BTS SIO **Août 2014**

Cours de graphes

Yann Barsamian
yann.barsamian@gmail.com

Ce document se veut un cours à destination des enseignants ayant à découvrir les graphes. Les notions abordées sont celles du programme de BTS SIO, sauf pour les parties ayant une ligne épaisse en bordure gauche : ce sont des paragraphes hors programme, qui sont là pour s'assurer que le professeur aille plus loin que le niveau qu'il a à enseigner à l'élève. Certains d'entre eux peuvent être présentés aux étudiants tout de même, bien sûr !

> *Exemple de paragraphe hors programme.*

Les démonstrations sont notamment hors programme, mais toutes sont abordables après une terminale scientifique : un étudiant de BTS peut ainsi également parcourir ce document.

Il est probable que ce cours contienne des coquilles voire de grossières erreurs. Je vous prie d'avance de bien vouloir m'en excuser. Si vous trouvez par ailleurs qu'il manque de krôbards, vous avez bien raison : il n'y a jamais suffisamment de dessins. Dans tous les cas n'hésitez pas : envoyez-moi vos remarques et/ou propositions d'amélioration, j'essayerai d'en tenir compte.

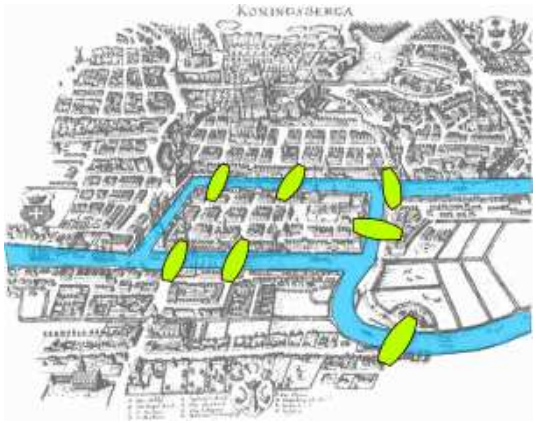
Table des matières

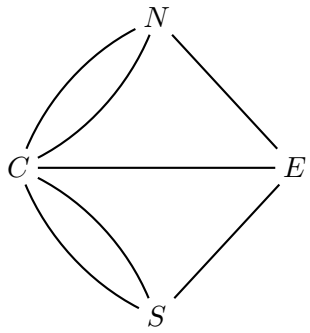
Chapitre 1

Généralités

1.1 Un peu d'histoire

En 1736, Euler s'intéresse au « problème des ponts de Königsberg » : existe-t-il un chemin passant une fois et une seule par chaque pont de la ville ? Il se rend compte qu'un tel chemin n'existe pas. Plus tard émerge la question de minimiser la distance parcourue par un postier voulant desservir toute une ville (« problème du voyageur de commerce »). Pour résoudre ce genre de problèmes, il est nécessaire de modéliser le réel : les points de visite sont modélisés par des sommets, les routes les reliant par des arcs ; et tout ce beau monde forme un graphe. Königsberg peut ainsi être représenté par le graphe suivant, dans lequel S est le côté sud de la rivière Pregolya, N le côté nord, C l'enclave centrale et E l'enclave est. C'est un graphe non-orienté et multiple, donc hors programme, mais il serait dommage de passer à côté :





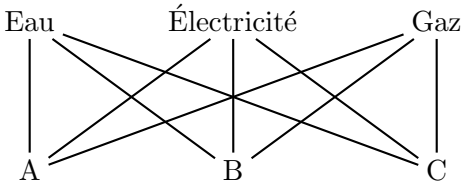
Source : Wikimedia Commons.

En 1856, Hamilton étudie un problème apparemment de même difficulté, celui de trouver un chemin passant une fois et une seule par chaque sommet d'un graphe. Il se trouve que ce problème est bien plus difficile à résoudre.

Plusieurs problèmes de grande envergure ont été soulevés puis résolus sur les graphes. Deux grands résultats récents traitent des graphes planaires : un graphe est planaire s'il existe une manière de le dessiner dans un plan sans que deux arcs se chevauchent.

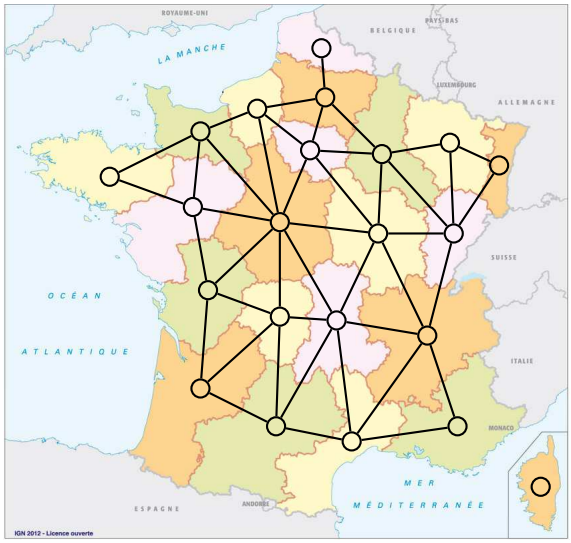
L'un d'eux est lié à un casse-tête connu : est-il possible de relier chacune des trois maisons à chacune des trois stations d'énergie (gaz, électricité, eau) de telle sorte que les canaux soient tous dans un seul plan ? Essayez, vous verrez que non. C'est le graphe connu sous le nom de $K_{3,3}$, qui n'est pas planaire. Cela peut paraître naturel avec le dessin du graphe : « il est évident » que la dernière arête va forcément couper l'une des huit premières dessinées ; mais ce résultat utilise un théorème de topologie pourtant difficile à démontrer : le théorème de Jordan. Kuratowski a montré en 1 930 que, pour savoir si un graphe est planaire, il suffit de

regarder s'il « contient » un graphe du type $K_{3,3}$ ou d'un autre type pas bien plus compliqué. Un résultat d'une simplicité déconcertante.



L'autre a été démontré en 1976 par Appel et Haken. Il s'agit du théorème des quatre couleurs : les sommets de tout graphe planaire peuvent être coloriés avec au plus quatre couleurs de telle sorte que deux sommets adjacents aient deux couleurs différentes. La preuve de ce résultat utilise l'outil informatique (réduction du cas général à un gros millier de cas particuliers, chacun colorié par l'ordinateur), et certains chercheurs essayent de la simplifier. Ci-dessous, les 22 régions de France - peut-être plus pour longtemps - coloriées avec 4 couleurs :

IGN 2012 - Licence ouverte



Le graphe associé.

1.2 Notion de graphe

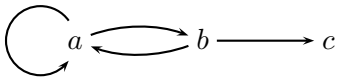
Un graphe simple orienté est la donnée d'un ensemble V (les sommets) et d'un ensemble $E \subset V \times V$ (les arcs).

Dans la suite, la taille de V sera notée n (c'est le nombre de sommets) et la taille de E sera notée m (c'est le nombre d'arcs).

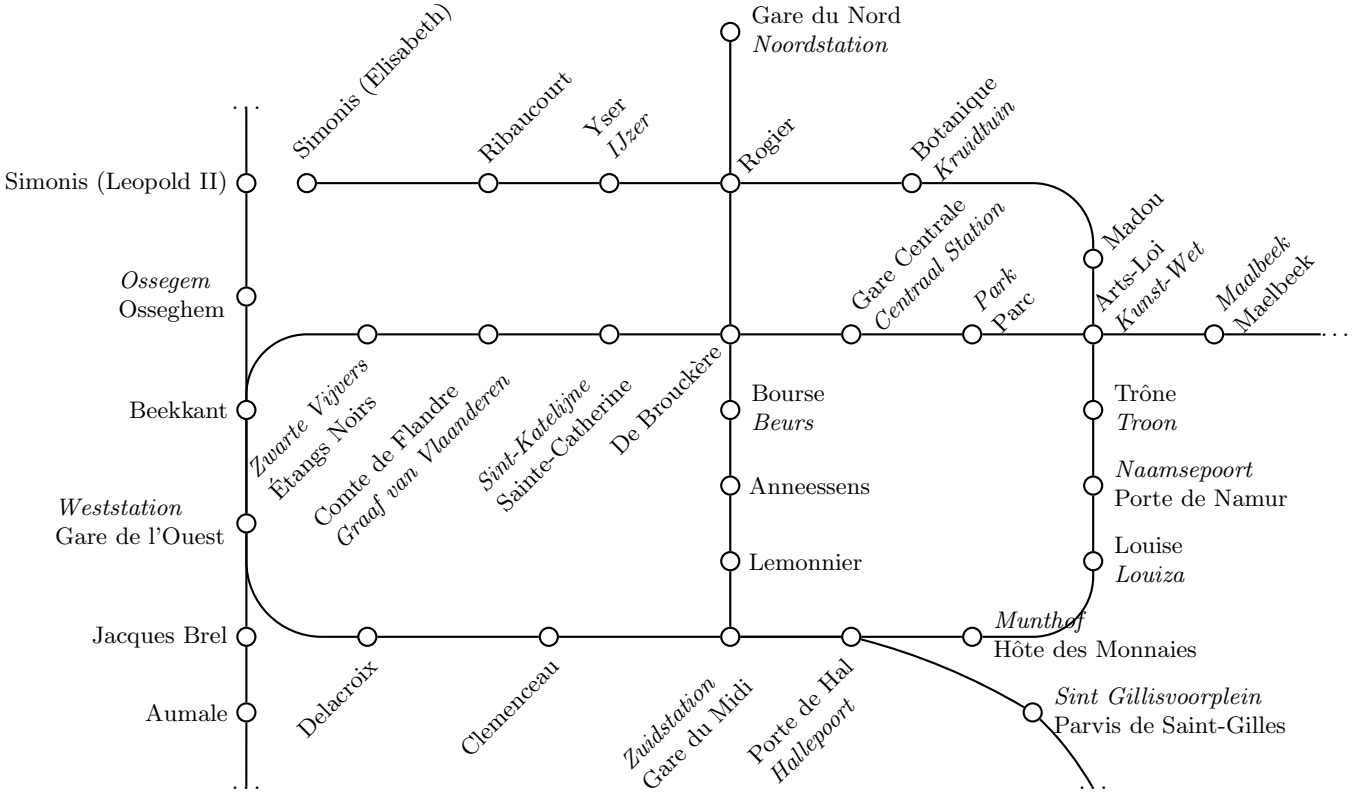
Remarques :

- la notation (V, E) provient de l'anglais : V pour *vertex* (sommet) et E pour *edge* (arc).
- une définition équivalente pour les arcs consiste à prendre non pas un ensemble E d'arcs inclus dans $V \times V$, mais une relation $\mathcal{R}$ sur V : dans les deux cas, il s'agit simplement de décrire quels sommets sont reliés à tels autres.

Pour donner un exemple « abstrait » illustrant la définition, le couple $(V = \{a; b; c\}\ ;\ E = \{(a, a); (a, b); (b, a); (b, c)\})$ est un graphe contenant 3 sommets et 4 arcs. Pour mieux comprendre cette notion de graphe, le plus simple reste de le dessiner. Pour cela, il faut écrire le nom des différents sommets, et mettre une flèche d'un sommet à un autre si l'arc correspondant existe :



Si vous êtes déjà allés dans une ville où il existe un métro, vous en avez forcément déjà vu un plan : c'est clairement un graphe ! La représentation des réseaux de transport en commun [1] est une application très concrète de ces derniers : les sommets sont les arrêts, et les arcs les connexions possibles entre deux arrêts. Par exemple, un extrait du plan du Métro de Bruxelles donne le graphe suivant :



Ce plan est particulièrement intéressant pour une autre raison. La Belgique est composée de la Flandre, et de la Wallonie ; la langue de la Wallonie est le français, et celle de la Flandre est le flamand. Les actualités encore récentes nous montrent que leurs querelles n'ont toujours pas cessé. Alors pour ne favoriser ni le français ni le flamand... il faut que les noms des arrêts de métro soient écrits dans les deux langues. Et ce n'est pas suffisant : il ne faut surtout pas qu'il y en ait plus qui commencent par le nom flamand, ou par le nom français. En fait l'idéal serait : un arrêt sur deux avec le nom flamand d'abord, puis le nom français ensuite, et un arrêt sur deux avec le nom français d'abord, puis le nom flamand ensuite. Pour réaliser cela, il faudrait que le graphe du métro bruxellois soit 2-coloriable : c'est-à-dire qu'il faudrait pouvoir colorier les sommets à l'aide de deux couleurs de telle sorte que deux sommets adjacents aient une couleur différente. Horreur et damnation : c'est impossible (essayez par vous-même, la présence d'un circuit de longueur impaire conclut facilement). Solution ? Certains noms ont la même écriture en français et en flamand ; pour ceux-là, pas de querelle. Il suffit d'en intercaler quelques-uns.

L'un des plus gros graphes sur lequel travailler est le graphe d'internet. C'est un graphe où l'ensemble des sommets est l'ensemble des pages internet et où un arc d'une page à une autre symbolise un lien pointant de la première vers la seconde. Ce graphe est parcouru par des robots en permanence afin de renvoyer les pages pertinentes sur les moteurs de recherche.

Un autre graphe intéressant est le graphe de connaissance : l'ensemble des sommets est l'ensemble des personnes sur terre ; un arc entre deux personnes symbolise le fait que la première

1. `http://www.le-cartographe.net/blog/archives/107-la-representation-cartographique-du-metro`

connaît la seconde. Les *stars* (les personnes connues d'énormément de monde) émergent alors directement. En 1 960, Milgram a montré que, comme le veut l'adage, « le monde est petit ». Il a demandé à 200 personnes de faire parvenir une lettre à une autre personne prise au hasard dans le monde, en ne connaissant que son nom, sa profession et sa ville de résidence. La contrainte était la suivante : chaque personne a le droit de faire passer la lettre par des intermédiaires, mais il faut absolument connaître la personne par qui faire transiter la lettre ; cette personne devra ensuite se plier à la même contrainte. Sur 200 lettres à faire parvenir, environ 100 sont arrivées (les autres n'ont peut être pas joué le jeu ?). Il a fallu en moyenne... 6 envois [2] pour faire arriver la lettre à destination.

Remarques :

- si i et j sont deux sommets, l'existence de l'arc (i, j) n'implique pas forcément l'existence de l'arc (j, i). En d'autres termes, ce n'est pas parce qu'un arc de i à j existe que l'arc retour (de j à i) existe également. [3]
- dans un graphe simple, il ne peut y avoir qu'un seul arc au maximum allant d'un sommet à un autre. [4]
- la définition du programme accepte qu'un graphe possède des boucles, c'est-à-dire des arcs allant d'un sommet vers lui-même. [5]

Finissons cette première entrevue sur les graphes par deux dernières définitions :
Soit (x, y) un arc de G. x est l'origine de cet arc et y est l'extrémité de cet arc.

2. Vous trouverez peut-être dans la littérature qu'aujourd'hui pour joindre deux individus pris au hasard sur un réseau social très connu, il faut en moyenne environ 4,5 arcs... cela veut-il dire que le monde s'est « resserré » ? Non, car ce n'est pas la même expérience : tout le monde n'est pas inscrit sur ce réseau social !

