temps) ?

Définition

Un graphe **cordal** ou **triangulé** est un graphe qui possède la propriété : tous les cycles de longueur supérieure ou égale à quatre ont des **raccourcis**. Ce que l'on appelle raccourci est une arête qui relie deux sommets non successifs du cycle. Par exemple (s_1, s_3) et (s_2, s_4) sont des raccourcis dans le cycle s_1, s_2, s_3, s_4, s_1.

Théorème

Tout graphe d'intervalles est triangulé.

Démonstration

Ce théorème est vrai parce que "le temps n'est pas cyclique". Supposons qu'on ait un cycle s_1, s_2, s_3, s_4, s_1 sans raccourcis. Alors les intervalles I_1 et I_3 sont disjoints. Mais I_2 intersecte I_1 et I_3, et I_4 intersecte également I_1 et I_3. Il n'y a pas d'autres possibilités que de faire s'intersecter I_2 et I_4 (faire un dessin pour s'en convaincre).

Définition

Un sommet est **simplicial** si l'ensemble de ses voisins forme une clique.

Théorème

Dans un graphe d'intervalles, il existe au moins deux sommets simpliciaux.

Démonstration

Considérerons l'intervalle I qui finit en premier, et notons $T(I)$ sa date de fin. Alors tout intervalle qui intersecte I intersecte nécessairement $T(I)$. Donc tous les intervalles qui intersectent I s'intersectent entre eux. Autrement dit, I est simplicial. On peut produire le même raisonnement avec l'intervalle qui commence en dernier, en considérant sa date de début.

Définition

Un **schéma d'élimination parfait** est un ordre des sommets du graphe, $s_1, ..., s_n$, tel que pour tout $i \in \{1, ..., n\}$, s_i est simplicial dans le sous-graphe induit par $s_1, ..., s_i$.

Un schéma d'élimination parfait se construit donc en cherchant un sommet s_n simplicial dans le graphe, puis en le supprimant du graphe, en cherchant un sommet s_{n-1} simplicial dans le sous-graphe obtenu, en le supprimant, etc...

Théorème

Un schéma d'élimination parfait donne un ordre optimal pour appliquer l'algorithme de coloration du graphe.

Démonstration

On remarque qu'à chaque étape de la procédure de coloration, l'ensemble de sommets constitué par le nouveau sommet à colorier avec ses voisins déjà coloriés forme une clique. Donc la palette des couleurs nécessaires à la coloration ne dépasse jamais la taille de la plus grande clique.

Théorème

[Admis] Un graphe est triangulé si et seulement s'il admet un schéma d'élimination parfait.

Remarque

Les graphes triangulés sont une généralisation des graphes d'intervalles si l'on remplace la structure linéaire du temps par un temps arborescent. Ce sont des graphes d'intersection de sous-arbres dans un arbre où les intervalles sont représenté par des arêtes...

Remarque

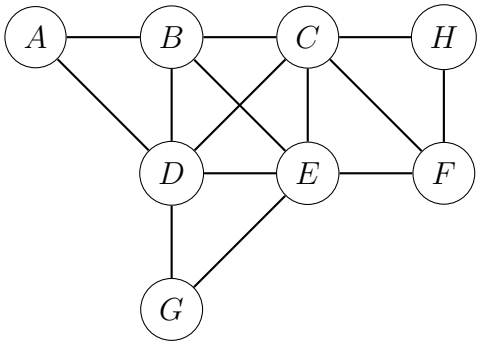
Le dernier théorème est la source de l'algorithme ci-dessous qui permet de déterminer si un graphe est triangulé ou non

Algorithme 5 : Reconnaissance d'un graphe triangulé de Fulkerson et Gross.

$G' \leftarrow G$;
tant que *il existe un sommet simplicial dans* G' **faire**
  retirer un sommet simplicial de G'
fin
si $G' == \emptyset$ **alors**
  G est triangulé
fin

Exemple

Déterminer à l'aide de l'algorithme de Fulkerson et Gross si le graphe ci-dessous est un graphe cordal.



V Exercices

Exercice 13

Soit G un graphe d'ordre 7 dont la matrice d'adjacence M est donnée ci-contre.
Ce graphe représente les 7 banc d'un parc relié entre eux par des allées permettant de passer de l'un à l'autre.

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

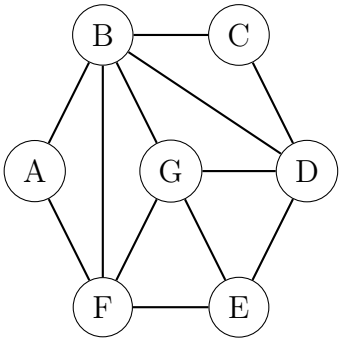
1. On veut peindre les bancs de manière que deux banc reliés par une allée soient de couleurs différentes. Donner un encadrement du nombre minimal de couleurs nécessaires.
2. Est-il possible de parcourir toutes les allées sans passer deux fois par la même allée ?
3. Est-il possible possible de parcourir toutes les allée en passant à coté de chaque banc une et une seule fois ?

Exercice 14

Un lycée doit organiser un examen composé de 7 épreuves à planifier, correspondant aux cours numéroté de 1 à 7, sachant que les paires de cours suivantes ont des étudiants en communs : $\{1;2\}$, $\{1;3\}$, $\{1;4\}$, $\{1;7\}$, $\{2;3\}$, $\{2;4\}$, $\{2;5\}$, $\{2;7\}$, $\{3;4\}$, $\{3;6\}$, $\{3;7\}$, $\{4;5\}$, $\{4;6\}$, $\{5;6\}$, $\{5;7\}$ et $\{6;7\}$.
Comment organiser ces épreuves sur une durée minimale ?

Exercice 15

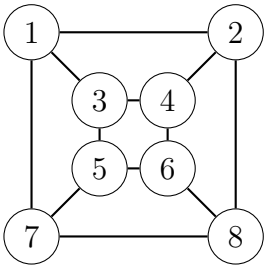
Appliquer l'algorithme DSATUR au graphe suivant.
Le nombre de couleur donné par l'algorithme DSATUR correspond-t-il au nombre chromatique du graphe ?



L'Algorithme DSATUR donne la coloration suivante :

Exercice 16

Montrer que le graphe suivant est biparti.



Exercice 17

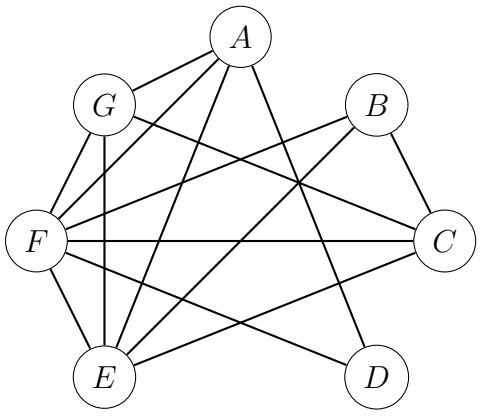
On veut transporter des produits chimiques noté A, B, C, D, E, F, G et H par le train. Le tableau ci-dessous indique donne les incompatibilités des produits (mélange explosif ...). En cas d'incompatibilité, les produit ne peuvent pas être entreposé dans le même wagon.

	A	B	C	D	E	F	G	H
A		X	X	X			X	X
B	X				X	X	X	
C	X			X		X	X	X
D	X		X		X			X
E		X		X		X	X	
F		X	X		X			
G	X	X	X		X			
H	X		X	X				

Quel est le nombre minimal de wagon nécessaire pour transporter ces produits ?

Exercice 18

Un concert de solidarité est organisé dans une grande salle de spectacle. A ce concert sont conviés sept artistes de renommée internationale : Luther Allunison (A), John Biaise (B), Phil Colline (C), Bob Ditlâne (D), Jimi Endisque (E), Robert Fripe (F) et Rory Garaguerre (G).
Les différents musiciens invités refusant de jouer avec certains autres, l'organisateur du concert doit prévoir plusieurs parties de spectacle. Les arêtes du graphe G ci-dessous indiquent quels sont les musiciens qui refusent de jouer entre eux.

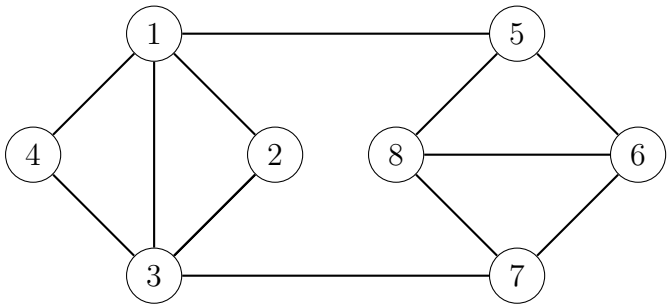


1. Déterminer la matrice associée au graphe G (les sommets de G étant classés dans l'ordre alphabétique).

2. Quelle est la nature du sous-graphe de G constitué des sommets A, E, F et G ? Que peut-on en déduire pour le nombre chromatique c(G) du graphe G ?
3. Quel est le sommet de plus haut degré de G ? En déduire un encadrement de c(G) .
4. Après avoir classé l'ensemble des sommets de G par ordre de degré décroissant, colorier le graphe G.
5. Combien de parties l'organisateur du concert doit-il prévoir ?
   Proposer une répartition des musiciens pour chacune de ces parties.

Exercice 19

Soit le graphe G représenté ci-dessous :



1. Donner un encadrement du nombre chromatique $\chi(G)$
2. Appliquer l'algorithme DSATUR pour colorer le graphe G.
3. Colorier le graphe avec seulement 3 couleurs.

L'algorithme DSATUR est un algorithme étant classé parmi les heuristiques il ne fournit pas forcement une solution optimale. DSATUR produit donc en temps polynomial une solution réalisable. Son auteur a montré qu'il était capable de fournir en un temps court (relativement aux autres heuristiques et aux méthodes exactes) une coloration optimale dans plus de 90 % des cas.

Exercice 20

Pendant une soirée au château de Moulinsart, au cours de laquelle plusieurs visiteurs se sont succédés à différents horaires, Bianca Castafiore s'est fait voler un précieux coffre à bijoux... Le voleur a subtilisé discrètement la clef de la chambre de la Castafiore (qui se trouvait dans une poche de son manteau posé au salon), il s'est absenté pour commettre le délit et il est revenu dans le salon pour remettre la clef à sa place. Tintin, qui tente de faire la lumière sur cette affaire, contacte le lendemain tous les invités, et tous lui affirment ne jamais avoir quitté le salon pendant la durée de leur visite. Il leur demande enfin qui ils avaient rencontré au salon, et note leurs réponses :

- Tournesol a vu Dupond, Dupont, Irma et Lampion,
- Dupond a vu Tournesol, Dupont, Haddock, Irma et Nestor,
- Dupont a vu Tournesol, Dupond et Haddock,
- Haddock a vu Dupond, Dupont et Irma,
- Irma a vu Tournesol, Dupond, Haddock et Nestor,
- Lampion a vu Tournesol et Nestor,
- Nestor a vu Dupond, Irma et Lampion.

