Graphes et algorithmes

Denis Lapoire

22 février 2005

2

Ces notes concernent un module "Graphes et Algorithmes" dispensé aux étudiants de 1ère année Filière Informatique de l'ENSEIRB.

Les deux fils conducteurs de cet enseignement sont d'une part les algorithmes et d'autre part les graphes. Ce cours s'inscrit dans la continuité d'un cours dispensé un semestre auparavant portant sur les algorithmes et structures de données élémentaires (pile, file, arbre, etc...). Après une brève introduction des graphes, il présente la résolution algorithme de problèmes de bases admettant une solution de complexité en temps polynomiale, voire linéaire (voir table des matières).

Ce document s'est librement inspiré d'ouvrages. Citons en quelques uns :

- Introduction à l'algorithmique, T. Cormen et al., Dunod (1994).
  La structure de ces notes doit beaucoup à cet ouvrage.
- Graphes et algorithmes, M. Gondran et M. Minoux, Eyrolles (1995).
  Plusieurs exercices étudiés en Travaux Ðirigés ont pour source cet ouvrage.
- Graph Theory, R. Diestel, (Springer) 2000.
  Un ouvrage qui sera mis davantage à contribution pour les versions futures de ce cours.
- Modern Graph Theory, B. Bollobás, (Springer) 1998.
  Même remarque que ci-dessus.
- Graphes et Algorithmes, Dicky, Cours du CNAM.
  Ce document en ligne est accessible à partir de l'adresse `http://dept-info.labri.fr/~dicky/ENSEIGNEMENT/GA-CNAM/`. Ce lieu de ressources pédagogiques est par son abondance à recommander absolument.

Ce document peut contenir des omissions, des contre-sens voire des erreurs. Merci de me les signaler à : `lapoire@enserb.fr`. La correction de celles-ci figurera à : `http://www.enseirb.fr/~lapoir`

Ces notes de cours ainsi que d'autres documents pédagogiques sont accessibles à l'adresse : `http://www.enseirb.fr/~lapoire/1ereAnnee/Graphes/`.

Table des matières

Chapitre 1

Exemples de problèmes

Nous présentons ici quelques problèmes "concrets" et leurs formalisations mathématiques. Celle-ci consiste :

1. à formaliser les informations sous la forme d'un graphe.
2. à formaliser en termes de graphes le problème lui-même.

Ce chapitre n'est qu'une introduction informelle. Plusieurs termes seront introduits : graphe, sommet, arête, arc, chemin, cycle, etc. mais ne seront définis que dans le prochain chapitre.

1.1 Coloration d'un graphe

On souhaite savoir si il est possible de colorier un ensemble de pays de façon à ce que deux pays limitrophes aient toujours deux couleurs distinctes et ce en utilisant seulement 3-couleurs.

Formalisation mathématique du problème

formalisation des données

En considérant l'exemple des 6 premiers pays de la communauté européenne (Pays-Bas, Belgique,Luxembourg,Allemagne, France et Italie), le graphe formalisant les données est représenté par la Figure 1.1 et est le graphe "simple" "non orienté" (V, E) défini par :

- $V := \{p, b, l, f, a, i\}$ est l'ensemble des "sommets" représentant chacun des pays.
- $E := \{\{a, b\}, \{a, f\}, \{a, l\}, \{a, p\}, \{b, f\}, \{b, l\}, \{b, p\}, \{f, i\}, \{f, l\}\}$ est l'ensemble des "arêtes" représentant chacune des frontières.

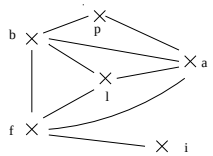

FIG. 1.1 – Graphe modélisant la carte de la C.E.E

formalisation du problème

Le problème consiste à décider de la 3-coloriabilité du graphe fourni en entrée. "3-Colorier" un graphe signifiant associer à chaque sommet une couleur parmi 3 fixées de telle façon que toute paire de sommets "adjacents" soient coloriés différemment. Plus formellement le problème est :

`3-coloriabilité`
Entrée : `un graphe` G `simple non orienté`
Sortie : `le booléen (`G `est 3-coloriable)`

1.2 Les ponts de Königsberg

La ville de Königsberg possède deux îles, au nord et au sud, deux rives, à l'est et à l'ouest et sept ponts. Un pont connecte ces deux îles. L'île au nord est reliée par un pont à chacune des deux rives. L'île au sud est reliée par deux ponts à chacune des deux rives. Deux problèmes se posent :

1. Est-il possible de se promener en passant une et une seule fois par chacun des ponts ?
2. Est-il possible de se promener en passant une et une seule fois par chacune des îles ou rives ?

Nous considérons ici qu'un promenade consiste à partir d'un point et à y revenir et ce sans traverser à la nage aucun des fleuves !

Formalisation mathématique du premier problème

formalisation des données

Le graphe à considérer ici est représenté par la Figure 1.2 et est le graphe non orienté "à arêtes multiples" (V, E, f) où :

- $V = \{n, s, e, o\}$ est l'ensemble des sommets représentant les 4 parties de la ville.
- $E = \{1, 2, 3, 4, 5, 6, 7\}$ est l'ensemble des arêtes représentant les ponts.
- f est la fonction $E \to \mathcal{P}(V)$ qui associe respectivement aux arêtes $1, \ldots, 7$ les paires de sommets $\{n, e\}$,$\{n, e\}$, $\{n, o\}$, $\{n, o\}$, $\{n, s\}$, $\{s, e\}$, $\{s, o\}$.

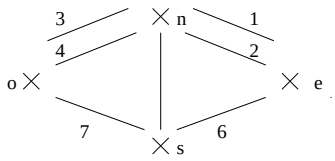

FIG. 1.2 – Graphe modélisant les ponts de Königsberg

formalisation du problème

Le problème se traduit en la recherche de l'existence d'un "cycle" "eulérien" dans le graphe, c'est à dire un cycle passant une et une seule fois par chacune des arêtes. Le problème général est donc :

`Cycle Eulérien`
Entrée : `un graphe` G `non orienté à arêtes multiples`
Sortie : `le booléen (`G `admet un cycle eulérien)`

Formalisation mathématique du second problème

formalisation des données

Pour formaliser le second problème, nous n'avons pas à nous soucier du nombre de ponts permettant de passer d'une rive à l'autre. La seule information à conserver est de savoir si l'on peut passer entre deux rives à l'aide d'un pont. Aussi, le graphe à considérer est le graphe simple induit par le graphe de la figure 1.2.

Ainsi, le graphe à considérer ici est représenté par la Figure 1.3 et est le graphe non orienté simple (V, E) où :

- $V = \{n, s, e, o\}$ est l'ensemble des sommets représentant les 4 parties de la ville.
- $E = \{\{n, e\}, \{n, o\}, \{n, s\}, \{s, e\}, \{s, o\}\}$ est l'ensemble des paires de rives admettant un pont les connectant.

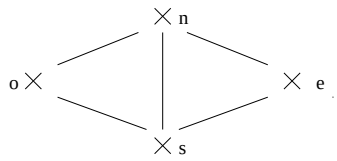

FIG. 1.3 – Graphe modélisant les ponts de Königsberg

formalisation du problème

Le problème se traduit en la recherche d'un cycle "hamiltonien" dans un tel graphe, c'est à dire un cycle passant une et une seule fois par chacun des sommets. Le prolème général est ainsi :

`Cycle Hamiltonien`
Entrée : `un graphe` G `simple non orienté`
Sortie : `le booléen (`G `admet un cycle hamiltonien)`

Remarque

Méfions nous des apparences ! Si les deux problèmes c'est à dire la recherche d'un cycle hamiltonien et la recherche d'un cycle eulérien paraissent semblables, il n'en est rien. En effet, le premier problème admet une solution algorithmique efficace (de complexité en temps

polynomialement bornée) alors que pour le second on en connait aucune. Certains même conjecturent qu'il n'en existe pas !

1.3 Choix d'un itinéraire

Soit le plan routier ainsi décrit :

9h Bordeaux-Lyon
8h Bordeaux-Marseille
5h Bordeaux-Paris
2h Grenoble-Lyon
3h Lyon-Marseille
3h Lyon-Paris

L'information ci-dessus indique que le temps d'un déplacement de Bordeaux à Lyon est de 9 heures ainsi que le temps d'un déplacement de Lyon à Bordeaux.

Souhaitant aller de Bordeaux à Grenoble le plus rapidement possible, comment dois je m'y prendre ?

Formalisation mathématique du problème

formalisation des données

Le graphe à considérer est représenté par la Figure 1.4 et est le graphe simple, non orienté et "étiqueté" (V, E, l) où :

- V l'ensemble des sommets représente les villes est $\{b, g, l, m, p\}$.
- E est l'ensemble de paire de sommets représentant chacune des routes, c'est à dire : $\{\{b, l\}, \{b, m\}, \{b, p\}, \{g, l\}, \{l, m\}, \{l, p\}\}$.
- l est une fonction qui associe aux paires $\{b, l\}, \{b, m\}, \{b, p\}, \{g, l\}, \{l, m\}, \{l, p\}$ respectivement les entiers : 9, 8, 5, 2, 3, 3.

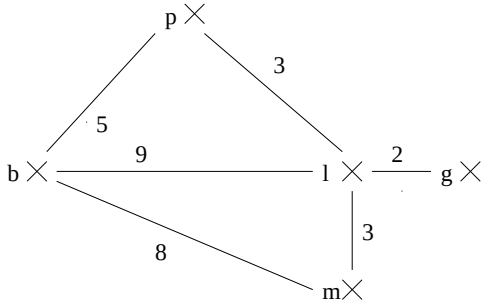

FIG. 1.4 – Graphe modélisant le réseau routier

formalisation du problème

Le problème consiste à calculer le "plus court" "chemin" (ou l'un des plus courts) allant de b à g.

`Plus court chemin`
Entrée : `un graphe` G `simple à arêtes pondérés,` s `et` t `deux sommets de` G
Sortie : `un plus court chemin de` G `allant de` s `à` t

1.4 Planification de travaux

Pour rénover une cuisine, il est prévu de **c**arreler (2 jours), de refaire l'installation **é**lectrique (3 jours), de **p**eindre quelques murs (4 jours) et de **t**apisser (1 jour). Le carrelage et la tapisserie ne doivent être faits qu'après la réfection de l'installation électrique. La tapisserie ne peut être fait qu'une fois les murs peints.

1. Si le propriétaire décide de tout faire de lui-même, dans quel ordre doit il procéder ?
2. Si la rénovation est faite par une entreprise et que chacune des tâches est accomplie par un employé différent, quelle est la durée minimale des travaux ?

Formalisation mathématique du premier problème

formalisation des données

Le graphe à considérer ici est représenté par la Figure 1.5 et est le graphe simple orienté (V, E) défini par :
- $V := \{c, e, p, t\}$ est l'ensemble des tâches à effectuer.
- $E := \{(e, c), (e, t), (p, t)\}$ est l'ensemble des arcs représentant les contraintes de précédence.