3. Quand les arcs sont à chaque fois présents dans les deux sens, la notion de sens n'a plus trop de nécessité : il s'agit alors de graphe non-orienté (hors programme). Si les arcs ont été définis par une relation, elle est dans ce cas symétrique, et il s'agit alors d'arêtes et non d'arcs.

4. Dans le cas contraire, il s'agit de graphe multiple (hors programme).

5. Les graphes non-orientés interdisent les boucles.

Chapitre 2

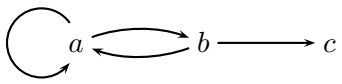
Modes de représentation d'un graphe

Nous avons découvert le mode de représentation le plus concret d'un graphe : le dessin (krôbard pour les intimes). Il en existe trois autres au programme.

2.1 Tableau de successeurs

Il s'agit de lister, pour chaque sommet i, l'ensemble des sommets j vers lesquels il existe un arc partant de i, c'est-à-dire $\{j \in V | (i,j) \in E\}$ [1]. Sur le dessin, c'est l'ensemble des sommets vers lesquels il existe une flèche partant de i.

Dans l'exemple qui est redessiné ici, le tableau de successeurs est le suivant :



Sommet	a	b	c
Successeurs	a, b	a, c	$-$

2.2 Tableau de prédécesseurs

Il s'agit de lister, pour chaque sommet i, l'ensemble des sommets j depuis lesquels il existe un arc partant vers i, c'est-à-dire $\{j \in V | (j,i) \in E\}$ [2]. Sur le dessin, c'est l'ensemble des sommets desquels il existe une flèche allant vers i.

Dans le même exemple, le tableau de prédécesseurs est le suivant :

Sommet	a	b	c
Prédécesseurs	a, b	a	b

2.3 Matrice d'adjacence

Pour construire cette matrice, il faut un ordre « naturel » sur l'ensemble des sommets (ordre lexicographique, ordre classique sur les nombres...), et chaque nom de sommet est alors remplacé par son numéro dans l'ordre choisi. La matrice d'adjacence M, de taille $n \times n$, est définie de la manière suivante :

$$(M)_{i,j} = \begin{cases} 1 & \text{lorsque l'arc } (i,j) \text{ est présent dans le graphe} \\ 0 & \text{sinon} \end{cases}$$

Dans l'exemple précédent, la matrice d'adjacence est la suivante :

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

1. Il existe une notation hors programme : $\Gamma^+(i)$ pour voisinage sortant de i.
2. Il existe une notation hors programme : $\Gamma^-(i)$ pour voisinage entrant de i.

D'après la définition des coefficients de la matrice, il est facile de voir les propriétés suivantes :

- pour lire l'ensemble des successeurs d'un sommet i, il suffit de regarder où sont les 1 dans la ligne i de M
- pour lire l'ensemble des prédécesseurs d'un sommet j, il suffit de regarder où sont les 1 dans la colonne j de M

Comme nous allons le découvrir, c'est cette dernière représentation qui permet de faire beaucoup de calculs utiles sur les graphes. La calculatrice effectuera ces calculs avec profit.

Il faut néanmoins noter que pour travailler en machine, ou pour déterminer la complexité des algorithmes utilisés, la représentation du graphe utilisée (tableau des successeurs ou matrice d'adjacence) donne lieu à une programmation différente et souvent à des complexités différentes. Par exemple :

- pour parcourir chacun des arcs : avec la matrice d'adjacence, il faut tester toutes les cases de la matrice (cela se fait en $O(n^2)$), alors qu'avec le tableau des successeurs, il faut parcourir chaque successeur de chaque sommet (cela se fait en $O(n + m)$)
- pour savoir si l'arc (i, j) existe : avec la matrice d'adjacence, il suffit de tester la case correspondante (cela se fait en $O(1)$), alors qu'avec le tableau des successeurs, il faut tester tous les successeurs de i (cela se fait en $O(n)$ à la louche - en fait cette approximation est trop grossière et pourra être amortie dans l'analyse de complexité car il n'y a pas n tests à faire à chaque sommet, puisque cela dépend du nombre de successeurs de i...)

✓Bulletin Officiel

Le B.O. attend qu'un étudiant soit capable de passer d'un mode de représentation à un autre. Il faut donc multiplier les exemples, notamment pour bien comprendre le sens des arcs dans la représentation matricielle.

Notons enfin qu'il existe un autre mode de représentation des graphes sans boucle : Δ, la matrice d'incidence. Il faut une nouvelle fois un ordre ≪ naturel ≫ sur l'ensemble des arcs, et chaque arc est alors remplacé par son numéro. La matrice d'incidence Δ, de taille $n \times m$, est définie de la manière suivante :

$$(\Delta)_{v,e} = \begin{cases} 1 & \text{lorsque le sommet } v \text{ est l'origine de l'arc } e \\ -1 & \text{lorsque le sommet } v \text{ est l'extrémité de l'arc } e \\ 0 & \text{sinon} \end{cases}$$

C'est un mode de représentation utile pour la théorie - par exemple les flots et la programmation linéaire - mais assez peu utilisé en pratique car gourmand en mémoire : la matrice est très creuse puisque seuls deux nombres sont non nuls par colonne. De plus, il n'est pas facile d'adapter cette représentation pour des graphes comportant des boucles : nous pourrions décider de mettre un 2 lorsque le sommet v est à la fois l'origine et l'extrémité de l'arc e, mais cela casserait une propriété intéressante de cette matrice (elle est totalement unimodulaire : le déterminant de toute matrice carrée extraite vaut 0, 1 ou -1).

Chapitre 3

Chemins

3.1 Définitions

Un chemin [1] est une suite d'arcs telle que chaque fin d'arc soit le début de l'arc suivant. Le cours se limitera à des suites finies d'arcs, et adoptera les notations $(v_0, v_1, \ldots, v_k)$ pour symboliser le chemin constitué des arcs $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ ainsi que $x \rightsquigarrow y$ pour un chemin allant de x à y.

La longueur d'un chemin est le nombre d'arcs qui le composent.

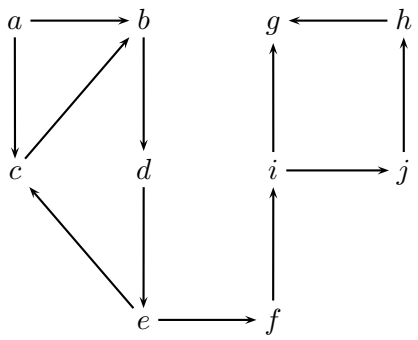
Pour $v \in V$, $v \rightsquigarrow v$ est un circuit [2] : c'est donc un chemin qui part d'un sommet pour y revenir.

Dans le cas où le chemin passe une fois et une seule par chaque sommet, c'est un chemin hamiltonien [3].

Lorsqu'il existe un chemin de i à j, j est dit accessible depuis i. La matrice d'accessibilité $\widehat{M}$ d'un graphe, de taille $n \times n$, est définie par :

$$(\widehat{M})_{i,j} = \begin{cases} 1 & \text{lorsque le sommet } j \text{ est accessible dans le graphe depuis le sommet } i \\ 0 & \text{sinon} \end{cases}$$

Exemple : soit le graphe suivant :



Le chemin $(a, c, b, d, e, f, i, j, h, g)$ est un chemin hamiltonien. Sa longueur est 9.

Le chemin (b, d, e, c, b) est un circuit. Du coup, le chemin $(b, d, e, c, b, d, e, c, b)$ est aussi un circuit, le chemin $(b, d, e, c, b, d, e, c, b, d, e, c, b)$ aussi, etc.

Le sommet g est accessible depuis le sommet f (depuis n'importe quel sommet d'ailleurs), mais aucun sommet n'est accessible depuis g (il n'a aucun successeur).

1. Pour les graphes non-orientés (hors programme), il s'agit de chaîne au lieu de chemin.
2. Pour les graphes non-orientés (hors programme), il s'agit de cycle au lieu de circuit, et un cycle doit de plus être simple, c'est-à-dire ne pas passer deux fois par la même arête ; sinon, toute arête prise dans un sens puis dans l'autre formerait un cycle pas très intéressant.
3. Dans le cas moins contraignant où le chemin n'a pas de répétition de sommet, c'est un chemin élémentaire (hors programme).

Remarques :
- il est évidemment équivalent de définir un chemin comme une suite de sommets dont chacun est un prédécesseur du suivant.

   Dans ce cas, la longueur d'un chemin, c'est le nombre de sommets qui le composent moins 1 (il y a toujours un intervalle de moins que de piquets).
- une boucle est (modulo un petit abus de langage) un circuit particulier, de longueur 1.
- nous verrons ensuite qu'il existe un lien entre la matrice d'adjacence d'un graphe et sa matrice d'accessibilité.
- dans le cas où le chemin passe une fois et une seule par chaque arc, c'est un chemin eulérien (notion hors programme, mais évoquée lors du problème des sept ponts de Königsberg).

3.2 Chemins de longueur donnée

Il est évident, de par la définition de la matrice d'adjacence, de voir que $(M)_{i,j}$ représente le nombre de chemins de longueur 1 allant de i à j. Effectivement, soit l'arc de i à j existe, et donc il y a un unique chemin de longueur 1 allant de i à j ; dans ce cas, $(M)_{i,j}$ vaut 1. Soit cet arc n'existe pas, et donc il n'existe pas de chemin de longueur 1 ; dans ce cas, $(M)_{i,j}$ vaut 0.

Ce résultat s'étend de la sorte :

Propriété
Pour tout $p \in \mathbb{N}^*$, $(M^p)_{i,j}$ est le nombre de chemins de longueur p allant de i à j.

Preuve

Raisonnons par récurrence sur p.

Initialisation : comme nous venons de le voir, la propriété est vraie au rang 1.

Hérédité : supposons la propriété vraie au rang p. Soient i et j deux sommets du graphe. Calculons alors le nombre de chemins de longueur $p+1$ allant de i à j. Un tel chemin commence nécessairement par un arc (i,k), pour un certain k entre 1 et n.

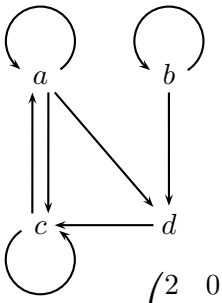
Soit $k \in [|1;n|]$. Dénombrons le nombre de chemins de longueur $p+1$ allant de i à j qui commencent par (i,k). Si cet arc n'existe pas, c'est 0. S'il existe, il y en a autant que de chemins de longueur p allant de k à j ; dans ce cas, par hypothèse de récurrence, il y en a $(M^p)_{k,j}$. Dans chacun de ces deux cas, il y en a donc bien $(M)_{i,k} \times (M^p)_{k,j}$.

Enfin, pour compter le nombre total de chemins de longueur $p+1$ allant de i à j, il suffit de faire la somme sur chaque valeur différente que peut prendre k, le second sommet du chemin. Ainsi il y en a $\sum_{k=1}^{n} (M)_{i,k} \times (M^p)_{k,j}$. C'est la définition de $(M \times M^p)_{i,j} = (M^{p+1})_{i,j}$.

Conclusion : la propriété est bien vraie pour tout $n \geq 1$. □

Remarques :
- il ne faut pas oublier les parenthèses dans $(M^p)_{i,j}$. Effectivement il s'agit d'élever d'abord M à la puissance p, puis de prendre le coefficient i,j ; et non pas de prendre d'abord le coefficient i,j de la matrice M, puis de l'élever à la puissance p.
- ce résultat nous permet de dénombrer les chemins de longueur quelconque allant d'un sommet à un autre, mais il ne nous dit en aucun cas par quels sommets passent ces chemins. Pour le savoir, il suffira de regarder le dessin et d'effectuer le parcours « à la main » ... néanmoins le B.O. tient à un algorithme : il est donné à la page suivante.

Exemple : soit le graphe suivant de matrice d'adjacence M :



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

Le calcul donne $M^2 = \begin{pmatrix} 2 & 0 & 3 & 1 \\ 0 & 1 & 1 & 1 \\ 2 & 0 & 2 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$, ce qui permet par exemple de déduire que :

- Il y a trois chemins de longueur 2 allant de a jusqu'à c car $(A^2)_{1,3} = 3$. Ce sont les chemins : (a, a, c) ; (a, d, c) et (a, c, c).
- Il y a 5 circuits de longueur 2 dans ce graphe : deux de a vers a car $(A^2)_{1,1} = 2$, un de b vers b car $(A^2)_{2,2} = 1$, deux de c vers c car $(A^2)_{3,3} = 2$ et aucun de d vers d car $(A^2)_{4,4} = 0$.
- Il y a 7 chemins de longueur 2 allant vers c dans ce graphe : trois depuis a car $(A^2)_{1,3} = 3$, un depuis b car $(A^2)_{2,3} = 1$, deux depuis c car $(A^2)_{3,3} = 2$ et un depuis d car $(A^2)_{4,3} = 1$.
- Il y a en tout 16 chemins de longueur 2 dans ce graphe.

L'algorithme pour trouver les chemins est le suivant :

Détermination des chemins de longueur donnée.

Entrée :
Un graphe $G = (V; E)$, un sommet $a \in V$ et un nombre entier non nul p.

Sortie et rôle de l'algorithme :
L'ensemble C des chemins du graphe G, de longueur p et commençant en a.

Variables utilisées :
C_1 et C_2, ensembles de chemins
b et c, sommets
k, nombre entier

Corps de l'algorithme :

```
1    C1 ← ∅
3    Pour chaque successeur b de a, faire
4        C1 ← C1 ∪ {(a, b)}
5    Fin Pour
6    Pour k de 2 à p, faire
7        C2 ← ∅
8        Pour chaque chemin (a, ..., b) de C1, faire
9            Pour chaque successeur c de b, faire
10               C2 ← C2 ∪ {(a, ..., b, c)}
11           Fin Pour
12       Fin Pour
13       C1 ← C2
14   Fin Pour
15   Renvoyer C1
```

✓ **Bulletin Officiel**
Le B.O. attend qu'un étudiant soit capable d'obtenir les chemins de longueur p donnée dans un graphe. En pratique, parce que l'épreuve ne dure que 2h, il est fort peu probable qu'il ait besoin de mettre en œuvre ce dernier algorithme : il faut par contre savoir interpréter les coefficients de M^p puis retrouver « à la main » les chemins.

3.3 Accessibilité

D'après la section précédente, il est évident que $(M^{[p]})_{i,j}$ est un booléen qui vaut 1 lorsqu'il existe un chemin de i à j de longueur p, 0 sinon. Effectivement, pour obtenir les coefficients de la matrice booléenne, il suffit de calculer M^p, puis de remplacer tout nombre non nul par 1. Et puisqu'il y a un nombre non nul s'il y a un nombre non nul de chemins entre i et j...

Nous venons de comprendre comment déterminer l'existence d'un chemin de longueur fixée entre deux sommets i et j. Une question naturelle est alors de déterminer l'existence d'un chemin *tout court* entre ces deux sommets, peu importe la longueur de ce chemin.