Muni de ces informations, Tintin pourra-t-il démasquer le coupable ?

1. Quel algorithme peut-on utiliser pour savoir s'il y a un menteur dans le groupe.
2. Déterminer le coupable.

Exercice 21

Cet exercice est inspiré de la nouvelle de Claude Berge Qui a tué le Duc de Densmore (Bibliothèque Oulipienne , Réédition Castor Astral, 2000). Dans cette nouvelle policière, le lecteur peut découvrir le meurtrier grâce à un théorème combinatoire dû au mathématicien hongrois . Hajós.

Un jour, Sherlock Holmes recoit la visite de son ami Watson que l'on avait chargé d'enquêter sur un assassinat mystérieux datant de plus de trois ans.

A l'époque, le Duc de Densmore avait été tué par l'explosion d'une bombe, qui avait entièrement détruit le château de Densmore où il s'était retiré. Les journaux d'alors relataient que le testament, détruit lui aussi dans l'explosion, avait tout pour déplaire à l'une de ses sept ex-épouses. Or, avant de mourir, le Duc les avait toutes invitées à passer quelques jours dans sa retraite écossaise.

Holmes : Je me souviens de cette affaire ; ce qui est étrange, c'est que la bombe avait été fabriquée spécialement pour être cachée dans l'armure de la chambre à coucher, ce qui suppose que l'assassin a nécessairement effectué plusieurs visites au château !

Watson : Certes, et pour cette raison, j'ai interrogé chacune des femmes : Ann, Betty, Charlotte, Edith, Félicia, Georgia et Helen. Elles ont toutes juré qu'elles n'avaient été au château de Densmore qu'une seule fois dans leur vie.

Holmes : Hum ! Leur avez-vous demandé à quelle période elles ont eu leur séjour respectif ?

Watson : Hélas ! Aucune ne se rappelait les dates exactes, après plus de trois ans ! Néanmoins, je leur ai demandé qui elles avaient rencontré :

- Ann a rencontré Betty, Charlotte, Félicia et Georgia.
- Betty a rencontré Ann, Charlotte, Edith, Félicia et Helen.
- Charlotte a rencontré Ann, Betty et Edith.
- Edith a rencontré Betty, Charlotte et Félicia.
- Félicia a rencontré Ann, Betty, Edith et Helen.
- Georgia a rencontré Ann et Helen.
- Helen a rencontré Betty, Félicia et Georgia.

Vous voyez, mon cher Holmes, les réponses sont concordantes !

C'est alors que Holmes prit un crayon et dessina un étrange petit dessin, avec des points marqué A, B, C, E, F, G, H et des lignes reliant certains de ces points. Puis, en moins de trente secondes, Holmes déclara :

- Tiens, tiens ! Ce que vous venez de me dire détermine l'assassin. Qui est l'assassin ?

1. Quel algorithme peut-on utiliser pour savoir s'il y a un menteur dans le groupe.
2. Déterminer l'assassin.

Exercice 22

L'institut Pasteur possède des milliers de candidats vaccins en cours d'expérimentation, qui doivent tous être conservés dans différents intervalles de températures très basses $[v_i^{min}, v_i^{max}]$ (où i est le numéro du vaccin). Chaque congélateur dont dispose l'institut doit être réglé à une température T_k bien précise (où k est le numéro du congélateur), et pour économiser l'énergie, on cherche à conserver tous les vaccins en faisant fonctionner le moins de congélateurs possibles. Modéliser le problème, et proposer un algorithme pour planifier la conservation.

Exercice 23

On dispose de 7 euros stockés sous la forme de 3 types de jetons, de valeur 1, 2 et 5 euros. On a donc six configurations possibles

$$A = (1)(1)(1)(1)(1)(1)(1), \quad B = (2)(1)(1)(1)(1)(1), \quad etc...$$

Une machine permet de faire trois types d'échanges, dans les deux sens, symbolisés par les doubles flèches :

$$(1)(1) \leftrightarrow (2) \quad ; \quad (1)(1)(1)(1)(1) \leftrightarrow (5) \quad ; \quad (2)(2)(1) \leftrightarrow (5)$$

1. Compléter les six configurations ($C = ..., D = ..., E = ..., F = ...$) et dessiner le graphe de sommets A, B, C, D, E, F représentant toutes les transitions possibles (par un échange avec la machine).
2. Montrer que ce graphe est biparti.
3. En partant de la configuration A, est-il possible de revenir à A après 999 échanges ?
4. (BONUS) Même question en considérant que l'on dispose de N euros, avec $N \geq 5$.

Exercice 24

Un tournoi d'échecs oppose 6 personnes. Chaque joueurs doit affronter tous les autres.

1. Construisez un graphe représentant toutes les parties possibles.
2. Quel type de graphe obtenez vous ?
3. Proposez un calendrier des matches.

Exercice 25

Exprimer la résolution d'un Sudoku classique en terme de coloration de graphe. Décrivez le graphe (nombre de sommet, nombres d'arêtes, ...)
Combien faut-il de couleurs ?

Chapitre 6

Plus courts chemins

Étant donnée une carte routière de France, avec les distances de chaque portion de route, comment une application peut-elle déterminer la route la plus courte entre deux localisations données ?
Un parcours en largeur du graphe associé à la carte de France ne permettra pas de résoudre ce problème : il permettra de trouver l'itinéraire comportant le moins d'étapes (traversant le moins de sommets), mais cet itinéraire n'est pas nécessairement le plus court en nombre de kilomètres.
On s'intéresse maintenant à la résolution de ce type de problème.

I Graphes valués et plus courts chemins

Définition

Un **réseau** ou **graphe valué** est un graphe $G = (\mathcal{S}, \mathcal{A})$ muni d'une fonction de pondération $c : \mathcal{A} \to \mathbb{R}$ qui à chaque arête ou chaque arc (x, y) associe un réel $c\big((x, y)\big)$, qui peut être positif ou négatif. Ce réel peut s'appeler coût, poids, longueur ou valuation de l'arête ou de l'arc suivant les situations... On note $G = (\mathcal{S}, \mathcal{A}, c)$ un tel réseau.

Remarque

Dans la suite, on considérera que les réseaux sont orientés et simples. En cas de réseau non orienté, on peut comme d'habitude dédoubler les arêtes pour en faire un réseau orienté.

Définition

Dans un réseau, le **coût d'un chemin** est la somme du poids des arcs qui le composent, donc, si $p = (s_0, s_1, ..., s_k)$ est un chemin d'un réseau, le poids de ce chemin est le réel défini par :

$$c(p) = \sum_{i=1}^{k} c(s_{i-1}, s_i).$$

Définition

Un **plus court chemin** entre deux sommets fixés est un chemin qui minimise le coût. On note

$\delta(s,t)$ le plus petit coût des chemins de s à t.

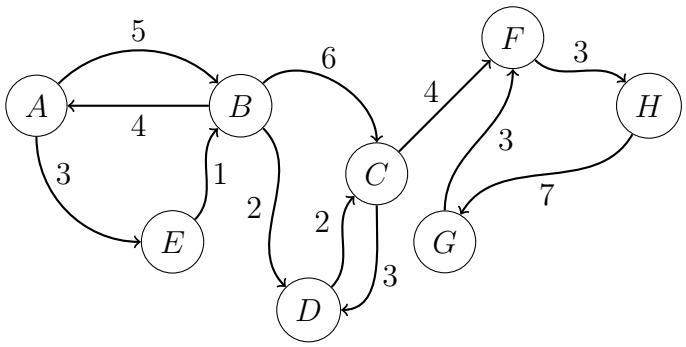
$$\delta(s,t) = \inf\{c(p) | p \text{ étant un chemin de } s \text{ à } t\}$$

Remarque

Il peut y avoir plusieurs chemins de même coût entre deux sommets fixés, donc un plus court chemin n'est pas nécessairement unique.

Exemple

Soit $G = (\mathcal{S}, \mathcal{A}, c)$ le réseau représenté ci-dessous.



Le coût du chemin $p = (A, B, C, D)$ est $c(p) = 5 + 6 + 3 = 14$.
Le plus court chemin entre les sommets A et D est le chemin (A, E, B, D) et son coût est $c((A, E, B, D)) = 3 + 1 + 2 = 6$ donc $\delta(A, D) = 6$

Théorème conditions d'existence d'un plus court chemin

Pour qu'un plus court chemin de s à t existe, il faut et il suffit que les deux conditions suivantes soient satisfaites :

1. il existe au moins un chemin de s à t (i.e. $t \in \widehat{\Gamma}^+(s)$),
2. aucun chemin de s à t ne comporte de circuit de coût négatif (on dit aussi **circuit absorbant**).

Remarque

En effet, si un chemin comporte un circuit de coût négatif, on peut diminuer d'autant que l'on veut le coût du chemin en parcourant plusieurs fois le circuit en boucle... L'ensemble des coûts des chemins entre s et t n'aura donc pas de minimum.
Une conséquence de ce théorème est qu'un plus court chemin peut toujours être choisi élémentaire. En effet, s'il passe deux fois par le même sommet, alors il comporte un circuit. Puisque ce circuit ne peut pas être de coût négatif, il est de coût positif ou nul. On obtient donc un chemin de coût moindre (ou égal) en supprimant le circuit.

Théorème principe de sous-optimalité

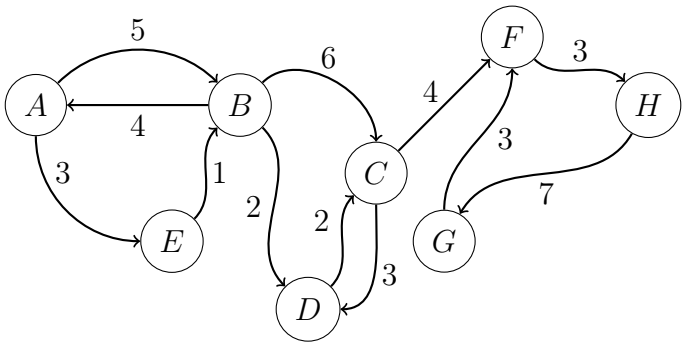
Si $(s_0, s_1, ..., s_k)$ est un plus court chemin, alors pour tout couple de sommets $\{s_i, s_j\}$ tel que $0 \leqslant i < j \leqslant k)$, le chemin $s_i, s_{i+1}, ..., s_j$ est un plus court chemin.