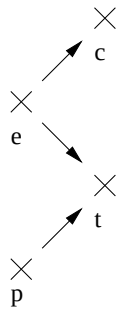

FIG. 1.5 – Graphe modélisant les tâches

formalisation du problème

Le problème consiste donc à calculer un "tri topologique" du graphe c'est à dire à numéroter les sommets du graphe de façon à ce pour tout arc (s, t) le numéro associé à s soit strictement inférieur à t. Le problème général est donc :

`Tri topologique`
Entrée : `un graphe` G `simple orienté`
Sortie : `un tri topologique de` G`, si il existe`

Formalisation mathématique du second problème

formalisation des données

Le graphe à considérer ici est représenté par la Figure 1.6 et est le graphe simple orienté à arcs étiquetés (V, E, l) défini par :

- $V := \{d, dc, fc, de, fe, dp, fp, dt, ft, f\}$ est l'ensemble des dates de début et fin des différentes tâches. d et f sont les dates de début et fin des travaux. dc et fc (resp. de et fe, dp et fp, dt et ft) sont les dates de début et fin des taches de carrelages (resp.électricité, peinture et tapisserie).
- E est l'ensemble des arcs représentant les contraintes de précédence qui indique que :
  - toute activité débute avant qu'elle ne finisse (arcs (dc, fc), (de, fe),(dp, fp),(dt, ft))
  - l'ensemble des travaux débute avant chaque début d'une tâche (arcs (d, dc), (d, de), (d, dp), (d, dt)) et termine après la fin de celle-ci (arcs (fc, f), (fe, f), (fp, f), (ft, f)).
  - les contraintes entre différentes tâches : arcs (fe, dc), (fe, dt) et (fp, dt).
- l associe respectivement à chaque arc (dc, fc), (de, fe),(dt, ft),(dp, fp) respectivement la valeur 2, 3, 1 et 4 et à tout autre arc 0.

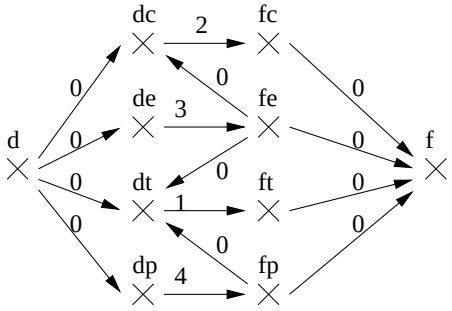

FIG. 1.6 – Graphe modélisant les tâches et leurs durées

formalisation du problème

Le problème consiste à calculer la longueur du plus long des chemins allant de d à f. Le problème général est donc :

`Plus long chemin`
Entrée : `un graphe` G `simple orienté à arcs pondérés,` s `et` t `deux sommets de` G
Sortie : `un plus long chemin dans` G `allant de` s `à` t`, si il en existe`

1.5 Premiers commentaires

Données pertinentes

Résoudre un problème sur un ensemble de données nécessite de faire le tri entre les données pertinentes et celles qui ne le sont pas. Selon que l'on résoud le premier ou le second

problème des ponts de Kónigsberg, distinguer si il existe un ou deux ponts entre deux rives est pertinent ou non.

Différents types de graphes

En conséquence, l'objet mathématique (ici des graphes) formalisant ces informations sera plus ou moins complexe. Il faudra alors choisir la structure la moins complexe. Nous manipulerons ainsi des graphes de types différents :

1. simples ou à arêtes multiples.
2. orientés ou non.
3. à sommets étiquetés ou non.
4. à arcs (ou arêtes) étiquetés (ées) ou non.

On peut s'en douter. D'autres exemples de graphes existent. Pour le simple plaisir des mots, citons en quelques-uns : les "hypergraphes" qui autorisent une arête à être incidente à plus de deux sommets, les "cartes" qui ordonnent partiellement les arêtes autour de chaque sommet. Un terme très présent dans la littérature, mais que nous n'emploierons pas, englobe ces différentes notions de graphes : celui de "structure relationnelle".

La variété de ces notions a une contrepartie : la nécessité de définir formellement et précisément le type de graphe utilisé ainsi que tous les outils et notions qui permettent de les manipuler. C'est l'objet du prochain chapitre.

Chapitre 2

Définitions générales

Nous présentons ici différents types de graphes, quelques notions et quelques opérations générales sur ceux-ci.

2.1 Graphes

Graphes orientés à arcs multiples

Un *graphe orienté* est un triplet (V, E, f) où :
- V, noté V_G, est un ensemble de *sommets* (*vertices* en anglais).
- E, noté E_G, est un ensemble d'*arcs* (*edges* en anglais).
- f est une application qui associe à chaque arc $e \in E$ un couple de sommets (s, t) de V : le premier sommet s est appelé l'*origine de* e, le second sommet t est appelé le *but de* e. On dit que t est un *successeur* de s et que s est un *prédécesseur* de t. L'arc e est un arc *entrant* de t et *sortant* de s. L'arc e est *incident* aux sommets s et t et inversement.

Deux sommets incidents à un même arc sont *adjacents*.

Exemple 1 Sur la Figure 2.1 est dessiné le graphe orienté à arcs multiples $G := (V, E, f)$ où :
- $V := \{1, 2, 3, 4\}$.
- $E := \{a, b, c, d, e\}$.
- $f := \{(a, (3, 2)), (b, (3, 2)), (c, (3, 4)), (d, (4, 3)), (e, (4, 4))\}$.

Graphes non orientés

Un graphe *non orienté* est un triplet (V, E, f) similaire à celui défini plus haut mais dans lequel tout élément $e \in E$ est associé à une paire de sommets $\{u, v\}$ de V ou à un singleton $\{u\}$ de V. Les éléments de E sont appelés les *arêtes* du graphes.

Evidemment, tout graphe orienté induit un unique graphe non orienté comme le montre l'exemple suivant :

Exemple 2 Sur la Figure 2.1 est dessiné le graphe non-orienté à arêtes multiples $H := (V, E, h)$ où :

- $V := \{1, 2, 3, 4\}$.
- $E := \{a, b, c, d, e\}$.
- $h := \{(a, \{2, 3\}), (b, \{2, 3\}), (c, \{3, 4\}), (d, \{3, 4\}), (e, \{4\})\}$.

Graphes simples

Un graphe (V, E, f) (orienté ou non) est *simple* si la fonction f est injective (c'est à dire si pour deux éléments distincts d et e de E on a : $f(d) \neq f(e)$). Un graphe simple est alors communément représenté par le couple (V, F) avec $F = f(V)$ c'est à dire par un ensemble de sommets V et un ensemble F :

- $\{(s_1, t_1), \ldots, (s_n, t_n)\}$ formé de couples de sommets si il s'agit d'un graphe orienté.
- $\{\{s_1, t_1\}, \ldots, \{s_n, t_n\}\}$ formé de paires ou de singletons de sommets si il s'agit d'un graphe non-orienté.

Exemple 3 Sur la Figure 2.1 est dessiné le graphe simple et orienté $I := (V, F)$ où :

- $V := \{1, 2, 3, 4\}$.
- $F := \{(3, 2), (3, 4), (4, 3), (4, 4)\}$.

Sur cette même figure, est dessiné le graphe simple et non orienté $J := (V, D)$ où :

- $V := \{1, 2, 3, 4\}$.
- $D := \{\{3, 2\}, \{3, 4\}, \{4\}\}$.

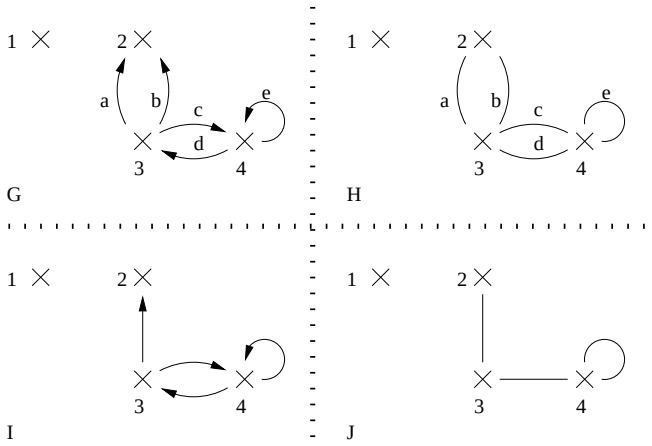

FIG. 2.1 – Graphes orientés ou non, simples ou non

Degrés et autres notions locales

Soit G un graphe orienté ou non.

Une *boucle* est un arc (ou une arête) incident à un unique sommet.

Un *sommet isolé* est un sommet adjacent à aucun autre sommet.

Le *degré* d'un sommet s, noté $deg_G(s)$, est le nombre d'arcs ou d'arêtes incidents à ce sommet (les boucles étant comptées double). Dans le cas des graphes orientés on peut

distinguer les arcs entrants de ceux sortants. Ainsi le *degré entrant* d'un sommet s est le nombre d'arcs entrants de s. Similairement, on définit le *degré sortant*.

Fait 1 Tout graphe G orienté ou non vérifie : $\sum_{s \in V_G} deg_G(s) = 2 \cdot \mathbf{card}\,(E_G)$.

preuve :
Fait en séance de TD. □

2.2 Une relation d'équivalence sur les graphes : l'isomorphisme

Il arrive très souvent quand on parle d'un graphe de penser non pas à un graphe mais à une classe de graphes isomorphes, c'est à dire à des graphes égaux à un renommage près des sommets et des arcs (ou arêtes) si ceux-ci ont un nom.

Par exemple : à la question "combien y-a t-il de graphes orientés simples sans boucles ayant exactement deux sommets ?", la réponse pourrait être une infinité (si l'on considère deux graphes isomorphes distincts comme par exemple $(\{1,2\},\emptyset)$ et $(\{7,8\},\emptyset)$ ou bien 2 (si l'on considère deux graphes isomorphes égaux).

Intuitivement, deux graphes sont égaux si on obtient le premier à partir du second en renommant chaque élément de son domaine. Nous donnerons ici une définition très générale de l'isomorphisme qui permet d'étendre cette notion à toute sorte de graphe (comme par exemple ceux orientés à arêtes multiples).

Rappelons ici qu'une *relation r d'arité k* sur un domaine E est une partie du produit cartésien E^k (c.a.d : $r \subseteq E^k$).

Définition 1 (Isomorphisme) Soient deux ensembles (E, r) et (F, s) deux ensembles munis de deux relations d'arité commune quelque entier k. Un *isomorphisme* de (E, r) dans (F, s), est une bijection $\varphi : E \to F$ telle que pour toute séquence $(e_1, \ldots, e_k)$ d'éléments de E on ait :

$$(e_1, \ldots, e_k) \in r \Leftrightarrow (\varphi(e_1), \ldots, \varphi(e_k)) \in s$$

Exemple 4 La Figure 2.2 présente trois exemples de relations isomorphes associées à des relations d'arité 1, 2 et 3. Sur chacune des lignes ($i = 1$, ou $i = 2$ ou $i = 3$), sont dessinés de gauche à droite une relation S_i ayant un domaine de 4 éléments et une relation d'arité i, une relation T_i isomorphe à S_i et la classe d'équivalence contenant S_i et T_i (en clair, le dessin de S_i et T_i débarassé des noms des éléments du domaine).