Pour cela, il suffit de savoir s'il existe un chemin de longueur 1 (facile : l'arc existe-t-il ?), ou bien s'il existe un chemin de longueur 2, ou bien s'il existe un chemin de longueur 3... Jusque quelle longueur aller comme cela ? Partons de i, et parcourons le graphe aussi longtemps que nous le désirons, pour essayer d'atteindre j. Au bout d'un certain temps, nous allons forcément repasser au même endroit, puisqu'il y a un nombre fini de sommets. Nous avons donc parcouru notre graphe « pour rien » : nous avons emprunté un circuit !

Propriété

Tout chemin de longueur supérieure ou égale à n contient un circuit.

Preuve

Étant donné qu'il y a n sommets dans le graphe, tout chemin qui passe par $n+1$ sommets passe donc forcément deux fois par le même sommet, et contient donc un circuit. □

Ainsi, pour savoir s'il existe un chemin entre i et j, il suffit de regarder les chemins de longueur 1, 2... n. Au-delà, c'est inutile, puisque s'il existe un chemin de longueur strictement supérieure à n entre i et j, il contient un circuit, et donc, en coupant ce circuit, cela donne un autre chemin de longueur strictement inférieure qui lie toujours i à j. Nous sommes cela dit obligés d'aller jusqu'à n puisque pour savoir si i est atteignable depuis lui-même, il faut parfois aller chercher précisément le circuit de taille n (il peut ne pas y en avoir de plus court).

Pour tout graphe, la matrice d'accessibilité booléenne $\widehat{M}$ peut donc se calculer par la somme booléenne des puissances booléennes de la matrice d'adjacence M :

$$\widehat{M} = M \oplus M^{[2]} \oplus \cdots \oplus M^{[n]}$$

Complexité

- en effectuant naïvement le calcul ci-dessus, le calcul de $\widehat{M}$ est en $O(n^4)$. Effectivement il y a $n-1$ multiplications matricielles à effectuer, chacune en $O(n^3)$ avec la formule $(AB)_{i,j} = \sum (A)_{i,k}(B)_{k,j}$. Il y a ensuite $n-1$ additions à effectuer, chacune en $O(n^2)$.
- avec des algorithmes de multiplication matricielle optimisés, cette complexité peut être améliorée. La meilleure complexité possible pour le produit matriciel est aujourd'hui encore inconnue, cela dit la méthode de Strassen [a] fournit du $O(n^{\log_2 7}) \approx O(n^{2,81})$ soit mieux que n^3.
- en remarquant que $M \oplus \left(M \oplus \cdots \oplus M^{[2^p]}\right)^{[2]} = M \oplus \cdots \oplus M^{[2^{p+1}]}$, le nombre de multiplications matricielles et le nombre d'additions matricielles à effectuer passe de n à $\log_2 n$; en combinant ces deux méthodes, la borne supérieure de la complexité de cet algorithme est $O(n^{2,81} \log_2 n)$.

a. L'algorithme de Strassen est de type « diviser pour régner » : il divise une matrice A en 4 blocs de taille moitié et réussit à calculer $A \times A$ avec 7 multiplications des blocs au lieu de 8 par méthode naïve. La démonstration de la complexité de cet algorithme utilise le « Master Theorem » qui permet de déterminer les complexités de ce type d'algorithmes. Se référer à l'excellente *Introduction to Algorithms* par Cormen, Leiserson, Rivest, et Stein.

✓**Bulletin Officiel**
Le B.O. attend qu'un étudiant soit capable d'obtenir et d'interpréter les coefficients de $M^{[p]}$.

3.4 Fermeture transitive

Soit $G = (V; E)$ un graphe. Appelons $\mathcal{R}$ la relation définie sur V par :

$$a\mathcal{R}b \iff (a, b) \in E$$

La fermeture transitive (ou clôture transitive) de G est le graphe $\widehat{G} = (V; \widehat{E})$ obtenu en ajoutant à partir de E un nombre minimum d'arcs possible de manière à ce que la relation $\widehat{\mathcal{R}}$ définie sur V par $a\widehat{\mathcal{R}}b \iff (a, b) \in \widehat{E}$ soit transitive.

Remarque : ajouter tous les arcs possibles rendrait la relation transitive... mais donnerait à chaque fois un graphe complet [4] : pas très intéressant.

L'algorithme naïf qui utilise la définition de la fermeture transitive consiste simplement à ajouter des arcs qui doivent être ajoutés pour que $\mathcal{R}$ soit transitive tant qu'il en existe :

Construction naïve de la fermeture transitive.

Entrée :
Un graphe $G = (V; E)$.

Sortie et rôle de l'algorithme :
Le graphe $\widehat{G}$, fermeture transitive de G.

Variables utilisées :
E', ensemble d'arcs
G', graphe

Corps de l'algorithme :
```
1   E' ← E
2   Répéter
3       G' ← (V; E')
4       Pour chaque arc (a, b) de G', faire
5           Pour chaque successeur c de b dans G', faire
6               Si (a, c) ∉ E', alors
7                   E' ← E' ∪ {(a, c)}
8               Fin Si
9           Fin Pour
10      Fin Pour
11  Jusqu'à ce que plus aucun arc ne soit ajouté
12  Renvoyer G'
```

Preuve de l'algorithme

1. Tous les arc de E' sont nécessaires. Au début tous les arcs de E sont dans E' : par définition de la fermeture transitive, ils sont nécessaires. Ensuite l'algorithme ajoute (a, c) dans E' si et seulement si $\exists b \in V, (a, b) \in E' \wedge (b, c) \in E'$. Cet arc est nécessaire à ajouter, sinon $\mathcal{R}'$ ne serait pas transitive.

2. De plus les arcs ajoutés sont suffisants. Quand l'algorithme n'en ajoute plus, c'est que pour chaque arc (a, b) de G', pour chaque successeur c de b dans G', $(a, c) \in E'$, ce qui veut bien dire que $\mathcal{R}'$ est transitive.

Conclusion : $\mathcal{R}$ est transitive en ayant ajouté le moins d'arcs possibles. □

4. Une clique (hors programme), ou graphe complet, est un graphe dans lequel tous les arcs sont présents.

Théorème

La matrice d'adjacence de $\widehat{G}$ est la matrice d'accessibilité $\widehat{M}$ de G.

Preuve

Il existe un chemin $(a, s_1, \ldots, s_n, b)$ dans G si et seulement si (a, b) est dans la fermeture transitive du graphe, tout simplement par transitivité. □

Théorème

L'algorithme naïf fait au plus $\log_2 n$ passages dans la boucle « Répéter ».

Preuve

Montrons par récurrence sur i, le nombre de passages dans la boucle « Répéter », l'invariant de boucle suivant : au début du i-ième passage dans la boucle, $(a, b) \in E'$ si et seulement s'il y a un chemin de longueur inférieur ou égale à 2^{i-1} de a à b dans G.

Initialisation : l'invariant est vrai au rang 1 car $(a, b) \in E' \iff (a, b) \in E$ puisque E' est une copie de E. Il n'y a pas d'autres chemins de longueur $2^{1-1} = 1$.

Hérédité : supposons l'invariant vrai au rang p. Soient a et b deux sommets du graphe. Le début du $p+1$-ième passage correspond à la fin du p-ième passage. Montrons qu'à la fin de ce p-ième passage, $(a, b) \in E'$ si et seulement s'il y a un chemin de longueur inférieur ou égale à 2^{i-1} de a à b dans G.

Sens direct : (a, b) sera dans E' si et seulement s'il existe c dans V avec (a, c) dans E' et (c, b) dans E' au début du p-ième passage ou que (a, b) y était déjà. Dans le premier cas, par hypothèse de récurrence il y a un chemin de longueur inférieure ou égale à 2^{p-1} de a à c et un autre de longueur inférieure ou égale à 2^{p-1} de c à b donc en les mettant bout à bout il existe un chemin de longueur inférieure ou égale à 2^p de a à c. Dans le second cas, par hypothèse de récurrence il existe un chemin de longueur inférieure ou égale à 2^{p-1} donc à 2^p de a à b.

Sens réciproque : s'il existe un chemin de longueur inférieure ou égale à 2^p de a à b : s'il est de longueur inférieure ou égale à 2^{p-1}, il y était au début du passage donc y est à la fin. Sinon, c'est la concaténation de deux (en coupant au milieu) chemins non vides $a \rightsquigarrow c$ et $c \rightsquigarrow b$ de longueurs chacune inférieure ou égale à 2^{p-1}, et donc par hypothèse de récurrence, (a, c) et (c, b) sont dans G' au début du p-ième passage, donc (a, b) sera ajouté.

Ainsi à la fin du p-ième passage, les arcs de E' vérifient la propriété : au début du $p+1$-ième, ils la vérifient donc !

Conclusion : l'invariant est bien vrai pour tout $i \geq 1$. Puisque nous avons déjà vu qu'il existe un chemin de a vers b si et seulement s'il existe un chemin de longueur $\leq n$ de a vers b, cela permet de déduire qu'il y a au plus $\log_2 n$ passages dans la boucle « Répéter ». □

Complexité

En majorant de manière grossière le nombre d'arcs dans E' par $\dfrac{n(n-1)}{2}$ (graphe complet), nous obtenons une borne supérieure en $O(n^3 \log_2 n)$ ($\log_2 n$ pour la boucle « Répéter » puis n^2 pour la boucle « Pour chaque arc » puis n pour la boucle « Pour chaque successeur »).

Nous pouvons aussi calculer $\widehat{M}$ par la méthode précédente pour obtenir ce graphe. Quelle que soit la méthode utilisée, une fois la fermeture du graphe construite, nous avons ensuite en

temps constant l'accessibilité d'un sommet à un autre : il suffit de regarder si l'arc correspondant se trouve dans le nouveau graphe ! Il est donc intéressant de construire ce graphe une fois pour toutes.

Exemple : sur la gauche un graphe G, et à droite sa fermeture transitive $\widehat{G}$:

Il a fallu ajouter (a, a) car (a, b) et (b, a) existent ; (b, b) car (b, a) et (a, b) existent ; (b, c) car (b, a) et (a, c) existent...

⚠ Il ne faut pas oublier d'ajouter (e, f) dans la fermeture transitive. Dans le graphe initial, il n'y a pas de sommet i tel que les arcs (e, i) et (i, f) existent à la fois. Cela dit, il faut ajouter l'arc (e, g) car (e, d) et (d, g) existent... et alors la transitivité oblige à ajouter (e, f).

Nous pouvons enfin présenter l'Algorithme de Roy-Warshall, qui a le bon goût d'être très simple à écrire. Cet algorithme est également intéressant car il s'étend facilement au calcul de chemins optimaux, que nous verrons plus tard.

Algorithme de Roy-Warshall.

Entrée :
La matrice d'adjacence A d'un graphe G.

Sortie et rôle de l'algorithme :
La matrice A' du graphe $\widehat{G}$, fermeture transitive de G.

Variables utilisées :
A', matrice de booléens
p, i, j, nombres entiers

Corps de l'algorithme :

```
1   n ← taille(A)          Étant entendu que c'est une matrice carrée...
2   A' ← A
3   Pour p de 1 à n, faire
4       Pour i de 1 à n, faire
5           Pour j de 1 à n, faire
6               (A')_{i,j} ← (A')_{i,j} ⊕ (A')_{i,p} × (A')_{p,j}
7           Fin Pour
8       Fin Pour
9   Fin Pour
10  Renvoyer A'
```

Preuve de l'algorithme

Nous montrerions par récurrence sur p l'invariant de boucle suivant : au début du p-ième passage dans la boucle (lignes 3-9), $(A')_{i,j} = 1$ si et seulement s'il existe un chemin de i à j dont tous les sommets intermédiaires s vérifient $s \leq p$. □

⌛ Complexité

Cet algorithme est en $O(n^3)$. Il n'a besoin d'aucune optimisation pour atteindre cette complexité, contrairement au calcul de la matrice d'accessibilité présenté plus haut en $O(n^{2,81} \log_2 n)$. Cela dit il est moins rapide, car $\log_2 n << n^{0,19}$.

Remarque : attention à ne pas confondre $\widehat{M}$ avec la matrice M^* définie par :

$$M^* = I_n \oplus M \oplus M^{[2]} \oplus \cdots \oplus M^{[n]}$$

Il s'agit ici de la matrice de la fermeture *réflexive* et transitive du graphe de matrice d'adjacence M, qui n'est pas au programme. Effectivement rajouter I_n fait que $\forall x, x\mathcal{R}x$ donc la relation a été rendue réflexive.

✓ **Bulletin Officiel**

Le B.O. attend qu'un étudiant soit capable de mettre en œuvre un algorithme pour la fermeture transitive d'un graphe : il est évident que toutes les méthodes ne peuvent pas être présentées en classe, et qu'il suffit donc que les étudiants maîtrisent bien l'une d'entre elles.

Chapitre 4

Graphes orientés sans circuit

4.1 Reconnaître un graphe sans circuit

Pour savoir si un graphe est sans circuit, il suffit de vérifier qu'il n'en contient pas... mais il est difficile de démontrer que quelque chose n'existe pas, alors partons du constat suivant :

Constat
Si $G = (V; E)$ est un graphe orienté sans circuit, alors il existe $v \in V$ tel que v n'a pas de prédécesseur.

Preuve

Puisque G n'a pas de circuit, alors la taille des chemins dans G est bornée (nous avons déjà vu ce résultat). Soit alors un chemin $(v_0, v_1, \ldots, v_k)$ de longueur maximale. Supposons que v_0 ait un prédécesseur p : alors $(p, v_0, v_1, \ldots, v_k)$ serait un chemin de G, ce qui contredit la maximalité du chemin $(v_0, v_1, \ldots, v_k)$. Ainsi v_0 est sans prédécesseur. □

Nous aboutissons alors au théorème, un peu plus intéressant, suivant :

Théorème
Soit $G = (V; E)$ un graphe. Alors G est sans circuit si et seulement s'il existe $v \in V$ tel que v soit sans prédécesseur et que pour tout $s \in V$ sans prédécesseur, le sous-graphe obtenu à partir de G en retirant le sommet s ainsi que tous les arcs ayant pour extrémité s soit sans circuit.

Preuve

Sens direct : si G est sans circuit, alors d'après notre constat, il existe $v \in V$ tel que v n'a pas de prédécesseur. Puisque G est sans circuit, en retirant un sommet et des arcs, il reste sans circuit.

Sens réciproque : soit $v \in V$ sans prédécesseur. Le sous-graphe obtenu à partir de G, en retirant le sommet v ainsi que tous les arcs ayant pour extrémité v, est donc sans circuit. Donc si G possédait un circuit, il passerait nécessairement par v, qui aurait donc un prédécesseur. D'où la contradiction, donc G est sans circuit. □

Remarque : les mêmes résultats sont bien sûr vrais en remplaçant « sans prédécesseur » par « sans successeur », la démonstration est similaire en tout point.

L'algorithme de reconnaissance d'un graphe sans circuit tombe alors naturellement. En entrée, un graphe $G = (V; E)$ et en sortie l'affichage d'un message indiquant si G contient des circuits ou non.

Reconnaissance naïve d'un graphe sans circuit.