Démonstration

S'il existait un autre chemin, plus court, entre s_i et s_j, alors $s_0, s_1, ..., s_k$ ne serait pas le plus court chemin

II Implémentation des graphes valués

Soit $G = (\mathcal{S}, \mathcal{A}, c)$ le réseau représenté ci-dessous.



Dans le cas d'une représentation par liste d'adjacence, les sommets seront remplacés par un couple (sommets,coût) comme dans l'exemple suivant qui utilise un dictionnaire.

```
Gdico={"A" : {"B": 5,"E":3}, "B" : {"A":4,"C":6,"D":2}, "C" : {"D":3,"F":4}, "D" : {"
    C":2}, "E" : {"B":1}, "F" : {"H":3}, "G" : {"F":3}, "H" : {"G":7}}
```

L'implémentation du graphe valué à l'aide d'une matrice d'adjacence est fait à l'aide de la matrice suivante où l'absence d'arête est représenté par un coût infini comme ci-dessous :

$$\begin{pmatrix} 0 & 5 & +\infty & +\infty & 3 & +\infty & +\infty & +\infty \\ 4 & 0 & 6 & 2 & +\infty & +\infty & +\infty & +\infty \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & +\infty \\ +\infty & +\infty & 2 & 0 & +\infty & +\infty & +\infty & +\infty \\ +\infty & 1 & +\infty & +\infty & 0 & +\infty & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 0 \end{pmatrix}$$

En python, cela donne :

```
G=[[0,5,inf,inf,3,inf,inf,inf],
   [4,0,6,2,inf,inf,inf,inf],
   [inf,inf,0,3,inf,4,inf,inf],
   [inf,inf,2,0,inf,inf,inf,inf]
   [inf,1,inf,inf,0,inf,inf,inf],
   [inf,inf,inf,inf,inf,0,inf,3],
```

```
7       [inf, inf, inf, inf, inf, 3, 0, inf],
8       [inf, inf, inf, inf, inf, inf, 7, 0]]
```

III Plus courts chemins à origine unique

Remarque

Dans cette partie, on suppose qu'un sommet x_0 est fixé dans le réseau. L'objectif est de déterminer, pour chaque sommet x , le coût du plus court chemin de x_0 à x. On note ce coût $d(x) = \delta(x_0, x)$ (penser à d comme "distance"). On a $d(x_0) = 0$, puisque par convention le sommet x_0 est relié à lui-même par un chemin qui ne contient pas d'arc, de coût 0. S'il n'existe pas de chemin entre x_0 et x, on pose $d(x) = +\infty$.

1 Arborescence des plus courts chemins

En plus de son coût, on cherche aussi à déterminer quels sont les sommets présents sur un plus court chemin. En cas de non-unicité, on choisit pour chaque sommet x un plus court chemin de manière à respecter le principe de sous-optimalité, ce qui implique que l'ensemble des plus courts chemins d'origine s forme une **arborescence couvrante** de l'ensemble des descendants de s.
De même que dans la Section IV , cette arborescence est mémorisée par une fonction π telle que $\pi(y) = x$ si et seulement si (x, y) est un arc de l'arborescence.
Autrement dit, $\pi(y)$ désigne le sommet précédant y dans le plus court chemin de s à y.

Propriété

Par définition, on a :

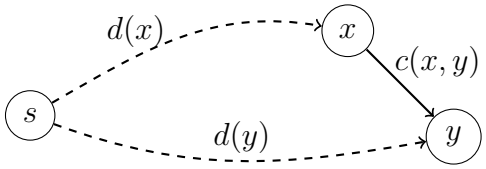
$$d(y) = d(\pi(y)) + c(\pi(y), y).$$

Remarque

De plus, pour tout arc $(x, y) \in \mathcal{A}$, on a :

$$d(y) \leq d(x) + c(x, y).$$

En effet, la somme $d(x) + c(x, y)$ correspond au coût d'un chemin de s à y qui passe par x. Puisque $d(y)$ est le coût du plus court chemin de s à y, il est nécessairement inférieur.



Les algorithmes que nous allons voir sont basé sur le principe de la détermination d'une arborescence couvrante comme vue précédemment et exploitent cette inégalité pour déterminer l'unique prédécesseur d'un sommet dans cette arborescence.
L'idée générale est de stocker les sommets y pour lesquels on peut encore diminuer leur distance

Algorithme 6 : Procédure : relâcher(x, y).

si $d(y) > d(x) + c(x, y)$ **alors**
  $d(y) \leftarrow d(x) + c(x, y)$;
  $\pi(y) \leftarrow x$;
fin

$d(y)$ dans une liste L, puis de diminuer progressivement les valeurs de $d(y)$ en examinant chaque arc (x, y) : si $d(y) > d(x) + c(x, y)$ alors on peut améliorer $d(y)$ en passant par x...

Appelons cette procédure "relâcher(x, y)". Elle prend en entrée : un arc (x, y) dans un réseau, le tableau L des bornes maximales des coûts (que l'on cherche à diminuer), et le tableau π qui représente l'arborescence des plus courts chemins. En sortie, elle retourne les tableaux L et π améliorés.

2 Coûts positifs : algorithme de Dijkstra

L'algorithme de Dijkstra est assez rapide pour trouver les plus courts chemins d'un graphe valué $G = (\mathcal{S}, \mathcal{A}, c)$, car il ne parcourt chaque arc du réseau qu'une seule fois.
Attention cependant, il n'est pas valable si certains arcs ont un coût négatif.
Dans cet algorithme, L représente la liste des sommets x pour lesquels on peut encore à priori diminuer la valeur de $d(x)$ et D représente la liste des distances minimales de s aux autres sommets de $\widehat{\Gamma}^+(s)$
On rajoute le sommet x de L ayant la plus petite distance au sommet initial s au tableau D des distances minimales, puis on relâche les successeurs de s n'appartenant pas à D.
On supprime ensuite le sommet x de L et on recommence.

Algorithme 7 : Algorithme de Dijkstra.

$L(s) \leftarrow 0$;
tant que $L \neq \emptyset$ **faire**
  trouver un sommet x de L tel que $d(x)$ soit minimale;
  $D(x) \leftarrow d(x)$;
  pour chaque $y \in \Gamma^+(x)$ *n'appartenant pas à* D **faire**
    relâcher (x,y)
  fin
  $L \leftarrow L - \{x\}$
fin

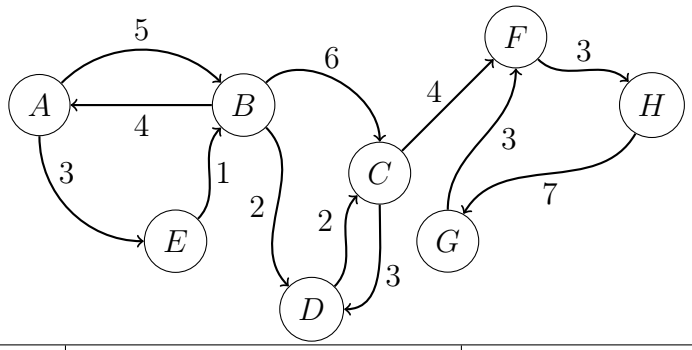
Démonstration 1. Lorsqu'on retire un sommet x de la liste L, la valeur de $d(x)$ ne peut plus être diminuée en passant par un chemin alternatif : on a alors bien déterminé le plus court chemin de s à x.
En effet, le premier sommet à sortir de L est s, pour lequel $d(s) = 0$ est bien la valeur correcte.
Supposons par récurrence que tous les sommets déjà sortis de L ont leurs plus courts chemins correctement fixés par les tableaux d et π. À chaque itération, on fait sortir de L un sommet x tel que $d(x)$ soit minimal parmi tous les $d(y),\ y \in L$.
Par hypothèse de récurrence, si $d(x)$ pouvait être amélioré, ce serait nécessairement en passant par un autre sommet $y \in L$, ce qui est impossible car $d(y) > d(x)$, et tout chemin de y à x étant de coût *positif*, la valeur de $d(x)$ ne pourrait qu'augmenter en passant par y.

Propriété

Si n désigne le nombre de sommets du réseau, la complexité de l'algorithme de Dijkstra est $\mathcal{O}(n^2)$. En effet, puisqu'à chaque étape de la boucle "tant que" on retire un sommet à la liste L, on aura de l'ordre de n étapes. De plus, dans chaque étape, on doit chercher le sommet avec la valeur de d minimale, ce qui a une complexité de l'ordre de n. Puisqu'on relâche une fois chaque arc, l'ensemble des opérations de relâchement aura une complexité de m, où m désigne le nombre d'arcs. Enfin, puisque $m \leq n^2$, la complexité globale sera $\mathcal{O}(n^2)$.

Exemple

Soit $G = (\mathcal{S}, \mathcal{A}, c)$ le réseau représenté ci-dessous.



L	D	action
		Le sommet de départ A est initialisé avec une distance nulle dans L
$(A, 0)$		Le sommet ayant la distance minimale de L est A, on le rajoute dans de D
$(A, 0)$	(A,0)	On relâche les successeurs de A n'appartenant pas à D
$(A, 0), (B, 5), (E, 3)$	(A,0)	On supprime A de L.
$(B, 5), (E, 3)$	(A,0)	Le sommet de L ayant la distance minimale est E, on le rajoute à D.
$(B, 5), (E, 3)$	(A,0),(E,3)	On relâche les successeurs de E n'appartenant pas à D
$(B, 4), (E, 3)$	(A,0),(E,3)	On supprime E de L
$(B, 4)$	(A,0),(E,3)	Le sommet de L ayant la distance minimale est B, on le rajoute à D.
$(B, 4)$	(A,0), (E,3), (B,4)	On relâche les successeurs de B n'appartenant pas à D
$(B, 4), (C, 10), (D, 6)$	(A,0), (E,3), (B,4)	On supprime B de L.
$(C, 10), (D, 6)$	(A,0), (E,3), (B,4)	Le sommet de L ayant la distance minimale est D, on le rajoute à D.
$(C, 10), (D, 6)$	(A,0), (E,3), (B,4), (D,6)	On relâche C.
$(C, 8), (D, 6)$	(A,0), (E,3), (B,4), (D,6)	On supprime D de L
$(C, 8)$	(A,0), (E,3), (B,4), (D,6)	On rajoute C à D
...	...	...