1. $S_1 = (\{1,2,3,4\}, \{(1),(2)\})$, $T_1 = (\{2,4,7,9\}, \{(2),(4)\})$,
2. $S_2 = (\{1,2,3,4\}, \{(1,2),(2,3),(1,3)\})$, $T_2 = (\{2,4,7,9\}, \{(2,4),(4,7),(2,7)\})$,
3. $S_3 = (\{1,2,3,4\}, \{(1,2,3),(3,4,4)\})$, $T_3 = (\{2,4,7,9\}, \{(2,4,7),(4,7,7)\})$,

Cette définition s'étend naturellement à toute sorte de graphe : par exemple les graphes orientés à arêtes multiples :

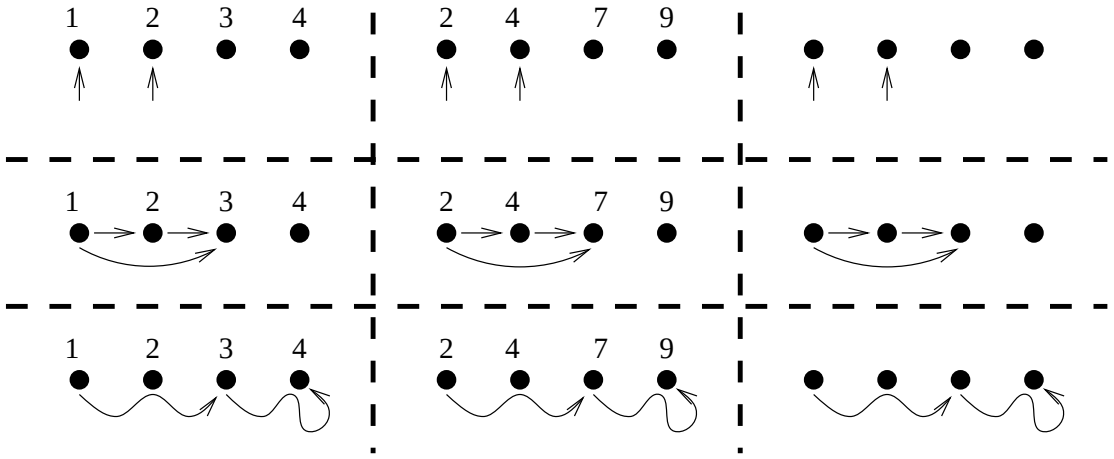

FIG. 2.2 – Quelques relations isomorphes

Exemple 5 Soient deux graphes orientés à arêtes multiples $G := (U, D, f)$ et (V, E, g). Ces deux graphes sont isomorphes si il existe deux bijections $\varphi : U \to V$ et $\psi : D \to E$ telles que pour tout arc $d \in D$ et pour tous sommets $(s, t) \in V^2$ on ait :

$$f(d) = (s, t) \Leftrightarrow g(\psi(d)) = (\varphi(s), \varphi(t))$$

2.3 Représentation des graphes

Nous avons vu l'existence d'une grande variété de types de graphes. Nous fournirons ici trois façons générales pour les représenter en vue de leurs manipulations algorithmiques.

Les graphes considérés ici auront pour ensemble de sommets des intervalles de la forme $[1, n]$ avec $n \geq 0$, et dans le cas de graphes à arcs ou arêtes multiples auront pour ensembles d'arcs ou arêtes des intervalles de la forme $[1, m]$ avec $m \geq 0$ (l'expression $[1, 0]$ désignant l'ensemble vide).

2.3.1 Représentation par matrice d'adjacence de graphes simples

Tout graphe orienté simple $([1, n], E)$ peut être représenté par sa *matrice d'adjacence*, c'est à dire la matrice M de booléens de taille $n \times n$ définie par :

$$M[i, j] := ((i, j) \in E)$$

Exemple 6 Le graphe orienté simple $([1, 4], \{(1, 2), (2, 4), (1, 3), (3, 4)\})$ admet pour codage la matrice

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

Avantages

- Cette représentation est un *codage* : tout graphe admet une unique représentation.
- Tester si un sommet est prédécesseur d'un second est réalisé en temps constant.

Inconvénient

- La taille de la représentation est élevée : $\Theta(n \cdot n)$. Un graphe peu "dense" (avec peu d'arcs) a une représentation de même taille qu'un graphe dense.

2.3.2 Représentation par matrice d'incidence

Tout graphe non orienté à arcs multiples $([1,n],[1,m],f)$ peut être représenté par sa *matrice d'incidence*, c'est à dire la matrice M de booléens de taille $n \times m$ définie par :

$$M[i,j] := (i \text{ est incident à j})$$

Exemple 7 Le graphe non orienté à arêtes multiples dessiné sur la figure ci-dessous et égal à $([1,4],['a','d'],('a',\{1\}),('b',\{2,3\}),('c',\{2,3\}),('d',\{1,3\}))$ a pour codage la matrice

	a	b	c	d
1	1	0	0	1
2	0	1	1	0
3	0	1	1	1
4	0	0	0	0

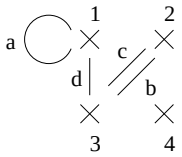

FIG. 2.3 – Un graphe à arêtes multiples

Avantages

- C'est un codage.
- Tester l'incidence d'un sommet et d'une arête est en temps constant.

Inconvénients

- Tester l'adjacence de deux sommets n'est pas en temps constant.
- La taille de la représentation est élevée $\Theta(n \cdot m)$. Un graphe peu "dense" tel un arbre a une représentation de taille $n \cdot (n-1)$.

2.3.3 Représentation par tableaux de listes

Un graphe orienté simple $([1,n],E)$ peut être représenté par un tableau T à indices dans $[1,n]$ et à valeurs des listes de sommets. Un tel tableau doit vérifier pour tout couple de sommets (i,j) :

$$j \in T[i] \Leftrightarrow (i,j) \in E$$

On suppose souvent dans une telle représentation que de telles listes sont sans répétition.

Exemple 8 Le graphe orienté $([1,4], \{(1,2),(2,4),(1,3),(3,4)\})$ a pour représentations les tableaux T et U définis par : $T(1) = (2,3)$, $T(2) = (4)$, $T(3) = (4)$, $T(4) = ()$ et : $U(1) = (3,2)$, $U(2) = (4)$, $U(3) = (4)$, $U(4) = ()$

Avantage

- L'espace mémoire est linéaire en la cardinalité du nombre de sommets et du nombre d'arêtes : $\Theta(n+m)$ (ce qui suppose les listes sans répétition).

Inconvénients

- Ce n'est pas un codage (voir exemple ci-dessus).
- Tester l'adjacence de deux sommets n est pas constant mais, dans le pire des cas, linéaire en le nombre d'arêtes.

Remarque

Pour obtenir un codage à partir d'une telle représentation, il suffirait d'exiger que les listes soient strictement croissantes. Avantage d'un côté, inconvénient de l'autre : maintenir ces listes triées a un coût !

2.4 Quelques opérations sur les graphes

2.4.1 Union disjointe

L'union de deux graphes sans sommet ni arc ou arête en commun est réalisé en faisant l'union des sommets et des arcs (ou arêtes). Formalisons cette définition trop peu précise dans le cas des graphes orientés :

Définition 2 Soient deux graphes orientés $G := (U, E, g)$ et $H := (V, F, h)$ vérifiant : $U \cap V = \emptyset$ et $E \cap F = \emptyset$. L'*union* de G et H est le graphe orienté $K := (U \cup V, E \cup F, k)$ où k associe à tout arc $e \in E$ le couple $g(e)$ et à tout arc $f \in F$ le couple $h(f)$.

2.4.2 Graphe partiel

On peut déterminer à partir d'un graphe G (orientée ou non, simple ou non) et d'un ensemble d'arcs D un nouveau graphe : il suffit de conserver tous les sommets et ne conserver que les arcs (ou arêtes) de D. Ce graphe est le *graphe partiel de G engendré par D*. Ce graphe sera noté $G|D$. Nous noterons aussi $G\backslash e$ le graphe $G|(D - \{e\})$ où e désigne une arête ou un arc de G).

Formalisons cette définition, sur l'exemple des graphes orientés à arcs multiples : si $G = (V, E, f)$ est un graphe orientée à arcs multiples et $D \subseteq E$ un ensemble d'arcs, le graphe partiel de G engendré par D est le triplet (V, D, g) où g est la restriction de f sur D.

Exemple 9 Sont représentés sur la figure ci-dessous deux graphes G et H ayant respectivement pour ensembles d'arêtes $['a',' g']$ et $['a',' d']$ et vérifiant $H = G|['a',' d']$.

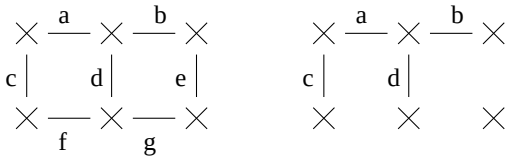

FIG. 2.4 – Un graphe et l'un de ses graphes partiels

2.4.3 Sous-graphe

On peut déterminer à partir d'un graphe G (orientée ou non, simple ou non) et d'un ensemble de sommets U un nouveau graphe : il suffit de conserver pour sommets ceux de U et pour arcs (ou arêtes) seulement ceux à extrémités toutes dans U. Ce graphe est le *sous-graphe de G induit par U* est noté $G|U$.

Formalisons cette définition, sur l'exemple des graphes orientés simples : si $G = (V, E)$ est un graphe orienté simple et si $U \subseteq V$ est un ensemble de sommets, le sous-graphe de G induit par U est le couple $(U, E \cap U \times U)$.

Exemple 10 Sont représentés sur la figure ci-dessous deux graphes G et H ayant respectivement pour ensembles de sommets $[1, 6]$ et $[2, 5]$ et vérifiant $H = G|[2, 5]$.

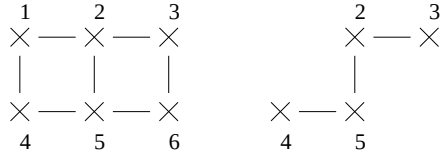

FIG. 2.5 – Un graphe et l'un de ses sous-graphes

2.4.4 Graphe quotient

On peut déterminer à partir d'un graphe G et d'une partition P de $V(G)$: il suffit de "fusionner" chaque partie $U \in P$ en un seul sommet.

Formaliser un telle définition dans le graphe orientée est très simple : si $G = (V, E, f)$ est un graphe orientée et $\sim$ une relation d'équivalence sur V, le graphe quotient de G par $\sim$ est le graphe (P, E, g) où :

- P est l'ensemble des classes d'équivalence de la forme $[u]_\sim$ avec $u \in V$.
- g est l'application qui associe à chaque arc $e \in E$ le couple $([u]_\sim, [v]_\sim)$ où $(u, v) = f(e)$.