Variables utilisées : v et s, sommets

Corps de l'algorithme :

```
1   Tant qu'il existe un sommet sans prédécesseur, faire
2       v ← un tel sommet
3       V ← V \ {v}
4       Pour chaque successeur s de v, faire
5           E ← E \ {(v, s)}
6       Fin Pour
7   Fin Tant que
8   Si V = ∅, alors
9       Afficher « G est sans circuit »
10  Sinon
11      Afficher « G contient des circuits »
12  Fin Si
```

Complexité

Ce premier jet d'algorithme est en $O(n^2)$. Effectivement, sans traitement particulier, trouver un sommet sans prédécesseur est en $O(n)$; cette opération s'effectue jusqu'à n fois, donc la complexité totale est en $O(n^2)$ (car l'algorithme supprime au plus $m \leq n^2$ arêtes). Cela dit, il existe une meilleure manière de procéder...

Reconnaissance d'un graphe sans circuit.

Variables supplémentaires utilisées : *degre_entrant*, tableau de n entiers ; *a_traiter*, ensemble de sommets ; *sommets_traites*, entier

Corps de l'algorithme :

```
1   Initialiser toutes les cases de degre_entrant par 0
2   Pour chaque sommet v de V, faire
3       Pour chaque successeur s de v, faire
4           degre_entrant(v) ← degre_entrant(v) + 1
5       Fin Pour
6   Fin Pour
7   a_traiter ← ∅
8   Pour chaque sommet v de V, faire
9       Si degre_entrant(v) = 0, alors
10          a_traiter ← a_traiter ∪ {v}
11      Fin Si
12  Fin Pour
13  sommets_traites ← 0
14  Tant que a_traiter ≠ ∅, faire
15      v ← un sommet de a_traiter
16      a_traiter ← a_traiter \ {v}
17      Pour chaque successeur s de v, faire
18          degre_entrant(s) = degre_entrant(s) − 1
19          Si degre_entrant(s) = 0, alors
20              a_traiter ← a_traiter ∪ {s}
21          Fin Si
22      Fin Pour
23      sommets_traites ← sommets_traites + 1
24  Fin Tant que
25  Si sommets_traites = |V|, alors
26      Afficher « G est sans circuit »
27  Sinon
28      Afficher « G contient des circuits »
29  Fin Si
```

Nous avons ici géré « intelligemment » le parcours des sommets à traiter, en introduisant la notion hors programme de degré entrant du sommet i : c'est le nombre de prédécesseurs de i (notée $|\Gamma^-(i)|$). Il suffit, plutôt que de parcourir tous les sommets à chaque fois, à la recherche d'un sommet sans prédécesseur, de tenir à jour, pour chaque sommet, une variable *degre_entrant*, et de ne traiter que ceux qui ont un *degre_entrant* nul. Cela revient à traiter les sommets par niveaux, comme nous le verrons dans la section suivante.

Complexité
Ce second algorithme est en $O(n + m)$. Effectivement, l'initialisation de *degre_entrant* (ligne 1) est en $O(n)$, la première boucle (lignes 2-6) est en $O(n + m)$, la seconde boucle (lignes 8-12) en $O(n)$ et la dernière boucle (lignes 14-24) en $O(n + m)$.

Pour travailler matriciellement, il suffit de faire les constats suivants :

- un sommet i n'a aucun prédécesseur si la colonne i n'est remplie que de 0
- un sommet i n'a aucun successeur si la ligne i n'est remplie que de 0.

L'algorithme de reconnaissance utilisant la matrice d'adjacence est donc le suivant :

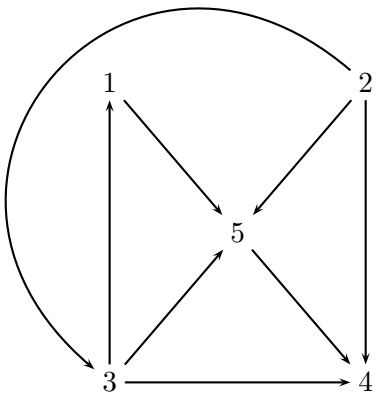
Reconnaissance d'une matrice d'adjacence de graphe sans circuit.

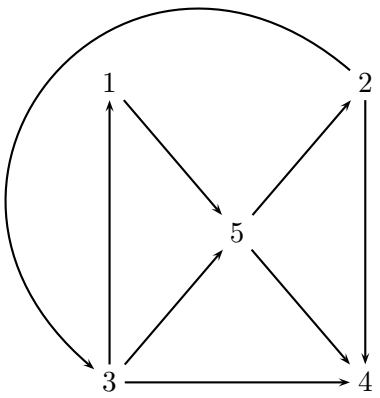
Entrée :
Une matrice d'adjacence M d'un graphe G.

Sortie et rôle de l'algorithme :
Afficher un message indiquant si G contient des circuits ou non.

Corps de l'algorithme :
1 Tant qu'il existe une colonne ou une ligne remplie de 0, faire
2 $i \leftarrow$ le numéro d'une telle ligne ou d'une telle colonne
3 Supprimer la i-ième ligne de M
4 Supprimer la i-ième colonne de M
5 Fin Tant que
6 Si M est de taille 0×0, alors
7 Afficher « G est sans circuit »
8 Sinon
9 Afficher « G contient des circuits »
10 Fin Si

Exemple : soient G et H les deux graphes suivants :





Il n'est pas évident de « voir » si l'un des graphes est sans circuit, alors qu'il n'y a que 5 sommets. Travaillons sur leurs matrices d'adjacence qui sont :

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

Pour le graphe G :

- M contient une colonne remplie de 0 (la 2^{eme}) : cela veut dire que le sommet 2 n'a pas de prédécesseur. L'algorithme peut le supprimer et regarder dans le graphe restant. Dans la matrice, il faudrait supprimer la 2^{eme} ligne ainsi que la 2^{eme} colonne.
- M contient une ligne remplie de 0 (la 4^{eme}) : cela veut dire que le sommet 4 n'a pas de successeur. L'algorithme peut le supprimer et regarder dans le graphe restant. Dans la matrice, il faudrait supprimer la 4^{eme} ligne ainsi que la 4^{eme} colonne.

Supprimons le sommet 2 ; la matrice devient $\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

Cette nouvelle matrice correspond donc au graphe G affranchi du sommet 2. Ainsi les sommets restants sont les sommets 1, 3, 4, 5.

- Il y a une colonne remplie de 0 (la 2^{eme}) : cela veut dire que le sommet 3 (attention au décalage !) n'a pas de prédécesseur. L'algorithme peut le supprimer et regarder dans le graphe restant. Dans la matrice, il faudrait supprimer la 2^{eme} ligne ainsi que la 2^{eme} colonne.
- Il y a une ligne remplie de 0 (la 3^{eme}) : cela veut dire que le sommet 4 (attention au décalage !) n'a pas de successeur. L'algorithme peut le supprimer et regarder dans le graphe restant. Dans la matrice, il faudrait supprimer la 3^{eme} ligne ainsi que la 3^{eme} colonne.

Supprimons le sommet 3 ; la matrice devient $\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

Cette nouvelle matrice correspond donc au graphe G affranchi des sommets 2 et 3. Ainsi les sommets restants sont les sommets 1, 4, 5.

- Il y a une colonne remplie de 0 (la 1^{ere}) : cela veut dire que le sommet 1 n'a pas de prédécesseur. L'algorithme peut le supprimer et regarder dans le graphe restant. Dans la matrice, il faudrait supprimer la 1^{ere} ligne ainsi que la 1^{ere} colonne.
- Il y a une ligne remplie de 0 (la 2^{eme}) : cela veut dire que le sommet 4 (attention au décalage !) n'a pas de successeur. L'algorithme peut le supprimer et regarder dans le graphe restant. Dans la matrice, il faudrait supprimer la 2^{eme} ligne ainsi que la 2^{eme} colonne.

Supprimons le sommet 1 ; la matrice devient $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$

Cette nouvelle matrice correspond donc au graphe G affranchi des sommets 1, 2 et 3. Ainsi les sommets restants sont les sommets 4 et 5.

- Il y a une ligne remplie de 0 (la 1^{ere}) : cela veut dire que le sommet 4 (attention au décalage !) n'a pas de prédécesseur. L'algorithme peut le supprimer et regarder dans le graphe restant. Dans la matrice, il faut supprimer la 1^{ere} ligne ainsi que la 1^{ere} colonne.

Supprimons le sommet 4 ; la matrice devient $\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = (0)$

Cette nouvelle matrice correspond donc au graphe G affranchi des sommets 1, 2, 3 et 4. Ainsi il ne reste plus que le sommet 5.

- Il y a une ligne remplie de 0 (la 1^{ere}) : cela veut dire que le sommet 5 (attention au décalage !) n'a pas de prédécesseur. L'algorithme peut le supprimer et regarder dans le

graphe restant. Dans la matrice, il faut supprimer la 1ere ligne ainsi que la 1ere colonne, ce qui aboutit clairement à la matrice vide : le graphe G est donc sans circuit.

Mêmes étapes pour H :

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

La matrice résultante ne contient plus ni ligne ni colonne remplie de 0.
Ainsi le graphe H contient un circuit.

Remarque : pour trouver un circuit, il suffit de parcourir le graphe H affranchi du sommet 4 (à partir de n'importe quel sommet), ce qui aboutira forcément à un circuit ; cela dit le circuit trouvé ne commencera pas nécessairement par le sommet choisi en premier.

Autre méthode : nous avons déjà remarqué qu'un circuit de longueur p se traduit par un nombre non nul sur la diagonale de la matrice M^p. Au lieu de barrer des lignes et des colonnes sur M, il suffit de calculer les puissances de la matrice d'adjacence, de M^1 à M^n et de s'assurer qu'aucune de ces n matrices n'a de coefficient non nul sur sa diagonale !

4.2 Niveaux

La représentation visuelle de certains graphes peut être parfois « fouillie » si le dessinateur n'a pris aucune précaution particulière. Dans un graphe sans circuit, c'est une bonne idée d'adopter une représentation qui rend très visuelle la hiérarchie des sommets dans le graphe : en haut (niveau 0) ceux qui sont sans prédécesseur, plus bas (niveau 1) ceux dont tous les prédécesseurs étaient parmi ceux du haut, plus bas (niveau 2) ceux dont tous les prédécesseurs étaient parmi ceux des deux niveaux précédents...

Puisqu'un graphe représente généralement un problème très concret, faire apparaître la hiérarchie entre les sommets permet de mieux comprendre le problème auquel il se rapporte. L'ordonnancement de tâches est un problème qui peut se résoudre par application directe de la décomposition en niveaux : étant données différentes tâches à exécuter en vue de mener un projet à bien, et différentes contraintes entre ces tâches, dans quel ordre les exécuter ?

Plus formellement, dans un graphe G sans circuit, le niveau d'un sommet i est la longueur d'un plus long chemin dans G se terminant en i.

Cette définition donne lieu à un calcul des niveaux pénible. Il existe une définition équivalente et plus simple à calculer : la fonction hauteur définie sur V et à valeurs dans $\mathbb{N}$ de manière récursive par :

$$x \mapsto \begin{cases} 0 & \text{si } x \text{ est sans prédécesseur} \\ 1 + \max\limits_{y \text{ prédécesseur de } x} h(y) & \text{sinon} \end{cases}$$

Nous verrons comme autre application directe de la décomposition en niveaux l'algorithme de Bellman (au chapitre suivant), qui a besoin de parcourir les sommets dans l'ordre d'un tri topologique (notion hors programme, mais dont la décomposition en niveaux est, à un petit abus de langage près, un cas particulier).

La définition de la fonction hauteur aboutit à l'algorithme suivant :

Calcul de niveaux.

Entrée :
Un graphe $G = (V; E)$.

Sortie et rôle de l'algorithme :
Étiqueter chaque sommet de V par son niveau.

Variables utilisées :
$Niveaux$, tableau de $n + 1$ ensembles de sommets
v et s, sommets
i, nombre entier

Corps de l'algorithme :

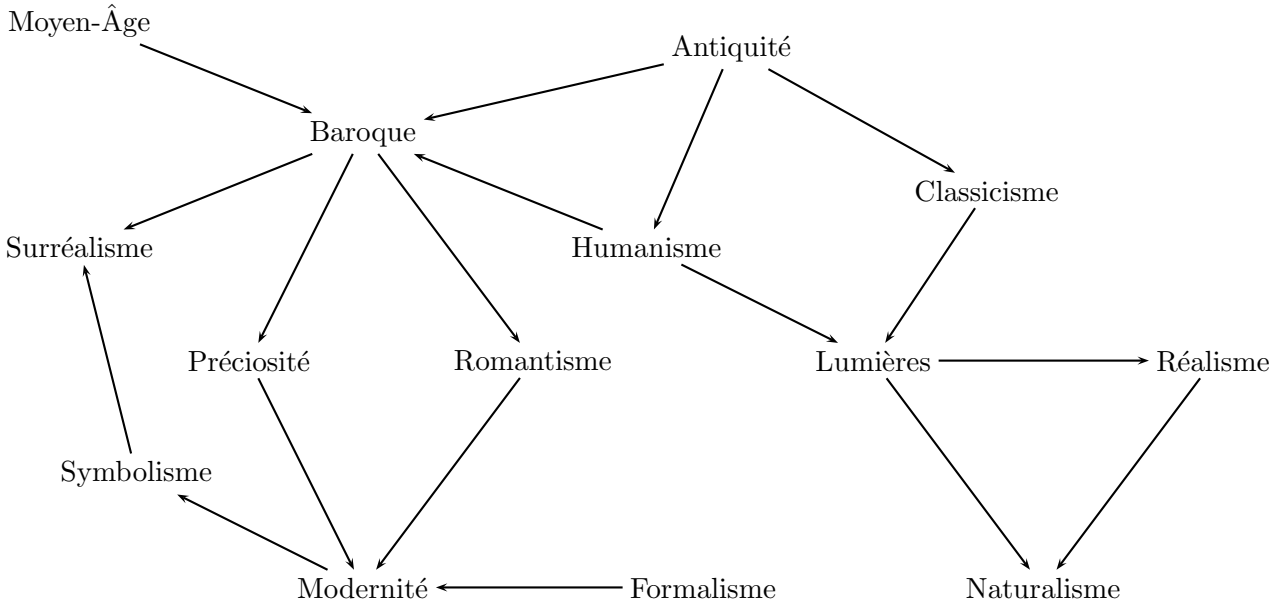
```
1   Initialiser toutes les cases de Niveaux par ∅
2   i ← 0
3   Pour chaque sommet v de V, faire
4       Si v est sans prédécesseur, alors
5           Niveaux[0] ← Niveaux[0] ∪ {v}
6       Fin Si
7   Fin Pour
8   Tant que Niveaux[i] ≠ ∅, faire
9       Pour chaque sommet v de Niveaux[i], faire
10          Étiqueter v par i
11          Pour chaque successeur s de v, faire
12              Retirer l'arc (v, s) de G
13              Si s est sans prédécesseur, alors
14                  Niveaux[i + 1] ← Niveaux[i + 1] ∪ {s}
15              Fin Si
16          Fin Pour
17      Fin Pour
18      i ← i + 1
19  Fin Tant que
```

⌛ Complexité
Comme dit précédemment, cet algorithme est en $O(n + m)$. Effectivement l'initialisation de la variable $Niveaux$ et la première boucle (lignes 3-7 : « Pour chaque sommet v de V ») sont en $O(n)$, puis la seconde boucle (lignes 8-19) parcourt exactement une fois chaque sommet (une fois qu'un sommet a été visité, il n'est pas ajouté dans une autre case de $Niveaux$) et une fois chaque arc (puisque pour chaque sommet visité, chaque arc sortant de ce sommet est parcouru une unique fois), donc est en $O(n + m)$.

