IN302
Graphes et algorithmes
Notes de cours et exercices

Michel Couprie

Le 11 janvier 2010

L'unité Graphes et Algorithmes a son site web !

`http://www.esiee.fr/~coupriem/IN302/`

Vous y trouverez le plan du cours, les sujets des TD et des TP, des lectures conseillées, des liens sur d'autres sites parlant de graphes et d'algorithmes ...

Table des matières

Chapitre 1

Notions de base

1.1 Première définition

Soit E un ensemble fini. L'ensemble des sous-ensembles d'un ensemble E, également appelé ensemble des parties de E, est noté $\mathcal{P}(E)$. Soit par exemple $E = \{1, 2, 3\}$, alors $\mathcal{P}(E) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}$. La notation $S \in \mathcal{P}(E)$ signifie que S est un sous-ensemble de E ; de manière équivalente on peut écrire $S \subset E$.

On appelle **graphe** sur E un couple $G = (E,\ \Gamma)$ où Γ est une application de $E \longrightarrow \mathcal{P}(E)$.

Exemple 1 : Soit un graphe $G = (E, \Gamma)$, avec $E = \{1, 2, 3, 4\}$ et Γ définie par :
– $\Gamma(1) = \{1, 2, 4\}$
– $\Gamma(2) = \{3, 1\}$
– $\Gamma(3) = \{4\}$
– $\Gamma(4) = \emptyset$

Il n'est pas facile de visualiser un graphe donné sous cette forme. Habituellement, on représente un graphe par un dessin tel que celui de la figure 1.1 page suivante. Notons que plusieurs dessins, comme pour cet exemple, peuvent représenter le même graphe.

• Tout élément de E est appelé un **sommet**.
• Tout couple ordonné $(x, y) \in E \times E$ tel que $y \in \Gamma(x)$ est appelé **arc** du graphe.
• Soit (x, y) un arc, on dit que y est un **successeur** de x.
• Soit (x, y) un arc, on dit que x est un **prédecesseur** de y.

Rappel : Dans un couple ordonné (x, y) l'ordre dans l'écriture — x puis y — est important. Ainsi $(x, y) \neq (y, x)$. Si l'ordre n'est pas important, il convient d'utiliser la notation ensembliste : $\{x, y\}$. Cette fois, nous avons : $\{x, y\} = \{y, x\}$.

Le graphe G de l'exemple 1 comporte quatre sommets et six arcs. On désigne par $\vec{\Gamma}$ l'ensemble des arcs du graphe $(E,\ \Gamma)$. L'ensemble des arcs est un sous-ensemble du produit

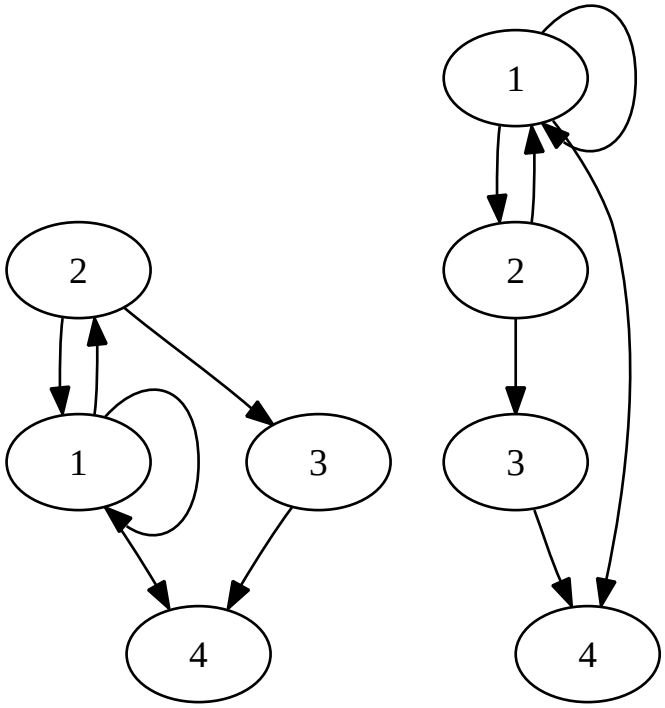


FIG. 1.1 – Graphe de l'exemple 1, dessiné de deux manières

cartésien $E \times E = \{(x, y) \mid x \in E, y \in E\}$, autrement dit, $\vec{\Gamma} \in \mathcal{P}(E \times E)$. On peut définir formellement $\vec{\Gamma}$ par :

$$\vec{\Gamma} = \{(x, y) \in \ E \times E \mid y \ \in \ \Gamma(x)\}$$

Suivant l'exemple précédent : $\vec{\Gamma} = \{(1,1), (1,2), (1,4), (2,3), (2,1), (3,4)\}$.

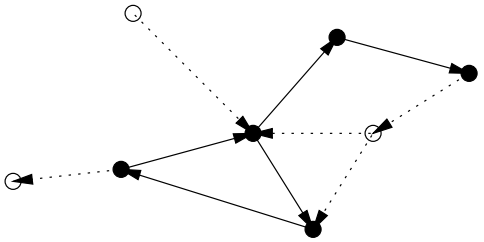
Remarque : On peut représenter G par (E, Γ) ou par $(E, \vec{\Gamma})$, les données de ces deux informations étant équivalentes. On appellera également graphe *le couple $(E, \vec{\Gamma})$.*

On note $|E|$ le cardinal (nombre d'éléments) de E. Pour un graphe $G = (E, \vec{\Gamma})$, on notera habituellement $n = |E|$ et $m = |\vec{\Gamma}|$, à savoir, n est le nombre de sommets et m le nombre d'arcs de G.

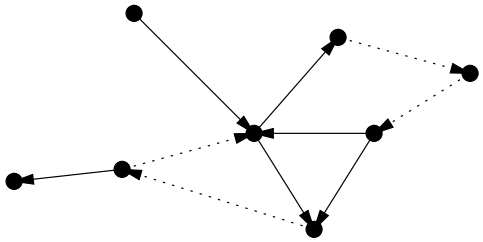
Un **sous-graphe** de $G = (E, \Gamma)$ est un graphe $H = (F, \Gamma_F)$ tel que F est un sous-ensemble de E et $\vec{\Gamma_F}$ est l'ensemble des arcs de $\vec{\Gamma}$ dont les deux sommets sont dans F. On dit que H est le **sous-graphe de G induit par F**, et que $\vec{\Gamma_F}$ est la **restriction de $\vec{\Gamma}$ à l'ensemble F**.

Un **graphe partiel** de G est un graphe $G' = (E, \vec{\Gamma'})$ tel que $\vec{\Gamma'}$ est un sous-ensemble de $\vec{\Gamma}$.

Exemple 2 :



Sous graphe



Graphe partiel

1.2 Représentation en mémoire

1.2.1 Représentation d'un sous-ensemble

On va chercher à représenter les éléments constituant un sous-ensemble S d'un ensemble donné $E \subset \mathbb{N}$.

Liste chaînée - LC

Les éléments présents dans le sous-ensemble sont contenus dans la liste.

Exemple 3 : Prenons le sous-ensemble $S = \{1, 2, 4\}$ de $\mathbb{N}$.

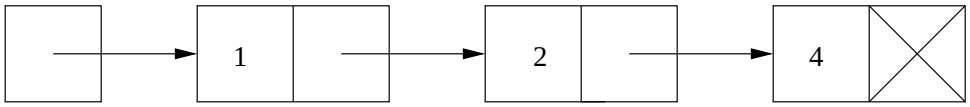


Tableau de booléens - TB

En supposant que $E = \{1 \dots n\}$, on utilise un tableau de booléens de taille n et tel que la case i du tableau vaut VRAI (1) si l'élément i appartient au sous-ensemble S et FAUX (0) sinon.

Exemple 4 : Prenons $E = \{1 \dots 9\}$ et $S = \{1, 2, 4\}$.

1	2	3	4	5	6	7	8	9
1	1	0	1	0	0	0	0	0

1.2.2 Opérations sur un ensemble

Le tableau suivant présente les principales opérations de base sur un ensemble ainsi que leur complexité selon le type de représentation mémoire. Le choix d'une structure de données adaptée est un facteur à prendre en compte lors de la conception d'un algorithme. Nous nous plaçons dans la configuration du pire cas où $|S| = |E| = n$.

Opération		LC	TB
Initialisation	$S \leftarrow \emptyset$	$O(1)$	$O(n)$
Existence/sélection	$\exists\, x \in S$?	$O(1)$	$O(n)$
Recherche	$x \in S$?	$O(n)$	$O(1)$
Insertion	$S \leftarrow S \cup \{x\}$	$O(n)/O(1)$	$O(1)$
Suppression	$S \leftarrow S \setminus \{x\}$	$O(1)$	$O(1)$
Parcours	$\forall x \in S$	$O(n)$	$O(n)$

L'initialisation consiste à mettre en place en mémoire la structure de données correspondante. Pour le TB, il faut initialiser toutes les cases à zéro ce qui impose une complexité en $O(n)$. Pour la LC, il suffit d'écrire NULL dans un pointeur $[O(1)]$.

La sélection permet de prendre un élément quelconque de l'ensemble ou de tester si l'ensemble est vide. Pour la LC il suffit de sélectionner le premier élément de la liste, par contre pour le TB il faut parcourir les cases jusqu'à trouver éventuellement un élément non nul $[O(n)]$.

La recherche permet de savoir si un élément donné est présent dans l'ensemble. Dans un TB il suffit de lire la case correspondante $[O(1)]$, pour une LC il faut parcourir l'ensemble des éléments présents $[O(n)]$.

La suppresion s'effectue en $O(1)$ pour la LC et le TB. Cependant l'insertion qui techniquement s'effectue aussi en $O(1)$ pour ces deux structures, pose un problème pour la LC. En effet, si nous voulons maintenir une liste d'éléments sans multiplicité : $\{1\} \cup \{1\} = \{1\}$, il faut avant l'insertion tester la présence de cet élément $[O(n)]$. Néanmoins, par exemple, si l'on sait que les éléments sont insérés une seule fois dans la liste, l'insertion peut s'effectuer en $O(1)$.

Le parcours est proportionnel au nombre d'éléments presents dans la LC $[O(n)]$ et au nombre de cases allouées dans le TB $[O(n)]$.

1.2.3 Représentation d'un graphe $(E, \vec{\Gamma})$

Double tableau - DT

Cette représentation consiste en deux tableaux I et T de taille $|\vec{\Gamma}|$, tels que $I(j)$ est le sommet initial de l'arc numéro j et $T(j)$ est le sommet terminal de l'arc numéro j.

Exemple 5 :

I	1	1	1	2	2	3
T	1	2	4	3	1	4

Représentation de $\vec{\Gamma}$ (voir exemple 1)

1.2.4 Représentation d'un graphe (E, Γ)

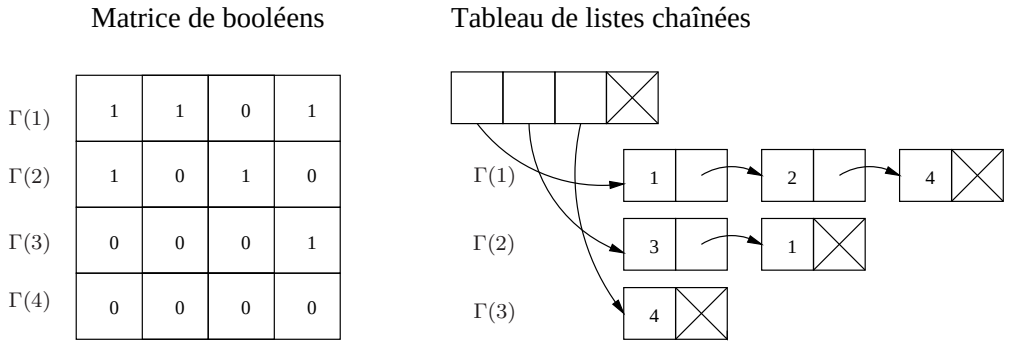
Tableau de listes chaînées - TLC

Cela consiste à prendre un tableau de taille $|E|$ contenant des pointeurs sur des listes chaînées tel que la i-ème liste chaînée corresponde à $\Gamma(i)$.

Matrice de booléens - MB

Il s'agit de la version bidimensionnelle des tableaux de booléens monodimensionnels (1.2.1). La matrice M est telle que $M_{ij} = 1$ si et seulement si (i, j) est un arc de G. La ligne M_i correspond au tableau de booléens monodimensionnel qui représente $\Gamma(i)$.

Exemple 6 :



Deux représentations possibles pour (E, Γ) (voir exemple 1)

Le choix des structures de données utilisées influe sur la complexité des programmes et donc sur leur efficacité.

Exercice 1
Décrivez par des diagrammes, comme dans les illustrations qui précèdent, les représentations possibles en mémoire des graphes de l'exemple 7 (section 1.3.1).

1.2.5 Évaluation de la complexité d'un algorithme

L'algorithme suivant a pour but de rechercher les successeurs d'une partie X d'un ensemble E.

Soit $G = (E, \vec{\Gamma})$, soit $X \subset E$. On définit l'ensemble des successeurs de la partie X ainsi :

$$\text{succ}(X) = \{\ y \in E \mid \exists x \in X\ ,\ (x, y) \in \vec{\Gamma}\ \} = \bigcup_{x \in X} \Gamma(x)$$

De la première forme de la définition, on déduit l'algorithme suivant :

Algorithme 1 : SuccPartie_1

Données : $X \subset E, \vec{\Gamma}$
Résultat : $Y \subset E$
1 $Y \leftarrow \emptyset$;
2 **pour chaque** $(x, y) \in \vec{\Gamma}$ **faire**
3 ⌊ **si** $x \in X$ **alors** $Y \leftarrow Y \cup \{y\}$;

Selon le choix du type de données pour X et Y, la complexité de l'algorithme précédent peut varier. Prenons par exemple Y sous forme d'un tableau de booléens et X sous forme de liste chaînée.
On note $n = |E|$ et $m = |\vec{\Gamma}|$.

L'instruction d'initialisation de Y à la ligne 1 est en $O(n)$. La boucle de la ligne 2 à 6 est executée m fois. Le test d'appartenance ligne 3 est en $O(n)$ car X est sous forme d'une liste chaînée, de plus il est executé m fois. On comptera donc $O(mn)$ pour la ligne 3. L'ajout d'un élément à un tableau de booléens, ligne 4, est en temps constant. On comptera donc pour la ligne 4 : $O(m \times 1)$. On peut conclure que la forme de la complexité de cet algorithme est en $O(n + m + mn + m) = O(mn)$.

Choisissons maintenant X sous forme d'un tableau de booléens. L'instruction de la ligne 3 devient en temps constant $O(1)$; le corps de boucle de la ligne 2-6 est en temps constant et est executé m fois. L'algorithme est donc en $O(m + n)$ ce qui est bien meilleur que la complexité précédente.

Exercice 2
Évaluez la complexité de l'algorithme suivant, inspiré de la seconde forme de la définition de succ(X), en supposant que Y est représenté par un tableau de booléens.

Algorithme 2 : SuccPartie_2

Données : $X \subset E, \Gamma$
Résultat : $Y \subset E$
1 $Y \leftarrow \emptyset$;
2 **pour chaque** $x \in X$ **faire**
3 **pour chaque** $y \in \Gamma(x)$ **faire**
4 $Y \leftarrow Y \cup \{y\}$;

1.3 Graphes orientés et non-orientés

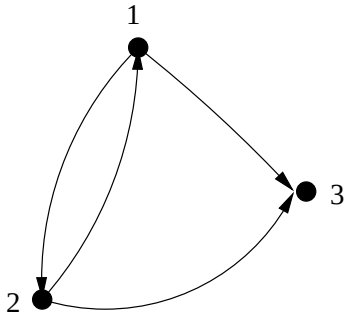
1.3.1 Le symétrique d'un graphe

Soit $G = (E, \Gamma)$. On appelle **symétrique** de G le graphe $G^{-1} = (E, \Gamma^{-1})$ défini par :

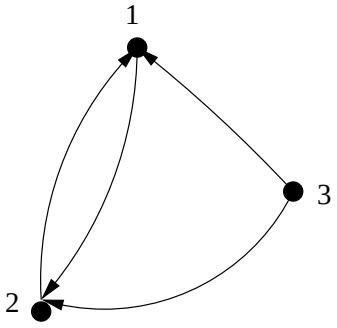
$$\forall x \in E \ , \ \Gamma^{-1}(x) = \{ \ y \in E \mid x \in \Gamma(y)\}$$

Il s'agit d'un graphe dont l'orientation des arcs a été inversée par rapport au graphe initial. Cette notation : Γ^{-1} est aussi utilisée pour représenter la "fonction prédécesseur", c'est à dire la fonction qui à tout sommet du graphe fait correspondre l'ensemble de ses prédecesseurs.