Exemple 11 Sont représentés sur la figure ci-dessous un graphe G, une partition des sommets $P = \{\{1\}, \{5\}, \{2, 3, 6, 7\}, \{4, 8\}\}$ et le graphe quotient H.

On peut bien entendu définir des variantes du graphe quotient, en supprimant les arcs multiples ou en supprimant les boucles.

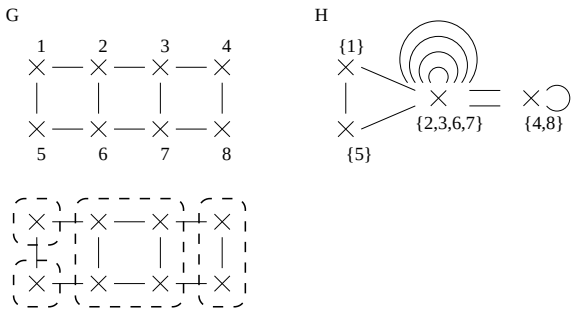

FIG. 2.6 – Graphe quotient

2.4.5 Conclusion

La première section a montré qu'il n'existe pas un "meilleur" type de graphe. Cette section a montré que, même ce type fixé, il n'existe pas une "meilleure" représentation de ce type de graphe. La représentation dépend en fait du problème à résoudre et est choisie afin de rendre minimal le coût en temps et en espace de l'algorithme résolvant ce problème.

Chapitre 3

Chemins et arbres

3.1 Chemins

3.1.1 Chemin

Un *chemin* dans un graphe orienté (resp. non orienté) (V, E, f) est une séquence w de la forme $(s_1, e_1, \ldots, e_l, s_{l+1})$ où pour tout $i \in [1, l]$, e_i est un arc allant du sommet s_i au sommet s_{i+1} (resp. une arête ayant pour extrémités les sommets s_i et s_{i+1}). L'entier l éventuellement nul est noté $|w|$ et est appelé la *longueur* de w. Le chemin est dit aller de s_1 à s_{l+1} ; s_1 (resp. s_{l+1}) est l'*extrémité initiale* (resp. *extrémité terminale*) de w ; les sommets $s_2, \ldots, s_l$ sont *internes* à w.

Un chemin est *simple* si il ne passe pas deux fois par le même arc. Il est *élémentaire* si il ne passe pas deux fois par le même sommet ; on autorise cependant l'égalité de ses deux extrémités.

remarque

Il est possible selon que le graphe est simple ou non orienté ou non, de représenter le chemin de façon plus ramassée :

- si le graphe est simple, un chemin $(s_1, e_1, \ldots, e_l, s_{l+1})$ peut être codé à l'aide de la séquence $(s_1, \ldots, s_{l+1})$. Dans ce cas là, on peut définir un chemin comme une séquence de sommets de la forme $(s_1, \ldots, s_{l+1})$ où pour tout entier $i \in [1, l]$, il existe un arc de s_i à s_{i+1} (cas orienté) ou une arête entre s_i et s_{i+1} (cas non orienté).
- si le graphe est orienté, un chemin $(s_1, e_1, \ldots, e_l, s_{l+1})$ de longueur non nulle peut être codée par la séquence $(e_1, \ldots, e_l)$.

De façon plus générale, un chemin $(s_1, e_1, s_2, e_2, \ldots, s_l, e_l, s_{l+1})$ d'un graphe (simple ou non, orienté ou non) peut être codé par la séquence $(s_1, e_1, e_2, \ldots, e_l)$.

3.1.2 Concaténation

Une opération naturelle sur les chemins est la concaténation : la concaténation de deux chemins u et v allant respectivement d'un sommet x à un sommet y et d'un sommet y à un sommet z est le chemin noté $u \cdot v$ obtenu en concaténant à la séquence u la séquence

v débarassée de son premier élément y. Clairement, le chemin $u \cdot v$ va de x à z et a pour longueur :

$$|u \cdot v| = |u| + |v|$$

3.1.3 Distance dans un graphe

La *distance* d'un sommet s à un sommet t dans un graphe G (orienté ou non) est notée $d_G(s,t)$ et est la longueur du plus court chemin allant de s à t si il en existe ou $+\infty$ sinon. D'une telle définition, il découle immédiatement :

$$\forall (s,t,u) \in V_G^3 \; d_G(s,u) \leq d_G(s,t) + d_G(t,u)$$

Le *diamêtre* d'un graphe est la quantité $\max_{s,t \in V_G} d_G(s,t)$.

3.2 Composantes connexes et fortement connexes

3.2.1 Cas non orienté

Soit G un graphe non orienté. Un ensemble de sommets U de G est *connexe* si pour tout couple de sommets (s,t) de U il existe un chemin à sommets dans U allant de s à t.

Une *composante connexe* de G est un ensemble de sommets connexe et maximal selon l'inclusion à vérifier cette propriété.

Le graphe G est *connexe* si son ensemble de sommets est connexe.

Exemple 12 Considérons le graphe non orienté G de la Figure 3.1. On observe que :

- $(1, e, 4, c, 3)$ est un chemin.
- les ensembles $\{1,3\}$ et $\{2,4\}$ ne sont pas connexes.
- les ensembles connexes du graphe sont tous les singletons (trivialement), toutes les paires d'extrémités d'arêtes ainsi que les ensembles $\{1,2,3\}$, $\{2,3,4\}$, $\{3,4,1\}$, $\{4,1,2\}$ et $\{1,2,3,4\}$.
- les somposantes connexes sont $\{1,2,3,4\}$ et $\{5,6\}$.
- G n'est pas connexe.

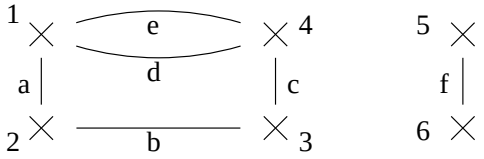

FIG. 3.1 – Un graphe non orienté

3.2.2 Dans le cas orienté, on parle de forte connexité

Les termes connexité, connexe, composante connexe sont remplacés respectivement par forte connexité, fortement connexe, composante fortement connexe.

Ainsi, si G est un graphe orienté. Un ensemble de sommets U de G est *fortement connexe* si pour tout couple de sommets (s,t) de U il existe un chemin à sommets dans U allant de s à t.

Une *composante fortement connexe* de G est un ensemble de sommets connexe et maximal selon l'inclusion à vérifier cette propriété.

Le graphe G est *fortement connexe* si son ensemble de sommets est fortement connexe.

Exemple 13 Considérons le graphe orienté G de la Figure 3.2. On observe que :
- $(1, b, 2, a, 1)$ est un chemin qui est un cycle simple et élémentaire.
- les ensembles $\{1,3\}$ et $\{4,5\}$ ne sont pas fortement connexes.
- les ensembles fortement connexes du graphe sont tous les singletons (trivialement) ainsi que les ensembles $\{1,2\}$, $\{5,6\}$ et $\{4,5,6\}$.
- les somposantes fortement connexes sont $\{1,2\}$, $\{3\}$ et $\{4,5,6\}$.
- G n'est pas fortement connexe.

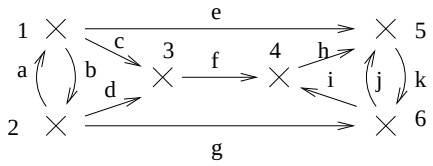

FIG. 3.2 – Un graphe orienté

3.3 Cycle

Un *cycle* dans un graphe est un chemin dont les deux extrémités sont égales. Les adjectifs *élémentaires* et *simples* sont étendus naturellement aux cycles : un cycle élémentaire est un chemin qui est un cycle et qui est élémentaire, un cycle simple est un chemin qui est un cycle et qui est simple.

Clairement si un graphe (ordonné ou non) contient un cycle, il contient un cycle élémentaire. Cette remarque ne s'étend pas à la propriété "être cycle simple". En effet cela dépend si il s'agit d'un graphe orienté ou non. Dans le cas orienté, la réponse est oui comme l'indique le fait suivant :

Fait 2 Si un graphe orienté possède un cycle, il possède un cycle simple et élémentaire.

Dans le cas non orienté, tout graphe possédant au moins une arête, possède un cycle : l'arête e d'extrémités s et t permet de construire le cycle (s, e, t, e, s).

Ainsi, la notion intéressante n'est pas "cycle" mais "cycle simple" (qui interdit de réutiliser deux fois la même arête ou deux fois le même arc).

Définition 3 Un graphe est *acyclique* si il ne possède aucun cycle simple de longueur non nulle.

circuit

Souvent dans la littérature, on emploie le terme cycle pour les graphes non orientés et *circuit* pour les graphes orientés.

3.4 Arborescence et arbre

Il existe un grand nombre de caractérisations des arbres et des arborescences : ce sont des structures qui possèdent un sommet à partir duquel on peut atteindre tous les autres sommets et minimales selon propriété (cette propriété se perd par la suppression de tout arc (ou toute arête)).

3.4.1 Arbre dans le cas orienté se dit arborescence

Une *arborescence* est un graphe orienté admettant un sommet, appelé la *racine*, tel que pour tout sommet il existe un unique chemin de la racine vers ce sommet. Clairement, tout sommet autre que la racine, admet un unique prédécesseur appelé son *père*.

Exemple 14 L'arborescence $(\{1,2,3,4,5,6\}, \{(2,1),(2,3),(3,4),(3,5),(3,6)\})$ de racine 2 est représentée sur la figure ci-dessous.

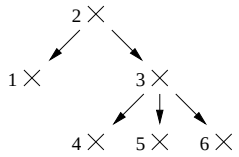

FIG. 3.3 – Une arborescence de racine 2

Une *forêt* est une union disjointe d'arborescences. Une caractérisation très naturelle est la suivante. La preuve est laissée au lecteur.

Fait 3 Pour tout graphe orienté G, les deux assertions suivantes sont équivalentes :

1. G est une forêt.
2. G est acyclique et le degré entrant de tout sommet est au plus 1.

3.4.2 Arbre

Un *arbre* est un graphe non orienté connexe et acyclique. Une *forêt* est une union d'arbres disjoints, c'est à dire un graphe acyclique.

Exemple 15 L'arbre $(\{1,2,3,4,5,6\}, \{\{1,2\},\{2,3\},\{2,4\},\{4,5\},\{4,6\}\})$ est représenté sur la figure ci-dessous.