Le tableau suivant donne un bon résumé des différentes étapes en donnant le coût minimal des chemins de A aux autres sommets du graphe valué.

	A	B	C	D	E	F	G	H
étape 1	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
étape 2		A - 5	$+\infty$	$+\infty$	A - 3	$+\infty$	$+\infty$	$+\infty$
étape 3		E - 4	$+\infty$	$+\infty$		$+\infty$	$+\infty$	$+\infty$
étape 4			B - 10	B - 6		$+\infty$	$+\infty$	$+\infty$
étape 5			D - 8			$+\infty$	$+\infty$	$+\infty$
étape 6						C - 12	$+\infty$	$+\infty$
étape 7							$+\infty$	F - 15
étape 6							H - 22	

Ce dernier tableau donne le coût minimal des chemins de A aux autres sommets du graphe valué ainsi que l'unique prédécesseur du sommet dans l'arborescence des chemins de coûts minimaux. En partant de **G**, on a **H** qui est son prédécesseur, puis **F** est celui de H, **C** celui de F, **D** celui de C et ainsi de suite. On en déduit que le chemin de coût minimal de A à G est (A,E,B,D,C,F,H,G) et sont coût est de 24.

3 Coûts quelconques : algorithme de Bellman-Ford

L'algorithme de Bellman-Ford est valable même si certains arcs sont de coût négatif. De plus, il a l'avantage de détecter la présence de circuits négatifs dans le réseau (qui, on l'a vu, empêchent l'existence des plus courts chemins).
Dans cet algorithme, $n = \text{card}(\mathcal{S})$ désigne le nombre de sommets du réseau. On effectue $n-1$ fois de suite un relâchement de l'ensemble des arcs (dans un ordre quelconque).

Algorithme 8 : Bellman-Ford.

pour chaque *sommet* $x \in \mathcal{S}$ **faire**
  $d(x) \leftarrow +\infty$;
fin
$d(s) \leftarrow 0$;
pour chaque $k = 1, ..., n-1$ **faire**
  pour chaque $(x, y) \in \mathcal{A}$ **faire**
    relâcher (x, y) ;
  fin
fin
pour chaque $(x, y) \in \mathcal{A}$ **faire**
  si $d(y) > d(x) + c(x, y)$ **alors**
    afficher : "circuit absorbant" ;
  fin
fin

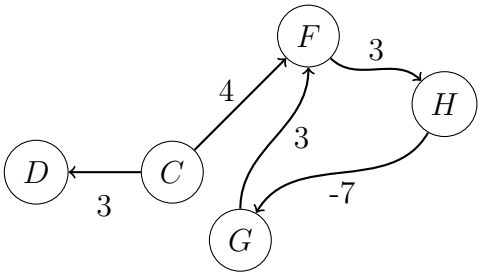
Démonstration

Quand on relâche k fois de suite l'ensemble des arcs, tous les plus courts chemins composés de k arcs ou moins sont détectés : cela se vérifie facilement par récurrence. Or, on a vu que les plus courts chemins, s'ils existent, sont élémentaires : ils ne passent pas plus d'une fois par chacun des

n sommets, et sont donc composés de $n-1$ arcs au maximum. S'il existe un circuit absorbant, les arcs du circuit peuvent être relâchés à l'infini. C'est ce qui est testé dans la dernière partie de l'algorithme.

Exemple

Soit $G = (\mathcal{S}, \mathcal{A}, c)$ le réseau représenté ci-dessous.



Le tableau suivant donne un bon résumé des différentes étapes.

		C	D	F	G	H
	Initialisation	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$k=1$	relâche (C,D)	0	C,3	$+\infty$	$+\infty$	$+\infty$
	relâche (C,F)	0	C,3	C,4	$+\infty$	$+\infty$
	relâche (G,F)	0	C,3	C,4	$+\infty$	$+\infty$
	relâche (F,H)	0	C,3	C,4	$+\infty$	F,7
	relâche (H,G)	0	C,3	C,4	H,0	F,7
$k=2$	relâche (C,D)	0	C,3	C,4	H,0	F,7
	relâche (C,F)	0	C,3	C,4	H,0	F,7
	relâche (G,F)	0	C,3	G,3	H,0	F,7
	relâche (F,H)	0	C,3	G,3	H,0	F,6
	relâche (H,G)	0	C,3	G,3	H,-1	F,6
$k=3$	relâche (C,D)	0	C,3	G,3	H,-1	F,6
	relâche (C,F)	0	C,3	G,3	H,-1	F,6
	relâche (G,F)	0	C,3	G,2	H,-1	F,6
	relâche (F,H)	0	C,3	G,2	H,-1	F,5
	relâche (H,G)	0	C,3	G,2	H,-2	F,5
$k=4$	relâche (C,D)	0	C,3	G,2	H,-2	F,5
	relâche (C,F)	0	C,3	G,2	H,-2	F,5
	relâche (G,F)	0	C,3	G,1	H,-2	F,5
	relâche (F,H)	0	C,3	G,1	H,-2	F,4
	relâche (H,G)	0	C,3	G,1	H,-3	F,4

Propriété

Puisque chacun des m arcs est relâché $n-1$ fois, la complexité est de l'ordre de $\mathcal{O}((n-1)m) = \mathcal{O}(nm)$ si le réseau est représenté par une liste d'adjacence. Si le réseau est représenté par une matrice d'adjacence, l'accès à tous les arcs nécessite de parcourir toute la matrice de taille $n \times n$, donc la complexité devient $\mathcal{O}(n^3)$.

IV Plus courts chemins entre toutes les paires de sommets

Dans cette partie, on suppose qu'on a un réseau sans circuit négatif, et que les sommets de ce réseau sont numérotés de 1 à n. L'objectif est de déterminer, pour toute paire de sommets i, j, le coût du plus court chemin de i à j. On note ce coût $d(i, j)$. Comme précédemment, on a toujours $d(i, i) = 0$, et s'il n'existe pas de chemin entre i et j, on pose $d(i, j) = +\infty$.
On peut penser à la fonction d comme une matrice carrée de taille $n \times n$:

$$D = \Big(d(i,j)\Big)_{1\leq i,j\leq n}.$$

Attention, en général cette matrice n'est pas symétrique : $d(i, j) \neq d(j, i)$, sauf si le réseau est non orienté...
En plus des coûts, on cherche aussi à déterminer quels sont les sommets présents sur les plus court chemin. Cette donnée va également être représentée par une matrice de taille $n \times n$:

$$\Pi = \Big(\pi(i,j)\Big)_{1\leq i,j\leq n},$$

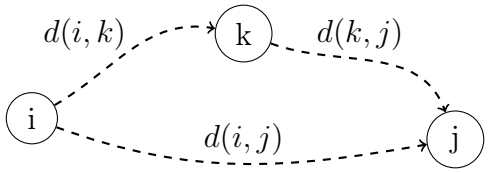
où $\pi(i, j)$ désigne le sommet précédant j dans le plus court chemin de i à j.

1 Inégalité triangulaire

Pour tous les triplets de sommets i, j, k, on a l'**inégalité triangulaire** :

$$d(i, k) + d(k, j) \geq d(i, j).$$

En effet, $d(i, j)$ étant le coût du plus court chemin de i à j, il est par définition inférieur ou égale à $d(i, k) + d(k, j)$ qui est le coût d'un chemin de i à j passant par k.



De manière similaire aux algorithmes vus précédemment, l'idée de Floyd-Warshall (section suivante) est de diminuer progressivement les valeurs de $d(i, j)$ en examinant chaque sommet k : si $d(i, j) > d(i, k) + d(k, j)$ alors on peut améliorer $d(i, j)$ en passant par k...

Algorithme 9 : Procédure : relâcher(i, k, j).

```
si d(i,j) > d(i,k) + d(k,j) alors
    d(i,j) ← d(i,k) + d(k,j);
    π(i,j) ← π(k,j);
fin
```

Appelons cette procédure "relâcher(i, k, j)". Elle prend en entrée : trois sommets i, k, j dans un réseau, la matrice D des bornes maximales des coûts (que l'on cherche à diminuer), et la matrice Π qui représente les "trajets" des plus courts chemins. En sortie, elle retourne les matrices D et Π améliorés.

2 Algorithme de Floyd-Warshall

Algorithme 10 : Floyd-Warshall.

pour chaque *sommets* i, j **faire**
  $d(i,j) \leftarrow +\infty$;
fin
pour chaque *sommet* i **faire**
  $d(i,i) \leftarrow 0$;
fin
pour chaque *arc* $(i,j) \in \mathcal{A}$ **faire**
  $d(i,j) \leftarrow c(i,j)$;
  $\pi(i,j) \leftarrow i$;
fin
pour chaque *sommet* $k = 1, ..., n$ **faire**
  pour chaque *sommet* $i = 1, ..., n$ **faire**
    pour chaque *sommet* $j = 1, ..., n$ **faire**
      relâcher (i, k, j) ;
    fin
  fin
fin

Propriété

Dans la boucle principale, on fait de l'ordre de n^3 opérations de relâchement (une pour chaque triplet i, k, j), donc la complexité est $\mathcal{O}(n^3)$.

Démonstration

Appelons $d_K(i,j)$ la valeur de $d(i,j)$ obtenue après avoir executé la boucle principale de l'algorithme jusqu'à l'étape $k = K$, autrement dit, après avoir fait relâcher(i,k,j) pour tous $i, j \leq n$ et pour tous $k \leq K$. On va montrer que :

Théorème

$d_K(i,j)$ correspond au coût du plus court chemin de i à j dont les sommets intermédiaires appartiennent l'ensemble $\{1, ..., K\}$.