Exemple 7 :



Graphe G



Symétrique de G

Remarque : $y \in \Gamma^{-1}(x) \Leftrightarrow x \in \Gamma(y)$

Algorithme de calcul du symétrique

Algorithme 3 : Sym_1

Données : E, Γ
Résultat : Γ^{-1}

```
1 pour chaque x ∈ E faire Γ^{-1}(x) ← ∅ ;
2 pour chaque y ∈ E faire
3 |   pour chaque x ∈ Γ(y) faire
4 |   |   Γ^{-1}(x) ← Γ^{-1}(x) ∪ {y} ;
```

Algorithme 4 : Sym_2

Données : $\vec{\Gamma}$
Résultat : $\vec{\Gamma^{-1}}$

```
1 Γ^{→-1} ← ∅ ;
2 pour chaque (x, y) ∈ Γ→ faire
3 |   Γ^{→-1} ← Γ^{→-1} ∪ {(y, x)} ;
```

Exercice 3
Évaluez la complexité de ces deux algorithmes.

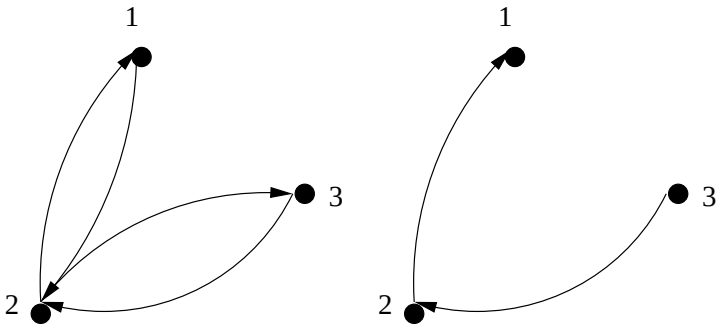
1.3.2 Graphe symétrique, graphe antisymétrique

Le graphe $G = (E, \Gamma)$ est un **graphe symétrique** si $\Gamma^{-1} = \Gamma$. Autrement dit, G est un graphe symétrique si

$$\forall x \in E\ ,\ \forall y \in E\ ,\ x \in \Gamma(y) \Leftrightarrow y \in \Gamma(x)$$

Un graphe $G = (E, \Gamma)$ est **antisymétrique** si $\forall x, y \in E\ ,\ [x \in \Gamma(y) \text{ et } y \in \Gamma(x)] \Rightarrow y = x$

Exemple 8 :



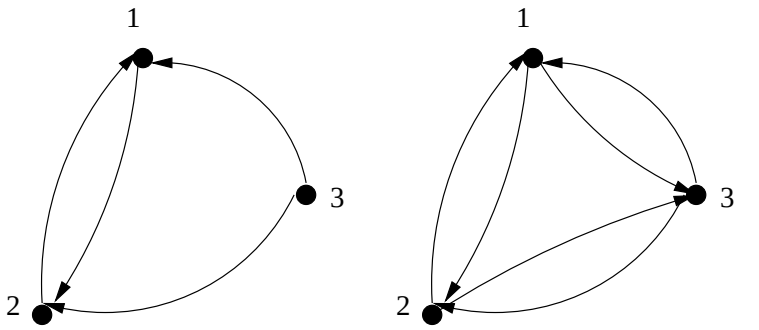
Graphe symétrique Graphe antisymétrique

1.3.3 Fermeture symétrique

Soit $G = (E, \Gamma)$. On appelle **fermeture symétrique** de G le graphe $G_s = (E, \Gamma_s)$ défini par :

$$\forall x \in E\ ,\ \ \Gamma_s(x) = \Gamma(x) \cup \Gamma^{-1}(x)$$

Exemple 9 :



Graphe G Fermeture symétrique de G

1.3.4 Graphe non-orienté

Définition

On appelle **graphe non-orienté** sur E un couple $G = (E,\ \overline{\Gamma})$ tel que $\overline{\Gamma}$ soit un sous-ensemble de $\{\ \{x, y\} \mid x \in E,\ y \in E\ \}$. Ainsi $\overline{\Gamma}$ est de la forme : $\{\{a, b\}, \{c, d\}, \{e, f\}, ...\}$. Tout élément de $\overline{\Gamma}$: $\{x, y\}$ est appelé **arête** du graphe. On dit que l'arête $\{x, y\}$ est **adjacente** au sommet x et au sommet y. A $\overline{\Gamma}$ on peut associer l'application $\Gamma_{n.o.}$ de $E \longrightarrow \mathcal{P}(E)$ définie par :

$$y \in \Gamma_{n.o.}(x) \Leftrightarrow \{x, y\} \in \overline{\Gamma}$$

Dans tous les cas, $(E, \Gamma_{n.o.})$ est un graphe symétrique.

Ainsi la donnée d'un graphe symétrique est équivalente à la donnée d'un graphe non-orienté.

Graphe non-orienté associé à un graphe orienté

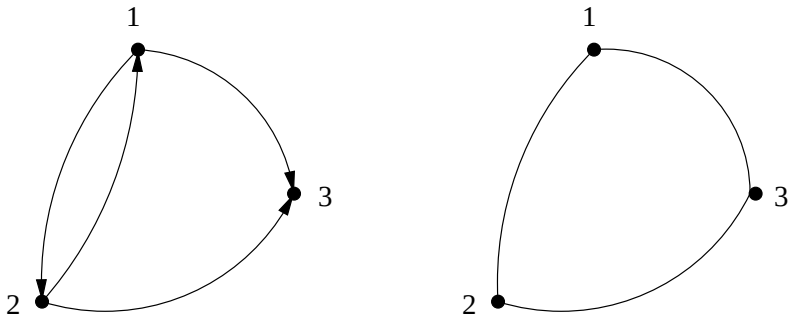
Il se peut qu'à partir d'un graphe orienté, l'on cherche à travailler dans un contexte non-orienté. Pour cela nous associons au graphe donné (E, Γ) sa version non-orientée $(E,\ \overline{\Gamma})$ définie par :

$$\{x, y\} \in \overline{\Gamma} \Leftrightarrow (x, y) \in \vec{\Gamma} \; ou \; (y, x) \in \vec{\Gamma}$$

ou encore par :

$$\Gamma_{n.o.} = \text{ fermeture symétrique } (\Gamma)$$

Exemple 10 :



Graphe G — Graphe non orienté associé à G

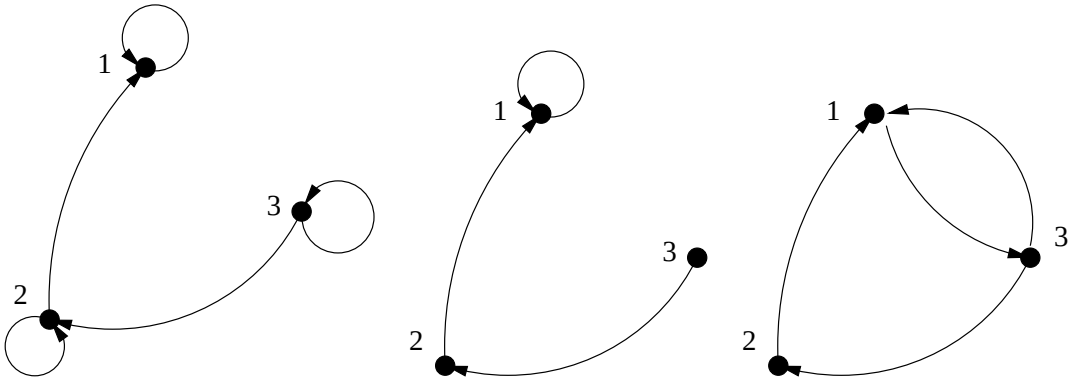
1.3.5 Réflexivité, antiréflexivité

Un graphe $G = (E, \Gamma)$ est **réflexif** si $\forall x \in E \ , \ x \in \Gamma(x)$.

Un graphe $G = (E, \Gamma)$ est **antiréflexif** si $\forall x \in E \ , \ x \notin \Gamma(x)$.

Un arc de la forme (x, x) est appelé une **boucle**.

Exemple 11 :

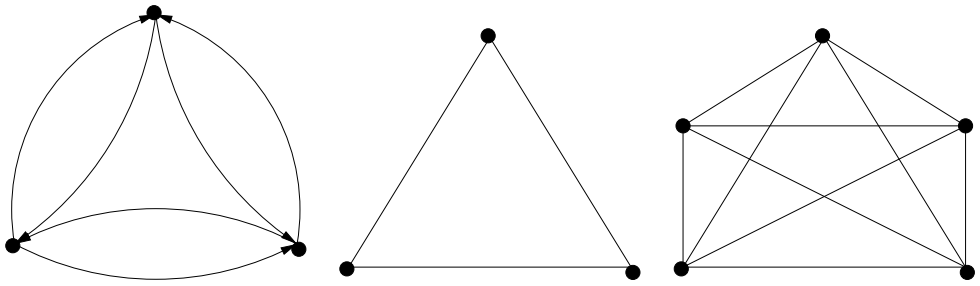


Graphe réfléxif — Graphe non réflexif et non antiréflexif — Graphe antiréflexif (sans boucle)

1.3.6 Graphe complet

Un graphe $G = (E, \vec{\Gamma})$ est dit **complet** si pour tout couple de sommets (x, y) distincts, on a $(x, y) \in \vec{\Gamma}$. Un graphe complet est aussi appelé **clique** à n sommets.

Exemple 12 :



Graphe complet (orienté) sur trois sommets

Graphe complet (non-orienté) sur trois sommets

Graphe complet (non-orienté) sur cinq sommets

Exercice 4
Soit $E = \{1, 2, \ldots, n, n+1\}$. Soit $G = (E, \overline{\Gamma})$ le graphe complet non-orienté sur E.
Décrivez deux façons différentes de compter les arêtes de G.
En déduire l'égalité $1 + 2 + \ldots + n = \frac{n(n+1)}{2}$.

1.4 Chemins, connexité

1.4.1 Chemins et chaînes

Soit $G = (E, \Gamma)$, soient x_0 et x_k deux sommets de E. Un **chemin** C de x_0 à x_k est une séquence ordonnée $C = (x_0, \ldots, x_k)$ de sommets de E telle que $\forall i \in [1, k]$, $x_i \in \Gamma(x_{i-1})$. En d'autres termes, chaque sommet x_i du chemin est un sucesseur du sommet x_{i-1}. Une séquence telle que $C = (x_0)$ est également appelée **chemin** de x_0 à x_0, un tel chemin est qualifié de **trivial**.

On dit que k est la **longueur** du chemin : c'est aussi le nombre d'arcs du chemin. La longueur d'un chemin trivial est nulle. Le chemin C est un **circuit** si $x_0 = x_k$, avec $k > 0$.

Chemin élémentaire

Le chemin C est dit **élémentaire** si les sommets x_i sont tous distincts (sauf éventuellement x_0 et x_k).

Proposition : Tout chemin C de x à y dans G contient (au sens d'une suite extraite) un chemin élémentaire de x à y dans G.

Exercice 5
Démontrez cette proposition.

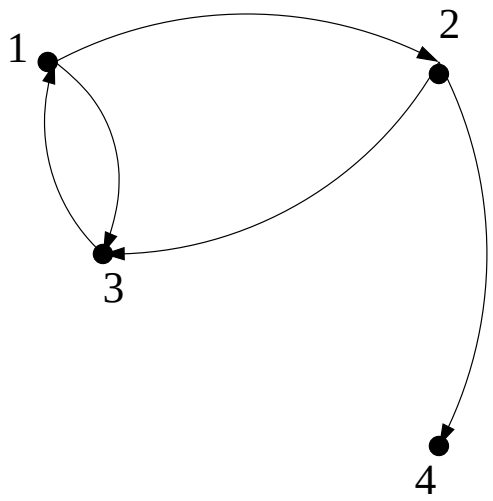
Exercice 6
Proposez un algorithme pour, étant donné un graphe et un chemin C quelconque dans ce graphe liant deux sommets x et y, retourne un chemin élémentaire de x à y extrait de C. Évaluez la complexité de votre algorithme.

Proposition : Soit $G = (E, \Gamma)$, avec $|E| = n$. La longueur d'un chemin élémentaire dans G qui n'est pas un circuit est inférieure ou égale à $n-1$. De même la longueur d'un circuit élémentaire est inférieure ou égale à n.

Chaînes et cycles

Définition : Soit $G = (E, \Gamma)$, et Γ_s la fermeture symétrique de Γ. On appelle **chaîne** tout chemin dans (E, Γ_s).
On appelle **cycle** tout circuit dans (E, Γ_s) ne passant pas deux fois par la même arête.

Exemple 13 :



- $(1, 2, 1)$ n'est pas un chemin.
- $(1, 2, 3)$ est un chemin.
- $(1, 2, 3, 1, 2, 4)$ est un chemin.
- $(4, 2, 1)$ n'est pas un chemin.
- $(4, 2, 1)$ est une chaîne.
- $(1, 2, 3, 1)$ est un circuit.
- $(1, 3, 2, 1)$ n'est pas un circuit.
- $(1, 3, 2, 1)$ est un cycle.
- $(1, 3, 1)$ n'est pas un cycle.

Le tableau suivant résume les différents termes employés selon la nature du graphe considéré :

Graphe orienté	Graphe non-orienté
arc	arête
chemin	chaîne
circuit	cycle

1.4.2 Composante connexe et fortement connexe

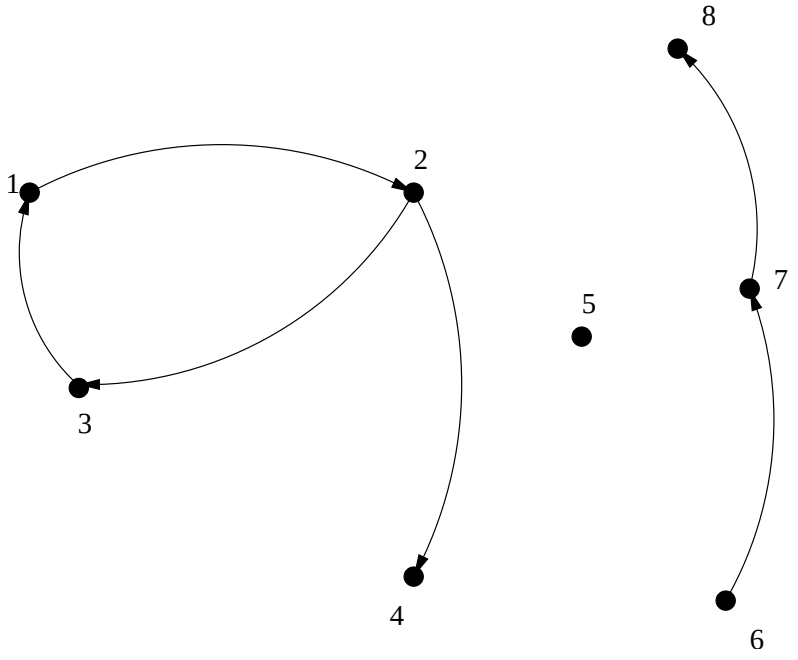
Composante connexe

Définition : Soit $G = (E, \Gamma)$ et $x \in E$. On définit C_x, la **composante connexe** de G contenant x par :

$$C_x = \{y \in E \mid \text{il existe une chaîne de } x \text{ à } y\}$$

Autrement dit, la composante connexe contenant le sommet x est l'ensemble des sommets qu'il est possible d'atteindre par une *chaîne* à partir de x.

Remarque : Cette définition sous-entend que l'on travaille toujours dans la version non-orientée du graphe. Remarquons que x appartient toujours à la composante C_x, en effet il existe une chaîne de longueur nulle (chaîne triviale) de x à x.



Exemple 14 :
- $C_1 = \{1, 2, 3, 4\} = C_2 = C_3 = C_4$
- $C_5 = \{5\}$
- $C_6 = C_7 = C_8 = \{6, 7, 8\}$

Composante fortement connexe

Définition : Soit $G = (E, \Gamma)$ et $x \in E$. On définit C'_x, la **composante fortement connexe** de G contenant x par :

$$C'_x = \{y \in E \mid \text{il existe un chemin de } x \text{ à } y \text{ et il existe un chemin de } y \text{ à } x\}$$

Remarque : Remarquons que x appartient toujours à la composante C'_x, en effet il existe un chemin de longueur nulle (chemin trivial) de x à x.