Remarque : la définition initiale permet également de travailler matriciellement : dans M, les colonnes vides correspondent aux sommets de niveau 0. Puis pour i de n à 1 (la boucle peut s'arrêter plus tôt, dès que tous les niveaux ont été calculés), si un sommet n'a pas encore de niveau et que dans une ligne de M^i il y a un nombre non nul à la colonne j, alors le sommet j a le niveau i. Cette méthode matricielle est par contre beaucoup plus coûteuse en complexité !

Comme nous l'avons dit en début de section, une fois le graphe décomposé par niveaux, nous pouvons le représenter de manière bien plus parlante. Nous allons détailler un exemple, où les sommets sont différents courants littéraires, et les arcs indiquent les influences [1]. La première représentation est à dessein dessinée sans souci particulier d'ordre, pour que l'algorithme de décomposition par niveaux prenne tout son sens.

1. Se référer à l'excellente *Histoire de la littérature française*, de Xavier Darcos, pour étoffer ce graphe.



Initialisation (lignes 1-2) :

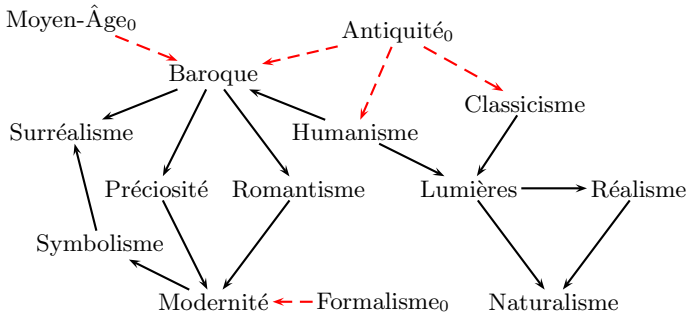
- i vaut 0, $Niveaux$ vaut $[\emptyset, \emptyset, \dots]$

Première boucle (lignes 3-7) :

- les sommets sans prédécesseurs sont Moyen-Âge, Antiquité et Formalisme. Ainsi l'algorithme change la case $Niveaux[0]$ en {Moyen-Âge; Antiquité; Formalisme}.

Seconde boucle (lignes 8-19), avec $i = 0$:

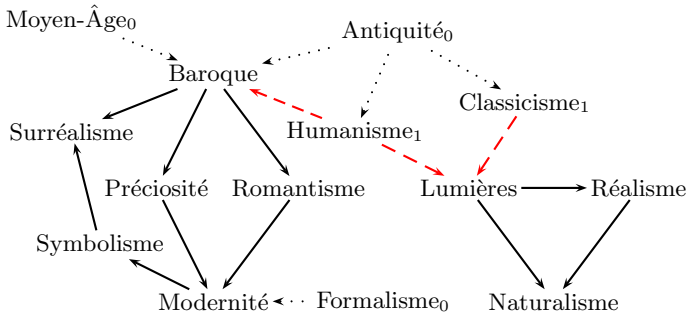
- l'algorithme étiquette Moyen-Âge, Antiquité et Formalisme par 0
- l'algorithme retire tous les arcs provenant de Moyen-Âge, Antiquité et Formalisme :



- l'algorithme regarde quels nouveaux sommets se retrouvent sans prédécesseurs : Humanisme et Classicisme.
- ainsi $Niveaux[1]$ vaut maintenant {Humanisme; Classicisme}.

Seconde boucle (lignes 8-19), avec $i = 1$:

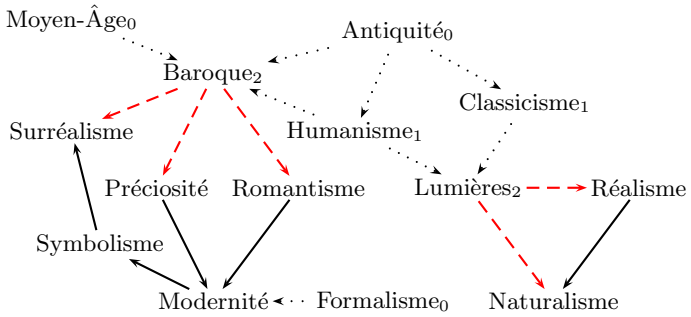
- l'algorithme étiquette Humanisme et Classicisme par 1
- l'algorithme retire tous les arcs provenant de Humanisme et Classicisme :



- l'algorithme regarde quels nouveaux sommets se retrouvent sans prédécesseurs : Baroque et Lumières.
- ainsi $Niveaux[2]$ vaut maintenant $\{\text{Baroque}; \text{Lumières}\}$.

Seconde boucle (lignes 8-19), avec $i = 2$:

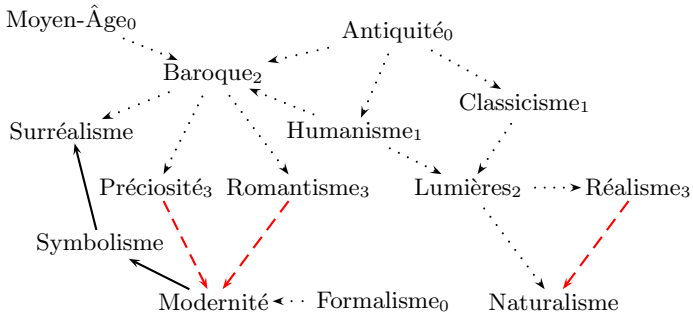
- l'algorithme étiquette Baroque et Lumières par 2
- l'algorithme retire tous les arcs provenant de Baroque et Lumières :



- l'algorithme regarde quels nouveaux sommets se retrouvent sans prédécesseurs : Préciosité, Romantisme et Réalisme.
- ainsi $Niveaux[3]$ vaut maintenant $\{\text{Préciosité}; \text{Romantisme}; \text{Réalisme}\}$.

Seconde boucle (lignes 8-19), avec $i = 3$:

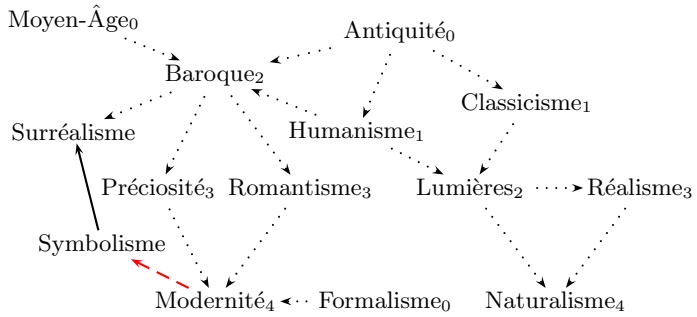
- l'algorithme étiquette Préciosité, Romantisme et Réalisme par 3
- l'algorithme retire tous les arcs provenant de Préciosité, Romantisme et Réalisme :



- l'algorithme regarde quels nouveaux sommets se retrouvent sans prédécesseurs : Modernité et Naturalisme.
- ainsi $Niveaux[4]$ vaut maintenant $\{\text{Modernité}; \text{Naturalisme}\}$.

Seconde boucle (lignes 8-19), avec $i = 4$:

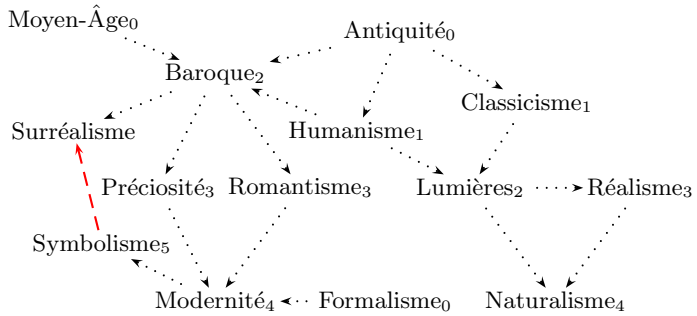
- l'algorithme étiquette Modernité et Naturalisme par 4
- l'algorithme retire tous les arcs provenant de Modernité et Naturalisme :



- l'algorithme regarde quels nouveaux sommets se retrouvent sans prédécesseurs : Symbolisme.
- ainsi $Niveaux[5]$ vaut maintenant {Symbolisme}.

Seconde boucle (lignes 8-19), avec $i = 5$:

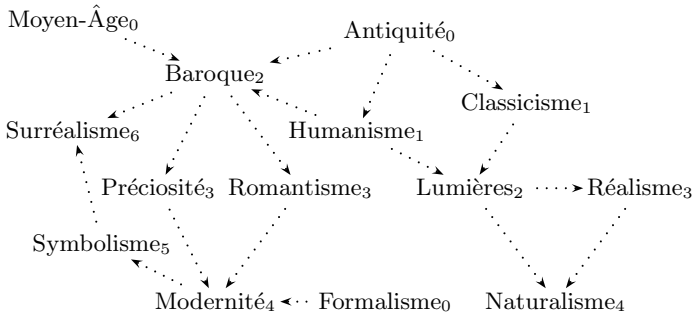
- l'algorithme étiquette Symbolisme par 5
- l'algorithme retire tous les arcs provenant de Symbolisme :



- l'algorithme regarde quels nouveaux sommets se retrouvent sans prédécesseurs : Surréalisme.
- ainsi $Niveaux[6]$ vaut maintenant {Surréalisme}.

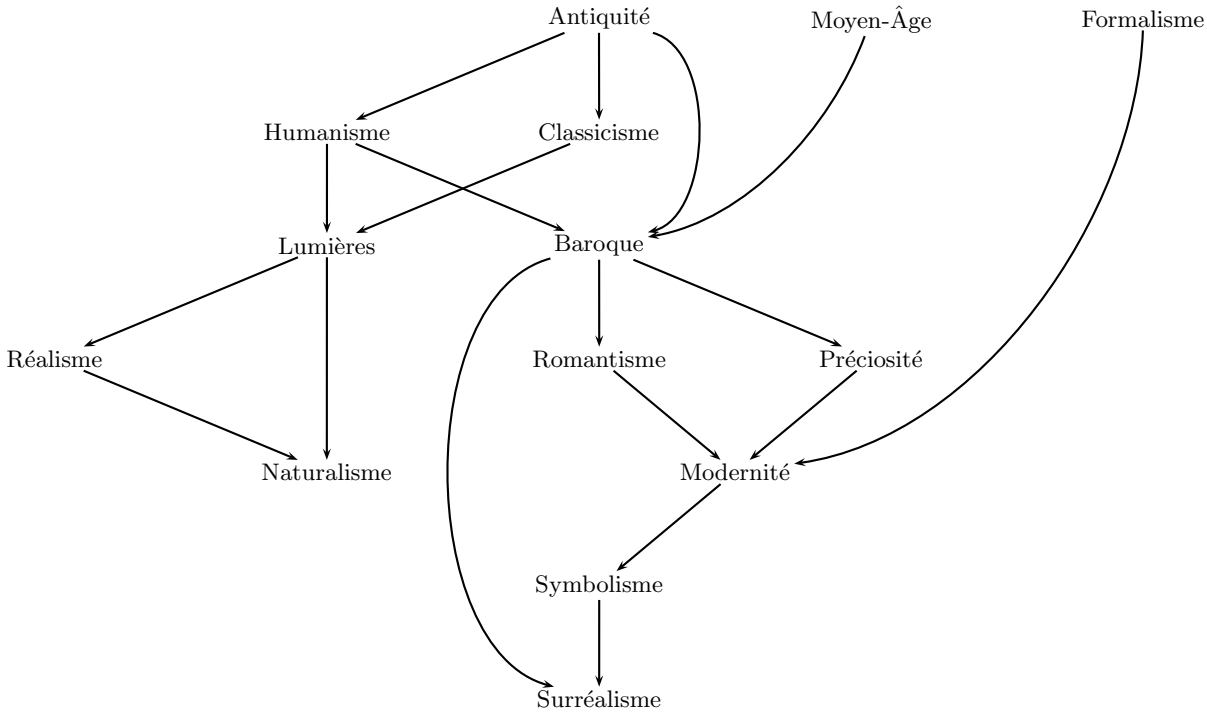
Seconde boucle (lignes 8-19), avec $i = 6$:

- l'algorithme étiquette Surréalisme par 6
- l'algorithme retire tous les arcs provenant de Surréalisme :



- l'algorithme regarde quels nouveaux sommets se retrouvent sans prédécesseurs : aucun.
- ainsi $Niveaux[7]$ reste égal à $\emptyset$ et c'est fini, il n'y aura pas de prochain tour de boucle !

Maintenant que tous les sommets sont étiquetés, nous pouvons représenter le graphe de manière hiérarchique, d'une direction vers son opposé. Ici nous choisissons de le représenter de haut en bas : d'abord une ligne pour les sommets de niveau 0, puis une ligne pour les sommets de niveau 1... et enfin les sommets de niveau le plus élevé :

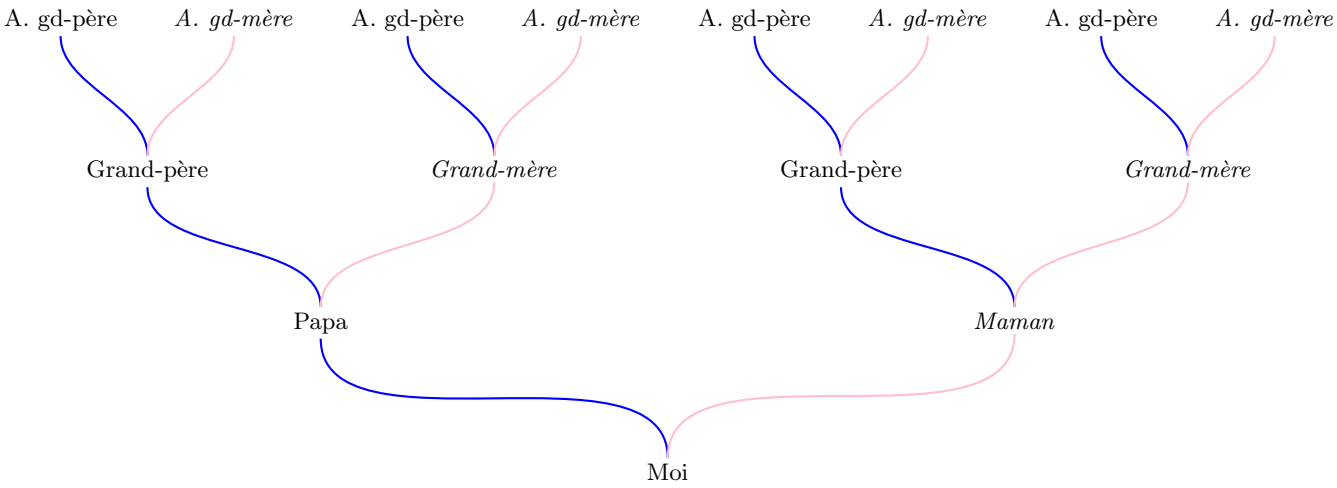


Remarque : nous pouvons voir sur le dessin deux propriétés : aucun niveau n'est vide, et tous les arcs vont strictement d'un sommet plus haut vers un sommet plus bas. C'est toujours le cas une fois le graphe ordonné par niveaux.