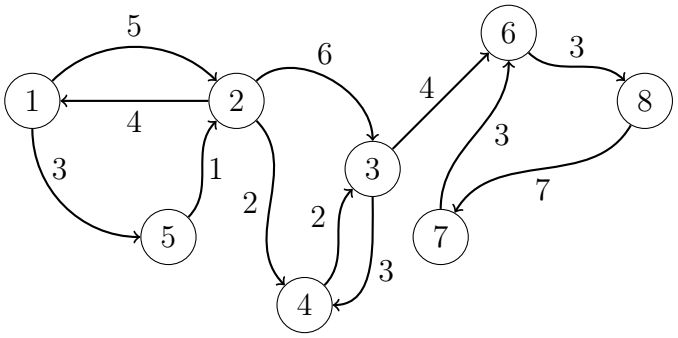
Pour abréger, notons $PCC_K(i,j)$ le plus court chemin de i à j dont les sommets intermédiaires appartiennent l'ensemble $\{1, ..., K\}$. On veut montrer que l'execution de l'algorithme jusqu'à l'étape $k = K$ détecte $PCC_K(i,j)$ pour toutes les paires de sommets i, j.
On vérifie que l'étape d'initialisation donne les coûts des plus courts chemins sans sommets intermédiaires (puisqu'un chemin sans sommet intermédiaire n'est autre qu'un arc du graphe).
Par récurrence, on suppose le théorème vrai jusqu'à l'étape $k = K - 1$.
On considère $PCC_K(i,j)$. On a deux possibilités : soit $PCC_K(i,j)$ passe par le sommet K, soit il ne passe pas par le sommet K. S'il ne passe pas par le sommet K, alors $PCC_K(i,j) = PCC_{K-1}(i,j)$. Dans ce cas, faire relâcher(i,K,j) ne change rien, et $d_K(i,j) = d_{K-1}(i,j)$. Si $PCC_K(i,j)$ passe par le sommet K, alors il n'y passe qu'une seule fois (puisque les plus courts chemins sont élémentaires). $PCC_K(i,j)$ est donc égal à $PCC_{K-1}(i,K)$ suivi de $PCC_{K-1}(K,j)$, et c'est bien ce que la procédure relâcher(i,K,j) va détecter.

Remarque

Pour exécuter l'algorithme à la main, on écrit les matrices $d(i,j)$ et $\pi(i,j)$ à l'initialisation, puis on les réécrit après chaque étape de la boucle $k = 1, ..., n$: on calcule successivement $d_1(i,j)$, $d_2(i,j)$, etc...

Exemple

Soit $G = (\mathcal{S}, \mathcal{A}, c)$ le réseau représenté ci-dessous.



$$D = \begin{pmatrix} 0 & 5 & +\infty & +\infty & 3 & +\infty & +\infty & +\infty \\ 4 & 0 & 6 & 2 & +\infty & +\infty & +\infty & +\infty \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & +\infty \\ +\infty & +\infty & 2 & 0 & +\infty & +\infty & +\infty & +\infty \\ +\infty & 1 & +\infty & +\infty & 0 & +\infty & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 1 & & & 1 & & & \\ 2 & & 2 & 2 & & & & \\ & & & 3 & & 3 & & \\ & & 4 & & & & & \\ & 5 & & & & & & \\ & & & & & & & 6 \\ & & & & & 7 & & \\ & & & & & & 8 & \end{pmatrix}$$

$$k = 1 \quad D = \begin{pmatrix} 0 & 5 & +\infty & +\infty & 3 & +\infty & +\infty & +\infty \\ 4 & 0 & 6 & 2 & 7 & +\infty & +\infty & +\infty \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & +\infty \\ +\infty & +\infty & 2 & 0 & +\infty & +\infty & +\infty & +\infty \\ +\infty & 1 & +\infty & +\infty & 0 & +\infty & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 1 & & & 1 & & & \\ 2 & & 2 & 2 & 1 & & & \\ & & & 3 & & 3 & & \\ & & 4 & & & & & \\ & 5 & & & & & & \\ & & & & & & & 6 \\ & & & & & 7 & & \\ & & & & & & 8 & \end{pmatrix}$$

$$k = 2 \quad D = \begin{pmatrix} 0 & 5 & 11 & 7 & 3 & +\infty & +\infty & +\infty \\ 4 & 0 & 6 & 2 & 7 & +\infty & +\infty & +\infty \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & +\infty \\ +\infty & +\infty & 2 & 0 & +\infty & +\infty & +\infty & +\infty \\ 5 & 1 & 7 & 3 & 0 & +\infty & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 1 & 2 & 2 & 1 & & & \\ 2 & & 2 & 2 & 1 & & & \\ & & & 3 & & 3 & & \\ & & 4 & & & & & \\ 2 & 5 & 2 & 2 & & & & \\ & & & & & & & 6 \\ & & & & & 7 & & \\ & & & & & & 8 & \end{pmatrix}$$

$$k=3 \quad D=\begin{pmatrix} 0 & 5 & 11 & 7 & 3 & 15 & +\infty & +\infty \\ 4 & 0 & 6 & 2 & 7 & +10 & +\infty & +\infty \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & +\infty \\ +\infty & +\infty & 2 & 0 & +\infty & 6 & +\infty & +\infty \\ 5 & 1 & 7 & 3 & 0 & 11 & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 1 & 2 & 2 & 1 & 3 & & \\ 2 & & 2 & 2 & 1 & 3 & & \\ & & & 3 & & 3 & & \\ & & 4 & & & 3 & & \\ 2 & 5 & 2 & 2 & & 3 & & \\ & & & & & & & 6 \\ & & & & & 7 & & \\ & & & & & & 8 & \end{pmatrix}$$

$$k=4 \quad D=\begin{pmatrix} 0 & 5 & 9 & 7 & 3 & 13 & +\infty & +\infty \\ 4 & 0 & 4 & 2 & 7 & 8 & +\infty & +\infty \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & +\infty \\ +\infty & +\infty & 2 & 0 & +\infty & 6 & +\infty & +\infty \\ 5 & 1 & 5 & 3 & 0 & 9 & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 1 & 4 & 2 & 1 & 3 & & \\ 2 & & 4 & 2 & 1 & 3 & & \\ & & & 3 & & 3 & & \\ & & 4 & & & 3 & & \\ 2 & 5 & 4 & 2 & & 3 & & \\ & & & & & & & 6 \\ & & & & & 7 & & \\ & & & & & & 8 & \end{pmatrix}$$

$$k=5 \quad D=\begin{pmatrix} 0 & 4 & 8 & 6 & 3 & 12 & +\infty & +\infty \\ 4 & 0 & 4 & 2 & 7 & 8 & +\infty & +\infty \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & +\infty \\ +\infty & +\infty & 2 & 0 & +\infty & 6 & +\infty & +\infty \\ 5 & 1 & 5 & 3 & 0 & 9 & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 5 & 4 & 2 & 1 & 1 & & \\ 2 & & 4 & 2 & 1 & 3 & & \\ & & & 3 & & 3 & & \\ & & 4 & & & 3 & & \\ 2 & 5 & 4 & 2 & & 3 & & \\ & & & & & & & 6 \\ & & & & & 7 & & \\ & & & & & & 8 & \end{pmatrix}$$

$$k=6 \quad D=\begin{pmatrix} 0 & 4 & 8 & 6 & 3 & 12 & +\infty & 15 \\ 4 & 0 & 4 & 2 & 7 & 8 & +\infty & 11 \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & 7 \\ +\infty & +\infty & 2 & 0 & +\infty & 6 & +\infty & 9 \\ 5 & 1 & 5 & 3 & 0 & 9 & +\infty & 12 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & 6 \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 5 & 4 & 2 & 1 & 3 & & 7 \\ 2 & & 4 & 2 & 1 & 3 & & 6 \\ & & & 3 & & 3 & & 6 \\ & & 4 & & & 3 & & 6 \\ 2 & 5 & 4 & 2 & & 3 & & 6 \\ & & & & & & & 6 \\ & & & & & 7 & & 6 \\ & & & & & & 8 & \end{pmatrix}$$

$$k=7 \quad D=\begin{pmatrix} 0 & 4 & 8 & 6 & 3 & 12 & +\infty & 15 \\ 4 & 0 & 4 & 2 & 7 & 8 & +\infty & 11 \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & +\infty & 7 \\ +\infty & +\infty & 2 & 0 & +\infty & 6 & +\infty & 9 \\ 5 & 1 & 5 & 3 & 0 & 9 & +\infty & 12 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & +\infty & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & 6 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 10 & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 5 & 4 & 2 & 1 & 3 & & 6 \\ 2 & & 4 & 2 & 1 & 3 & & 6 \\ & & & 3 & & 3 & & 6 \\ & & 4 & & & 3 & & 6 \\ 2 & 5 & 4 & 2 & & 3 & & 6 \\ & & & & & & & 6 \\ & & & & & 7 & & 6 \\ & & & & & 7 & 8 & \end{pmatrix}$$

$$k=8 \quad D=\begin{pmatrix} 0 & 4 & 8 & 6 & 3 & 12 & 22 & 15 \\ 4 & 0 & 4 & 2 & 7 & 8 & 18 & 11 \\ +\infty & +\infty & 0 & 3 & +\infty & 4 & 14 & 7 \\ +\infty & +\infty & 2 & 0 & +\infty & 6 & 16 & 9 \\ 5 & 1 & 5 & 3 & 0 & 9 & 19 & 12 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 0 & 10 & 3 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 3 & 0 & 6 \\ +\infty & +\infty & +\infty & +\infty & +\infty & 10 & 7 & 0 \end{pmatrix} \text{ et pi} = \begin{pmatrix} & 5 & 4 & 2 & 1 & 3 & 8 & 6 \\ 2 & & 4 & 2 & 1 & 3 & 8 & 6 \\ & & & 3 & & 3 & 8 & 6 \\ & & 4 & & & 3 & 8 & 6 \\ 2 & 5 & 4 & 2 & & 3 & 8 & 6 \\ & & & & & & 8 & 6 \\ & & & & & 7 & & 6 \\ & & & & & 7 & 8 & \end{pmatrix}$$

Le coefficient ligne 5 et colonne 6 de la matrice D est $d_{5,6}$ est 9, donc le coût du plus court chemin pour aller du sommet 5 au sommet 6 est 9.
D'autre part, la matrice pi permet de retrouver ce plus court chemin qui est donc $(5, 2, 4, 3, 6)$

Remarque

Comme on l'a vu, il faut supposer que le réseau ne contient pas de circuit négatifs pour que les plus courts chemins existent et que l'algorithme rende bien le résultat voulu. Dans le cas contraire, l'exécution de l'algorithme conduira à obtenir certaines valeurs $d(i, i)$ négatives (si le sommet i appartient à un circuit négatif). Ce critère est d'ailleurs une manière de compléter l'algorithme pour détecter la présence d'un circuit négatif.

V Exercices

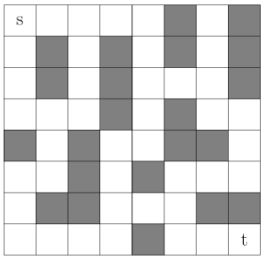
Exercice 26

Soit G un réseau (graphe orienté valué) dont certains arcs ont un coût négatif, mais sans circuit négatif.