Exemple 15 : Dans le graphe de l'exemple précédent :
- $C'_1 = \{1, 2, 3\} = C'_2 = C'_3$
- $C'_4 = \{4\}$
- $C'_5 = \{5\}$
- $C'_6 = \{6\}$
- $C'_7 = \{7\}$
- $C'_8 = \{8\}$

1.4.3 Calcul des composantes fortement connexes et des composantes connexes

Définitions préliminaires

Soit un graphe $G = (E, \Gamma)$, on définit la famille d'ensembles de sommets de E :
- $\Gamma_0(x) = \{x\}$
- $\Gamma_1(x) = \Gamma(x)$
- $\Gamma_2(x) = \text{succ}(\Gamma_1(x))$
- ...
- $\Gamma_p(x) = \text{succ}(\Gamma_{p-1}(x))$

où succ(X) désigne l'ensemble des successeurs de la partie X, autrement dit, succ$(X) = \cup_{x \in X} \Gamma(x)$ (voir aussi la section 1.2.5).

De même, on peut définir pred$(X) = \cup_{x \in X} \Gamma^{-1}(x)$ ainsi que la famille $\Gamma_p^{-1}(x) = \text{pred}(\Gamma_{p-1}^{-1}(x))$.

Proposition :
$\Gamma_p(x) = \{y \in E$, il existe un chemin de x à y dans G de longueur $p\}$
$\Gamma_p^{-1}(x) = \{y \in E$, il existe un chemin de y à x dans G de longueur $p\}$

On définit, pour tout $x \in E$,

$$\hat{\Gamma}_p(x) = \bigcup_{i=0}^{p} \Gamma_i(x)$$

$$\hat{\Gamma}_p^{-1}(x) = \bigcup_{i=0}^{p} \Gamma_i^{-1}(x)$$

Proposition :
$\hat{\Gamma_p}(x) = \{y \in E\ ,\ $ il existe un chemin de x à y dans G de longueur $\leq p\}$
$\hat{\Gamma_p^{-1}}(x) = \{y \in E\ ,\ $ il existe un chemin de y à x dans G de longueur $\leq p\}$

On définit aussi $\hat{\Gamma_\infty}(x)$ comme l'ensemble des $y \in E$ tel qu'il existe un chemin de x à y dans G, et $\hat{\Gamma_\infty^{-1}}(x)$ comme l'ensemble des $y \in E$ tel qu'il existe un chemin de y à x dans G.

Proposition : Soit $n = |E|$, la composante fortement connexe C'_x de G contenant x se calcule par la formule :

$$C'_x = \hat{\Gamma}_{n-1}(x) \cap \hat{\Gamma}^{-1}_{n-1}(x)$$

La composante connexe de G contenant x s'écrit :

$$C_x = [\hat{\Gamma_s}]_{n-1}(x) = [\hat{\Gamma_s}^{-1}]_{n-1}(x)$$

où Γ_s est la fermeture symétrique de Γ.

Démonstration : par définition nous avons $C'_x = \hat{\Gamma}_\infty(x) \cap \hat{\Gamma}^{-1}_\infty(x)$. Soit un chemin de i à j. Soit le chemin élémentaire associé, c'est un chemin de i à j ayant au plus $n-1$ arcs. Ainsi la connaissance des chemins de longueur $n-1$ est suffisante (puisque cet ensemble contient tous les chemins élémentaires) pour déterminer $\hat{\Gamma}_\infty (= \hat{\Gamma}_{n-1})$.

Calcul de la composante fortement connexe

Algorithme 5 : CompFoCo

Données : $(E, \Gamma, \Gamma^{-1}); a \in E$
Résultat : C'_a

$\hat{\Gamma}^\infty(a) \leftarrow \emptyset$; $T \leftarrow \{a\}$;
tant que $\exists x \in T$ **faire**
  $T \leftarrow T \setminus \{x\}$;
  pour chaque $y \in \Gamma(x)$ **tel que** $y \notin \hat{\Gamma}^\infty(a)$ **faire**
    $\hat{\Gamma}^\infty(a) \leftarrow \hat{\Gamma}^\infty(a) \cup \{y\}$;
    $T \leftarrow T \cup \{y\}$;
$\hat{\Gamma}^{-\infty}(a) \leftarrow \emptyset$, $T \leftarrow \{a\}$;
tant que $\exists x \in T$ **faire**
  $T \leftarrow T \setminus \{x\}$;
  pour chaque $y \in \Gamma^{-1}(x)$ **tel que** $y \notin \hat{\Gamma}^{-\infty}(a)$ **faire**
    $\hat{\Gamma}^{-\infty}(a) \leftarrow \hat{\Gamma}^{-\infty}(a) \cup \{y\}$;
    $T \leftarrow T \cup \{y\}$;
$C'_a \leftarrow \left[\hat{\Gamma}^\infty(a) \cap \hat{\Gamma}^{-\infty}(a)\right] \cup \{a\}$;

Exercice 7
Évaluez la complexité de cet algorithme.

Exercice 8
En vous inspirant de l'algorithme ci-dessus, proposez un algorithme pour tester l'existence d'un circuit passant par un sommet donné x dans un graphe donné.
Proposez une variante permettant de détecter un cycle.
Donnez la complexité de ces deux algorithmes.

Calcul de la composante connexe

Algo COMPCO (**Données :** $(E, \Gamma, \Gamma^{-1}); a \in E$; **Résultat :** C_a)

1: $C_a \leftarrow \{a\}$; $T \leftarrow \{a\}$
2: **while** $\exists x \in T$ **do**
3: $T \leftarrow T \setminus \{x\}$
4: **for all** $y \in \Gamma(x)$ tel que $y \notin C_a$ **do**
5: $C_a \leftarrow C_a \cup \{y\}$
6: $T \leftarrow T \cup \{y\}$
7: **end for**
8: **for all** $y \in \Gamma^{-1}(x)$ tel que $y \notin C_a$ **do**
9: $C_a \leftarrow C_a \cup \{y\}$
10: $T \leftarrow T \cup \{y\}$
11: **end for**
12: **end while**

1.4.4 Degrés

Soit un graphe (orienté) $G = (E, \Gamma)$ et un sommet $x \in E$. On définit le **degré exterieur** de x, noté $d^+(x)$ par :

$$d^+(x) = |\Gamma(x)|$$

Le degré extérieur est le nombre de successeurs de x.
On définit le **degré intérieur** de G, noté $d^-(x)$ par :

$$d^-(x) = |\Gamma^{-1}(x)|$$

Le degré intérieur est le nombre de prédecesseurs de x.
Le **degré** du sommet x est défini par $d(x) = d^+(x) + d^-(x)$.

Soit un graphe non-orienté $G = (E, \overline{\Gamma})$ et un sommet $x \in E$. On définit le **degré** de x, noté $d(x)$ par :

$$d(x) = |\{a \in \overline{\Gamma} \mid x \in a\}|$$

En d'autres termes, dans un graphe non-orienté, le degré du sommet x est le nombre d'arêtes adjacentes à x.

Exercice 9
Démontrez les deux affirmations suivantes :
– La somme des degrés de tous les sommets d'un graphe G quelconque est paire.
– Dans tout graphe, le nombre de sommets de degré impair est pair.

Exercice 10
Peut-on tracer dans le plan cinq droites distinctes, telles que chacune d'entre elles ait exactement trois points d'intersection avec les autres ? En modélisant ce problème par un graphe, démontrez que cela n'est pas possible.

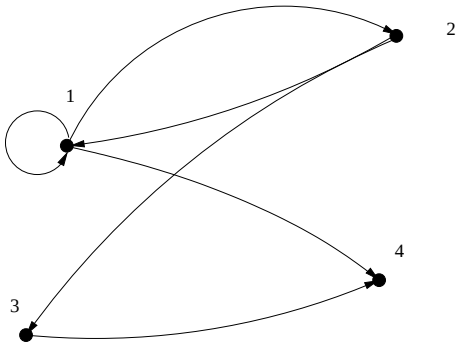
Exercice 11
Voici une affirmation : "dans toute réunion mondaine, il y a au moins deux personnes ayant le même nombre d'amis présents" (on suppose la relation d'amitié symétrique).
Dans le cadre des graphes, à quelle propriété correspond cette affirmation ?
Cette propriété est elle fausse ou vraie ? Donnez une preuve de ce que vous avancez.

1.5 Représentation matricielle

Soit $G = (E, \Gamma)$, avec $|E| = n$. On peut représenter G par la matrice booléenne $\mathcal{M}_\Gamma = [m_{ij}]_{i,j=1\ldots n}$, dite **matrice d'adjacence** de G, telle que :

$$\begin{cases} m_{ij} & = & 1 \text{ si } j \in \Gamma(i) \\ m_{ij} & = & 0 \text{ sinon} \end{cases}$$

Exemple 16 : Soit le graphe $G = (E, \Gamma)$ suivant :



La matrice d'adjacence $\mathcal{M}_\Gamma$ associée à G est la suivante :

	1	2	3	4
1	1	1	0	1
2	1	0	1	0
3	0	0	0	1
4	0	0	0	0

On définit le produit booléen de matrices à partir des opérateurs $\vee$ (ou) et $\wedge$ (et).
Soient $A = [a_{ij}]_{i,j=1\ldots n}$ et $B = [b_{ij}]_{i,j=1\ldots n}$ deux matrices booléennes, alors $C = A \times B =$

$[c_{ij}]_{i,j=1\ldots n}$ est définie par :

$$\forall i,j = 1\ldots n\ ,\ c_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj})$$

Proposition : Soit $M^p = M \times \ldots \times M$ (p fois), où M est la matrice booléenne associée au graphe G. On a : $M^p_{ij} = 1 \Leftrightarrow$ il existe un chemin de longueur p dans G de i à j.

Exercice 12
Démontrez cette proposition.

Remarque : Pour tout graphe G, il n'existe pas en général de nombre $k \in \mathbb{N}$ tel que $M^k = M^{k+1}$. (Par exemple, il suffit de prendre le graphe formé par deux sommets a et b et deux arcs (a,b) et (b,a)).

Proposition : Soit $\hat{M}^p = I \vee M \vee M^2 \vee \ldots M^p$., où I désigne la matrice identité de la même taille que M.
On a : $\hat{M}^p_{ij} = 1 \Leftrightarrow$ il existe un chemin de longueur $\leq p$ dans G de i à j.

Remarque : Si $|E| = n$ alors $\forall p \geq n-1$, $\hat{M}^p = \hat{M}^{n-1} = \hat{M}^\infty$.

Exercice 13
Soit M la matrice booléenne :

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

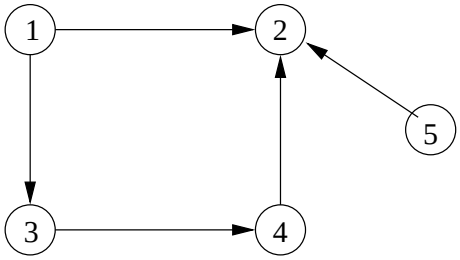
A quoi est égal M^{2006} ? Justifiez votre réponse. On évitera, autant que possible, les calculs !

1.6 Graphe biparti

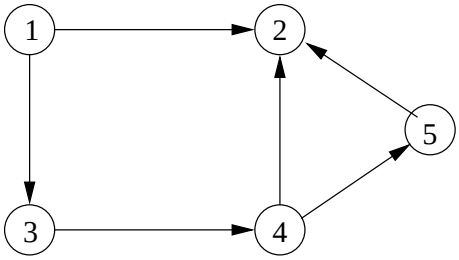
On dit qu'un graphe $G = (E, \Gamma)$ est **biparti** si l'ensemble E des sommets peut être partitionné en deux sous-ensembles distincts E_1 et E_2 de telle sorte que :

$$\forall (x,y) \in \vec{\Gamma}, \begin{cases} x \in E_1 & \Rightarrow & y \in E_2 \\ x \in E_2 & \Rightarrow & y \in E_1 \end{cases}$$

Exemple 17 :



G_1



G_2

On voit que G_1 est biparti avec $E_1 = \{1, 4, 5\}$ et $E_2 = \{2, 3\}$. Le graphe G_2 n'est pas biparti.

Exercice 14
Montrez qu'un graphe est biparti si et seulement si il ne contient pas de cycle impair.

Indication : L'implication "biparti $\Rightarrow$ sans cycle impair" se démontre facilement (par l'absurde). Dans l'autre sens c'est plus dur. On peut penser à choisir un sommet x arbitrairement et, en supposant le graphe connexe, à considérer les longueurs des chaînes les plus courtes de x à tout autre sommet du graphe.

Chapitre 2

Arbres et arborescences

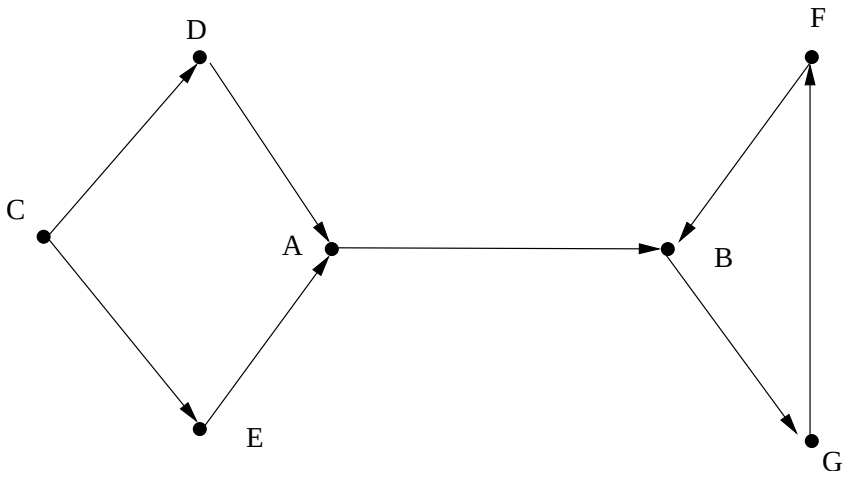
Dans ce chapitre, on considèrera un graphe G antisymétrique.

2.1 Définitions

2.1.1 Isthme

Soit $G = (E, \vec{\Gamma})$, $u \in \vec{\Gamma}$. On dit que u est un **isthme** si sa suppression de $\vec{\Gamma}$ augmente le nombre de composantes connexes de G.

Exemple 18 :



L'arc $u = (A, B)$ est un isthme. Aucun autre arc de ce graphe n'est un isthme.

Proposition : $G = (E, \vec{\Gamma}),\ a \in E,\ b \in E$. L'adjonction d'un nouvel arc $u = (a, b)$ à G pour donner un nouveau graphe $G' = (E, \vec{\Gamma} \cup \{u\})$ a pour effet :
- soit de diminuer de 1 le nombre de composantes connexes, dans ce cas u n'appartient à

aucun cycle de G' ;
- soit de laisser inchangé le nombre de composantes connexes, dans ce cas u appartient à un cycle de G'.

Démonstration :
- Si a et b appartiennent à deux composantes connexes distinctes, le nombre de composantes connexes diminue de 1. L'arc u ne peut appartenir à un cycle de G' car si on supprime u dans ce cycle, on aurait une chaîne joignant les deux composantes connexes dans G.
- Si a et b appartiennent à la même composante connexe, le nombre de composantes connexes reste inchangé. La concaténation de la chaîne joignant a et b dans G et de u donne un cycle dans G'.

Proposition : u est un isthme $\Leftrightarrow$ u n'appartient à aucun cycle de G.

Proposition : Soit $G = (E, \vec{\Gamma})$, tel que $|E| = n$ et $|\vec{\Gamma}| = m$:
1- G connexe $\Rightarrow m \geq n - 1$
2- G est sans cycle $\Rightarrow m \leq n - 1$.

2.1.2 Arbre

Un **arbre** est un graphe connexe sans cycle.

Remarque : Un arbre ne comporte pas de boucles. En effet, toute boucle est un cycle.

Remarque : Dans un arbre, chaque arc est un isthme.

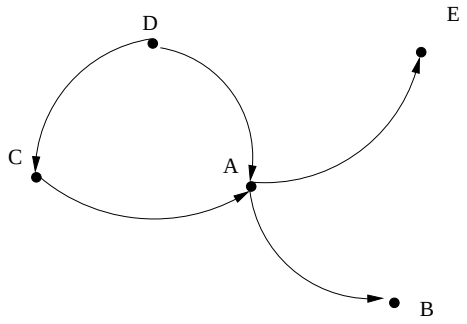
Proposition : Soit $G = (E, \vec{\Gamma})$, tel que $|E| = n \geq 2$ et $|\vec{\Gamma}| = m$. Les propriétés suivantes sont équivalentes :

1. G est connexe et sans cycle,
2. G est connexe et $m = n - 1$,
3. G est sans cycle et $m = n - 1$,
4. G est sans boucles et il existe une unique chaîne élémentaire joignant tout couple de sommets a et b distincts.

2.1.3 Racine

Soit $G = (E, \vec{\Gamma})$, $x \in E$. On dit que x est une **racine** de G si $\forall y \in E \setminus \{x\}$, il existe un chemin de x à y.

Exemple 19 :



Ce graphe a pour racine D.