✓ Bulletin Officiel
Le B.O. attend qu'un étudiant soit capable de mettre en œuvre un algorithme pour obtenir les niveaux, et qu'il puisse ensuite représenter géométriquement un graphe orienté par niveaux.

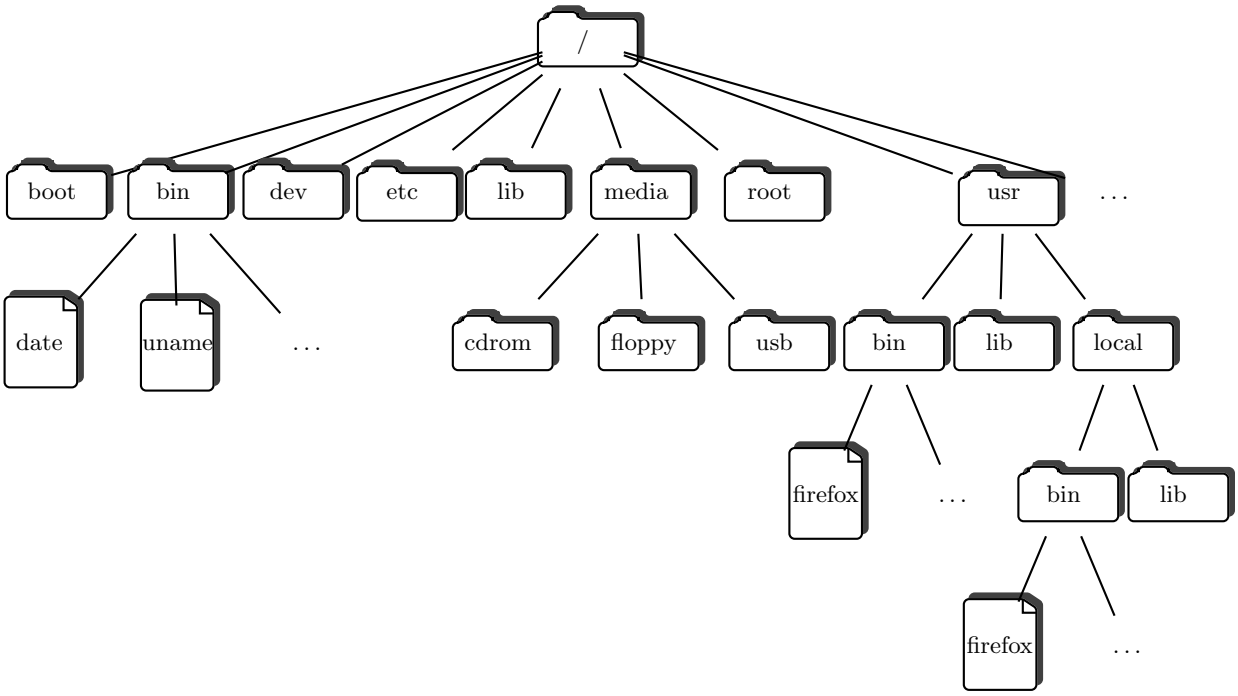
4.3 Arbres

Certains des graphes sans circuit sont d'un type encore plus particulier : ce sont les arbres. Avant de donner une définition, commençons par donner quelques exemples. Vous avez sûrement déjà construit un arbre généalogique. Cela tombe bien, c'est également un arbre au sens de la théorie des graphes :

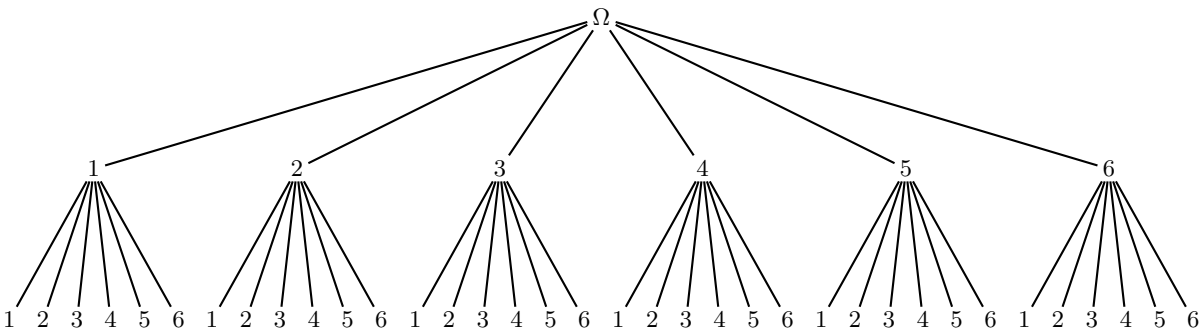


Et, puisque cela s'inscrit pleinement dans les thématiques de ce BTS, vous connaissez certainement l'arborescence de fichiers d'un ordinateur. Voici un extrait d'une arborescence de fichiers

sous Unix :



Enfin, nous avons rencontré depuis la seconde les arbres de probabilités, qui permettent, dans leur version non pondérée, de calculer des probabilités dans le cas équiprobable (car il permettent de dénombrer les possibilités). Par exemple, dans l'expérience aléatoire « lancer deux dés équilibrés à 6 faces numérotées de 1 à 6 », l'arbre de probabilité suivant montre que la somme des deux faces sera plus souvent 7 que n'importe quel autre nombre :



Les documents produits en html, xml... respectent aussi ce genre de structure. Il est également possible de représenter des objets qui respectent une certaine grammaire par des arbres (par exemple des expressions mathématiques, des phrases de la langue française...)

Les arbres présentés plus haut sont des graphes non-orientés. Effectivement, par définition un arbre est un graphe non-orienté, connexe et sans cycle (comme vu en chapitre 3, le cycle est l'équivalent pour les graphes non-orientés du circuit pour les graphes orientés ; dans un cycle toutes les arêtes sont différentes, sinon toute arête prise dans un sens puis dans l'autre formerait un cycle pas très intéressant.).

Le B.O. est cela dit clair : la notion de connexité est hors programme (elle n'est cela dit pas compliquée à comprendre : cela veut dire qu'il est possible d'atteindre n'importe quel sommet depuis n'importe quel autre ; dans le cas orienté il faut faire la distinction entre connexité et forte connexité). La notion d'arbre « tout court » (non-orienté) est hors-programme aussi,

mais il est possible de définir les arbres sans parler de connexité, puisqu'il y a équivalence entre les six propositions suivantes :

1. G est connexe sans cycle
2. G est sans cycle avec $m = n - 1$
3. G est connexe avec $m = n - 1$
4. G est sans cycle maximal (l'ajout d'une quelconque arête donnerait lieu à un cycle)
5. G est connexe minimal (le retrait d'une quelconque arête enlève la connexité)
6. deux sommets quelconques de G sont reliés par une unique chaîne élémentaire (dans laquelle un sommet apparaît au plus une fois)

Preuve de l'équivalence

$1 \Rightarrow 2$: Raisonnons par récurrence sur n.

Initialisation : un graphe sans cycle avec un unique sommet ($n = 1$) n'a aucune arête ($m = 0$), car la seule arête possible serait une boucle. Ainsi $m = n - 1$.

Hérédité : soit $n \geq 1$ et supposons la propriété vraie au rang n, c'est-à-dire que tout graphe à n sommets connexe et sans cycle a $n - 1$ arêtes.

Soit G un graphe connexe sans cycle à $n + 1$ sommets. Le pendant du constat fait en début de chapitre pour les graphes non-orientés est qu'il existe alors au moins un sommet de degré 1 (au lieu de considérer un chemin de longueur maximale, il faut considérer une chaîne élémentaire de longueur maximale). Appelons v un tel sommet, et considérons G' construit à partir de G en retirant v ainsi que l'unique arête qui a v comme extrémité. Ainsi G' est un graphe connexe (puisque v est de degré 1, retirer l'arête n'enlève pas la connexité), sans cycle (puisqu'on n'a fait qu'enlever une arête et un sommet), donc par hypothèse de récurrence G', connexe sans cycle qui possède n sommets, possède donc $n - 1$ arêtes.

Donc G possède bien n arêtes. □

$2 \Rightarrow 3$: Raisonnons encore par récurrence sur n.

Initialisation : un graphe avec un unique sommet ($n = 1$) est évidemment connexe.

Hérédité : soit $n \geq 1$ et supposons la propriété vraie au rang n, c'est-à-dire que tout graphe à n sommets, sans cycle et avec $n - 1$ arêtes est connexe.

Soit G un graphe sans cycle à $n+1$ sommets et n arêtes. Soit, comme précédemment, v un sommet de G de degré 1, et G' construit à partir de G en retirant v ainsi que l'unique arête qui a v comme extrémité. Ainsi G' est un graphe sans cycle (puisqu'il a perdu une arête et un sommet), donc par hypothèse de récurrence G' est connexe.

G est alors connexe car chaque sommet de G' est atteignable depuis le sommet auquel est relié v. □

$3 \Rightarrow 4$: Raisonnons une nouvelle fois par récurrence sur n.

Initialisation : un graphe avec un unique sommet et aucune arête est évidemment sans cycle maximal, car la seule arête qu'il est possible d'ajouter serait une boucle, qui rajouterait donc un cycle.

Hérédité : soit $n \geq 1$ et supposons la propriété vraie au rang n, c'est-à-dire que tout graphe à n sommets, connexe et à $n - 1$ arêtes est sans cycle maximal.

Soit G un graphe sans cycle à $n + 1$ sommets. Soit, comme précédemment, v un sommet de G de degré 1, et G' construit à partir de G en retirant v ainsi que l'unique arête qui a v comme extrémité. Par hypothèse de récurrence, G' est connexe maximal. Donc, si G avait un cycle, il passerait par v... ce qui est impossible car v est de degré 1. Donc G est sans cycle.

Montrons maintenant que rajouter une arête à G crée un cycle. Soient x et y deux sommets de G non reliés par une arête. Puisque G est connexe, il existe une chaîne élémentaire de x à y. Donc rajouter l'arête (x, y) crée un cycle [2].

Donc G est bien sans cycle maximal. □

$4 \Rightarrow 5$: soit G un graphe sans cycle maximal. Soient x et y deux sommets de G. Si (x, y) est une arête de G, ils sont reliés. Sinon, rajouter l'arête (x, y) créerait un cycle (par maximalité), ce qui veut également dire que x est atteignable depuis y. Ainsi G est connexe.

Montrons maintenant qu'enlever une arête de G enlèverait sa connexité en raisonnant par l'absurde. Supposons qu'en retirant une arête (x, y) de G, G reste connexe. Cela veut dire qu'il existe dans ce nouveau graphe une chaîne élémentaire de x à y. Donc, comme précédemment, remettre l'arête (x, y) créerait un cycle, ce qui est absurde.

Donc G est bien connexe minimal. □

$5 \Rightarrow 6$: soit G un graphe connexe minimal. Soient x et y deux sommets de G. Il existe une chaîne élémentaire de x à y puisque G est connexe.

Montrons maintenant que cette chaîne élémentaire est unique en raisonnant par l'absurde. Supposons qu'il existe c_1 et c_2 deux chaînes élémentaires distinctes reliant x à y. Considérons (z, t) une arête dans c_2 qui ne soit pas dans c_1. Alors le graphe G privé de (z, t) est lui aussi connexe. Effectivement pour atteindre t depuis z dans ce nouveau graphe il suffit de suivre la chaîne c_2 à rebours de z à x, puis la chaîne c_1, puis la chaîne c_2 à rebours de y à t. Cela vient donc contredire la minimalité de la connexité de G.

Ainsi deux sommets quelconques de G sont donc bien reliés par une unique chaîne élémentaire. □

$6 \Rightarrow 1$: soit G un graphe dans lequel deux sommets quelconques sont reliés par une unique chaîne élémentaire.

G est donc évidemment connexe.

Montrons que G est sans cycle en raisonnant par l'absurde. Supposons qu'il existe un cycle dans G. Soient x et y deux sommets de G tels que ce cycle s'écrive $(x, \ldots, y, x)$. Alors $(x, \ldots, y)$ et (x, y) sont deux chaînes élémentaires distinctes (car dans un cycle les arêtes sont toutes différentes) reliant x à y, ce qui est absurde.

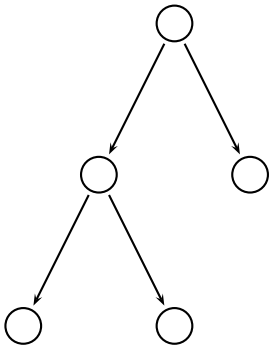
Ainsi G est bien connexe sans cycle. □

Remarque : pour continuer dans le champ lexical, une forêt est un graphe non-orienté qui est simplement acyclique. Il a donc le droit de ne pas être connexe, c'est-à-dire précisément d'être une sorte d'union d'arbres, exactement comme son nom le laisse présager.

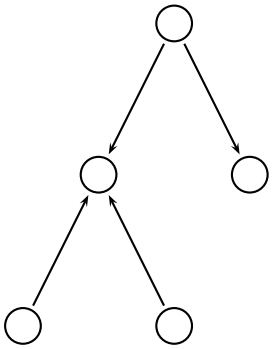
Pour utiliser une orientation particulière, il faut « enraciner » l'arbre, c'est-à-dire imposer le sens des arcs. Il est évident que pour le premier exemple, les arcs vont être orientés de bas en haut (racine en « Moi »), et pour les deux suivants, de haut en bas (racine en « / » et en « Ω »). Pour filer la métaphore :

- un arbre enraciné est un graphe orienté sans circuit dans lequel il y a un unique sommet sans prédécesseur, et dans lequel chacun des autres sommets a un unique prédécesseur.
- la racine d'un arbre enraciné est son unique sommet sans prédécesseur
- une feuille d'un arbre enraciné est un sommet sans successeur

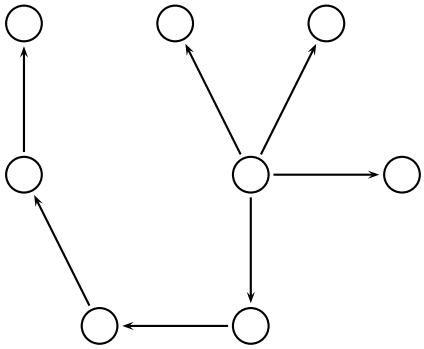
 Il faut être vigilant avec les arbres enracinés. Car si, au premier abord, certaines représentations de graphes peuvent faire penser à un arbre, ce n'en sont pas. Et à l'inverse certaines représentations qui n'ont rien a priori de l'arbre peuvent en être. Pour fixer les idées :



Je suis un arbre.



Je ne suis pas un arbre.



Je suis un arbre.