1. Illustrer grâce à un exemple le fait que l'algorithme de Dijkstra ne détecte pas toujours les plus courts chemins dans G.
2. Pour contourner le problème induit par les arcs négatifs, un étudiant propose l'idée suivante : « soit $-c \in \mathbb{R}$ le coût minimal dans G. Alors on rajoute $+c$ à tous les coûts : plus précisément, pour chaque arc (i, j), on pose $c(i, j) \leftarrow c(i, j) + c$. Ainsi tous les coûts sont maintenant positifs, et on peut appliquer Dijkstra pour calculer les plus courts chemins. » L'idée fonctionne-t-elle ? Si oui, le prouver. Si non, expliquer avec un exemple.

Exercice 27

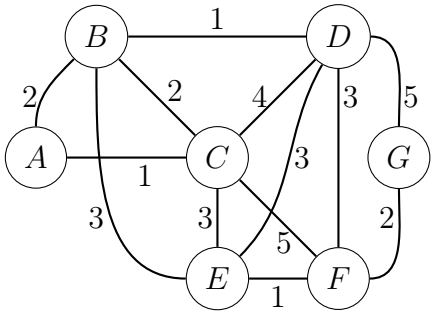
Les cases grisées de l'échiquier de la figure ci-dessous sont interdites. Le but du problème est de déplacer une tour de la case s à la case t sans passer sur les cases interdites (Rappel : une tour ne peut se déplacer qu ?horizontalement et verticalement, mais d ?un nombre de cases libre). L ?objectif est de trouver un chemin qui minimise le nombre de cases visitées (on entend par cases visitées les cases sur lesquelles se pose la tour dans son déplacement se s à t).



1. Formuler ce problème comme un problème de plus courts chemins (expliquer votre méthode pour construire le graphe correspondant).
2. En déduire un chemin de longueur minimale en nombre de déplacements que doit suivre la tour pour atteindre la case t à partir de la case s.

Exercice 28

Le graphe ci-dessous représente le plan d'une ville.
Le sommet A désigne l'emplacement des services techniques.
Les sommets B, C, D, E, F et G désignent les emplacements de jardins publics. Une arête représente l'avenue reliant deux emplacements et est pondérée par le nombre de feux tricolores situés sur le trajet.



Partie I

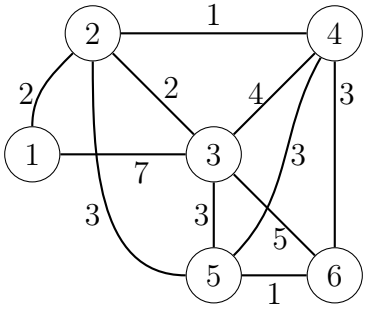
1. Répondre sans justification aux quatre questions suivantes :
   (a) Ce graphe est-il connexe ?
   (b) Ce graphe est-il complet ?
   (c) Ce graphe admet-il une chaîne eulérienne ?
   (d) Ce graphe admet-il un cycle eulérien ?
2. Déterminer, en justifiant, le nombre chromatique de ce graphe.

Partie II

1. Appliquer l'algorithme de Dijkstra pour donner les plus courtes distances du sommet A aux autres sommets.
2. En déduire le chemin le plus court du sommet A au sommet G.

Proposer un trajet comportant un minimum de feux tricolores reliant A à G. La réponse sera justifiée par un algorithme.

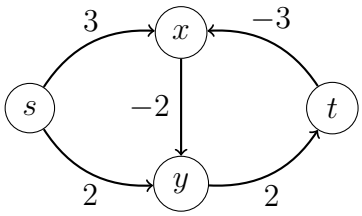
Exercice 29



1. Appliquer l'algorithme de Floyd-Warshall au graphe précédent.
2. Déterminer le sommet au "centre" du graphe, c'est à dire le sommet s minimisant la somme des distances de s aux autres sommets.

Exercice 30

Exécuter l'algorithme de Bellman-Ford sur le réseau ci-dessous, à partir du sommet s. Vérifier que l'algorithme détecte la présence d'un circuit absorbant.



Exercice 31

[algorithme de Sedgewick-Vitter]
Nous avons vu en cours que pour calculer le plus court chemin entre deux sommets fixés s et t, il n'existait pas d'algorithme plus efficace que Dijkstra, qui calcule tous les plus courts chemins d'origine s. Ceci est vrai dans le cas général, mais il existe cependant des cas particuliers où l'on peut améliorer la recherche d'un plus court chemin à destination fixée... Un *graphe euclidien* est un graphe non orienté dont les sommets sont des points d'un espace euclidien, et le coûts de chaque arête est égale à la longueur de l'arête :

$$c(x, y) := \|y - x\|.$$

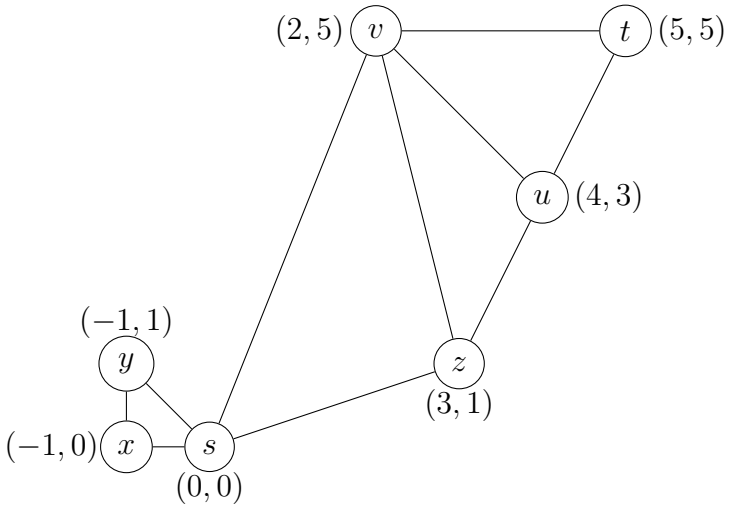
On rappelle que l'algorithme de Dijkstra visite les sommets x dans l'ordre croissant de $d(x)$. L'algorithme de Sedgewick-Vitter est une variante qui permet de calculer plus rapidement le plus court chemin vers un sommet t fixé. Pour cela, on visite les sommets x dans l'ordre croissant de $d(x) + \|t - x\|$. Plus précisément, on remplace la ligne

"trouver un sommet x de L tel que $d(x)$ soit minimal"

dans l'algorithme de Dijkstra par :

"trouver un sommet x de L tel que $d(x) + \|t - x\|$ soit minimal".

Montrer la correction de l'algorithme : à chaque étape, le sommet x selectionné est tel que $d(x)$ soit optimale. Puis, comparer les algorithmes de Dijkstra et de Sedgewick-Vitter pour le calcul du plus court chemin de s à t dans le graphe du plan euclidien ci-après. (On a indiqué les coordonnées des sommets dans un repère orthonormé.)



Exercice 32

On considère un réseau de télécommunication, composé d'émetteurs-récepteurs pouvant s'envoyer des messages, avec une certaine fiabilité.
On modélise ce réseau à l'aide d'un graphe orienté et valué, où chaque sommet représente un émetteur-récepteur, chaque arc (x, y) indique la possibilité d'une transmission de x à y, et la valuation $p(x, y)$ indique la probabilité pour que la communication se passe sans problème de x à y.

1. Comment calculer la *fiabilité* d'un chemin $x_1, x_2, \ldots, x_n$, c'est-à-dire la probabilité qu'un message soit bien transmis de x_1 à x_n par l'intermédiaire de ce chemin ?
2. Montrer que la recherche d'un chemin de fiabilité maximale entre deux sommets peut se rapporter à un problème de plus court chemin (indice : utiliser une fonction qui transforme les produits en sommes).

Exercice 33

[chemin de capacité maximale]
On représente un réseau routier par un graphe orienté valué, dont la valuation de chaque arc $c(x, y)$ représente la *capacité de trafic* sur la route de x à y. On définit alors la capacité de trafic sur un chemin comme le minimum des capacités des arcs qui composent le chemin. Modifier l'algorithme de Dijkstra pour obtenir un algorithme qui calcule la capacité maximale des chemins d'origine s dans un réseau. Montrer la correction de ce nouvel algorithme.

Exercice 34

On considère un système de contraintes :

$$x_j - x_i \leq w_{ij}, \quad x_i \leq w_i \quad (i, j = 1, ...n). \tag{6.1}$$

Par exemple :

$$\begin{array}{lcllcl} x_1 - x_2 & \leq & 3, & x_1 & \leq & 3, \\ x_2 - x_3 & \leq & -2, & x_2 & \leq & 2, \\ x_3 - x_1 & \leq & 2. & x_3 & \leq & 1. \end{array} \tag{6.2}$$

On associe à ce système un *graphe de contrainte*, ayant un sommet pour chaque variable x_i et un arc (x_i, x_j) de coût w_{ij} pour chaque contrainte $x_j - x_i \leq w_{ij}$. On ajoute à ce graphe un sommet source s, relié à chaque autre sommet par un arc (s, x_i) de coût w_i.

1. Dessiner le graphe de contrainte correspondant au système 6.2 donné en exemple.
2. Résoudre le système 6.2 grâce à l'algorithme de Bellman-Ford.
3. Montrer qu'en toute généralité, le système de contraintes 6.1 admet des solutions si et seulement si le graphe de contrainte associé ne possède pas de circuit négatif.

Exercice 35

Dans un réseau (graphe orienté valué), on note $d(i, j)$ le coût d'un plus court chemin allant du sommet i au sommet j, et on pose $d(i, j) = +\infty$ s'il n'existe pas de chemin de i à j.

1. Est-il possible qu'il existe un chemin de i à j, mais pas de plus court chemin ? Justifier.
2. Étant donné trois sommets i, j, k, quelle inégalité relie $d(i, j)$, $d(i, k)$ et $d(k, j)$?
3. En cours d'exécution de l'algorithme de Floyd-Warshall sur un réseau à quatre sommets, on obtient la matrice

$$D = (d(i,j))_{i,j} = \begin{pmatrix} 0 & 8 & 4 & 1 & 7 \\ +\infty & 0 & 3 & -1 & 1 \\ +\infty & +\infty & 0 & 2 & 4 \\ +\infty & +\infty & +\infty & 0 & 4 \\ +\infty & +\infty & +\infty & +\infty & 0 \end{pmatrix}$$

   L'algorithme est-il terminé, ou peut-on encore améliorer certaines entrées de la matrice ? Justifier.