Exercice 15
Un graphe G est dit *quasi fortement connexe* si, pour toute paire de sommets distincts x, y de G, on peut trouver un troisième sommet z tel qu'il existe un chemin de z à x et un chemin de z à y.
Montrer que si G est quasi fortement connexe alors G admet une racine.

2.1.4 Arborescences

Soit $G = (E, \Gamma)$ un graphe, soit $r \in E$. Le graphe G est une **arborescence de racine** r si :

1. G est antisymétrique, et
2. G est un arbre, et
3. r est une racine de G.

Un sommet d'une arborescence est appelé une **feuille** s'il n'a aucun successeur.

Exercice 16
Montrez que si un graphe G est une arborescence de racine r, alors aucun autre sommet de G n'est une racine de G.

2.1.5 Découpage en niveaux

Une arborescence peut être découpée en niveaux (voir figure 2.1 page suivante).

Exercice 17
Proposez une définition formelle de la notion d'"ensemble de sommets de niveau k" d'une arborescence.
Proposez un algorithme pour afficher les ensembles de sommets correspondant aux différents niveaux d'une arborescence (connaissant sa racine). Évaluez sa complexité.

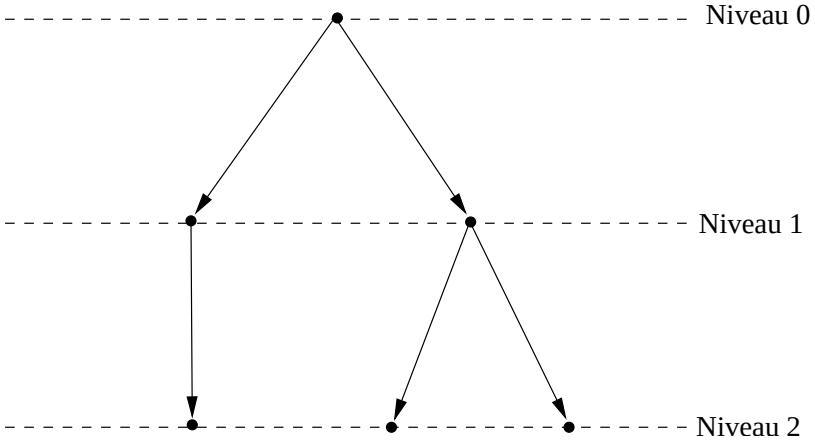


FIG. 2.1 – Représentation en niveaux

2.2 Exemples et applications

2.2.1 Fausse monnaie

Soit 8 pièces de monnaies, dont une est fausse (elle est plus légère que les autres). On possède deux balances. La balance 1 permet de savoir quel plateau est le plus lourd et ne possède que deux états $(<, \geq)$; la balance 2 possède trois états ($>$, $<$, $=$). On cherche à trouver la fausse pièce en minimisant le nombre de pesées. Pour modéliser la résolution de ce problème, on va utiliser une arborescence.

2.2.2 Arborescence de décision

Soit $G = (E, \vec{\Gamma})$ une arborescence de racine $r \in E$. Soit X un ensemble fini (dont les éléments représenteront pour nous les "possibilités"). On dit que G est une **arborescence de décision** sur X s'il existe une application $f : E \to \mathcal{P}(X)$ telle que :
– $f(r) = X$
– $\forall a \in E \ / \ \Gamma(a) \neq \emptyset$, $f(a) = \bigcup_{b \in \Gamma(a)} f(b)$,
– $\forall a \in E \ / \ \Gamma(a) = \emptyset$, $|f(a)| = 1$.
Dans le cas de notre exemple de la balance 1, on pose 4 pièces sur chaque plateau. Un des deux plateaux sera forcément moins lourd car il contient une fausse pièce. On prend le lot de 4 pièces les moins lourdes et on le divise en 2 et ainsi de suite. Le nombre de pesées pour trouver la fausse pièce est égale au nombre de niveau du graphe 2.2 page suivante moins 1, soit ici 3.

On peut suivre la même approche pour la balance 2. On aura alors un arbre à 3 niveaux et une solution en deux pesées.

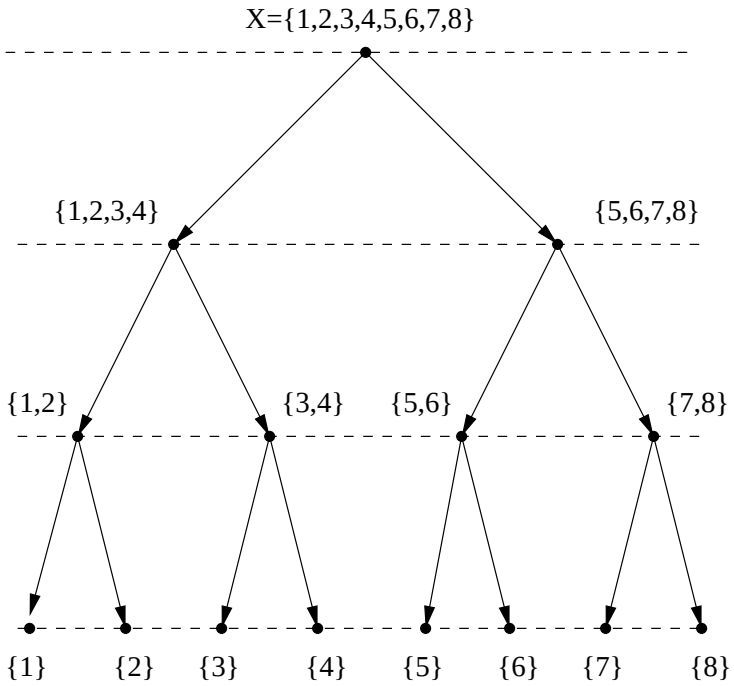


FIG. 2.2 – Arbre de décision pour la balance 1

2.2.3 Arithmétique

On s'intéresse à l'évaluation d'expressions arithmétiques, en respectant les priorités des différents opérateurs. Soit l'expression arithmétique (A) :

$$(A) = \frac{ae^x - b(y+1)}{2}$$

Sur l'arbre de la figure 2.3 page suivante correpondant à l'expression (A), chaque sommet du graphe est soit une donnée (symbolique ou numérique), soit un opérateur. Les opérateurs binaires possédent deux successeurs, les opérateurs unaires en possèdent un. Les données n'ont aucun sucesseur, ce sont les feuilles de l'arbre. Le problème est l'ordre d'évaluation, car $a - b \neq b - a$. Il est donc nécessaire d'ordonner les successeurs d'un sommet.

2.2.4 Arborescence ordonnée

Soit E un ensemble fini, on note $O(E)$ l'ensemble des $k-$uplets ordonnés d'éléments de E. Un couple $\mathring{G} = (E, \mathring{\Gamma})$, où $\mathring{\Gamma}$ est une application de E dans $O(E)$ est appelé un **graphe ordonné** sur E. Ainsi :

$$\forall a \in E,\ \mathring{\Gamma}(a) = (x_1, ..., x_i)$$

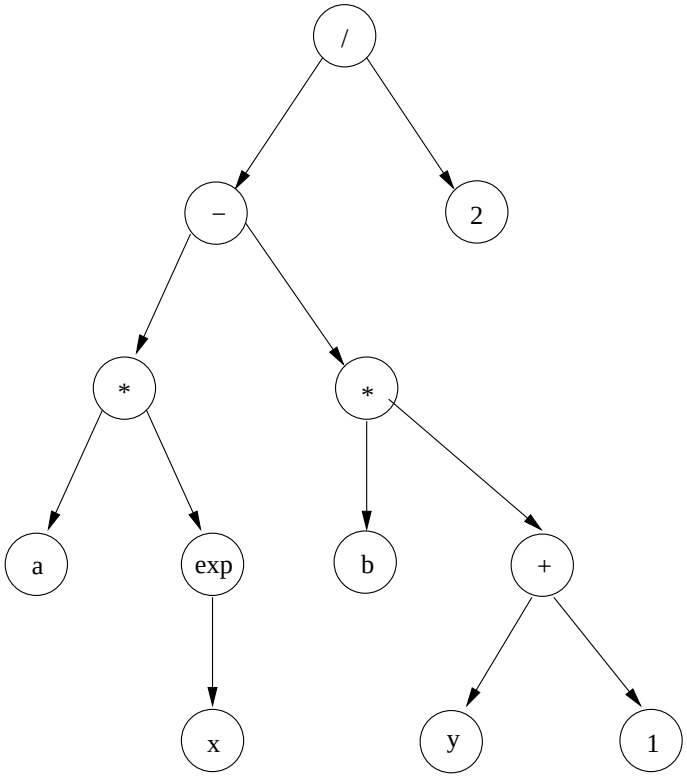


FIG. 2.3 – Arbre d'évaluation arithmétique

A tout graphe ordonné $(E, \mathring{\Gamma})$, on peut faire correspondre le graphe (E, Γ) tel que $\Gamma(a) = \{x_1, ..., x_i\} = \{$ éléments du $k-$uplet $\mathring{\Gamma}(a)\}$. On dit que $\mathring{G} = (E, \mathring{\Gamma})$ est une **arborescence ordonnée** (ou arborescence plane) si (E, Γ) est une arborescence.

2.2.5 Arborescence de recherche

Soit D un ensemble appelé **domaine**, muni d'une relation d'ordre total (que l'on notera $\leq$). Soit $X \subset D$ et $n \in D$. La question que l'on se pose est de savoir si n appartient à l'ensemble X.

On peut par exemple prendre D égal à l'ensemble des chaînes de caractères alphabétiques, X une partie de ces chaînes (par exemple l'ensemble des mots d'un dictionnaire), et $\leq$ représentant l'ordre lexicographique usuel. On peut aussi, plus classiquement, considérer le domaine $D = \mathbb{N}$ et l'ordre habituel sur les entiers naturels.

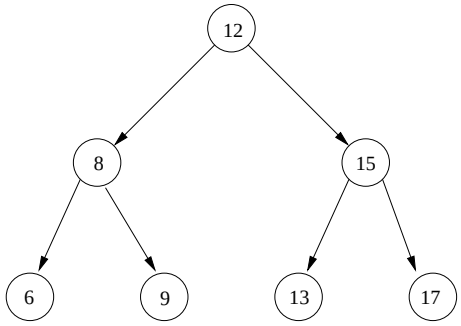
Si X est représenté sous forme d'une liste, la résolution du problème $n \in X$? s'effectue en $O(|X|)$ comparaisons entre éléments de D. Si X est représenté sous forme d'un tableau de booléens, la résolution s'effectue en $O(1)$ comparaisons mais l'encombrement mémoire est en $O(|D|)$ (impraticable si le domaine est très grand). Si X est sous forme d'une arborescence binaire, c'est-à-dire que le nombre de successeurs de tout sommet est compris

entre 0 et 2, la recherche peut sous certaines conditions s'effectuer en $O(log|X|)$.

Il faut pour cela imposer les conditions suivantes :
– l'arborescence est ordonnée (cf. section 2.2.4).
– pour tout sommet x, toute valeur située dans la sous-arborescence gauche de x est inférieure à la valeur associée à x, et toute valeur située dans la sous-arborescence droite de x est supérieure à la valeur associée à x.
Une arborescence binaire respectant les conditions ci-dessus est appelée **arborescence de recherche**.

Il faut de plus que l'arborescence soit équilibrée, autrement la recherche s'effectue dans le pire des cas en $O(|X|)$.

Exemple 20 :



Arborescence de recherche pour la relation $\leq$ définie sur le domaine $\mathbb{N}$, avec $X = \{6, 8, 9, 12, 15, 13, 17\}$.

Exercice 18
Écrivez en pseudo-langage un algorithme permettant de tester la présence d'une valeur donnée v dans une arborescence de recherche.

Exercice 19
Écrivez deux programmes récursifs, l'un permettant d'afficher les valeurs d'une arborescence de recherche dans l'ordre croissant, l'autre dans l'ordre décroissant.

2.3 Arbre de poids extrémum

2.3.1 Graphe Pondéré

Soit $G = (E, \vec{\Gamma})$, un graphe connexe et sans boucles. Soit P une application de $\vec{\Gamma} \rightarrow \mathbb{R}$. Soit $u \in \vec{\Gamma}$, $P(u)$ est appelé **poids** de l'arc u.

Le graphe $(E, \vec{\Gamma}, P)$ est appelé **graphe pondéré**. Soit $\vec{\Gamma}' \subset \vec{\Gamma}$ un graphe partiel de G, $P(\vec{\Gamma}') = \sum_{u \in \vec{\Gamma}'} P(u)$ définit le **poids du graphe partiel** $\vec{\Gamma}'$.

Etant donné un graphe non orienté dont les arêtes sont munies d'une valuation, on cherche un graphe partiel de ce graphe qui soit un arbre et qui soit de poids maximum. Puisqu'il s'agit d'un graphe partiel, cet arbre a pour ensemble de sommets E tout entier. On appelle **arbre couvrant** un tel arbre.

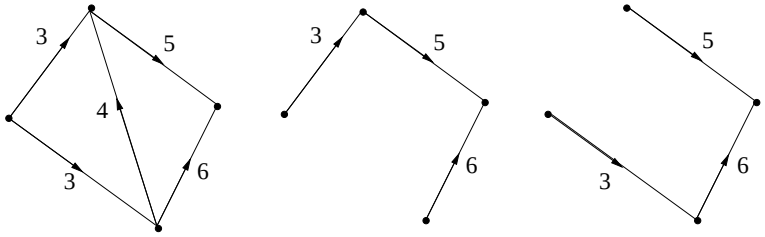
Soit $(E, \vec{\Gamma}')$ un graphe partiel de G qui soit un arbre. On dit que $(E, \vec{\Gamma}')$ est un **arbre de poids maximum** sur G, si $\forall \vec{\Gamma}'' \subset \vec{\Gamma}$ tel que $(E, \vec{\Gamma}'')$ soit un arbre, on a la relation suivante :

$$P(\vec{\Gamma}'') \leq P(\vec{\Gamma}')$$

Soit $(E, \vec{\Gamma}')$ un graphe partiel de G qui soit un arbre. On dit que $(E, \vec{\Gamma}')$ est un **arbre de poids minimum** sur G, si $\forall \vec{\Gamma}'' \subset \vec{\Gamma}$ tel que $(E, \vec{\Gamma}'')$ soit un arbre, on a la relation suivante :

$$P(\vec{\Gamma}'') \geq P(\vec{\Gamma}')$$

Exemple 21 :



Un graphe pondéré G et des arbres de poids maximum sur G.

Proposition : Un arbre de poids maximum pour $G = (E, \vec{\Gamma}, P)$ est un arbre de poids minimum pour $G' = (E, \vec{\Gamma}, -P)$.

Remarque : On peut se demander combien d'arbres différents on peut construire à partir de n sommets. Cayley *a apporté la réponse en 1889 : on peut construire n^{n-2} arbres couvrant n sommets.*

Remarque : Une méthode naïve pour trouver un arbre de poids maximum pour un graphe G consiste à construire tous les graphes partiels de G qui sont des arbres et à évaluer leur poids. On voit d'après la remarque précédente que cette méthode a une complexité exponentielle.

2.3.2 Propriétés des arbres de poids maximum

Soit $(E, \vec{\Gamma})$ un graphe connexe et sans boucle. Soit P l'application de pondération associée à ce graphe. Soit $\vec{A} \subset \vec{\Gamma}$ tel que $(E, \vec{A})$ soit un arbre.

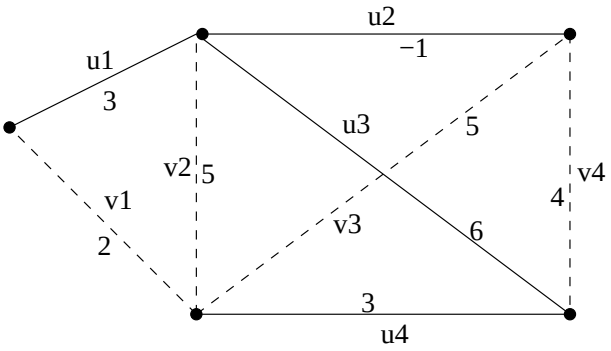
Soit $u = (x, y)$ un arc appartenant à $\vec{\Gamma}$ mais non à $\vec{A}$, on nomme C_u l'ensemble des sommets constituant la chaîne dans $\vec{A}$ ayant pour extrémités x et y.

Soit v un arc appartenant à $\vec{A}$, on nomme Ω_v l'ensemble des arcs de $\vec{\Gamma} \setminus \vec{A}$ ayant leurs extrémités dans les deux composantes connexes de $(E, \vec{A} \setminus \{v\})$.

Proposition : Les deux propriétés suivantes sont équivalentes et caractérisent un arbre de poids maximum :

1. $\forall u \in \vec{\Gamma} \setminus \vec{A}$, $P(u) \leq \min_{\omega \in C_u} P(\omega)$.
2. $\forall v \in \vec{A}$, $P(v) \geq \max_{\omega \in \Omega_v} P(\omega)$.