Ceux qui aiment la combinatoire peuvent se demander comment compter le nombre d'arbres « distincts » à n sommets, ainsi que le nombre d'arbres enracinés « distincts » à n sommets. Il faut sous-entendre bien sûr de ne pas compter comme distincts deux graphes si seuls les noms des sommets ont été modifiés (ou si seule la représentation graphique est différente...).

Chapitre 5

Chemin optimal

Dans cette section, nous cherchons comment aller « le plus rapidement possible » d'un sommet source vers un autre sommet puits (avec une vitesse à déterminer selon les cas). Nous allons utiliser des algorithmes qui vont calculer les chemins optimaux de ce sommet source vers chacun des autres sommets du graphe. Il est possible bien sûr de s'arrêter dès que le chemin optimal vers le puits a été découvert, mais cela ne coûte en fait pas moins cher en terme de complexité (dans le pire cas).

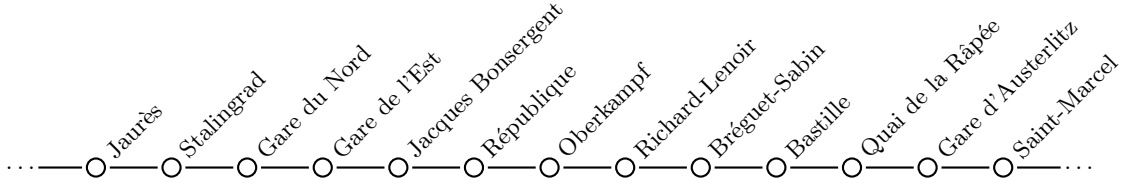
5.1 Chemin de longueur minimale

Lorsque nous empruntons le métro, une question naturelle à laquelle nous sommes confontés en regardant le plan est : « Combien de stations nous séparent de notre destination ? » Nous souhaitons donc connaître le nombre minimal d'arcs séparant un sommet d'un autre. Dans cette section, un chemin entre deux sommets est optimal s'il joint ces sommets en le moins d'arcs possible et la distance d'un sommet à un autre est définie par :

$$d(x, y) = \min_{c \,:\, x \rightsquigarrow y} \text{longueur}(c)$$

Bien sûr l'ensemble des chemins de x à y peut être vide, dans ce cas $d(x, y) = +\infty$. Cet ensemble peut également être infini (s'il existe un circuit entre x et y), mais cela ne pose pas de problème, car les longueurs sont toutes des entiers positifs et donc le minimum est bien défini (toute partie non vide de $\mathbb{N}$ admet un plus petit élément).

Par exemple, sur le graphe composé d'un fragment de la ligne 5 de métro parisien suivant...



... il est facile de voir qu'un voyageur arrivant à Gare d'Austerlitz est à distance 8 de la Gare de l'Est, à distance 1 de Saint-Marcel, à distance 5 d'Oberkampf, etc. Bien sûr, le problème se complexifie énormément dès qu'il faut prendre en compte les correspondances. Ce d'autant que lors des correspondances, il y a souvent un temps d'attente, et que ce temps d'attente dépend de l'heure, de l'endroit de la correspondance [1]... Bref : le modèle de graphe choisi pour l'instant ne semble pas optimal pour déterminer le plus court (en terme de temps) chemin possible d'une station à une autre. Contentons-nous cela dit pour l'instant de ce modèle, et voyons ce qu'il est capable de résoudre.

1. Pour modéliser plus finement le graphe du métro parisien, il faudrait mettre par exemple non pas un sommet pour Gare d'Austerlitz, mais trois : un pour Gare d'Austerlitz ligne 5, un pour Gare d'Austerlitz ligne 10, un pour Gare d'Austerlitz ligne C.

Un algorithme calculant les distances d'un sommet s vers les autres est présenté ci-après. Son principe est simple : s est à distance 0 de lui-même. Ses successeurs sont à distance 1. Les successeurs de ces successeurs, s'ils n'étaient pas déjà successeurs directs, sont à distance 2. Etc.

L'algorithme est basé sur la proposition fondamentale suivante :

Proposition
Si $x \rightsquigarrow y$ est un chemin de longueur minimale de x à y et que a et b sont deux sommets rencontrés dans cet ordre sur ce chemin (c'est-à-dire que $x \rightsquigarrow y = x \rightsquigarrow a \rightsquigarrow b \rightsquigarrow y$), alors $a \rightsquigarrow b$ est un chemin de longueur minimale de a à b.
Autrement dit, tout sous-chemin d'un chemin optimal est optimal.

Preuve

Il suffit de raisonner par l'absurde : s'il existait un chemin $a \overset{*}{\rightsquigarrow} b$ de longueur strictement inférieure à $a \rightsquigarrow b$, alors $x \rightsquigarrow a \overset{*}{\rightsquigarrow} b \rightsquigarrow y$ serait un chemin de longueur strictement inférieure à $x \rightsquigarrow y$, ce qui est absurde par minimalité de $x \rightsquigarrow y$. □

Calcul de distances.

Entrée :
Un graphe $G = (V; E)$.
Un sommet $s \in V$.

Sortie et rôle de l'algorithme :
Étiqueter chaque sommet de V par sa distance à s.

Variables utilisées :
d, tableau de $n + 1$ ensembles de sommets
v et x, sommets
i, nombre entier

Corps de l'algorithme :

```
1   Initialiser toutes les cases de d par ∅
2   d[0] ← {s}
3   i ← 0
4   Tant que d[i] ≠ ∅, faire
5       Pour chaque sommet v de d[i], faire
6           Étiqueter v par i
7           Pour chaque successeur x de v, faire
8               Si x n'est pas étiqueté, alors
9                   d[i + 1] ← d[i + 1] ∪ {x}
10              Fin Si
11          Fin Pour
12      Fin Pour
13      i ← i + 1
14  Fin Tant que
15  Pour chaque sommet v de V, faire
16      Si v n'est pas étiqueté, alors
17          Étiqueter v par +∞
18      Fin Si
19  Fin Pour
```

Complexité
Cet algorithme est en $O(n + m)$. Effectivement, l'initialisation de d (ligne 1) est en $O(n)$, la première boucle (lignes 4-14) est en $O(n + m)$ et la seconde boucle (lignes 15-19) en $O(n)$.

Remarque : qui dit graphes dit aussi parcours de graphes (ne serait-ce que pour aller voir ce qui se passe partout). Il y a deux parcours « classiques », dont l'un est le parcours en largeur. Son principe est exactement de parcourir le sommet source, puis de parcourir les sommets à distance 1, les sommets à distance 2, ... Pour réaliser un tel parcours de manière plus efficace, il faut utiliser une structure de données hors programme : la file. C'est une structure qui porte très bien son nom, puisqu'elle fonctionne exactement comme les files d'attente pour les caisses : tout élément qui a rejoint la file avant un autre va être traité avant cet autre élément. C'est avec un parcours en largeur que certaines « intelligences artificielles » fonctionnent : jouer aux échecs (explorer les coups possibles, avec plusieurs coups d'avance), sortir d'un labyrinthe (explorer les différentes cases atteignables)...

Remarque : l'algorithme calcule les distances mais ne dit absolument pas par quels sommets passe un chemin optimal. Pour cela, il suffit de rajouter un tableau d'entiers *pere* de taille n tel que $pere[i]$ représente le sommet précédant i dans un chemin optimal partant de s et passant par i. Pour mettre à jour ce tableau, il faut rajouter $pere[x] = v$ à la ligne 9. Pour connaître un chemin optimal de s à p, il suffit d'aller voir $pere[p]$, $pere[pere[p]]$, etc. jusqu'à trouver s.

Une application de la distance est le nombre d'Erdős. Paul Erdős était un chercheur très prolifique (environ 1 500 articles) qui avait signé des articles avec environ 500 chercheurs différents. Dans le graphe non-orienté où les sommets sont constitués des chercheurs et une arête entre deux chercheurs symbolise qu'ils ont cosigné un article, le nombre d'Erdős d'un chercheur est sa distance à Paul Erdős.

5.2 Chemin de coût minimal

Nous nous intéressons cette fois à des graphes valués. Un graphe est valué s'il est doté d'une valuation, c'est-à-dire une fonction qui à chaque arc associe son coût :

$$\begin{array}{rcl} c : E & \to & \mathbb{R} \\ e & \mapsto & c(e) \end{array}$$

Notre *GPS* calcule en permanence pour nous des chemins de coût minimal. Il nous laisse d'ailleurs même le choix : souhaitons-nous un chemin qui minimise la distance de parcours (moins d'essence consommée) ou le temps de parcours (plus de temps dans le lit à l'arrivée) ? Comme lui, nous allons essayer de minimiser les coûts entre deux sommets. Pour cela, il nous reste à définir le coût d'un chemin, de manière naturelle par :

$$c(v_0, v_1, \ldots, v_k) = c(v_0, v_1) + c(v_1, v_2) + \ldots + c(v_{k-1}, v_k) = \sum_{n=0}^{k-1} c(v_n, v_{n+1})$$

Nous souhaitons donc connaître le coût minimal d'un chemin séparant un sommet d'un autre. Dans cette section, un chemin entre deux sommets est optimal s'il joint ces sommets avec un coût minimum et la distance d'un sommet à un autre est définie par :

$$\delta(x, y) = \inf_{p \,:\, x \rightsquigarrow y} c(p)$$

Bien sûr l'ensemble des chemins de x à y peut être vide, dans ce cas $\delta(x, y) = +\infty$. Cet ensemble peut également être infini (s'il existe un circuit entre x et y), et cela peut cette fois-ci poser problème, car les coûts ne sont pas forcément positifs ! C'est pourquoi la définition utilise la borne inférieure dans cette section et pas le minimum : dans le cas où il existe un circuit de coût strictement négatif entre x et y, il faut alors poser $\delta(x, y) = -\infty$, et alors il n'existe pas de chemin optimal entre x et y.

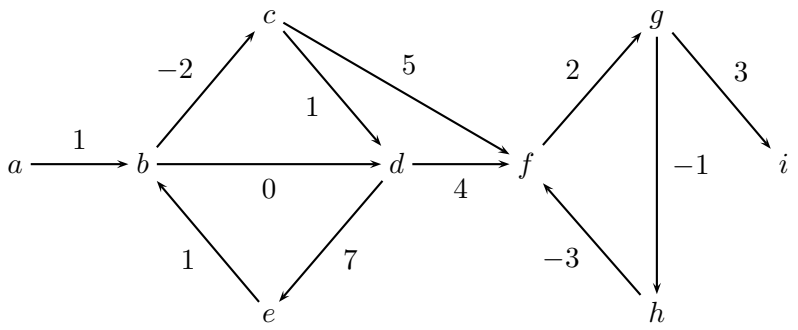
Preuve

Soit a un sommet tel que :
- il existe un chemin $c = x \rightsquigarrow a \rightsquigarrow y$ de x à y
- il existe un circuit $a \rightsquigarrow a$ de coût strictement négatif

Alors le chemin $c' = x \rightsquigarrow a \rightsquigarrow a \rightsquigarrow y$ est un chemin de x à y de coût strictement inférieur à c : aucun chemin de x à y ne peut donc être optimal.

De plus, comme l'ajout de ce circuit dans un chemin de x à y permet de trouver des chemins de x à y de coût aussi négatif que souhaité, cela justifie de poser $\delta(x, y) = -\infty$. □

Exemple : un graphe valué G :



Voici plusieurs calculs de distance pour se familiariser avec la notion :

- il existe clairement un circuit de coût strictement négatif : le circuit (f, g, h, f) de coût -2. Ainsi, par exemple, $\delta(d, f) = -\infty$ mais il n'existe pas de chemin de coût minimal de d à f. De même $\delta(a, i) = -\infty$, etc.
- il existe un autre circuit : (b, d, e, b) de coût 8. Il n'est donc pas intéressant d'emprunter ce circuit et aucun chemin optimal ne passera par lui.
- $\delta(a, d) = 0$ et un chemin optimal de a à d est (a, b, c, d) (il est de plus unique).
- $\delta(f, b) = +\infty$ car b n'est pas atteignable depuis f.
- $\delta(c, f) = 5$ et un chemin optimal de c à f est (c, f) (le chemin (c, d, f) est optimal également)

Un algorithme de calcul de chemins optimaux devra donc, dans le cas général, savoir reconnaître un circuit de coût négatif (on parle de circuit absorbant (notation hors programme)). Cela dit, dans le cours nous nous restreindrons à deux cas particuliers pour lesquels nous sommes assurés de la non-existence de tels circuits :

- lorsque la fonction de coût est à valeurs dans $\mathbb{R}^+$, l'algorithme de Dijkstra fera ces calculs
- lorsque le graphe est sans circuit, l'algorithme de Bellman fera ces calculs

Remarque : une nouvelle fois, soulignons que ces algorithmes sont justifiés par la proposition de la section précédente, encore valable pour les graphes valués : tout sous-chemin d'un chemin optimal est optimal.

5.2.1 Algorithme de Dijkstra

Tout d'abord évitons d'écorcher le nom de cet informaticien néerlandais : il se prononce « Daïkstra ».

Pour cet algorithme, nous rappelons que la fonction de coût c est supposée positive.

Algorithme de Dijkstra.

Entrée :
Un graphe $G = (V; E)$.
Un sommet $s \in V$.
La fonction de coût c, positive.

Sortie et rôle de l'algorithme :
Un tableau d où $d[i]$ contient la distance de s à i.
Un tableau $pere$ où $pere[i]$ contient le sommet précédant i dans un plus court chemin de s vers i.

Variables supplémentaires utilisées :
$Fixes$, ensemble de sommets
x et y, sommets

Corps de l'algorithme :

```
1   Fixes ← ∅
2   Initialiser toutes les cases de d par +∞
3   Initialiser toutes les cases de pere par s
4   d[s] = 0
5   Tant que Fixes ≠ V, faire
6       Soit x un sommet de V \ Fixes tel que d[x] soit minimal
7       Fixes ← Fixes ∪ {x}
8       Pour chaque successeur y de x, faire
9           Si y ∉ Fixes et d[x] + c(x, y) < d[y], alors
10              d[y] ← d[x] + c(x, y)
11              pere[y] ← x
12          Fin Si
13      Fin Pour
14  Fin Tant que
15  Renvoyer d et pere
```

Preuve de l'algorithme

Montrons par récurrence sur p, le nombre de passages dans la boucle « Répéter », l'invariant de boucle suivant : à la fin du p-ième passage dans la boucle, pour chaque sommet i :

- si $i \in Fixes$, alors $d[i] = \delta(s, i)$
- sinon, si i a un prédécesseur dans $Fixes$, alors $d[i] = \min_{\substack{k \in Fixes \\ (k,i) \in E}} \{d[k] + c(k, i)\}$
- sinon $d[i] = +\infty$

Initialisation : l'invariant est vrai au rang 1 car :

- $Fixes = \{s\}$
- $d[s] = 0 = \delta(s, s)$ et il n'y a pas d'autre sommet dans $Fixes$
- si $i \notin Fixes$ et $(s, i) \in E$, alors $d[i] = c(s, i)$ (car $0 + c(s, i) < +\infty$ donc l'algorithme a mis à jour $d[i]$) qui vaut bien le minimum annoncé car il n'y a que s dans $Fixes$
- si $i \notin Fixes$ et $(s, i) \notin E$, alors $d[i] = +\infty$ (car l'algorithme ne l'a alors pas mis à jour)

Hérédité : supposons l'invariant vrai au rang p. Soit i un sommet de V. Montrons que l'invariant reste vrai au rang $p + 1$ en distinguant selon la nature de i :

Si i est un sommet qui était déjà dans $Fixes$ au p-ième passage, alors par hypothèse de récurrence $d[i] = \delta(s, i)$.