Une propriété remarquable des arbres est de pouvoir les caractériser de différentes façons :

Fait 4 Pour tout graphe $G = (V, E, f)$ non orienté, les assertions suivantes sont équivalentes :

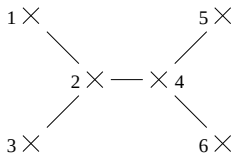

FIG. 3.4 – Un arbre

1. G est un arbre (c.a.d connexe et acyclique).
2. G est connexe et est minimal à vérifier cette propriété (c.a.d la suppression d'une arête quelconque déconnecte G).
3. G est connexe et vérifie $|V| \geq |E| + 1$.
4. G est acyclique et est maximal à vérifier cette propriété (c.a.d l'ajout de toute nouvelle arête crée un cycle simple).
5. G est acyclique et vérifie : $|V| \leq |E| + 1$.
6. pour tout couple de sommets (s, t) de G, il existe un unique chemin simple allant de s à t.

preuve :
Voir TD. □

3.5 Un petit lexique

Pour conclure, présentons quelques familles de graphes. Leurs propriétés ne seront pas étudiées.

graphes discrets

Un graphe est *discret* si il ne contient aucune arête.

FIG. 3.5 – Le graphe discret à 4 sommets

étoiles, peignes et chenilles

Nous avons dans une section précédente présenté la famille des forêts, des arbres et des chemins. Quelques sous familles se distinguent :

1. une *étoile* est un arbre dont un sommet est adjacent à tous les autres.
2. une *chenille* est un arbre tel que tout sommet de degré ≥ 2 est adjacent à au plus deux sommets de degré ≥ 2.

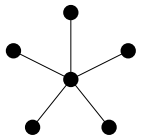

FIG. 3.6 – L'étoile à 6 sommets

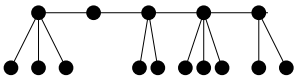

FIG. 3.7 – Une chenille à 15 sommets

3. un *peigne* est une chenille à sommets de degré 1 ou 3, exceptés exactement deux sommets de degré 2.

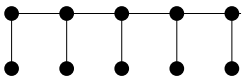

FIG. 3.8 – Un peigne à 10 sommets

graphes planaires

Un graphe est *planaire* si il peut être dessiné de telle façon qu'aucune paire d'arêtes ne se croisent. Un exemple de graphe planaire est la *grille* $n \times n$ ayant pour ensemble de sommets $V := [1,n] \times [1,n]$ et pour ensemble d'arêtes toute paire $\{(i,j),(i',j')\} \in V$ vérifiant $(i = i' \wedge |j - j'| = 1) \vee (j = j' \wedge |i - i'| = 1)$.

Citons ici un des résultats mathématique les plus célèbres : la 4-coloriabilité des graphes planaires (les sommets d'un graphe planaire peuvent être coloriés de 4 couleurs différentes sans que deux sommets adjacents n'aient la même couleur). Ce résultat n'a été prouvé que très récemment ; la preuve présentée est très longue et nécessite l'utilisation d'un ordinateur !

graphes complets et tournois

Le graphe *complet* à n sommets est un graphe simple non orienté sans boucle où tous les sommets sont deux à deux adjacents. Un *tournoi* est un graphe orienté obtenu à partir d'un graphe complet en orientant chaque arête $\{x,y\}$ en choisissant ou bien l'arc (x,y) ou bien l'arc (y,x).

Propriété qui tient du folklore : le graphe K_5 est le plus petit graphe à ne pas être planaire.

graphes bipartis

Un graphe *biparti* est un graphe simple non orienté (V,E) admettant une partition $\{A,B\}$ de son ensemble de sommets tel que :

$$\{a,b\} \in E \Rightarrow (a,b) \in A \times B \cup B \times A$$

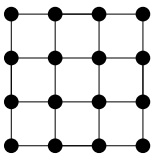

FIG. 3.9 – La grille à 4×4 sommets

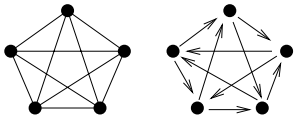

FIG. 3.10 – Le graphe complet K_5 et un tournoi à 5 sommets

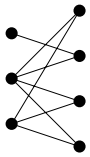

FIG. 3.11 – Un graphe biparti

Un ensemble remarquable de bipartis est l'ensemble des arbres ou plus généralement celui des forêts.

graphes bipartis complets

Un graphe est *biparti-complet* si il existe une partition $\{A, B\}$ de V tel que :

$$\{a, b\} \in E \Leftrightarrow (a, b) \in A \times B \cup B \times A$$

Observons qu'une étoile est un biparti complet dont l'une des parties est un singleton.

Un biparti complet célèbre est celui possèdant deux parties de 3 sommets excatement. Ce graphe noté $K_{3,3}$ est un graphe non planaire qui de plus est minimal à ne pas être planaire (toute suppression d'arête le rend planaire).

graphes réguliers

Un graphe *régulier* est un graphe non orienté simple dont tous les sommets ont même degré.

hypercubes

L'*hypercube* H_n à 2^n sommets est défini comme le graphe ayant pour ensemble de sommets les mots binaires de longueur n où deux sommets sont adjacents si ils diffèrent par exactement

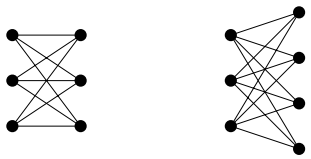

FIG. 3.12 – Les graphe bipartis complets $K_{3,3}$ et $K_{3,4}$

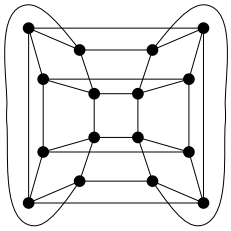

FIG. 3.13 – Un graphe régulier

un bit. Il est aisé d'observer que le graphe H_n est régulier de degré n.

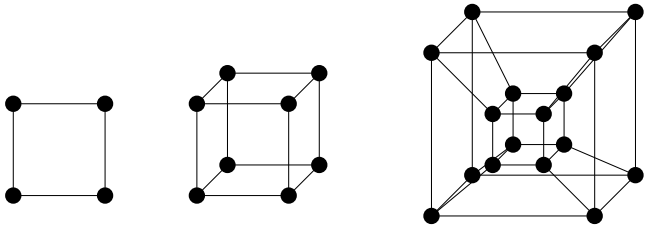

FIG. 3.14 – Les hypercubes H_2, H_3 et H_4

Chapitre 4

Le problème de l'arbre couvrant minimal

Dans ce chapitre, les graphes sont non orientés. Afin de gagner en lisibilité, ils sont supposés simples. Mais cette restriction n'est pas impérative. Un *arbre couvrant* d'un graphe (simple et non orienté) $G := (V, E, f)$ est un ensemble d'arêtes $D \subseteq E$ pour lequel (V, D) est un arbre. Si l'on suppose que le graphe est *pondéré*, c'est à dire est muni d'une fonction p qui associe à toute arête un réel > 0 appelé son *poids*, un arbre couvrant D est dit *minimal* si son *poids*, $p(D) := \sum_{d \in D} p(d)$, est au plus égale au poids de tout arbre couvrant. Le problème de l'*arbre couvrant minimal* (noté en abrégé `ACM`) est alors naturellement :

`ACM`
`ENTRÉE : un graphe` G `non orienté connexe`
`SORTIE : un arbre couvrant minimal de` G

Lors de ce chapitre, le terme graphe désignera un graphe simple non orienté muni d'une fonction de pondération sur les arêtes ($E_G \to \mathbb{R}$) notée p_G.

4.1 Un algorithme générique

Avant de présenter un algorithme générique résolvant le problème `ACM`, un simple fait qui inspire directement ce même algorithme et qui prouve sa correction. La présentation de ce fait nécessite la définition suivante :

Définition 4 Une *coupure* d'un graphe G est un couple partitionnant l'ensemble des sommets, c'est à dire un couple de la forme $(P, V_G - P)$ avec $P \subseteq V_G$. Une arête $e \in E_G$ *traverse* la coupure $(P, V_G - P)$ si l'une des extrémités est dans P est l'autre non. Une coupure *respecte* un ensemble d'arêtes $D \subseteq E_G$ si aucune arête $e \in D$ ne traverse la coupure.

Fait 5 Soit $G := (V, E, f, p)$ un graphe connexe. Soit D un ensemble d'arêtes contenu dans un ACM. Si $(P, V - P)$ est une coupure respectant D et si e est une arête de poids minimal traversant $(P, V - P)$, alors $D \cup \{e\}$ est contenu dans un arbre couvrant minimal.

preuve :
Reprenant les notations et hypothèses de l'énoncé, nous notons T un ACM contenant D. Si

$e \in T$, la conclusion est immédiate. Considérons le cas contraire : $e \notin T$. Puisque T est un arbre, $T \cup \{e\}$ contient un cycle c. Le cycle c contient e, ainsi le chemin w obtenu à partir de c en supprimant e contient un sommet de P et un autre de $V - P$. L'une de ses arêtes, notons la f, traverse ainsi $(P, V - P)$.

L'ensemble d'arêtes $U := (T \backslash f) \cup \{e\}$ est connexe, car $T \cup \{e\}$ est connexe et possède un cycle contenant e et f, a même cardinalité que T et est donc un arbre couvrant de G.

De plus, le poids de f est au moins celui de e. On en déduit que le poids de U, égal à $p(U) = p(T) - p(f) + p(e)$, et au plus égal à celui de $p(T)$. Ainsi, U est une solution contenant e. □

Conséquence directe de ce fait l'algorithme suivant (Figure 4.1) qui résoud de façon récursive `ACM`.

```
fonction Acm(G:graphe):ensemble d'arêtes
    retourner AcmRécursif(G,∅);

fonction AcmRécursif(G:graphe ; D: ens. d'arêtes):ensemble d'arêtes
    si D est un arbre couvrant de G
        retourner D
    sinon
        choisir une coupure (P, V_G − P) respectant D et
                une arête e de poids minimal traversant (P, V_G − P);
        retourner AcmRécursif(G; D ∪ {e}) ;
```

FIG. 4.1 – Algorithme `AcmRécursif`

Corollaire 6 L'algorithme `Acm` se termine et résoud le problème `ACM`.

preuve :
Établir ceci revient à établir que l'algorithme `AcmRécursif` se termine et résoud le problème suivant :

`ACM'`
`ENTRÉE : un graphe` G `non orienté connexe, une partie` D `d'un ACM de` G
`SORTIE : un arbre couvrant minimal de` G `contenant` D

Il vient :

1. l'algorithme `AcmRécursif` termine.
   Conséquence directe de la Définition 4, toute arête traversant une coupure respectant un ensemble d'arêtes D ne peut appartenir à D, il en découle l'inclusion stricte $D \subset D \cup \{e\}$. L'ensemble D étant contenu dans l'ensemble fini E_G, l'algorithme termine. De facçon plus précise, l'ensemble étant une partie d'un arbre couvrant, il ne peut contenir plus de $|V_G| - 1$ arêtes. Le nombre d'appels récursifs est donc exactement $|V_G| - 1$.
2. l'algorithme `AcmRécursif` résoud `ACM'`.
   Démontrons que pour toute partie D d'un ACm de G, l'ensemble d'arêtes retournée est un ACM de G contenant D. Deux cas apparaissent :

(a) D couvre G.
Clairement D est un ACM de G.