Exemple 22 : On se propose d'étudier le graphe suivant :



On veut vérifier si l'arbre representé par les arcs en pointillés est bien un arbre de poids maximum.

$C_{u1} = \{v_1, v_2\}$ $\quad \Omega_{v1} = \{u_1\}$
$C_{u2} = \{v_2, v_3\}$ $\quad \Omega_{v2} = \{u_1, u_2, u_3\}$
$C_{u3} = \{v_2, v_3, v_4\}$ $\quad \Omega_{v3} = \{u_2, u_3, u_4\}$
$C_{u4} = \{v_3, v_4\}$ $\quad \Omega_{v4} = \{u_3, u_4\}$

$P(u_1) = 3 > \min_{\omega \in C_{u_1}} P(\omega) = \min\{5, 2\} = 2$, la propriété 1 n'est pas vérifiée.
$P(v_2) = 5 < \max_{\omega \in \Omega_{v_2}} P(\omega) = \max\{3, -1, 6\} = 6$, la propriété 2 n'est pas vérifiée.
L'arbre representé n'est donc pas un arbre de poids maximum.

Remarque : L'idée consiste à remplacer l'arête étudiée par celle contredisant la propriété 1 ou 2. Si tel était le cas nous obtiendrions un arbre de poids strictement supérieur.

Démonstration : $APMAX \Rightarrow 1$
Supposons que $\vec{A}$ est un APMAX sur $(E, \vec{\Gamma}, P)$ et il existe $u \in \vec{\Gamma} \setminus \vec{A}$ tel que $P(u) > \min_{w \in C_u}(P(w))$. Soit $v \in C_u$ tel que $p(v) = \min_{w \in C_u}(P(w))$. Alors $\vec{A'} = (\vec{(A)} \cup \{u\}) \setminus \{v\}$ est un arbre sur E (sans cycles (facile à justifier) et $n-1$ arcs) et de plus $P(\vec{A'}) > P(\vec{A})$: contradiction.

Exercice 20
Compléter la démonstration.

Nous allons étudier des algorithmes qui permettent d'extraire les arbres de poids maximum d'un graphe. La construction d'arbres couvrants de poids minimum relève des mêmes algorithmes, il suffit juste d'utiliser la relation d'ordre inverse.

2.3.3 Algorithme de Kruskal_1

Soit $G = (E, \vec{\Gamma})$ un graphe connexe et sans boucle, et P une application de pondération de $\vec{\Gamma}$ sur $\mathbb{R}$. On note $|E| = n$ et $\vec{\Gamma} = \{u_1, u_2, \cdots u_m\}$ avec $\forall i, P(u_i) \geq P(u_{i+1})$ (les sommets sont triés par ordre décroissant).

Proposition : Soit la famille $\vec{\Gamma}_i$ définie par :

$$\begin{array}{rcl} \vec{\Gamma}_0 & = & \emptyset \\ \forall i > 0,\ \vec{\Gamma}_i & = & \left\{ \begin{array}{ll} \vec{\Gamma}_{i-1} \cup \{u_i\} & \text{si } (E, \vec{\Gamma}_{i-1} \cup \{u_i\}) \text{ est sans cycles} \\ \vec{\Gamma}_{i-1} & \text{sinon} \end{array} \right. \end{array}$$

Alors $\vec{\Gamma}_m$ est un arbre de poids maximum.

Démonstration : Chaque $\vec{\Gamma}_i$ se construit en rajoutant un arc u_i qui ne fait pas apparaître de cycle dans $(E, \vec{\Gamma}_i)$. Par conséquent, le graphe partiel résultant $(E, \vec{\Gamma_m})$ ne contient pas de cycle.
Montrons que le graphe partiel $(E, \vec{\Gamma_m})$ est connexe. Par l'absurde, supposons qu'il y ait $k = 2$ composantes connexes distinctes (le cas $k > 2$ se déduit aisément). Comme $(E, \vec{\Gamma})$ est connexe, il existe une arête u de $\vec{\Gamma} \setminus \vec{\Gamma_m}$ tel que $(E, \vec{\Gamma_m} \cup \{u\})$ soit connexe. Comme par définition cette arête est un isthme dans $(E, \vec{\Gamma_m} \cup \{u\})$, elle n'appartient à aucun cycle de $(E, \vec{\Gamma_m})$ et a fortiori de $(E, \vec{\Gamma_i})$, $\forall i \leq m$; et aurait dû être insérée par l'algorithme, d'où une contradiction. Le graphe partiel obtenu est connexe et sans cycle, il s'agit donc d'un arbre.
Il reste à démontrer que cet arbre est de poids maximum.

Exercice 21
Compléter la démonstration.

Indication : On pourra raisonner par l'absurde, et utiliser la propriété 2 de la section 2.3.2.

Remarque : Il est inutile de calculer tous les $\vec{\Gamma}_i$, il suffit de s'arrêter dès lors que $|\vec{\Gamma}_i| = |E| - 1$.

Mise en œuvre

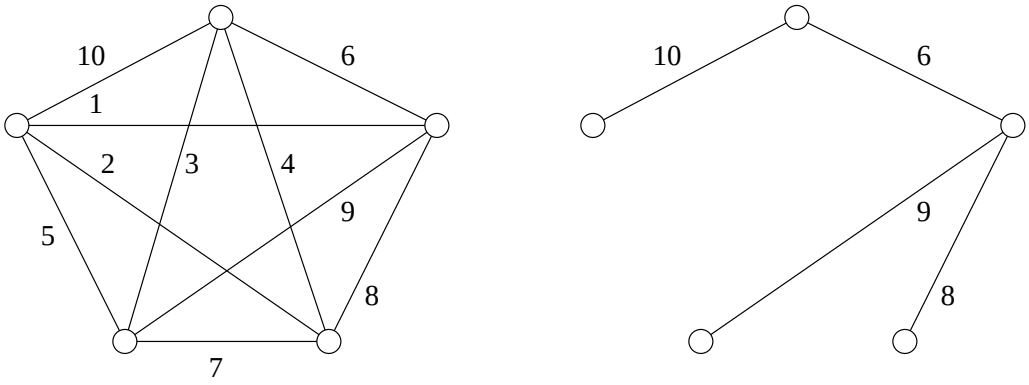
La mise en œuvre de l'algorithme de KRUSKAL_1 se déroule en deux phases :
- Trier les arêtes par ordre des poids décroissants,
- Tant que l'on n'a pas retenu $n-1$ arêtes, procéder aux opérations suivantes : considérer dans l'ordre du tri la première arête non examinée. Si elle forme un cycle avec les arêtes précedemment choisies, la rejeter ; sinon la garder.

Complexité

Il faut d'abord constituer une liste triée par ordre décroissant des arcs du graphe. Cette opération s'effectue —par des algorithmes de tri classiques [Cormen]— en $O(m.log(m))$. Le test du caractère sans cycle est en $O(n+m)$ (voir l'exercice 7) et doit être effectué au pire pour les m arêtes. La complexité de l'algorithme de KRUSKAL_1 (sous cette forme) est en :

$$O(m.(n+m) + m.\log(m)) = O(m.(n+m))$$

Exemple 23 :



Un graphe pondéré G et l'unique arbre de poids maximum sur G.

Exercice 22
Exécutez "à la main" l'algorithme de KRUSKAL_1 sur le graphe de l'exemple 23, ou sur un graphe de votre choix.

Remarque : Le test pour savoir si un graphe est sans cycles peut être remplacé par une recherche de composantes connexes. Pour chaque arête que l'on est en train d'examiner, on regarde dans le graphe en cours de construction si les deux extrémités de l'arête sont dans la même composante connexe ou non. En utilisant une structure de données adaptée (ensembles disjoints, voir [Cormen]) il est possible de réduire le coût de ce test à une quasi-constante. La complexité de KRUSKAL_*1 est alors dominée par celle du tri.*

2.3.4 Algorithme de Kruskal_2

Cet algorithme procède de façon similaire, mais au lieu de construire un nouveau graphe en ajoutant en priorité les arcs les plus lourds, il procède en enlevant du graphe d'origine les arcs les plus légers, à condition de préserver la connexité.

Dans cette variante, les arcs doivent donc être triés par ordre croissant. La complexité de l'algorithme de KRUSKAL_2 est la même que celle de KRUSKAL_1.

On note $|E| = n$ et $\vec{\Gamma} = \{u_1, u_2, \cdots u_m\}$ avec $\forall i, P(u_i) \leq P(u_{i+1})$ (les sommets sont triés par ordre croissant).

Proposition : Soit la famille $\vec{\Gamma}_i$ définie par :

$$\begin{array}{rcl} \vec{\Gamma}_0 & = & \vec{\Gamma} \\ \forall i > 0, \ \vec{\Gamma}_i & = & \begin{cases} \vec{\Gamma}_{i-1} \setminus \{u_i\} & \text{si } (E, \vec{\Gamma}_{i-1} \setminus \{u_i\}) \text{ est connexe} \\ \vec{\Gamma}_{i-1} & \text{sinon} \end{cases} \end{array}$$

Alors $\vec{\Gamma}_m$ est un arbre de poids maximum.

Exercice 23
Exécutez "à la main" l'algorithme de KRUSKAL_2 sur le graphe de l'exemple 23, ou sur un graphe de votre choix.

Exercice 24
Démontrez cette proposition.

2.3.5 Algorithme de Prim

Définition : Soit (E, Γ) un graphe, soit $X \subset E$, on considère :
- l'ensemble des **arcs sortant de** X :

$$\overrightarrow{\delta}(X) = \{ \ u \in \vec{\Gamma} \mid \exists x \in X, \exists y \in E \setminus X, u = (x, y)\}$$

- l'ensemble des **arcs entrant dans** X :

$$\overleftarrow{\delta}(X) = \{u \in \vec{\Gamma} \mid \exists x \in X, \exists y \in E \setminus X, u = (y, x)\}$$

- l'ensemble des arcs **frontière de** X :

$$\delta(X) = \overleftarrow{\delta}(X) \cup \overrightarrow{\delta}(X)$$

Soit (E, Γ) un graphe connexe sans boucle, $n = |E|$ et $x \in E$. On considère les familles $(E_i)_{1 \leq i \leq n}$ et $(\vec{\Gamma}_i)_{1 \leq i \leq n}$ définies par :
- $E_1 = \{x\}$, $\Gamma_1 = \emptyset$
- $\vec{\Gamma}_i = \vec{\Gamma}_i \cup \{u\}$ avec $u \in \delta(E_{i-1})$ tel que $p(u) = \max\{p(v) \mid v \in \delta(E_{i-1})\}$
- soit y l'extrémité de u non dans E_{i-1}, $E_i = E_{i-1} \cup \{y\}$

Alors : $(E, \vec{\Gamma}_n)$ est un APMAX

Exercice 25
Écrivez l'algorithme de PRIM en utilisant le pseudo-langage habituel. Exécutez "à la main" l'algorithme de PRIM sur le graphe de l'exemple 23, ou sur un graphe de votre choix. Déterminez sa complexité.

Chapitre 3

Plus courts chemins

On considère dans ce chapitre des graphes orientés.

3.1 Définition

3.1.1 Réseau

Nous allons travailler sur un graphe sans boucle $G = (E, \vec{\Gamma})$ et une application l de $\vec{\Gamma}$ dans $\mathbb{R}$. L'application l associe à chaque arc u un réel $l(u)$ appelé **longueur** de l'arc u. Le triplet $R = (E, \vec{\Gamma}, l)$ est appelé un **réseau**.

Par simplicité, si $u = (x, y)$, on notera $l(x, y)$ la longueur de u (plutôt que $l((x, y))$, correct mais lourd).

Soit $\sigma = (x_0, x_1, \cdots x_n)$ un chemin dans G. On définit la **longueur de σ (dans R)** comme la somme des longueurs des arcs de σ, autrement dit :

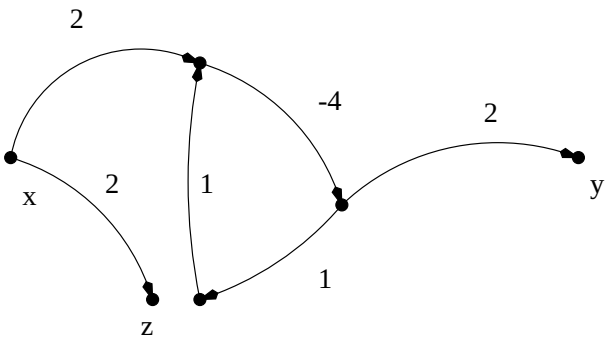
$$l(\sigma) = \sum_{0 \leq i \leq n-1} l(x_i, x_{i+1})$$

3.1.2 Plus court chemin

Soient x et y deux sommets de E. Un chemin σ de x à y dans R est un **plus court chemin de x à y** si pour tout chemin σ' de x à y, on a $l(\sigma) \leq l(\sigma')$.

Dans la recherche d'un plus court chemin de x à y, trois cas peuvent se présenter :
- Il n'existe aucun chemin de x à y (par exemple, si x et y appartiennent à deux composantes connexes différentes de G).
- Il existe des chemins de x à y mais pas de plus court chemin.
- Il existe un plus court chemin de x à y.

Exemple 24 :



Il existe des chemins (combien ?) de x à y, mais il n'existe pas de plus court chemin de x à y. C'est-à-dire que pour tout chemin σ de x à y, on peut trouver un chemin σ' tel que $l(\sigma') < l(\sigma)$. Il existe un plus court chemin de x à z mais pas de chemin de z à x.

Remarque : Un plus court chemin dans $R = (E, \vec{\Gamma}, l)$ est un plus long chemin dans $R' = (E, \vec{\Gamma}, -l)$, et réciproquement.

Circuit absorbant

Un circuit de longueur négative est appelé **circuit absorbant**. Dans l'exemple précédent, il existe un circuit absorbant.

*Remarque : S*i une composante fortement connexe présente un circuit absorbant, alors il n'existe pas de plus court chemin entre deux sommets quelconques de cette composante.

Proposition : Soit $R = (E, \vec{\Gamma}, l)$ un réseau. Soit $s \in E$, une condition nécessaire et suffisante pour qu'il existe un plus court chemin entre s et tout sommet de $E \setminus \{s\}$ est :

1. s est une racine de $(E, \vec{\Gamma})$, et
2. il n'y a pas de circuit absorbant dans R.

3.2 Problématique du plus court chemin

Proposition : Soit R un réseau, soit $c = (x_0, ..., x_k)$ un plus court chemin de x_0 à x_k et soit $c' = (x_i, ..., x_j)$, $0 \leq i < j \leq k$, un sous-chemin extrait de c, alors c' est un plus court chemin de x_i à x_j.

Remarque : Il s'agit ici de la propriété très importante d'optimalité des sous-chemins.

Exercice 26
Démontrez cette proposition :
1) par l'absurde,
2) par une preuve directe.

3.2.1 Graphe des plus courts chemins

Soit $R = (E, \vec{\Gamma}, l)$ un réseau sans circuit absorbant, soit $i \in E$. On définit l'application $\pi_i : E \to \mathbb{R} \cup \{\infty\}$ par :

$$\pi_i(x) = \begin{cases} \text{la longueur d'un plus court chemin de } i \text{ à } x \text{ s'il en existe} \\ \infty \text{ sinon} \end{cases}$$

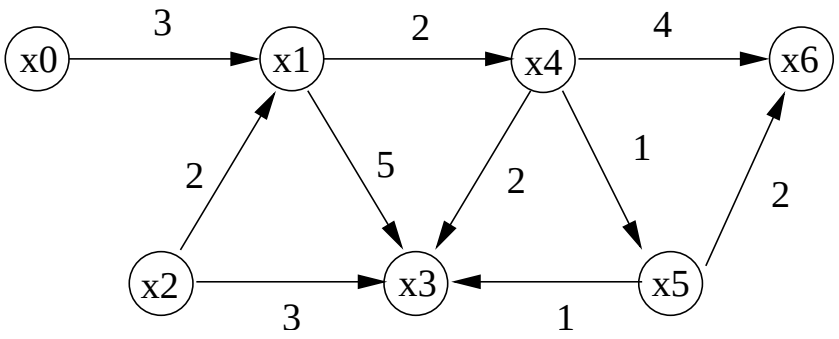
Soit $(E, \vec{\Gamma}')$ le graphe défini par :

$$\vec{\Gamma}' = \left\{ (x, y) \in \vec{\Gamma} \mid l(x, y) = \pi_i(y) - \pi_i(x) \right\}$$

Le graphe $(E, \vec{\Gamma}')$ est appelé **graphe des plus courts chemins** relativement au sommet i. L'ensemble $\vec{\Gamma}'$ est constitué des arcs faisant partie d'un plus court chemin de i à un sommet de E.

Proposition : Un plus court chemin de i à x dans R est un chemin de i à x dans $(E, \vec{\Gamma}')$.