Si $i = x$, le sommet rajouté à $Fixes$ au début du $p+1$-ième passage, il faut démontrer que $d[i] = \delta(s, i)$. Soit $(x_1, x_2, \dots x_n)$ un plus court chemin de s à i (avec donc $x_1 = s$ et $x_n = i$). Soit x_k le premier sommet dans ce chemin qui n'était pas dans $Fixes$ au début de ce passage.

- Puisque $x_k \notin Fixes$ au début du passage, $d[i] \leq d[x_k]$ (car $d[i]$ est minimal).
- $x_1, \dots, x_{k-1}$ sont tous dans $Fixes$. Or par hypothèse de récurrence $d[k]$ est le minimum des coûts des chemins de s à x_k dont tous les sommets intermédiaires sont dans $Fixes$. Donc $d[x_k] \leq c(x_1, \dots, x_k)$.

- Puisque tous les poids sont positifs, $c(x_k, \ldots, x_n) \geq 0$ et on peut donc conclure que $c(x_1, \ldots, x_k) \leq c(x_1, \ldots, x_n)$.

Ainsi $d[i] \leq d[x_k] \leq c(x_1, \ldots, x_k) \leq c(x_1, \ldots, x_n) = \delta(s, i)$ donc $d[i] \leq \delta(s, i)$ (par transitivité), ce qui n'est possible que si $d[i] = \delta(s, i)$.

Pour la suite, nous noterons $Fixes(p)$ la valeur de l'ensemble $Fixes$ au début du passage, et $Fixes(p+1) = Fixes(p) \cup \{x\}$ la valeur de cet ensemble à la fin du passage.

Si i est un sommet qui reste dans $V \setminus Fixes$ et que i n'est pas successeur de x, alors $d[i]$ reste inchangé. Puisque $(x, i) \notin E$, l'ensemble $\{y \in V | y \in Fixes(p) \wedge (y, i) \in E\}$ est égal à l'ensemble $\{y \in V | y \in Fixes(p+1) \wedge (y, i) \in E\}$, donc l'invariant reste vrai.

Si i est un sommet qui reste dans $V \setminus Fixes$ et que i est successeur de x : par hypothèse de récurrence, au début du passage $d[i] = \min_{\substack{k \in Fixes(p) \\ (k,i) \in E}} \{d[k] + c(k, i)\}$. Si $d[i]$ reste inchangé, c'est que le minimum sur $Fixes(p+1)$ est encore atteint pour un sommet de $Fixes(p)$ donc le minimum sur $Fixes(p)$ est le même que celui sur $Fixes(p+1)$; si $d[i]$ est mis à jour c'est que $d[x] + c(x, i) < d[i]$ et que le minimum sur $Fixes(p+1)$ est atteint pour $k = x$. La nouvelle valeur est bien dans les deux cas $d[i] = \min_{\substack{k \in Fixes(p+1) \\ (k,i) \in E}} \{d[k] + c(k, i)\}$.

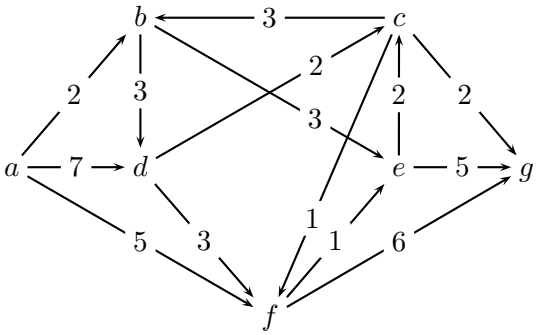
Conclusion : l'invariant est bien vrai pour tout $i \geq 1$. Quand tous les sommets sont dans $Fixes$, l'algorithme a donc bien calculé tous les coûts minimaux ! □

Complexité

Il y a clairement n passages dans la boucle « Tant que » (lignes 5-14). Le calcul de la complexité est alors ramené à la manière de gérer l'ensemble des sommets pour en extraire à chaque étape celui qui minimise $d[i]$ efficacement.
De manière naïve, chercher le minimum est en $O(n)$, et donc la complexité totale est majorée par $O(n^2 + m)$.
Un tas (structure de données arborescente qui permet l'extraction du minimum, l'ajout d'une nouvelle valeur, ou la mise à jour d'une valeur en $O(\log n)$) permet d'arriver à $O((n + m) \log n)$. Un tas de Fibonacci (structure de données avancée qui permet de faire décroître une valeur en temps amorti $O(1)$) permet même d'arriver à $O(n \log n + m)$.

Remarque : cet algorithme fait partie de la famille des algorithmes gloutons : il choisit à chaque étape le meilleur sommet, le traite, et continue. La théorie des matroïdes permet d'expliquer quand un algorithme glouton permet d'arriver à une solution optimale, et quand il ne permet que d'arriver à une approximation (exemple classique : problème du sac à dos).

Pour voir l'algorithme évoluer « à la main », il est possible de tenir à jour un tableau qui contient une colonne par sommet, et une ligne par étape. Chaque ligne contient les valeurs de d pour les différents sommets (il faut donc un autre tableau pour suivre les valeurs de *pere*). Voyons l'algorithme fonctionner sur l'exemple suivant, avec a comme sommet source :



- À l'initialisation, le tableau est rempli de la sorte :

Étape \ Sommet	a	b	c	d	e	f	g
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

- À l'étape 1, a est le sommet minimal. On le fixe, et on met à jour ses successeurs :

Étape \ Sommet	a	b	c	d	e	f	g
0	**0**	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	.	2	$+\infty$	7	$+\infty$	5	$+\infty$

- À l'étape 2, b est le sommet minimal. On le fixe, et on met à jour ses successeurs :

Étape \ Sommet	a	b	c	d	e	f	g
0	**0**	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	.	**2**	$+\infty$	7	$+\infty$	5	$+\infty$
2	.	.	$+\infty$	5	5	5	$+\infty$

- À l'étape 3, d, e et f sont les sommets minimaux. On en fixe un, d par exemple, et on met à jour ses successeurs :

Étape \ Sommet	a	b	c	d	e	f	g
0	**0**	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	.	**2**	$+\infty$	7	$+\infty$	5	$+\infty$
2	.	.	$+\infty$	**5**	5	5	$+\infty$
3	.	.	9	.	5	5	$+\infty$

- À l'étape 4, e et f sont les sommets minimaux. On en fixe un, f par exemple, et on met à jour ses successeurs :

Étape \ Sommet	a	b	c	d	e	f	g
0	**0**	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	.	**2**	$+\infty$	7	$+\infty$	5	$+\infty$
2	.	.	$+\infty$	**5**	5	5	$+\infty$
3	.	.	9	.	5	**5**	$+\infty$
4	.	.	9	.	5	.	11

- À l'étape 5, e est le sommet minimal. On le fixe, et on met à jour ses successeurs :

Étape \ Sommet	a	b	c	d	e	f	g
0	**0**	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	.	**2**	$+\infty$	7	$+\infty$	5	$+\infty$
2	.	.	$+\infty$	**5**	5	5	$+\infty$
3	.	.	9	.	5	**5**	$+\infty$
4	.	.	9	.	**5**	.	11
5	.	.	7	.	.	.	10

- À l'étape 6, c est le sommet minimal. On le fixe, et on met à jour ses successeurs :

Étape \ Sommet	a	b	c	d	e	f	g
0	**0**	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	.	**2**	$+\infty$	7	$+\infty$	5	$+\infty$
2	.	.	$+\infty$	**5**	5	5	$+\infty$
3	.	.	9	.	5	**5**	$+\infty$
4	.	.	9	.	**5**	.	11
5	.	.	**7**	.	.	.	10
6	.	.	.	.	.	.	9

- À l'étape 7, il ne reste plus que g. On le fixe, et on peut conclure pour les distances :

Étape \ Sommet	a	b	c	d	e	f	g
0	**0**	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	.	**2**	$+\infty$	7	$+\infty$	5	$+\infty$
2	.	.	$+\infty$	**5**	5	5	$+\infty$
3	.	.	9	.	5	**5**	$+\infty$
4	.	.	9	.	**5**	.	11
5	.	.	**7**	.	.	.	10
6	.	.	.	.	.	.	**9**
	0	2	7	5	5	5	9

Tant que nous sommes dans le cas où la fonction de coût est positive, présentons une modification de l'algorithme de Roy-Warshall calculant les coûts des chemins optimaux dans ce cas, comme annoncé au chapitre 3 :

Algorithme de Roy-Warshall modifié.

Entrée :
La matrice d'adjacence A d'un graphe G.
La fonction de coût c.

Sortie et rôle de l'algorithme :
La matrice δ, contenant les distances de chaque sommet à chaque autre.

Variables utilisées :
δ, matrice à coefficients dans $\overline{\mathbb{R}}$
p, i, j, nombres entiers

Corps de l'algorithme :

```
1   n ← taille(A)          Étant entendu que c'est une matrice carrée...
2   Pour i de 1 à n, faire
3       Pour j de 1 à n, faire
4           Si A_{i,j} = 0, alors
5               δ_{i,j} = +∞
6           Sinon
7               δ_{i,j} = A_{i,j}
8           Fin Si
9       Fin Pour
10  Fin Pour
11  Pour p de 1 à n, faire
12      Pour i de 1 à n, faire
13          Pour j de 1 à n, faire
14              δ_{i,j} ← min(δ_{i,j}, δ_{i,p} + δ_{p,j})
15          Fin Pour
16      Fin Pour
17  Fin Pour
18  Renvoyer δ
```

Remarques :

- le + est entendu sur $\overline{\mathbb{R}}$ avec en particulier :

$$\forall x \in \mathbb{R}, x + (+\infty) = (+\infty) + x = (+\infty) + (+\infty) = +\infty$$

De même le min est entendu sur $\overline{\mathbb{R}}$ avec en particulier :

$$\forall x \in \overline{\mathbb{R}}, \min(x, +\infty) = \min(+\infty, x) = x$$

- cet algorithme calcule les coûts des chemins optimaux de tous les sommets vers tous les sommets, contrairement à l'algorithme de Dijkstra qui ne calcule « que » les coûts des chemins optimaux d'un sommet donné vers tous les sommets.
- en terme de complexité, calculer les coûts des chemins optimaux de tous les sommets vers tous les sommets est plus rapide que de calculer, pour chaque sommet, les coûts des chemins optimaux de ce sommet vers tous les sommets.
- cet algorithme peut encore être adapté dans le cas général (avec gestion des circuits absorbants) : c'est l'algorithme de Warshall-Floyd.

5.2.2 Algorithme de Bellman

Pour cet algorithme, nous rappelons que le graphe G est supposé sans circuit.

Algorithme de Bellman.

Entrée :
Un graphe $G = (V; E)$, sans circuit.
Un sommet $s \in V$.
La fonction de coût c.

Sortie et rôle de l'algorithme :
Un tableau d où $d[i]$ contient la distance de s à i.
Un tableau *pere* où *pere*$[i]$ contient le sommet précédant i dans un plus court chemin de s vers i.

Variables supplémentaires utilisées :
i, entier
x et y, sommets

Corps de l'algorithme :

```
1   Initialiser toutes les cases de d par +∞
2   Initialiser toutes les cases de pere par s
3   d[s] = 0
4   Ordonner le graphe par niveaux
5   Pour i de niveau(s) + 1 à nombre de niveaux - 1, faire
6       Pour chaque sommet x du niveau i, faire
7           Pour chaque prédécesseur y de x, faire
8               Si d[y] + c(y,x) < d[x], alors
9                   d[x] ← d[y] + c(y,x)
10                  pere[x] ← y
11              Fin Si
12          Fin Pour
13      Fin Pour
14  Fin Pour
15  Renvoyer d et pere
```

Preuve de l'algorithme

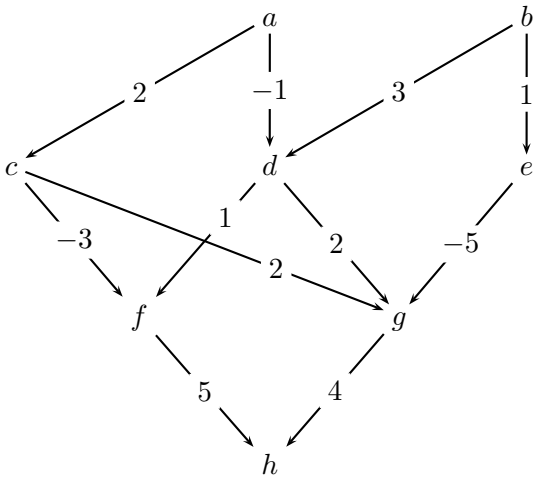
Soit x un sommet (atteignable depuis s). Un plus court chemin de s à x passe forcément par un prédécesseur de x. Ainsi, avec la connaissance des plus courts chemins de s vers tous les prédécesseurs de x, le plus court chemin de s à x est de longueur :

$$\delta(s,x) = \min_{(y,x)\in E}\{\delta(s,y) + c(y,x)\}$$

Le graphe est sans circuit ; en le parcourant par niveaux, au moment d'évaluer le sommet x, tous ses prédécesseurs auront déjà été évalués, ce qui permet d'utiliser cette formule. □

Complexité
Ordonner le graphe par niveaux prend un temps $O(n + m)$.
Ensuite, la boucle « Pour » (lignes 5-14) examine tous les sommets (ayant un niveau strictement supérieur au niveau de s), et, pour chacun de ces sommets, examine tous ses prédécesseurs. Cette boucle est donc là aussi en $O(n + m)$.
Cet algorithme a donc une complexité de $O(n + m)$ et est donc plus efficace que les optimisations les plus ingénieuses de l'algorithme de Dijkstra.

Voyons l'algorithme fonctionner sur l'exemple suivant, avec a comme sommet source :



- À l'initialisation, $d[a] = 0$ et $d[b] = +\infty$.
- Lorsque l'algorithme évalue le niveau 1 :
  - ◇ c n'a qu'un seul prédécesseur donc $d[c] = d[a] + 2 = 2$.
  - ◇ d a deux prédécesseurs donc $d[d] = \min(d[a] - 1, d[b] + 3) = -1$.
  - ◇ e n'a qu'un prédécesseur donc $d[e] = d[b] + 1 = +\infty$
- Lorsque l'algorithme évalue le niveau 2 :
  - ◇ f a deux prédécesseurs donc $d[f] = \min(d[c] - 3, d[d] + 1) = -1$.
  - ◇ g a trois prédécesseurs donc $d[g] = \min(d[c] + 2, d[d] + 2, d[e] - 5) = 1$
- Lorsque l'algorithme évalue le niveau 3, h a deux prédécesseurs donc $d[h] = \min(d[f] + 5, d[g] + 4) = 4$.