Exercice 36

Répondre par `vrai` ou `faux` aux quatre affirmations suivantes. Justifier la réponse par une courte preuve (si `vrai`) ou un contre-exemple (si `faux`). On se place dans un graphe orienté valué...

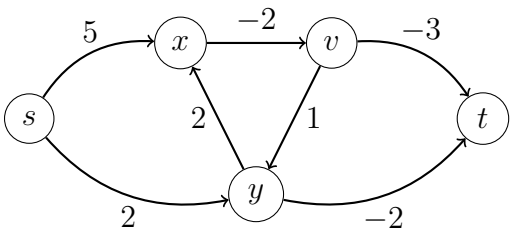
1. S'il existe un plus court chemin entre deux sommets s et t, alors celui-ci est unique.
2. S'il existe un plus court chemin entre deux sommets s et t, alors il est élémentaire (il ne passe pas deux fois par le même sommet).

Pour les deux questions suivantes, on suppose que α est un chemin de s_0 à s_1, et β est un chemin de s_1 à s_2. On note $\alpha\beta$ le chemin de s_0 à s_2 obtenu en parcourant le chemin α suivi du chemin β.

3. Si α et β sont des plus courts chemins, alors $\alpha\beta$ est un plus court chemin.
4. Si $\alpha\beta$ est un plus court chemin, alors α et β sont des plus courts chemins.

Exercice 37

On considère le graphe orienté valué G ci-dessous.



1. Quel algorithme utiliseriez-vous pour trouver un plus court chemin de s à t ? Pourquoi ?
2. Décrire succinctement le principe de cet algorithme.
3. Exécuter l'algorithme et en déduire un plus court chemin de s à t.
4. D'après l'exécution de l'algorithme, qu'est-ce qui nous assure qu'il n'existe pas de circuit négatif dans le graphe G ? (autrement dit, que se passerait-il s'il existait un circuit négatif ?)

Chapitre 7

Arbre couvrant de poids minimal

I Arbre couvrant de poids minimal

Les deux algorithmes suivants permettent de trouver un arbre couvrant de poids minimal dans un graphe non orienté et pondéré.
En d'autres termes, cet algorithme trouve un sous-ensemble d'arêtes formant un arbre sur l'ensemble des sommets du graphe initial et tel que la somme des poids de ces arêtes soit minimale. Si le graphe n'est pas connexe, alors l'algorithme détermine un arbre couvrant minimal d'une composante connexe du graphe.
Ce problème a de nombreuses applications, par exemple simplifier un câblage ou supprimer les liaisons maritimes les moins rentables en préservant l'accessibilité aux différents ports. ...

1 Algorithme de Prim

L'algorithme a été développé en 1930 par le mathématicien tchèque Vojtech Jarnik1, puis a été redécouvert et republié par Robert C. Prim et Edsger W. Dijkstra en 1959.

2 Algorithme de Kruskal

L'algorithme de Kruskal est un algorithme de recherche d'arbre recouvrant de poids minimum ou arbre couvrant minimum dans un graphe connexe non-orienté et pondéré. Il a été créé en 1956 par Joseph Kruskal.
Le principe est le suivant :

II Exercices

Chapitre 8

Ordonnancement : planification de projet par les réseaux

On s'intéresse ici au problème de la planification des différentes étapes d'un projet. L'objectif de cette planification est notamment de déterminer, pour chaque étape du projet, sa date de début de réalisation. Cette date doit être fixée en tenant compte d'un certain nombre de paramètres liés à la tâche (en particulier sa durée), ainsi que des différents contraintes imposées par le projet.
Dans le contexte de ce cours, on va plus particulièrement étudier comment les réseaux (graphes valués) peuvent aider à la modélisation et la résolution de problèmes de planification.

I Durée d'une tâche, et contraintes

Un projet est décomposé en différentes tâches à effectuer. Chaque tâche correspond à une étape "élémentaire" dans la réalisation du projet et a une durée. D'autre part, ces données ne sont généralement pas "certaines", mais elles sont estimées, notamment à partir des expériences passées dans la gestion de projets similaires. Ici, on considérera que la durée d'une tâche est une donnée fixe, et on notera $durée(i)$ la durée de la tâche i.
La réalisation des différentes tâches doit être planifiée en respectant les contraintes du problème que l'on peut classer en plusieurs catégories.

- Les **contraintes de précédence** ou d'**antériorité** imposent que certaines tâches ne peuvent commencer que si d'autres tâches sont déjà terminées. Par exemple, le toit d ?une maison ne peut être posé que si les murs sont construits.
- Les **contraintes de localisation temporelle** expriment le fait qu'une tâche doit être réalisée à l'intérieur d'une période de temps imposée. Par exemple, le tournage d'une scène extérieure nocturne pour un film devra évidemment se faire de nuit.
- Les **contraintes disjonctives** ou d'**exclusion** expriment le fait que plusieurs tâches ne peuvent être faites en même temps. Ces contraintes interviennent généralement lorsque l'exécution des tâches dépend de la disponibilité de certaines ressources. Par exemple, dans la cuisine d'un restaurant, l'avancement des préparations sera conditionnée par le nombre de cuisiniers, le nombre de fourneaux et de feux, le nombre d'ustensiles, etc.

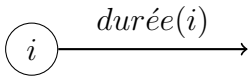
La réalité de la planification peut donc s'avérer très complexe. Dans le cadre de ce cours, on simplifie un peu : on ne prendra en compte que la durée des tâches ainsi que les contraintes de précédence dans la modélisation d'un projet.
Prenons un exemple - le projet le plus simple : faire des pâtes. On a une tâche A "acheter les pâtes", une tâche B "faire bouillir l'eau" et une tâche C "cuire les pâtes". Sans contrainte de localisation temporelle (le supermarché est ouvert), et sans contrainte disjonctive (une personne peut sortir

acheter les pâtes tandis que l'autre fait bouillir l'eau), on peut résumer le projet par le tableau suivant.

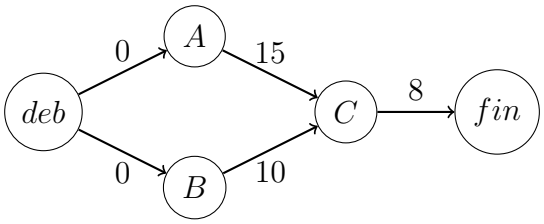
tâche	tâches antérieures	durée (minutes)
A	-	15
B	-	10
C	A,B	8

II Graphes potentiels-tâches

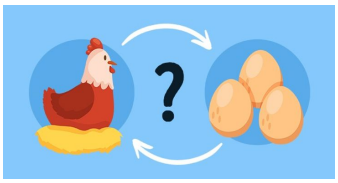
L'idée est d'associer un sommet du graphe à chaque tâche du projet, tandis que les arcs traduisent les contraintes de précédence : une tâche i devra être effectuée avant la tâche j si et seulement si $j \in \Gamma^+(i)$ (i est un ancêtre de j) dans le graphe. À chaque arc (i, j) du graphe sera associé le coût $durée(i)$, c'est-à-dire la durée de réalisation de la tâche correspondant au sommet de départ de l'arc :



Pour plus de clarté on rajoute un sommet initial deb et un sommet final fin, et on rajoute un arc (de durée 0) entre le sommet deb et tout sommet n'ayant pas de prédécesseur, ainsi qu'un arc entre tout sommet n ?ayant pas de successeur et le sommet fin.
Pour notre exemple "faire des pâtes", le graphe potentiels-tâches est donc le suivant :



Remarque
Les graphes potentiels-tâches ne comportent pas de circuit. En effet, la présence d'un circuit traduirait une incohérence dans les contraintes de précédence : une tâche i serait ancêtre d'elle-même, donc nécessiterait d'être déjà effectuée avant de pouvoir commencer.



III Dates au plus tôt et au plus tard

1 Date de début au plus tôt

La **date de début au plus tôt** d'une tâche i, notée $t(i)$, est le temps minimum qui sépare le début de cette tâche du début du projet. Attention : il faut bien comprendre que :

$$t(i) = \text{durée du } \textbf{plus long chemin} \text{ entre } (deb) \text{ et } i.$$

En effet, pour arriver à la tâche i, il faut avoir eu le temps de parcourir tous les chemins possibles entre (deb) et i.
On aura donc toujours $t(deb) = 0$, et la date au plus tôt de fin de réalisation du projet sera notée $t(fin)$. Cette date $t(fin)$ correspond aussi à la **durée minimale** du projet.
Dans notre exemple, on a $t(A) = t(B) = 0$, $t(C) = 15$, et $t(fin) = 8 + 15 = 23$.

2 Date de début au plus tard

La **date de début au plus tard** d'une tâche i, notée $T(i)$, est le temps maximum qui peut séparer le début de cette tâche du début du projet, **sans retarder la date** $t(fin)$ **de fin du projet**. En termes de graphes,

$$T(i) = t(fin) - \text{durée du plus long chemin entre } i \text{ et } (fin).$$

On aura donc toujours $T(fin) = t(fin)$, et $T(deb) = 0$.
Dans notre exemple, on a $T(A) = t(A) = 0$, $T(B) = 23 - (10 + 8) = 5$, et $T(C) = t(C) = 15$.

3 Marges

À partir des dates de début au plus tôt et au plus tard d'une tâche i, on peut calculer la marge associée à cette tâche, c'est-à-dire le battement maximum dont on dispose pour retarder sa date de début (par rapport à $t(i)$) sans pour autant perturber la date de fin du projet. La marge associée à une tâche i, notée $M(i)$, est donc définie par

$$M(i) = T(i) - t(i).$$

Dans notre exemple, on a $M(A) = M(C) = 0$, et $M(B) = 5$.

4 Tâches et chemins critiques

Certaines tâches ne disposent d'aucune marge pour leur réalisation. Le moindre retard dans la réalisation d'une telle tâche entraînera nécessairement un retard global du projet. C'est pourquoi de telles tâches sont dites **critiques**. Sur l'exemple des pâtes, il s'agit des tâches A et C.
D'une manière plus générale, on appelle **chemin critique** tout chemin dans le graphe allant du sommet de début au sommet de fin du projet et ne passant que par des tâches critiques. Tout projet possède au moins un chemin critique, correspondant au plus long chemin entre les sommets (deb) et (fin). Certains projets peuvent avoir plusieurs chemins critiques, s'il y a plusieurs plus longs chemins.

IV Un peu de méthode

Pour planifier un projet important, composé de milliers voire de millions de tâches (par exemple : installer une base pour vivre sur la planète Mars), on a besoin d'une méthode algorithmique qui permette de générer une représentation du projet par un graphe potentiel-tâches, et de calculer les dates de début au plus tôt et au plus tard automatiquement.