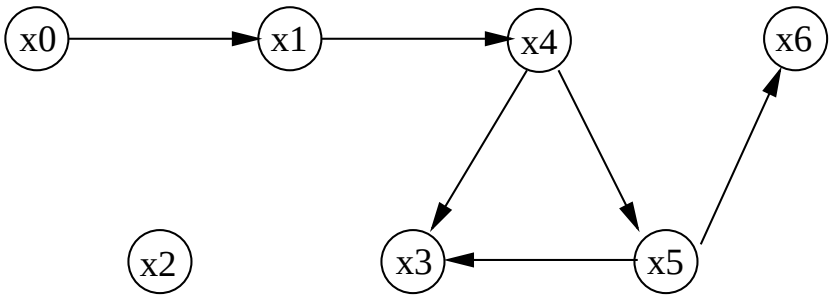
Exemple 25 :



On peut construire "à la main" l'application π_{x_0} :

x	$\pi_{x_0}(x)$
x_0	0
x_1	3
x_2	∞
x_3	7
x_4	5
x_5	6
x_6	8

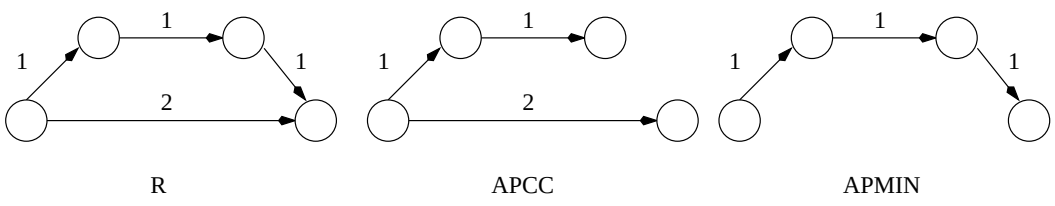
Le graphe des plus courts chemins est alors :



Arborescence des plus courts chemins

Soit $R = (E, \vec{\Gamma}, l)$ un réseau sans circuit absorbant. Soit $i \in E$ et $(E, \vec{\Gamma}')$ le graphe des plus courts chemins défini précédemment. On dit que $(E', \vec{A})$ est une **arborescence des plus courts chemins** pour R si :
– $E' = \{x \in E, \pi_i(x) \neq \infty\}$, et
– $(E', \vec{A})$ est une arborescence de racine i, et
– $\vec{A} \subset \vec{\Gamma}'$.
Remarque : En général une arborescence des plus courts chemins n'est pas un arbre de poids minimum.

Exemple 26 :



R APCC APMIN

3.3 Réseaux à longueurs quelconques (Bellman)

Nous allons étudier l'algorithme de Bellman pour calculer les longueurs de plus courts chemins dans un réseau à longueurs quelconques. Cet algorithme calcule la longueur des plus courts chemins d'un sommet inital i à tout autre sommet x du graphe.

But : soit $R = (E, \vec{\Gamma}, l)$, $i \in E$, calculer π_i.
L'idée est qu'un chemin de i à x dans R passe nécessairement par un sommet $y^* \in \Gamma^{-1}(x)$. La longueur d'un plus court chemin de i à x passant par y^* est :

$$\pi_i(x) = \pi_i(y^*) + l(y^*, x)$$

Et donc, on a la propriété suivante, dite propriété de Bellman.

Proposition :

$$\pi_i(x) = \min_{y \in \Gamma^{-1}(x)} \{\pi_i(y) + l(y, x)\}$$

3.3.1 Algorithme en pseudo language

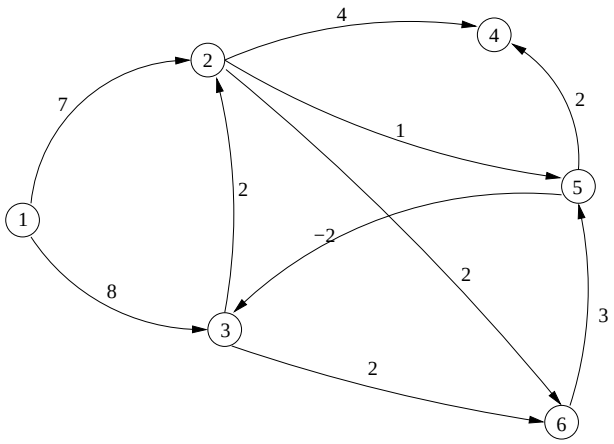
Pour alléger les notations, on notera dans la suite $\pi = \pi_i$, sauf en cas de nécessité.

Algo BELLMAN (**Données :** $E, \Gamma^{-1}, l, i \in E$; **Résultats :** $\pi, CIRCABS$)

1: $CIRCABS = FAUX$; $\pi^0(i) = 0$; $\pi^1(i) = 0$; $k = 1$
2: **for all** $x \in E \setminus \{i\}$ **do**
3: $\quad \pi^0(x) = \infty$
4: $\quad \pi^1(x) = \infty$
5: **end for**
6: **for all** $x \in E$ tel que $i \in \Gamma^{-1}(x)$ **do**
7: $\quad \pi^1(x) = l(i, x)$
8: **end for**
9: **while** $k < n + 1$ et $\exists x \in E$ tel que $\pi^k(x) \neq \pi^{k-1}(x)$ **do**
10: $\quad k = k + 1$
11: $\quad$ **for all** $x \in E$ **do**
12: $\quad\quad \pi^k(x) = \min\left[\pi^{k-1}(x), \pi^{k-1}(y) + l(y, x); y \in \Gamma^{-1}(x)\right]$
13: $\quad$ **end for**
14: **end while**
15: **if** $k = n + 1$ **then**
16: $\quad CIRCABS = VRAI$
17: **end if**
18: $\pi = \pi^k$

Cet algorithme renvoie également dans la variable $CIRCABS$ un booléen indiquant la présence d'un circuit absorbant.

Exercice 27
Exécutez "à la main" l'algorithme de Bellman sur le graphe suivant (avec $i = 1$), ou sur un graphe de votre choix.



Complexité

- L'initialisation de la ligne 1 est en temps constant $O(1)$
- La boucle d'initialisation des lignes 2 à 5 est en $O(n)$
- La boucle d'initialisation des lignes 6 à 8 est en $O(n + m)$
- Le corps de boucle principale est executé $n + 1$ fois au maximum. La complexité de la ligne 9 est en $O(n^2)$ et le corps de la boucle 10 à 14 est en $O(n(n + m))$.

Au total, la complexité de l'algorithme de Bellman est $O(n(n + m))$.

Remarque : La complexité de l'algorithme de Bellman *dans le cas d'un graphe dense ($m \sim n^2$) est donc $O(n^3)$. Il existe des variantes plus efficaces, toutefois notons que celle-ci peut facilement être parallélisée.*

3.3.2 Justification

L'idée est que la longueur d'un plus court chemin du sommet i à n'importe quel sommet x est imposée par :

$$\pi(x) = \min_{y \in \Gamma^{-1}(x)} \{\pi(y) + l(y, x)\}$$

Cette formule locale est appliquée jusqu'à convergence (ou jusqu'à n itérations, dans ce cas il y a présence d'un circuit absorbant).

On appelle $k-$chemin un chemin possédant au plus k arcs. Dans cet algorithme, $\pi^k(x)$ représente la longueur d'un plus court $k-$chemin de i à x (s'il en existe, ∞ sinon).
Pour prouver cette proposition, nous allons procéder par récurrence.

La proposition est trivialement vérifiée pour $k = 0$ et $k = 1$. Supposons qu'elle est vérifiée jusqu'à l'étape $k - 1$ et plaçons-nous à l'étape k. Soit $\mathcal{C}_k$ l'ensemble des sommets y tels

qu'il existe un $k-$chemin de i à y.

- Si $x \notin \mathcal{C}_k$, $\forall y \in \Gamma^{-1}(x)$, $y \notin \mathcal{C}_{k-1}$. Donc d'après l'hypothèse de récurrence, $\forall y \in \Gamma^{-1}(x)\pi^{k-1}(y) = \infty$, d'où $\pi^k(x) = \infty$.
- Si $x \in \mathcal{C}_k$ et $x \neq i$ alors $\exists y \in \Gamma^{-1}(x)$ tel que $y \in \mathcal{C}_{k-1}$. Donc $\pi^{k-1}(y) < \infty$.
  Tout $k-$chemin de i à x passe par un $y \in \Gamma^{-1}(x)$. Un plus court $k-$chemin de i à x est composé d'un plus court $(k-1)-$chemin de i à un sommet $y \in \Gamma^{-1}(x)$ et de l'arc (y, x). Il a pour longueur $\pi^{k-1}(y) + l(y, x)$. La longueur d'un plus court $k-$chemin de i à x est donc :
  $$\min_{y \in \Gamma^{-1}(x)} \pi^{k-1}(y) + l(y, x)$$

De plus, la longueur d'un plus court k-chemin de i à x ne peut être supérieure à celle d'un plus court $(k-1)$-chemin de i à x (puisqu'un $(k-1)$-chemin est aussi un k-chemin), d'où l'instruction de la ligne 12.

On peut facilement voir que s'il existe un plus court chemin σ de i à x alors il existe un plus court chemin de i à x qui est un chemin élémentaire (en effet si σ comporte un circuit de longueur positive ou nulle, alors en ôtant ce circuit de σ on obtient un chemin de longueur inférieure ou égale à $l(\sigma)$). Or un tel chemin possède au plus $n-1$ arcs (avec $n = |E|$). Donc s'il n'existe pas de circuit absorbant, on a $\forall k \geq n,\ \pi^k = \pi^{n-1}$.

Par conséquent, si $\pi^n \neq \pi^{n-1}$, alors le graphe présente un circuit absorbant.

Remarque : Pour implémenter l'algorithme de Bellman*, il suffit de deux tableaux :* π*_courant et* π*_précédent.*

Exercice 28
Proposez un algorithme qui prend en entrée un réseau R et un couple de sommets (i, j), et qui retourne en sortie un chemin (liste ordonnée de sommets) de longueur minimale de i à j s'il en existe.

Indication : On commencera par calculer les longueurs $\pi_i(x)$ pour tous les sommets x par l'algorithme de Bellman.

3.4 Réseaux à longueurs positives (Dijkstra)

L'algorithme de Dijkstra permet de trouver les longueurs des plus courts chemins d'un sommet donné i à tous les autres sommets dans un réseau (E, Γ, l) à longueurs positives. Dans un tel réseau on a l'équivalence suivante :

$$\text{il existe un chemin de } x \text{ à } y \Leftrightarrow \text{ il existe un plus court chemin de } x \text{ à } y$$

pour tout couple de sommets (x, y). Cette équivalence est due au fait qu'il n'existe pas de circuit absorbant.

Le principe de l'algorithme est le suivant :
Soit $R = (E, \vec{\Gamma}, l)$, avec pour tout $u \in \vec{\Gamma}$, $l(u) \geq 0$. Soit $i \in E$ sommet initial. On désigne

par S l'ensemble des sommets x de E pour lesquels la longueur d'un plus court chemin de i à x a été calculée. Initalement $S = \{i\}$, $\pi(i) = 0$.
On appelle $S-$chemin un chemin partant de i et ayant tous ses sommets dans S sauf le dernier.
Soit $Y = E \setminus S$. Pour tout sommet y dans Y, on désigne par $\pi'(y)$ la longueur d'un plus court $S-$chemin de i à y. On sélectionne y^* dans Y tel que $\pi'(y^*) = \min\{\pi'(y), y \in Y\}$.
On est alors assuré que $\pi'(y^*) = \pi(y^*)$.

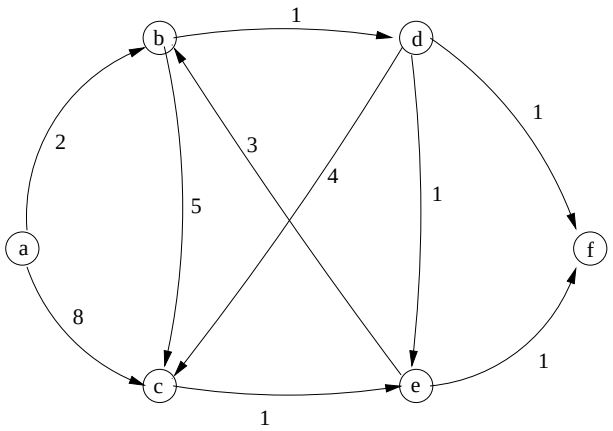
Exercice 29
Expliquez pourquoi cette dernière affirmation est vraie.

3.4.1 Algorithme en pseudo language

Algo DIJKSTRA (**Données :** $E, \Gamma, l, i \in E$; **Résultats :** π, S)

1: $S = \{i\}$, $\pi(i) = 0$, $k = 1$, $x_1 = i$
2: **for all** $x \in E \setminus \{i\}$ **do**
3: $\quad \pi(x) = \infty$
4: **end for**
5: **while** $k < n$ et $\pi(x_k) < \infty$ **do**
6: $\quad$ **for all** $y \in \Gamma(x_k)$ tel que $y \notin S$ **do**
7: $\quad\quad \pi(y) = \min[\pi(y), \pi(x_k) + l(x_k, y)]$
8: $\quad$ **end for**
9: $\quad$ Extraire un sommet $x \notin S$ tel que $\pi(x) = \min\{\pi(y), y \notin S\}$
10: $\quad k = k + 1$, $x_k = x$, $S = S \cup \{x_k\}$.
11: **end while**

Exercice 30
Exécutez "à la main" l'algorithme de Dijkstra sur le graphe suivant (avec $i = a$), ou sur un graphe de votre choix.



Complexité

Nous prendrons S sous la forme d'un tableau de booléens.
– Les boucles d'initialisation des lignes 1 et 2 à 4 sont en $O(n)$
– La boucle principale (ligne 5) est executée n fois.
– Les lignes 6 et 7 sont en $O(n+m)$.
– La ligne 9 est en $O(n^2)$ (si on l'implémente naïvement).
La complexité de l'algorithme de DIJSKTRA (sous cette forme) est en $O(n^2)$. Il est facile, en utilisant une structure de données adaptée (par exemple un arbre de recherche équilibré, voir [Cormen]) pour la ligne 9, de ramener cette complexité à $O(n\log(n)+m)$. Il existe des implémentations encore plus efficaces.

3.4.2 Réseaux à longueurs uniformes

Nous nous plaçons dans le cas d'un réseau $R=(E,\Gamma,l)$ tel que pour tout arc u dans $\vec{\Gamma}$, la longueur de l'arc soit égale à une constante strictement positive. On prendra $l(u)=c=1$.

Comme pour Bellman et Dijkstra, on fixe un sommet initial i. Pour trouver la longueur d'un plus court chemin de i à tout autre sommet x, il existe un algorithme en temps linéaire $O(n+m)$. Cet algorithme s'appuie sur une stratégie d'exploration du graphe dite *d'exploration-largeur*. Il consiste à parcourir le graphe "par couches successives" à partir de i et de proche en proche en suivant les arcs non déjà utilisés. La longueur du plus court chemin à x est alors le nombre de *couches* parcourues pour aller de i à x (multiplié par la constante c).

Exercice 31
Écrivez cet algorithme en pseudo-langage et évaluez sa complexité.

3.5 Graphes et réseaux sans circuits

3.5.1 Définition

Un **graphe sans circuit** (ou **GSC**) est comme son nom l'indique un graphe ne contenant aucun circuit. Si $G=(E,\Gamma)$ est un graphe sans circuit alors son symétrique (E,Γ^{-1}) l'est aussi.

Dire qu'un graphe $G=(E,\Gamma)$ est un graphe sans circuit équivaut à dire que toutes les composantes fortement connexes de G sont réduites à un seul sommet. C'est une des caractérisations des GSC.

3.5.2 Sources et puits

Soit $G = (E, \Gamma)$ un graphe sans circuit et $x \in E$.
On dit que x est une **source** de G si $d^-(x) = |\Gamma^{-1}(x)| = 0$.
On dit que x est un **puits** de G si $d^+(x) = |\Gamma(x)| = 0$.

Autrement dit, une source est un sommet sans prédécesseur, et un puits est un sommet sans successeur.

Proposition : Tout GSC fini possède au moins une source et un puits.

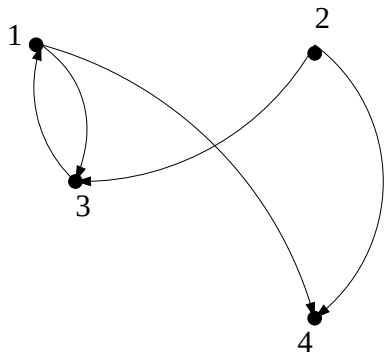
Exercice 32
Démontrez cette propriété, et donnez-en un contre-exemple dans le cas infini.