(b) D ne couvre pas G.
Conséquence du fait que pour toute partie D contenue strictement dans un ACM, D n'est pas connexe ($G|D$ n'est pas un arbre (car est contenue strictement dans un arbre), est non connexe car acyclique), ainsi admet une coupure (formée d'une composante connexe de $G|D$ et du restant non vide de sommets G) traversée par un ensemble non vide d'arêtes. Ainsi il existe une arête e de poids minimal parmi celles traversant une coupure respectant D. L'instruction `choisir une coupure ...` peut donc être exécutée. Conséquence du Fait 5, pour une telle arête e il existe une ACM de G contenant $D \cup \{e\}$. En conséquence de quoi, l'ensemble retourné par `AcmRécursif(\(G\); \(D\cup\{e\}\))` ; par induction un ACM de G contenant $D \cup \{e\}$ est bien un ACM de G contenant D.

□

L'algorithme précédent peut être écrit de façon itérative de la façon décrite par la Figure 4.2. Nous étendons très naturellement la terminologie des graphes aux ensembles

```
fonction AcmItératif(G:graphe pondéré non orienté): ens. d'arêtes
   D ← ∅ ;

   tantque D n'est pas un arbre couvrant de G
        choisir une coupure (P, V_G − P) respectant D et
                une arête e de poids minimal traversant (P, V_G − P);
        D ← D ∪ {e} ;

   retourner D ;
```

FIG. 4.2 – Algorithme `AcmItératif`

d'arêtes : ainsi, une partie D d'un ensemble d'arêtes d'un graphe G est *acyclique* (resp. *connexe*) si le graphe partiel de G engendré par D (noté $G \mid D$ voir sous-section 2.4.2) est acyclique (resp. *connexe*).

Corollaire 7 Si le graphe en entrée est connexe, l'algorithme `AcmItératif` se termine et retourne un arbre couvrant minimal de G.

preuve :
Démontrons la terminaison puis la correction de l'algorithme :

1. l'algorithme se termine.
   En effet la cardinalité de D est bornée par celle, finie de E, et à chaque passage de boucle, la cardinalité de D est incrémentée : D est augmenté d'une arête e qui ne peut trivialement pas appartenir à D (une arête e ne peut pas traverser une coupure respectant un ensemble contenant e).

2. l'algorithme est correct.
   Observons que si le graphe est connexe, toute coupure respectant quelque partie D ne couvrant pas G admet une arête traversant cette coupure. Ainsi, on peut toujours choisir une telle arête. En utilisant le précédent fait, on démontre par récurrence que l'ensemble D est à tout moment contenu dans un ACM. Ainsi, l'ensemble D retourné est un ensemble couvrant G contenu dans un arbre couvrant minimal : il est donc un arbre couvrant minimal de G.

□

Il existe différentes façons d'implémenter l'algorithme `AcmItératif`. Nous en étudions deux :

`Kruskal` L'arête choisie est de poids minimal parmi toutes celles à avoir deux extrémités non déjà connectées par la solution courante D c'est à dire une arête de poids minimal parmi toutes celles qui traverse une coupe respectant D.

`Prim` la seconde façon consiste à avoir à tout instant une coupure $(V - P, P)$ de la forme suivante :
- D connecte tous les sommets de $V - P$.
- aucune arête de D n'est incident à un quelconque sommet de P.

En d'autres terme, le graphe $G|D$ est composé d'un arbre ayant pour sommets ceux de $P - V$ et d'une union de sommets isolés P.

4.1.1 Algorithme `Kruskal`

Comme nous l'avons déjà dit, la première implémentation de `AcmItératif` par `Kruskal` consiste à choisir une arête de poids maximal parmi toutes celles qui traversent une coupe respectant les arêtes déjà choisies, c'est à dire choisir une arête qui ne crée pas un cycle dans la solution courante. Cet algorithme a pour définition celle de la Figure 4.3.

```
fonction Acm-Kruskal1(G:graphe):ensemble d'arêtes

    trier l'ensemble des arêtes E_G par poids croissant ;
    D ← ∅ ;

    pour chaque arête e pris par poids croissant
        si D ∪ {e} est acyclique alors
            D ← D ∪ {e} ;

    retourner D ;
```

FIG. 4.3 – Algorithme `Acm-Kruskal1`

Pour tester si l'ajout d'une arête e menace ou non l'acyclicité de la solution courante D, il suffit de tester l'égalité des composantes connexes dans le sous-graphe $G|D$ contenant

les deux extrémité de l'arête e. Cela requiert un type `partition` possédant les opérations suivantes :

- `partitionDiscrète` : `ensemble` $\rightarrow$ `partition` qui retourne la partition d'un ensemble E formée de ses singletons.
- `équiv` : `partition`$\times$`élément`$\times$`élément`$\rightarrow$`booléen` qui décide si deux éléments appartiennent à une même partie.
- `union` : `partition`$\times$`élément`$\times$`élément`$\rightarrow$`partition` qui transforme une partition en faisant l'union des deux parties contenant les deux éléments fournis en entrée.

Le type `partition` ainsi défini, l'algorithme `Acm-Kruskal1` se réécrit en l'algorithme défini sur la Figure 4.4.

```
fonction Acm-Kruskal2(G:graphe pondéré):ensemble d'arêtes

    trier l'ensemble d'arêtes E_G par poids croissant ;
    D ← ∅ ;
    P ← partitionDiscrète(V_G);

    pour chaque arête e = {u,v} pris par poids croissant
        si non équiv(P,u,v)
            P ← union(P,u,v) ;
            D ← {e} ∪ D ;

    retourner D ;
```

FIG. 4.4 – Algorithme `Acm-Kruskal2`

Pour implémenter efficacement selon des considérations de temps et d'espace ce dernier algorithme, il nous suffit de choisir une bonne implémentation du type `partition`. Cette étude sera réalisée en TD.

4.1.2 Algorithme `Prim`

L'algorithme `Prim` construit une coupure $(V-P, P)$ et un ensemble D de telles sorte qu'à chaque instant l'ensemble acyclique D connecte les sommets de $V-P$ en un arbre et laisse chacun des sommets de P isolés. Une première version de cet algorithme est celle définie sur la Figure 4.5.

L'arbre couvrant minimal de G sera représenté à l'aide d'une fonction `père` qui associe à tout sommet, sauf un (la racine), un sommet. Pour choisir rapidement une arête traversant $(V-P, P)$ de poids minimal, on fait en sorte qu'à chaque instant :

- l'arête $\{$`père`$(x), x\}$ soit une arête de poids minimal à traverser la coupe $(V-P, P)$
- on connaisse le poids de cette arête. C'est l'objet de la fonction `clé`(x).

L'algorithme `Acm-Prim2` est défini sur la figure 4.6.

Pour obtenir une implémentation efficace en temps et en espace, il nous faut choisir notamment une bonne implémentation de l'ensemble P. Son étude sera réalisée en TD.

```
fonction Acm-Prim1(G:graphe pondéré): ensemble d'arêtes

    D ← ∅ ;
    choisir un sommet r dans V_G ;
    P ← P\r ;

    tantque non(estVide(P)) faire
        choisir une arête e de poids minimal à traverser (V − P, P) ;
        D ← D ∪ {e} ;
        P ← P\x où x est l'extrémité de e appartenant à P ;

    retourner D ;
```

FIG. 4.5 – L'algorithme Acm-Prim1

```
fonction Acm-Prim2(G:graphe pondéré): fonction V_G → V_G

    clé ← tableau indicé par V_G initialisé à +∞ ;
    père ← tableau indicé par V_G initialisé à NULL ;

    P ← V_G ;

    choisir un sommet r dans V_G ;
    clé[r] ← 0 ;

    tantque non(estVide(P)) faire
        extraire de P un élément x de clef minimale ;
        pour chaque voisin y de x faire
            si y ∈ P et p_G(x, y) <clé(y) alors
                clé[v] ← p_G(x, y) ;
                père[y] ← x ;

    retourner père ;
```

FIG. 4.6 – L'algorithme Acm-Prim2

Chapitre 5

Le problème du plus court chemin

Le problème du plus court chemin considéré ici porte sur des graphes orientés à arcs pondérés (dont chaque arc est associé à un réel positif ou nul). Le plus court chemin d'un sommet s à un sommet t est alors un chemin de s à t dont le poids, la somme des poids des arcs qu'il contient, est minimale.

Si l'on cherche à préciser exactement le problème, différents choix se présentent à nous. Ainsi, il existe plusieurs problèmes du plus court chemin qui se distinguent par les propriétés des graphes fournis en entrée :

- possèdent-ils des arcs négatifs ou non ?
- possèdent-ils des circuits de poids strictement négatifs ?

et de l'information à fournir en sortie, selon qu'il s'agit de calculer :

- un plus court chemin d'un sommet s donné à un sommet t donné.
- des plus courts chemins d'un sommet s donné à tous les autres sommets.
- pour tout couple de sommets (s, t) d'un plus court chemin de s à t.

Nous nous intéresserons dans ce cours qu'au deuxième problème (une unique source). Les graphes considérés dans ce chapitre sont, sauf mention contraire, simples et orientés.

5.1 Définitions

Définition 5 Soit $G = (V, E, p)$ un graphe à arcs pondérés. La *longueur pondérée* d'un chemin $(s_0, \ldots, s_l)$ est la somme des poids de ses arcs $\sum_{i \in [1,l]} p(s_{i-1}, s_i)$. Un chemin w allant de s à t est un *plus court chemin* si tout chemin de s à t est de longueur pondérée au moins égale à celle de w. La *distance pondérée* entre deux sommets s et t est la longueur d'un plus court chemin allant de s à t, si il en existe. Elle est notée $\delta(s, t)$. Afin de simplifier certaines notations, dans le cas d'absence de chemins de s à t, $\delta(s, t)$ sera supposée égale à $+\infty$.

Fait 8 Tout chemin extrait d'un plus plus court chemin d'un graphe à arcs pondérés est aussi un plus court chemin.

preuve :
Soit $u = (s_0, \ldots, s_l)$ un plus court chemin d'un graphe à arcs pondérés $G = (V, E, p)$. Soit v un chemin extrait de u, c'est à dire une séquence de la forme $(s_i, \ldots, s_j)$ avec $0 \leq i \leq j \leq l$.

Pour conclure, démontrons que tout chemin w de s_i à s_j est de longueur pondérée égale au moins à celle de v.

Soit $w = (t_0, \ldots, t_m)$ un chemin de s_i à s_j. La séquence x obtenue à partir de u en remplaçant sa sous-séquence $(s_i, \ldots, s_j)$ par la séquence $(t_0, \ldots, t_m)$ est un chemin de s à t (on a : $t_0 = s_i$ et $s_j = t_m$) de longueur pondérée $p(x) = p(u) - p(v) + p(w)$. Par hypothèse u est un plus court chemin. Ce qui entraîne $p(x) \geq p(u)$ et donc $p(w) \geq p(v)$. □

Fait 9 Si w est un plus court chemin allant d'un sommet s à un sommet $u \neq s$, alors le sommet t qui précède u dans w vérifie :

$$\delta(s, u) = \delta(s, t) + p(t, u)$$

preuve :
Soient $G = (V, E, p)$ un graphe à arcs pondérés, s et u deux sommets distincts et $w = (s_0, \ldots, s_l)$ un plus court chemin de $s = s_0$ à $u = s_l$. Soit v le sommet s_{l-1}.