1 Tri topologique et classement des sommets par niveaux

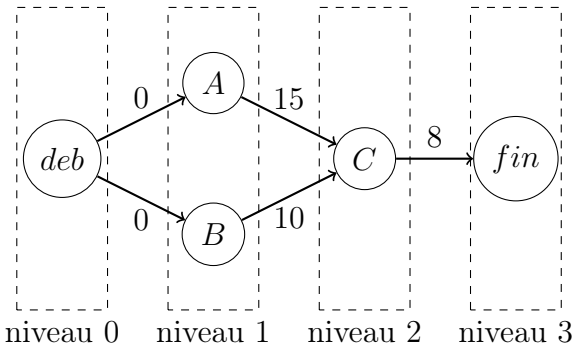
Un **tri topologique** des sommets/tâches est une manière d'ordonner les tâches qui soit cohérente avec les contraintes de précédence : si i précède j, alors i doit être classée avant j. Autrement dit,

si l'on ordonne les tâches par un tri topologique de gauche à droite, alors tous les arcs du graphe potentiels-tâches doivent être orientés de gauche à droite.
Il est nécessaire d'effectuer un tri topologique des tâches avant de calculer les dates au plus tôt et au plus tard.
Une manière d'effectuer ce tri est de classer les tâches par **niveaux**. Le niveau d'un sommet i du graphe est le nombre maximal d'arcs dans un chemin de (deb) à i.
Les tâches de niveau 1 sont celles qui n'ont pas de contraintes de précédence. Les tâches de niveau 2 sont celles qui peuvent être effectuées une fois que toutes les tâches de niveau 1 sont terminées, et ainsi de suite... Suivant cette idée, le tri par niveaux est décrit dans l'algorithme 11 ci-dessous.

Algorithme 11 : Tri par niveaux.

$L \leftarrow$liste des tâches;
$n \leftarrow 0$;
classer le sommet (deb) au niveau $n = 0$;
tant que $L \neq \emptyset$ **faire**
  $n \leftarrow n + 1$;
  retirer de L les taches sans précédence et les classer au niveau n;
fin

En pratique, on peut placer les sommets de même niveau dans une même colonne, et on ordonne les niveaux de gauche à droite. Dans notre exemple, les niveaux apparaissent ainsi :



2 Recherche des dates $t(i)$

Partir de $t(deb) = 0$. Puis, prendre les sommets par niveaux croissants et faire

$$t(i) = \max_{k \in \Gamma^-(i)} \Big(t(k) + durée(k) \Big).$$

3 Recherche des dates $T(i)$

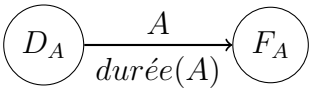
Partir de $T(fin) = t(fin)$. Puis, prendre les sommets par niveaux décroissants et faire

$$T(i) = \min_{k \in \Gamma^+(i)} \Big(T(k) - durée(i) \Big).$$

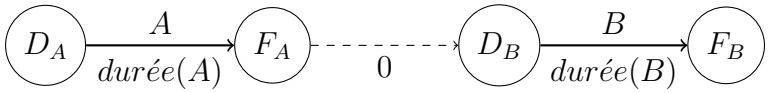
S'il n'y a pas d'erreur, on doit bien retomber sur $T(deb) = 0$.

V Graphes potentiels-étapes (méthode PERT)

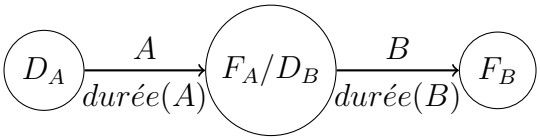
L'idée est d'associer un arc du graphe à chaque tâche A du projet, tandis que les sommets du graphe représentent des **étapes** dans l'avancement des travaux :



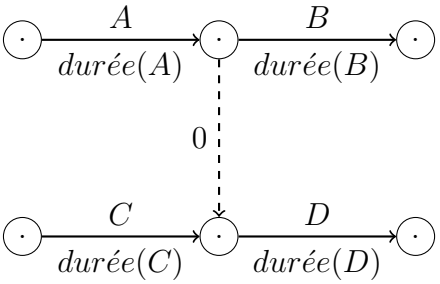
Ici l'étape D_A signifie "début de A", et l'étape F_A signifie "fin de A".
La précédence est modélisée par un arc fictif (ou tâche fictive) de durée égale à 0. Dans l'exemple ci-dessous, l'arc fictif en pointillés modélise le fait que la tâche A doit être terminée avant de pouvoir commencer la tâche B.



Dans certains cas, on peut fusionner les étapes :



Dans d'autres cas, comme ci-dessous, la présence d'arc fictif est nécessaire et ne peut être évitée sans modifier les précédences :



La seule condition à respecter est que pour chaque sommet/étape, les tâches qui y arrivent doivent être terminées avant que ne débutent les tâches qui en partent. Les symboles D_A, F_A, etc, sont facultatifs. On pourra, à la place, numéroter les sommets.

Remarque

Cette méthode a l'avantage de donner une représentation visuelle très lisible de l'enchaînement des tâches. En revanche, elle est assez rigide, dans la mesure où elle ne permet pas de prendre en compte facilement l'introduction, ou la suppression, de contraintes de précédence.

VI Exercices

Exercice 38

On a 6 tâches A,B,C,D,E,F à réaliser. Pour être traitées, les tâches D,E,F nécessitent toutes les trois que A,B,C soient préalablement effectuées.

1. Tracer le graphe potentiels-tâches et le graphe potentiels-étapes (méthode PERT) du projet.
2. Lequel de ces graphes donne, selon vous, la représentation la plus visuellement lisible de l'enchaînement des tâches ?
3. On se rend compte, en cours de préparation du projet, que finalement la tâche D n'a pas besoin de l'antériorité de A pour pouvoir être traitée. Comment modifier les deux graphes ? Lequel est le plus flexible (i.e. peut être modifié le plus facilement) ?

Exercice 39

On décompose un projet en différentes tâches A,B,...,N. On donne les contraintes de précédence ainsi que la durée de chaque tâche dans le tableau ci-dessous.

tâche	tâches antérieures	durée (jours)
A	F	6
B	-	9
C	B,G,I,N	7
D	E,F,J	5
E	K	6
F	-	7
G	J,K	2
H	K	5
I	A,L	3
J	B,H,M	4
K	-	3
L	F	8
M	F	3
N	A,L	4

1. Classer les tâches par niveaux et déterminer les tâches immédiatement antérieures.
2. Tracer le graphe potentiels-tâches du projet.
3. Calculer la date de début au plus tôt de chaque tâche, et la durée minimale du projet.
4. Calculer la date de début au plus tard et la marge de chaque tâche.
5. Déterminer le chemin critique.
6. Dans les 3 cas suivants, indiquer ce que devient la durée minimale du projet :
   - la durée de C passe à 11 jours ;
   - la durée de E passe à 11 jours ;
   - la durée de I passe à 11 jours.

Exercice 40

On décompose un projet en différentes tâches A,B,...,L. On donne les contraintes de précédence ainsi que la durée de chaque tâche dans le tableau ci-dessous.

tâche	tâches antérieures	durée (jours)
A	B	4
B	-	8
C	-	11
D	B	5
E	C	3
F	-	2
G	E,K	5
H	A,B,C	2
I	C,D,H,L	10
J	B,C,L	5
K	F	10
L	A,C	3

1. Classer les tâches par niveaux et déterminer les tâches immédiatement antérieures.
2. Tracer le graphe potentiels-tâches du projet.
3. Calculer la date de début au plus tôt de chaque tâche, et la durée minimale du projet.
4. Calculer la date de début au plus tard et la marge de chaque tâche.
5. Déterminer le chemin critique.
6. Dans les 3 cas suivants, indiquer ce que devient la durée minimale du projet :
   - la durée de E passe à 10 jours ;
   - la durée de L passe à 5 jours ;
   - la durée de G passe à 10 jours.

Exercice 41

Appliquer la méthode PERT au projet suivant.

tâche	tâches antérieures	durée (jours)
A	-	3
B	-	9
C	-	5
D	A	8
E	B	4
F	B	7
G	B	20
H	C,F	6
I	D,E	5

Exercice 42

On décompose un projet en différentes tâches A,B,C,D,E,F,G. On donne les contraintes de précédence ainsi que la durée de chaque tâche dans le tableau ci-dessous.

tâche	tâches antérieures	durée (jours)
A	-	3
B	A	2
C	-	4
D	A,E	1
E	A,C	6
F	-	8
G	A,B,C,F	7

1. Tracer le graphe potentiels-tâches du projet. On y fera apparaître le classement des tâches par niveaux.
2. Calculer la date de début au plus tôt de chaque tâche, et la durée minimale du projet.
3. Calculer la date de début au plus tard, et la marge de chaque tâche.
4. Déterminer le chemin critique.
5. Suite à un imprévu, on se voit dans l'obligation de faire un choix entre deux options : (1) rajouter 1 jour à la durée de la tâche E, ou (2) rajouter 5 jours à la durée de la tâche B. Quelle option est préférable ? justifier.

Exercice 43

Un festival de musique a lieu pendant 5 jours consécutifs, avec des concerts à Nantes et à Angers. Le tableau suivant récapitule le nombre de concerts qui ont lieu dans chacune des deux villes en fonction du numéro n de la journée.

jour numéro...	1	2	3	4	5
nombre de concerts à Nantes	1	2	4	2	1
nombre de concerts à Angers	4	2	0	1	4

Marcellin souhaite assister à un maximum de concerts possible. Mais il ne se déplace qu'à vélo : il lui faut donc sacrifier une journée entière pour voyager d'une ville à l'autre. Le jour numéro 0, il a le choix entre pédaler vers Nantes ou vers Angers pour être sur place au début du jour numéro 1. Puis, chaque jour, il peut soit changer de ville (et n'assister à aucun concert), soit assister à tous les concerts dans la ville où il se trouve.

1. Montrer que le problème du choix du planning optimal de Marcellin peut être vu comme un problème de *plus long chemin* d'un sommet `"début"` à un sommet `"fin"`, dans un graphe G que l'on dessinera.
2. Dans un graphe potentiel-tâches, comment appelle-t-on le plus long chemin de `début` à `fin` ?
3. En considérant G comme un graphe potentiel-tâches, trouver les dates de début au plus tôt pour chaque sommet, et la durée minimale du projet.

4. Trouver les dates de début au plus tard, et le chemin critique.
5. Conclure sur le planning que devra suivre Marcellin.