3.5.3 Rang et hauteur

Soit $G = (E, \Gamma)$ un graphe quelconque, et x un sommet de E. Le **rang** du sommet x, noté $r(x)$ est la longueur maximale d'un chemin élémentaire se terminant par x.

Soit $G = (E, \Gamma)$ un graphe quelconque, et x un sommet de E. La **hauteur** du sommet x, noté $h(x)$ est la longueur maximale d'un chemin élémentaire débutant en x.

Exemple 27 : Soit le graphe G :



Le rang du sommet 2 est 0, le rang du sommet 4 est 3, la hauteur du sommet 4 est 0, la hauteur du sommet 2 est 3.

Dire que $r(x) = 0$ équivaut à dire que x est une source. De même, si $h(x) = 0$ alors x est un puits, et réciproquement.

3.5.4 Partition en niveaux

Rappel : Soit E un ensemble quelconque, et $E_0, \ldots, E_p$ une famille finie de sous-ensembles de E. On dit que $\{E_i\}_{i=0}^p$ est une **partition** de E si $\bigcup_{i=0}^p E_i = E$ et si les E_i sont tous disjoints deux à deux.

Soit $G = (E, \Gamma)$ un graphe quelconque. Le graphe G admet une **partition en niveaux** s'il existe une partition de E telle que :

$$E = \bigcup_{i=0}^{p} E_i$$

où E_0 est l'ensemble des sources de G, et $\forall i > 0$, E_i est l'ensemble des sources de $(E \setminus [E_0 \cup \ldots \cup E_{i-1}], \vec{\Gamma}_i)$, $\vec{\Gamma}_i$ étant la restriction de G à $E \setminus [E_0 \cup \ldots \cup E_{i-1}]$ (c'est-à-dire l'ensemble des arcs de $\vec{\Gamma}$ ayant leurs 2 extremités dans $E \setminus [E_0 \cup \ldots \cup E_{i-1}]$).

Proposition : Un graphe admet une partition en niveaux **si et seulement si** c'est un graphe sans circuit.

Proposition : Soit $G = (E, \Gamma)$ un graphe. Si elle existe, la partition en niveaux $\{E_i\}_{i=0}^{p}$ de G est unique et telle que pour tout sommet x appartenant à E_i, $r(x) = i$.

Exercice 33
Démontrez ces deux propriétés.

3.5.5 Algorithme circuit-niveaux

Voici un algorithme qui permet de déterminer la partition en niveaux d'un graphe $G = (E, \Gamma)$, ou de détecter la présence d'un circuit.

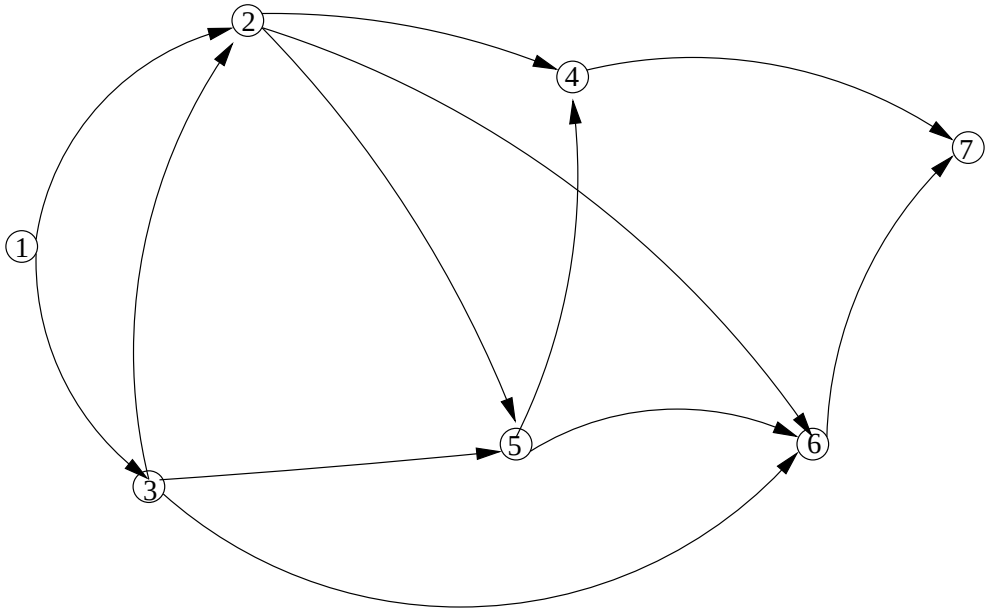
Remarque : Dans la littérature anglo-saxonne, cet algorithme est appelé tri topologique *(*topological sort*), et les GSC sont appelés DAG (*Directed Acyclic Graphs*).*

Algo CIRCUIT-NIVEAUX (**Données :** E, Γ, Γ^{-1} ; Résultat : $E_i \subset E, CIRC$)

1: $E_0 = \emptyset, N = 0, i = 0$
2: **for all** $x \in E$ **do**
3: $\quad d^-(x) = |\Gamma^{-1}(x)|$
4: **end for**
5: **for all** $x \in E$ tel que $d^-(x) = 0$ **do**
6: $\quad E_0 = E_0 \cup \{x\}$
7: $\quad N = N + 1$
8: **end for**
9: **while** $N < n$ et $E_i \neq \emptyset$ **do**
10: $\quad E_{i+1} = \emptyset$
11: $\quad$ **for all** $x \in E_i$ **do**
12: $\quad\quad$ **for all** $y \in \Gamma(x)$ **do**
13: $\quad\quad\quad d^-(y) = d^-(y) - 1$
14: $\quad\quad\quad$ **if** $d^-(y) = 0$ **then**
15: $\quad\quad\quad\quad E_{i+1} = E_{i+1} \cup \{y\}$
16: $\quad\quad\quad\quad N = N + 1$
17: $\quad\quad\quad$ **end if**
18: $\quad\quad$ **end for**

19: **end for**
20: $i = i + 1$
21: **end while**
22: **if** $N < n$ **then**
23: $CIRC = VRAI$
24: **else**
25: $CIRC = FAUX$
26: **end if**

Exercice 34
Exécutez "à la main" cet algorithme sur le graphe suivant (avec $i = 1$), ou sur un graphe sans circuit de votre choix.



Exercice 35
Exécutez "à la main" cet algorithme sur un graphe de votre choix comportant au moins un circuit.

Exercice 36
Évaluez la complexité de cet algorithme.

3.5.6 Plus courts chemins dans les réseaux sans circuits

Dans le cas d'un réseaux sans circuit, le calcul des longueurs des plus courts chemins depuis un sommet i peut se faire en temps linéaire (complexité $O(n + m)$) grâce à l'algorithme suivant, dérivé de Bellman. Le calcul des rangs doit être effectué préalablement (algorithme linéaire CircuitsNiveaux), et un tri par dénombrement (voir [Cormen]) des sommets selon les rangs croissants peut être réalisé en temps linéaire également.

Algorithme 6 : BellmanSansCircuit

Données : $E = \{x_1 = i, x_2, \ldots, x_n\}, \Gamma^{-1}, \ell$; avec i racine et les sommets de E triés suivant leur rang croissant

Résultat : π

```
1 π(i) ← 0 ; j ← 2 ;
2 tant que j ≤ n faire
3     x ← x_j ;
4     π(x) ← min{π(y) + ℓ(y, x); y ∈ Γ^{-1}(x)} ;
5     j ← j + 1 ;
```

Exercice 37
Démontrez que chaque valeur $\pi(x)$ calculée par cet algorithme est bien la longueur d'un plus court chemin de i à x.

Chapitre 4

Flots et réseaux de transport

Les applications visées sont par exemple : les réseaux de transport routier, ferroviaire, de marchandises, de gaz, d'eau, de pétrole, d'électricité, d'information ...

4.1 Modélisation du réseau de transport

Un réseau de transport est constitué par :
- un graphe $(E, \vec{\Gamma})$,
- un puits $p \in E$,
- une source $s \in E \setminus \{p\}$,
- une application c de $\vec{\Gamma}$ dans $\mathbb{R}^+ \cup \{\infty\}$. Pour tout u dans $\vec{\Gamma}$, $c(u)$ est appelé **capacité** de u. Typiquement $c(u)$ représente le débit maximal pouvant transiter par l'arc u.

4.1.1 Modélisation du traffic (flot)

Soit f une application $\vec{\Gamma} \to \mathbb{R}^+$. La valeur $f(u)$ représente le débit, supposé constant, sur l'arc u. Si $u = (x, y)$ on note $f(u) = f(x, y)$.

On définit également les applications f^+ et f^- sur E définies par :

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

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

4.1.2 Équilibre global

La propriété d'équilibre global énonce que la somme de toutes les quantités entrantes est égale à la somme de toutes les quantités sortantes, plus précisément :

$$\sum_{x \in E} f^+(x) = \sum_{x \in E} f^-(x)$$

Remarque : Cette propriété est automatiquement vérifiée pour tout graphe et pour toute application f.

4.1.3 Équilibre local

La propriété d'équilibre local énonce que pour tout sommet x de E, la somme des quantités entrant dans x est égal à la somme des quantités sortant de x, soit :

$$\forall x \in E \quad f^+(x) = f^-(x)$$

La fonction f est un **flot** sur $(E, \vec{\Gamma})$ si elle vérifie la propriété d'équilibre local.

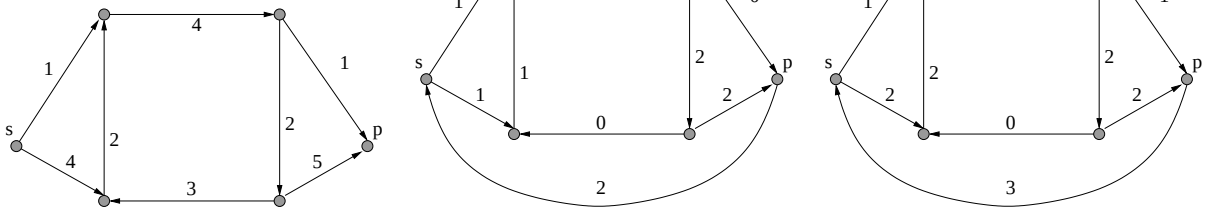
4.1.4 Arc de retour

Soit $R = (E, \vec{\Gamma}, p, s, c)$ un réseau de transport et f un flot sur $(E, \vec{\Gamma} \cup \{(p, s)\})$. L'arc (p, s) est appelé **arc de retour**. La quantité $f(p, s)$ est appelée **valeur du flot**. Par la propriété d'équilibre, on voit que la valeur du flot est égale à la quantité qui transite par le réseau de la source jusqu'au puits.

4.1.5 Flot compatible avec un réseau de transport

Le flot f est **compatible avec le réseau** R si pour chaque arc, le flot passant par cet arc est inférieur ou égal à la capacité de cet arc ; autrement dit si $\forall u \in \vec{\Gamma}, \quad f(u) \leq c(u)$.

Exemple 28 :



Réseau de transport R | Flot compatible avec R, non maximal | Flot maximal compatible avec R

Soit R un réseau de transport. Le problème du flot maximum est de trouver un flot f qui soit compatible avec R et de valeur maximale.

4.2 Algorithme de Ford et Fulkerson

Cet algorithme a pour but de trouver un flot maximal compatible avec un réseau donné R. L'algorithme de Ford et Fulkerson procède par améliorations succesives du flot f, en partant d'un flot compatible. Le flot nul par exemple est compatible et peut servir comme flot initial.

Un arc $u \in \vec{\Gamma}$ est dit **saturé** si $f(u) = c(u)$. On ne peut plus augmenter le flot passant sur cet arc. Un flot est dit **complet** si tout chemin de s à p passe au moins par un arc saturé. Pour améliorer un flot compatible, on peut chercher s'il un chemin de s à p sur lequel aucun arc n'est saturé.

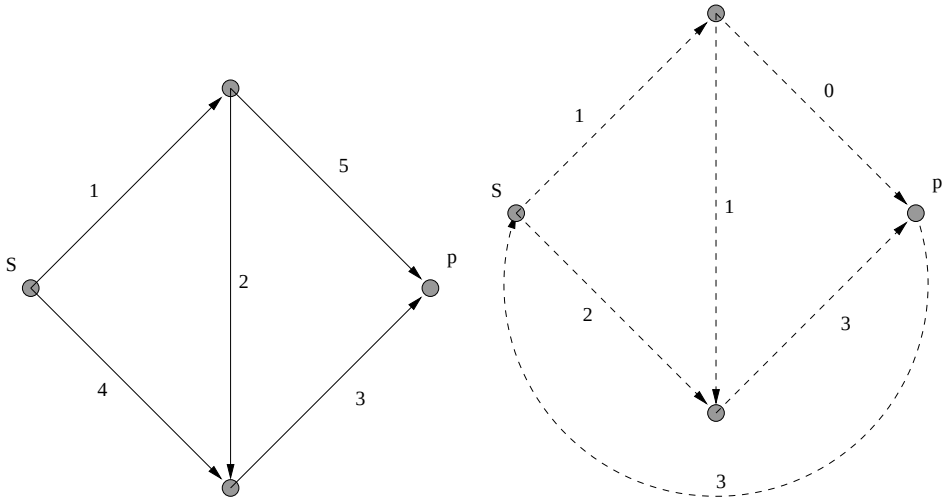
Supposons qu'il existe un chemin σ de s à p tel qu'aucun arc de σ n'est saturé. On appelle $r(\sigma)$ la **valeur résiduelle** de σ le nombre :

$$r(\sigma) = \min\{c(u) - f(u), u \text{ dans } \sigma\}$$

Le flot f sur le réseau R peut alors être amélioré de $r(\sigma)$ (on augmente f de $r(\sigma)$ sur tous les arcs de σ et sur l'arc de retour (p, s)).

Il est tentant d'affirmer qu'un flot complet est obligatoirement maximal. Avant de lire la suite, essayez de chercher par vous-même un contre-exemple à cette affirmation.

Exemple 29 :



Un réseau R (à gauche) et un flot f compatible (à droite).

Le flot de l'exemple 29 est complet (tous les chemins de s à p ont une valeur résiduelle nulle) mais n'est pas maximal. Il existe un flot de valeur 4 qui lui, est maximal.

On voit par là que le problème de la recherche d'un flot maximal n'est pas facile, en particulier il ne suffit pas de rechercher des chemins composés d'arcs non saturés pour trouver toutes les possibilités d'améliorer un flot.

4.2.1 Réseau d'écart

L'algorithme de Ford et Fulkerson utilise un réseau auxiliaire appelé réseau d'écart pour améliorer le flot courant. Le **réseau d'écart** associé à R est défini par :

$$\overline{R} = (E, \overline{\Gamma}, \overline{c})$$

Pour tout arc $u = (x, y) \in \vec{\Gamma}$, on associe au plus 2 arcs dans le réseau d'écart :
- $u^+ = (x, y)$ et tel que $\overline{c}(u^+) = c(u) - f(u)$, si u est non saturé. C'est l'**arc direct** associé à u.
- $u^- = (y, x)$ et tel que $\overline{c}(u^-) = f(u)$, si $f(u) > 0$. C'est l'**arc rétrograde** associé à u.

Soit $\overline{\sigma}$ un chemin de s à p dans le réseau d'écart $\overline{R}$. On appelle capacité résiduelle de $\overline{\sigma}$ le minimum des capacités $\overline{c}(u)$ pour tous les arcs u de $\overline{\sigma}$. Tout chemin de s à p dans le réseau d'écart fournit un moyen d'augmenter le flot.

Exercice 38
Construire le réseau d'écart associé au réseau et au flot de l'exemple 29. En déduire un flot de valeur supérieure.

4.2.2 Algorithme

Algo FORD & FULKERSON (**Données :** $R = (E, \vec{\Gamma}, p, s, c)$; **Résultat :** f_k)
0. Initialisation
Soit $k = 0$, et soit f_0 un flot compatible avec R (par exemple le flot nul).
1. Recherche d'un moyen d'améliorer le flot courant f_k
Soit f_k le flot courant. Soit $\overline{R_k} = (E, \overline{\Gamma}, \overline{c})$ le réseau d'écart associé à R et f_k. On recherche un chemin $\overline{\sigma_k}$ de s à p dans le réseau d'écart $\overline{R}$. S'il n'existe pas de chemin, alors l'algorithme s'arrête et f_k est un flot maximal.
2. Amélioration du flot courant
Soit ϵ_k la capacité résiduelle de $\overline{\sigma_k}$, autrement dit $\epsilon_k = \min\{\overline{c}(u), u \text{ dans } \overline{\sigma_k}\})$.
Pour tout arc u dans $\overline{\sigma_k}$, on applique les règles suivantes :
– si u^+ est un arc direct dans $\overline{R_k}$, alors $f_{k+1}(u) = f_k(u) + \epsilon_k$,
– si u^- est un arc rétrograde dans $\overline{R_k}$, alors $f_{k+1}(u) = f_k(u) - \epsilon_k$.
Faire $f_{k+1}(p, s) = f_k(p, s) + \epsilon_k$;
Faire $k = k + 1$;
Retourner en 1.