D'après le Fait 8, $(s_0, \ldots, s_{l-1})$ est un plus court chemin. Ainsi, $\delta(s, t) = p(s_0, \ldots, s_{l-1}) = p(w) - p(t, u)$. Par hypothèse, w est un plus court chemin, ainsi $\delta(s, u) = p(w)$. Il vient : $\delta(s, u) = \delta(s, t) + p(t, u)$. □

Fait 10 Soient s et t deux sommets d'un graphe à arcs pondérés. Les deux assertions suivantes sont équivalentes :

- il existe un plus court chemin de s à t.
- il existe un chemin de s à t. De plus, tout cycle d'extrémité un sommet contenu dans un chemin allant de s à t est de longueur positive ou nulle.

preuve :
Soient $G := (V, E, p)$ un graphe à arcs pondérés et s et t deux sommets.

- Supposons l'existence d'un plus court chemin de s à t.
  Soient x un sommet d'un chemin allant de s à t de poids a et c un circuit d'extrémité x de poids b. Pour conclure, il suffit de démontrer $b \geq 0$. Clairement, pour tout entier $i \geq 0$ on peut construire un chemin allant de s à t et contenant i fois le circuit c ; ce chemin a pour poids $a + b \cdot i$. Ainsi, si $b < 0$, il n'existe pas de plus court chemin. Donc, $b \geq 0$.
- supposons t accessible à partir de s et que tout circuit d'extrémité un sommet appartenant à un chemin allant de s à t est de poids non strictement nul.
  Soit WAC l'ensemble des chemins élémentaires de s à t (c'est à dire ne passant pas deux fois par le même sommet). Par hypothèse, WAC est non vide. Clairement, chaque séquence de WAC est de longueur au plus le nombre de sommets de G. Ainsi, WAC est fini. La quantité $d = min\{p(w) \mid w \in WAC\}$ est donc défini. Soit w un chemin de WAC de longueur pondérée d.
  Démontrons qu'il s'agit d'un plus court chemin. Soit u un chemin de s à t. Si u ne possède pas de cycle, u appartient à WAC et est de longueur pondérée égale à au moins celle de w. Sinon, on peut successivement enlever à u chacun de ses cycles et obtenir finalement un chemin v sans cycle de longueur pondérée inférieure ou égale à celle de u et supérieure ou égale à celle de w.

□

5.2 À source unique

Conséquence immédiate que tout chemin extrait d'un plus court chemin est aussi un plus court chemin, on peut représenter un ensemble de plus courts chemins allant d'un sommet s fixé à l'ensemble des autres sommets par une arborescence dont la racine est s, ou plus exactement par une forêt dont l'une des arborescence est de racine s et dont toutes les autres arborescences se réduisent aux sommets inaccessibles à partir de s. Représenter une telle forêt se fait à l'aide d'une fonction `père` qui associe à chaque sommet non racine de l'une des arborescences son sommet père. Ainsi, cet ensemble de plus courts chemins sera représenté par une fonction `père` qui indiquera pour chaque sommet $x \neq s$ accessible à partir de s le dernier sommet utilisé par le plus court chemin de s à ce sommet x.

5.2.1 Relâchement

Les algorithmes dans cette section utilisent le procédé dit de relâchement qui consiste d'une part à initialiser tous les sommets autre que la source à $+\infty$.

```
procédure relacherInit(G:graphe à arcs pondérés; s : sommet)
                      : (tableau V_G → ℝ,tableau V_G → V_G)

    d ← tableau indicé par V_G initialisé à +∞ ;
    père ← tableau indicé par V_G initialisé à NULL ;

    d[s] ← 0 ;

    retourner (d,père) ;
fin
```

puis selon un procédé qui reste à définir à "relâcher" des arcs, c'est à dire à réévaluer l'estimation de l'extrémité terminale v de l'arc en fonction de celle initiale u :

```
procédure relacher(u:sommet,v:sommet, G: graphe à arcs pondérés,
                   ES d: tableau V_G → ℝ, ES père: tableau V_G → V_G
                   )
    si d(v) > d(u) + p_G(u,v) ;
        d[v] ← d(u) + p_G(u,v) ;
        père[v] ← u ;
```

Propriétés

Fait 11 Soient G un graphe à arcs pondérés et s un sommet. Supposons que la fonction d n'a été modifié que par l'appel des procédures `relâcherInit` et `relâcher`. Alors pour tout sommet u de G :

- si $d[u] \neq +\infty$, alors il existe un chemin de s à u de poids $d[u]$.
- $\delta(s,u) \leq d[u]$.
- les deux valeurs à deux instants $t < t'$ de $d[u]$ vérifient : $d_{t'}[u] \leq d_t[u]$.
- si à un instant $d[u]$ atteint $\delta(s,u)$, alors cette valeur est conservée.

preuve :
Conséquence immédiate des définitions de `relâcherInit` et `relâcher`. □

Fait 12 Soient G un graphe à arcs pondérés et s un sommet. Supposons que la fonction d n'a été modifié que par l'appel des procédures `relâcherInit` et `relâcher`. Soient deux sommets u et v tels qu'il existe un plus court chemin de s à v d'avant dernier sommet u et tels qu'à une date on ait :
- $\delta(s,u) = d[u]$.
- éxécution de l'instruction `relâcher(u,v,G,d,père)`.

alors on a : $\delta(s,v) = d[v]$.

preuve :
Soit t' une date $> t$. Conséquence du fait précédent, à la date t' on a : $\delta(s,v) \leq d[v]$. Par définition de `relâcher`, à la date t' on a : $d_{t'}[v] \leq d_t[u] + p(u,v) \leq \delta(s,u) + p(u,v)$. Conséquence du Fait 9, on a : $\delta(s,u)+p(u,v) = \delta(s,v)$. Ainsi, à la date t', on a : $d[v] = \delta(s,v)$.
□

5.2.2 Bellman-Ford

Le premier algorithme étudié est un algorithme qui garantit que pour tout chemin $(s_1, \ldots, s_l)$ avec $l \leq n$ les arcs $(s_1,s_2), \ldots, (s_{l-1}, s_l)$ seront relachés dans cet ordre (avec éventuellement d'autres relachements). L'intérêt de cet algorithme est double :

1. sa définition est très simple.
2. il retourne un résultat correct pour des graphes possèdant des arcs de poids négatifs (mais sans circuit de poids < 0).
3. il détecte un cycle de poids < 0 si il en existe.

Plus formellement, la Figure 5.1 définit le problème résolu ici. L'algorithme `Bellman-Ford` est définie par la figure de même nom. Sa correction est l'objet des deux derniers faits.

Fait 13 Si G ne possède aucun circuit de poids < 0, la fonction `d` retournée est la fonction $x \rightarrow \delta_G(s,x)$.

preuve :
Soit G un graphe à arcs pondérés sans circuit de poids strictement négatif. Soient s et t deux sommets. Deux cas apparaissent :
- t n'est pas accessible à partir de s.
  Ceci peut être noté $\delta(s,t) = +\infty$. Le Fait 11 entraîne $d[t] = +\infty$.
- t est accessible à partir de s.
  D'après le Fait 10, il existe un plus court chemin $w = (s_0, \ldots, s_l)$ de $s = s_0$ à $t = s_l$.

```
àSourceUnique
Entrée : un graphe G à arcs pondérés dans ℝ, un sommet s
Sortie : un triplet (b,d,père) tel que:
             si G possède un circuit de poids négatif
                 b=faux
             sinon
                 b=vrai,
                 d est le tableau V_G → ℝ défini par d[x]=δ_G(s,x),
                 père est un tableau V_G → V_G décrivant
                     un ensemble de plus courts chemins de source s.
```

FIG. 5.1 – Le problème `àSourceUnique`

```
fonction Bellman-Ford(G : graphe à arcs pondérés, s : sommet de G)
                     :(booléen,tableau V_G → ℝ,tableau V_G → V_G)

    (d,père) ← relâcherInit(G,s) ;

    faire |V_G| − 1 fois
        pour chaque arc (u,v) de G
            relâcher(u,v,G,d,père) ;

    pour chaque arc (u,v) de G faire
        si d(v) > d(u) + p_G(u,v)
            retourner (FAUX,_,_) ;

    retourner (VRAI,d,père) ;
fin
```

FIG. 5.2 – L'algorithme `Bellman-Ford`

Démontrons par récurrence que tout entier $i \in [0,l]$ vérifie la propriété $\mathcal{P}(i)$: après la i^{e}exécution de `pour chaque arc (u,v) relâcher(u,v)` on a : $d[s_i] = \delta(s,s_i)$.
Initialement $d[s]$ est initialisé à 0. Ainsi $\mathcal{P}(0)$ est vrai.
Démontrons que pour tout $i \in [0,l-1]$ on a : $\mathcal{P}[i] \Rightarrow \mathcal{P}[i+1]$. Soit $i \in [0,l-1]$ un entier vérifiant $\mathcal{P}$. Par hypothèse, l'arc (s_i, s_{i+1}) est relâchée à une date où $d[s_i] = \delta(s,s_i)$. Le Fait 12 entraîne $d[s_{i+1}] = \delta(s,s_{i+1})$ et donc $\mathcal{P}[i+1]$.

□

Fait 14 Les deux assertions suivantes sont équivalentes :

1. tout circuit de G est de poids ≥ 0.
2. `Bellman-Ford` retourne le booléen vrai.

preuve :
Soit $G = (V, E, p)$ un graphe à arcs pondérés. Il vient :

$1 \Rightarrow 2$

Si tout circuit de G est de poids ≥ 0, d'après le Fait 13, la fonction `d` est la fonction qui associe à tout sommet x la valeur $\delta_G(s, x)$. Or cette fonction distance vérifie l'équation :

$$\forall (u, v) \in E_G \delta_G(s, v) \leq \delta_G(s, u) + p(u, v)$$

Il en est donc de même pour `d`. Ainsi, `Bellman-Ford` retourne vrai.

$2 \Rightarrow 1$

Soit $c = (s_1, \ldots, s_l)$ un cycle de G. Par hypothèse, pour tout arc (u, v) de G on a : $\texttt{d}[v] \leq \texttt{d}[u] + p(u, v)$. En appliquant cette propriété aux l arcs $(s_1, s_2), \ldots, (s_{l-1}, s_l), (s_l, s_1)$, on obtient $\sum_{i \in [1,l]} \texttt{d}[s_i] \leq \sum_{i \in [1,l]} \texttt{d}[s_{i+1}] + \sum_{i \in [1,l]} p(s_i, s_{i+1})$ où s_{l+1} désigne s_1 et donc $0 \leq \sum_{i \in [1,l]} p(s_i, s_{i+1}) = p(c)$.