4.2.3 Preuve de l'algorithme de Ford et Fulkerson

Soit $G = (E, \vec{\Gamma})$ et $F \subset E$. On note $\vec{\delta}^+(F)$ l'ensemble des arcs **sortant** de F, et $\vec{\delta}^-(F)$ l'ensemble des arcs **entrant** dans F. Plus précisément,

$$\vec{\delta}^+(F) = \{(x, y) \in E \times E \mid x \in F \ , \ y \notin F\}$$

$$\vec{\delta}^-(F) = \{(x, y) \in E \times E \mid x \notin F \ , \ y \in F\}$$

Par conséquent, $\vec{\delta}^-(F) = \vec{\delta}^+(E \setminus F)$.

Soit $R = (E, \vec{\Gamma}, p, s, c)$ un réseau de transport. On appelle **coupe** sur R un ensemble d'arcs $\vec{S} \subset \vec{\Gamma}$ de la forme $S = \vec{\delta}^+(F)$ avec $F \subset E$, $s \in F$, $p \notin F$.

On appelle capacité de la coupe $\vec{S}$ la quantité $c(\vec{S}) = \sum_{u \in \vec{S}} c(u)$.

Proposition : Soit $R = (E, \vec{\Gamma}, p, s, c)$ un réseau de transport. Pour tout flot f compatible avec R et pour toute coupe $\vec{S}$ sur R, on a $f(p, s) \leq c(\vec{S})$.

Exercice 39
Démontrez cette proposition.

Indication : Utiliser la conservation du flux (propriété d'équilibre local, que l'on généralisera à une partie F de E).

Proposition : Quand l'algorithme de Ford et Fulkerson se termine, le flot courant est maximum.

Exercice 40
Démontrez cette proposition.

Indication : Utiliser la propriété précédente, en utilisant la coupe induite par la partie $F = \{x \in E \mid$ il existe un chemin de s à x dans $\overline{R}\}$.

Proposition : Si les capacités sont des multiples entiers d'un réel strictement positif et s'il existe une coupe de capacité finie, alors l'algorithme de Ford et Fulkerson se termine après un nombre fini d'itérations.

Exercice 41
Démontrez cette proposition.

La proposition suivante a une importance particulière puisqu'elle permet de montrer l'équivalence de deux problèmes : la recherche d'un flot maximum et celle d'une coupe minimum. Elle se déduit facilement de ce qui précède.

Proposition : (théorème de la coupe minimum)
Soit $R = (E, \vec{\Gamma}, p, s, c)$ un réseau de transport. La valeur maximum de $f(p, s)$ pour un flot f compatible est égale à la capacité d'une coupe de capacité minimum.

Exercice 42
Démontrez cette proposition.

Chapitre 5

Résolution de problèmes en intelligence artificielle et optimisation combinatoire

Cette partie est plus informelle que les précédentes. Elle vise à donner un avant-goût de ces applications de la théorie des graphes, mais ne prétend nullement faire le tour de la question. Pour aller plus loin, on se reportera à [Nilsson].

5.1 Exemple 1 : le problème des 8 reines

On se place dans le cas d'un échiquier standard (de taille 8×8). Le problème est de placer 8 reines sur cet échiquier de telle sorte qu'aucune reine ne peut se faire prendre par une autre (une reine peut capturer toutes les pièces se trouvant sur sa rangée, sa colonne ou ses diagonales).

La figure 5.1 page suivante présente le placement de 2 reines sur un échiquier. On voit que le nombre de cases disponibles pour le placement de 6 autres reines est très restreint.

Voici deux approches possibles pour le placement des huit reines :

1. Recherche aveugle : On place la n-ième reine sur une case choisie aléatoirement parmis les cases libres (une case est dite libre si elle n'est ni occupée, ni "menacée" par une reine). A un moment, il n'y aura plus de cases libres pour le placement des reines. S'il reste des reines à placer, il y aura alors remise en cause d'un ou de plusieurs choix faits précédemment (retour arrière).
2. Recherche heuristique : On place la n-ième reine sur une des cases restées libres, de telle manière que le nombre de cases "menacées" (susceptibles d'être prises par cette reine) soit minimal. Autrement dit, on cherche à maximiser à chaque étape le nombre de cases libres restantes.

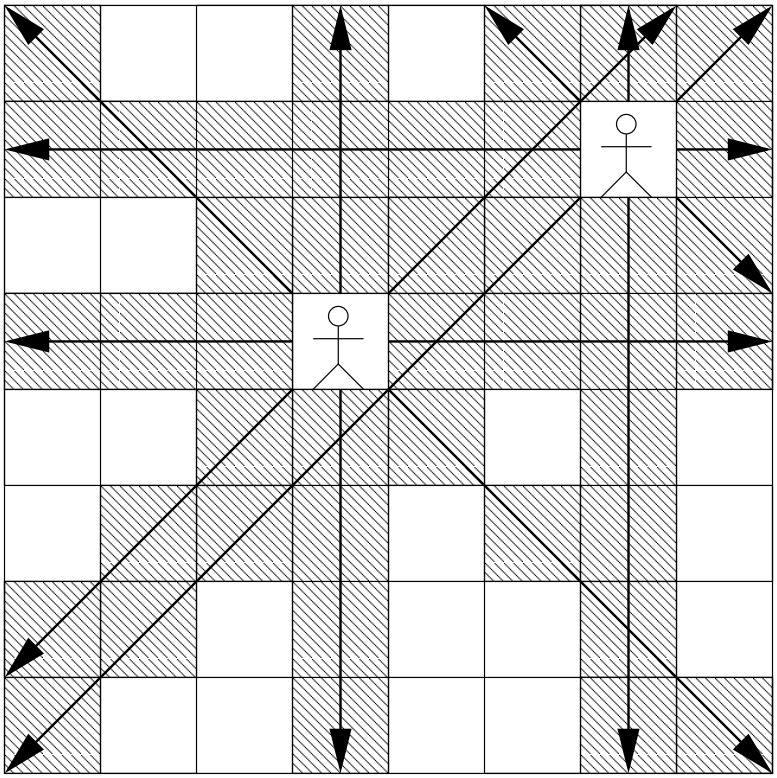


FIG. 5.1 – Exemple de placement pour le jeu des 8 reines

5.2 Graphe de résolution de problème

Les problèmes que l'on cherche à résoudre par cette approche consistent à faire évoluer un système, d'un état dit initial à l'un des états dits terminaux ou buts, au moyen d'un ensemble de règles.

Dans l'exemple précédent, le système est un échiquier portant un certain nombre de reines. L'état initial est l'échiquier vide, les états terminaux sont les configurations de 8 reines ne se menaçant pas mutuellement.

Un **graphe de résolution de problème** ou **GRP** est un graphe dont les sommets sont les états possibles du problème. Il y a un arc $u = (i, j)$ dans le graphe si une règle permet de passer de l'état i à l'état j. On associe un coût $c(u)$ à cet arc. On doit distinguer le sommet initial et les sommets terminaux.

Formellement, un graphe de résolution de problème est un quintuplet $(S, \vec{\nabla}, d, B, c)$:
– S est l'ensemble des sommets du graphe (ou états du problème),
– $\vec{\nabla}$ l'ensemble des arcs,
– $d \in S$ le sommet de départ (état inital du problème),
– $B \subset S$ l'ensemble des sommets buts,
– $c : \vec{\nabla} \longrightarrow \mathbb{R}$, où $c(i, j)$ est le coût de l'application de la règle permettant de passer de

l'état i à l'état j.
Pour un même problème, on peut en général associer plusieurs GRP.

Dans l'exemple des 8 reines, on peut prendre pour états toutes les combinaisons de $0, 1, 2, \ldots, 8$ reines telles que les reines ne se menacent pas mutuellement. Le nombre de telles configurations est de l'ordre du million.

Toutefois, on peut voir que l'on ne réduit pas la généralité du modèle en contraignant la première reine placée à se trouver sur la première rangée, la seconde sur la seconde rangée, etc, puisqu'une solution doit obligatoirement comporter une reine sur chaque rangée.

Les états sont nettement moins nombreux avec ce second modèle, leur nombre est de l'ordre de quelques milliers.

Un graphe de résolution de problème est souvent énorme, parfois même infini, ainsi le but de la recherche heuristique est d'explorer partiellement le graphe pour aboutir à une solution (c'est à dire trouver un plus court chemin de i à un sommet $s \in B$), en cherchant à limiter le plus possible la partie explorée.

5.3 Exemple 2 : jeu du taquin

Le jeu du *taquin* est un jeu de plateau dans lequel il faut passer d'une configuration de départ à une configuration d'arrivée sachant que les cases numérotées ne peuvent se déplacer que sur une case vide immédiatement adjacente.

	1	3
7	5	8
4	2	6

Configuration de départ

1	2	3
4		5
6	7	8

Configuration d'arrivée

FIG. 5.2 – Jeu de taquin

Les contraintes étant :
– Passer de la configuration de départ à la configuration d'arrivée
– Minimiser le nombre de déplacements
Un graphe de résolution de problème possible associé à ce jeu est celui tel que :
– les sommets du GRP sont les états possibles du jeu
– le sommet initial est la configuration de départ
– le sommet but (unique) est la configuration d'arrivée
– il existe un arc de la configuration x à la configuration y si un mouvement autorisé permet de passer de x à y.

Remarque : Ce GRP possède à la fois des cycles : il peut exister 2 chemins différents pour passer d'un sommet x à un sommet y ; et des circuits : les règles permettent aussi de passer d'une configuration x à y puis de revenir à x.

5.4 Stratégies de recherche

5.4.1 Stratégie sans mémoire : la recherche arrière (*backtrack*)

Cette stratégie consiste à explorer le GRP via un chemin issu de d jusqu'à atteindre
– le but (c'est gagné)
– un puits (c'est perdu)
– une profondeur limite fixée d'avance
Si le but n'est pas atteint, on revient au dernier choix effectué chronologiquement (en *oubliant* la partie venant d'être explorée). Pour eviter les circuits on mémorise les états du graphe se trouvant sur le chemin courant. Tout nouvel état atteint est comparé à la liste des états du chemin.

Cette stratégie pose toutefois des problèmes : si le but ne peut être atteint dans la limite de profondeur fixée, il faut augmenter celle-ci et tout recommencer.

5.4.2 Stratégie avec mémoire : la recherche avec graphe (RAG)

On a un GRP qui est défini implicitement, en général trop grand pour être representé en mémoire et être exploré en totalité.

On va développer (c'est à dire représenter explicitement) une partie du GRP suffisante pour trouver le sommet but. On évitera ainsi les explorations redondantes.

Méthode (d'après [Nilsson])

1. **Créer un graphe de recherche** G qui consiste uniquement en un sommet de départ d. Mettre d sur une liste appelée OUVERT.
2. **Créer une liste appelée** FERME qui est initialement vide.
3. **BOUCLE**, si OUVERT est non-vide, sinon échec.
4. **Sélectionner le premier sommet d'OUVERT**, l'enlever d'OUVERT, et le mettre dans FERME, Appeler ce sommet n.
5. **Si** n **est un sommet but, terminer la procédure** avec succès.
6. **Développer le sommet** n, produisant l'ensemble M de ses successeurs et les mémoriser comme successeurs de n dans G.
7. **Mémoriser un pointeur vers** n **à partir des éléments de** M qui n'étaient pas déjà dans G (c'est à dire pas déjà dans OUVERT ou FERME). Ajouter ces éléments de M à OUVERT. Pour chaque élément de M qui était déjà dans OUVERT ou FERME, décider si l'on redirige ou non le pointeur vers n (voir ci-dessous). Pour chaque membre de M déjà dans FERME, décider pour chacun de ses descendants dans G si l'on redirige ou non le pointeur.
8. **Réordonner la liste OUVERT**, soit arbitrairement, soit selon des heuristiques.
9. **Aller à BOUCLE**

Les *pointeurs* dont il est question à l'étape 7 servent à retrouver, lorsqu'un sommet but b est atteint, un plus court chemin de d à b. Il suffit de "remonter" à partir de b en suivant les pointeurs.

Les heuristiques utilisées à l'étape 8 peuvent s'appuyer sur des connaissances spécifiques au problème.

5.4.3 Algorithmes A et A^*

Dans la recherche avec graphe (RAG), le choix de la fonction d'évaluation f pour le tri de OUVERT s'avère être un point crucial. Dans le cas de la méthode appelée **algorithme A**, on choisit f telle que pour tout sommet n du GRP, $f(n)$ estime le coût d'un chemin optimal de d à un but passant par n.

L'application f peut se mettre sous la forme :

$$f(n) = g(n) + h(n)$$

– $g(n)$ estime le coût d'un chemin optimal de d à n.
– $h(n)$ estime le coût d'un chemin optimal de n à un but.
Posons $g^*(n)$ le coût d'un chemin optimal dans le GRP de d à n et $h^*(n)$ le coût d'un chemin optimal de n à un but. On a $f^*(n) = g^*(n) + h^*(n)$. On veut que f estime (au mieux) f^*.

Pour $g(n)$ il est naturel de prendre le coût de d à n dans le graphe de recherche (partie du GRP déjà developpé). Pour $h(n)$ on s'appuie sur l'information spécifique au problème. On nomme h fonction heuristique.

Par exemple dans le cas du jeu de Taquin, on pourra prendre :
– $g(n)$ = distance (nombre d'arcs) de d à n dans le graphe de recherche.
– $h(n)$ = nombre de cases mal placées par rapport à la configuration but.
Si pour tout sommet n, $h(n) \leq h^*(n)$, alors on démontre que l'algorithme **A** trouvera un chemin optimal de d à un but (s'il en existe). Dans ce cas, on parle d'**algorithme A***. Dans le cas particulier où $h = 0$, on a bien un algorithme **A*** qui n'est autre que l'exploration en largeur.

Chapitre 6

Compléments

6.1 Exploration en profondeur

Algo EXPLORATION-PROFONDEUR (**Données :** arborescence $=(E, \Gamma)$, r racine)

```
1:  L = (r)
2:  while L ≠ ∅ do
3:     x = PREM(L)
4:     if Γ(x) = ∅ then
5:        L = RESTE(L)
6:     else
7:        L = (PREM(Γ(x)), L)
8:        Γ(x) = RESTE(Γ(x))
9:     end if
10: end while
```

Cet algorithme ne construit pas de résultat. Il explore tous les sommets d'une arborescence en se dirigeant vers le premier sommet fils (par rapport au sommet courant) non encore exploré. Lorsque plus aucun sommet fils ne reste à explorer, l'algorithme remonte à un sommet parent ayant des sommets fils non encore explorés.

La liste L correspond en fait à une pile, les opérations en ligne 5 et 7 correspondent à un POP et un PUSH. La complexité de ces opérations est trivialement en $O(1)$.

La condition en ligne 4 correspond à l'absence de sommets fils à explorer. Si cette dernière est vraie, l'action consécutive est une remontée et une descente dans le cas contraire.

Cet algorithme est une sorte de modèle (patron serait le terme exact) autour duquel peuvent être construites de nombreuses autres méthodes importantes. En effet, il est possible d'adapter très facilement cet algorithme pour qu'il parcoure l'ensemble des arêtes

d'un graphe. Cette nouvelle version consiste à éviter de repasser par un sommet déjà exploré. Par exemple, cette modification permet de mettre en place un algorithme simple et efficace de recherche de circuits dans un graphe.

6.2 Exploration en largeur

Algo EXPLORATION-LARGEUR (**Données :** arborescence $=(E, \Gamma)$, r racine)

1: $S = (r)$, $T = \emptyset$
2: **while** $S \neq \emptyset$ **do**
3: **for all** $x \in S$ **do**
4: **for all** $y \in \Gamma(x)$ **do**
5: $T = T \cup \{y\}$
6: **end for**
7: **end for**
8: $S = T$;
9: $T = \emptyset$
10: **end while**

Cet algorithme ne construit pas de résultat. Il explore tous les sommets d'une arborescence en les parcourant par niveau croissant. Les listes S et T correspondent à deux piles stockant les sommets restant à parcourir. L'une sert à parcourir les sommets du niveau courant, l'autre reçoit les sommets du niveau suivant et à la ligne 8 le niveau suivant devient le niveau courant. Cet algorithme est un classique à partir duquel peuvent être dérivées de multiples solutions algorithmiques.

Bibliographie

[Cormen] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction à l'algorithmique*, Dunod.

[Gondran] M. Gondran, M. Minoux, *Graphes et algorithmes*, Eyrolles.

[Nilsson] N. Nilsson, *Principes d'Intelligence artificielle*, Cépadues.

Index