□

5.2.3 Dijkstra

Le second algorithme étudié a pour principales propriétés :

1. sa correction est établie que pour des graphes sans arcs de poids < 0.
2. la simplicité de sa définition.
3. une faible complexité en temps (analysée en TD).

Plus formellemnt, le problème résolu ici est le suivant :

`àSourceUniquePoidsPositifs`
`Entrée : un graphe` G `à arcs pondérés dans` $\mathbb{R}^+$`, un sommet` s
`Sortie : un couple (d,père) tel que:`
`` $d$ `est le tableau` $V_G \to \mathbb{R}$ `défini par` $d[x] = \delta_G(s, x)$ ``
`père est un tableau` $V_G \to V_G$ `décrivant`
`un ensemble de plus courts chemins de source` s

L'algorithme `Dijkstra` est définie par la figure de même nom. Sa correction est établie par le prochain fait.

Fait 15 Si G ne possède aucun arc de poids négatif, l'algorithme de Dijkstra se termine et est correct.

preuve :
Soit $G := (V, E, p)$ un graphe à arcs pondérés ayant n sommets. Clairement, pour conclure, il nous suffit de démontrer que tout entier $i \in [1, n+1]$ vérifie la propriété $\mathcal{P}(i)$ à savoir : avant le i^{e}passage dans la boucle `tantque` tout sommet $x \notin Y$ vérifie $d[y] = \delta(s, y)$. Initialement, Y est vide. Ainsi, $\mathcal{P}(1)$ est vrai.

Démontrons que tout entier $i \in [1, n]$ vérifie $\mathcal{P}(i) \Rightarrow \mathcal{P}(i+1)$. Soit $i \in [1, n]$ un entier vérifiant $\mathcal{P}$. Soit u le sommet extrait par `Dijkstra` de Y au i^{e}passage dans la boucle `tantque`. Pour conclure, démontrons qu'à cet instant t, $d_t[u] = \delta(s, u)$. Deux cas apparaissent :

```
fonction Dijkstra(G:graphe à arcs pondérés ; s :sommet)
                 :(fonction V_G → ℝ, fonction V_G → V_G)

    (d,père) ← relâcherInit(G,s) ;

    Y ← V_G ;

    tantque Y ≠ ∅ faire
        extraire un élément u de Y de valeur d minimale ;
        pour chaque successeur v de u faire
            si v appartient à Y faire
                relâcher(u,v,G,d,père);

    retourner (d,père) ;
fin
```

FIG. 5.3 – L'algorithme `Dijkstra`

- $u = s$.
  Conséquence de $d_t[s] = 0 = \delta(s, s)$.
- $u \neq s$ et $d_t[u] = +\infty$.
  u est un sommet de valeur d minimale, ainsi tout sommet $y \in Y$ est de valeur $d + \infty$. On en conclut qu'aucun arc de G ne connecte un sommet de $V - Y$ accessible à partir de s à un sommet de Y. On en déduit qu'aucun chemin de G relie s à u.
- $u \neq s$ et $d_t[u] < +\infty$.
  Clairement u est accessible à partir de s. Conséquence du Fait 12, pour conclure il suffit de démontrer l'existence d'un plus court chemin de s à u à sommets tous dans $V - Y$ (exception faite de u). Soit w un plus court chemin de s à u. Par hypothèse, $s \neq u$. Ainsi $s \notin Y$ et $u \in Y$. Soit x le premier sommet de w qui n'appartient pas à Y mais dont le successeur y dans w appartient à Y. La sous-séquence de w allant de s à y est un plus court chemin (Fait 8) à sommets tous dans $V - Y$ sauf l'extrémité terminale y. Aucun arc du chemin w est strictement négatif : ainsi, $\delta(s, y) \leq \delta(s, u)$. Le Fait 12 entraîne $\delta(s, y) = d_t[y]$. Par définition, u est de valeur d minimale. Ainsi, $d_t[u] \leq d_t[y]$. L'inégalité $\delta(s, u) \leq d_t[u]$ du Fait 11 entraîne :

  $$d_t[y] = \delta(s, y) \leq \delta(s, u) \leq d_t[u] \leq d_t[y]$$

  On en déduit : $\delta(s, u) = d_t[u]$

□

5.3 À sources multiples

Ce problème sera étudié en TD.

Chapitre 6

Parcours en largeur

Deux façons générales de parcourir un graphe sont le parcours en largeur et le parcours en profondeur. L'importance de ces parcours est factuelle :

1. un grand nombre de problèmes sur les graphes admet pour solutions des algorithmes utilisant ces deux parcours ;
2. la complexité en temps de ces algorithmes est linéaire (ou presque).

Si dans un arbre (structure récursive par excellence), le parcours le plus simple et donc le plus naturel est celui en profondeur, dans le cas des graphes, il semble que ce soit le contraire. Nous commençons donc pas le parcours en largeur. Le parcours en profondeur sera étudié dans un prochain chapître.

6.1 Le parcours en largeur

Parcourir en largeur un arbre a pour effet de parcourir la racine, puis les sommets de hauteur 1, puis ceux de hauteur 2 et ainsi de suite.

Le parcours en largeur d'un graphe G à partir d'un sommet s est similaire. On parcourt s, puis ses voisins, puis les voisins des voisins non déjà visités et ainsi de suite.

On l'aperçoit ici. Le parcours en largeur est intimement lié au calcul des "distances" des sommets au sommet s (la distance de deux sommets s et t dans un graphe (V, E) est la longueur du plus court chemin de s à t) : dans un tel parcours, on visite le sommet s, puis les sommets à distance 1, puis ceux à distance 2 et ainsi de suite.

Pour définir algorithmiquement un tel parcours, on utilise, comme pour les arbres, une file (notée ici `GRIS`). A tout instant de l'algorithme, l'ensemble des sommets est partitionné en trois ensembles disjoints `BLANC`,`GRIS`,`NOIR` dont l'union forme l'ensemble des sommets de G et dont la sémantique est la suivante :

`BLANC` est l'ensemble des sommets non découverts.

`GRIS` est l'ensemble des sommets découverts mais dont certains des voisins n'ont pas été éventuellement découverts.

`NOIR` est l'ensemble des sommets découverts et dont tous les voisins ont été découverts.

```
procédure ParcoursLargeur(G: graphe, s : sommet)
    BLANC ← V_G\ s ;
    GRIS  ← enfiler(fileVide(),s) ;
    NOIR  ← ∅ ;

    tantque non(estVide(GRIS)) faire
        u ← premier(GRIS) ;
        pour tout sommet v successeur de u faire
            si v ∈ BLANC
                BLANC ← BLANC \ s ;
                GRIS  ← enfiler(GRIS,v) ;
        GRIS ← défiler(GRIS) ;
        NOIR ← NOIR ∪{u} ;
```

FIG. 6.1 – Parcours en largeur

L'ensemble de ces considérations induit naturellement l'algorithme suivant défini par la Figure 6.1. La première propriété de cet algorithme est qu'il parcourt l'ensemble des sommets *accessibles* à partir du sommet s (c'est à dire des sommets extrémités terminales de chemin d'extrémité initiale s) :

Fait 16 Après exécution de ParcoursLargeur, l'ensemble `NOIR` contient tous les sommets de G accessibles à partir de s.

preuve :
Conséquence du fait que :

- tout sommet possède l'une des trois couleurs blanc, gris ou noir.
- tout sommet colorié noir conserve définitivement cette couleur.
- tout sommet perdant la couleur blanche la perd définitivement.
- tout sommet n'est colorié noir qu'après que tous ses successeurs blancs aient été coloriés en gris

on prouve aisément par réccurence qu'à chaque instant du déroulement de l'algorithme aucun sommet noir ne possède un successeur blanc. Ainsi, à la fin de l'algorithme puisque tout sommet est soit noir soit blanc, tout successeur d'un sommet noir est noir. Ce qui entraîne par transivité et du fait que s n'est pas blanc et est donc noir, que tout sommet accessible à partir de s est noir. □

Fait 17 En choisissant une bonne implémentation des types ensembles `BLANC`, `GRIS`, `NOIR` et du type `Graphe`, `ParcoursLargeur` a une complexité (dans le pire des cas) en temps $\Theta(|E(G)|)$ et en espace $\Theta(|V(G)|)$.

preuve :
Soit G un graphe, n son nombre de sommets et m son nombre d'arêtes (ou d'arcs). Pour avoir une complexité en temps égale à $\Theta(1)$ pour les instructions : `enlever s à BLANC`, `v in BLANC`, `enlever v à BLANC`, `ajouter u à NOIR` il suffit de représenter les ensembles `BLANC` et `NOIR` à l'aide de deux tableaux `blan`, `noir` de booléens de taille n : décider

`v in BLANC` est réalisé en calculant `blanc[v]`, `enlever s à BLANC` est réalisé par l'affectation —blan[s]=0—... En choisissant une implémentation usuelle du type `file` (voir cours ASD du premier semestre), on obtient une complexité en temps $\Theta(1)$ pour les instructions `u <- premier(GRIS)`, `GRIS <- enfiler(GRIS,v)` et `GRIS <- défiler(GRIS)`.

En conséquence, la complexité en temps de l'instruction `pour tout sommet ...` est égal au nombre d'arêtes incidentes à u (ou d'arcs sortant de u). Observant qu'après exécution de cette instruction, le sommet u est définitivement enlevé de la file `GRIS` (il est noirci et ne peut donc être rajouter à la file), ce traitement s'applique au plus une fois pour chacun des sommets u. En conséquence, la complexité en temps de la boucle `tant que` est $O(m)$. Il est aisé de définir un graphe pour lequel chaque sommet u est placé au moins une fois dans la file, il suffit de prendre un graphe connexe dans le cas orienté (fortement connexe dans le cas non orienté) par exemple un chemin. On obtient ainsi une complexité en temps de la boucle `tant que` (et donc de l'algorithme) égale à $\Theta(m)$ (pour plus de détails voir Fait 16.

Conséquence des remarques précédentes, la taille de la file est au plus $O(n)$. Ainsi, l'espace utilisée est $\Theta(n)$, à savoir l'espace nécessaire à l'implémentation des tableaux. □

6.2 Première application : calcul de distances

Comme nous l'avons dit en introduction, cet algorithme permet de calculer les plus courts chemins dans un graphe G ayant comme extrémité initiale un sommet s. Attention : ici les arcs du graphes ne sont pas pondérés ; pour calculer les distances dans un graphe tenant compte des poids des arcs, se reporter au chapître précédent. Il permet aussi de calculer un sous-graphe partiel de G qui contient pour tout sommet accessible t à partir de s un des plus courts chemins de s à t. Pour ce faire, il suffit d'utiliser deux tableaux :

- un premier tableau, `d`, qui fournit la distance de s à t.
- un second tableau, `père`, qui fournit une arborescence des plus courts chemins.

La Figure 6.2 fournit une définition de l'algorithme du parcours en largeur.

Exemple 16 Soit G le graphe représenté sur la figure ci-dessous. Ses arêtes sont représentées par des traits en pointillés. Ses sommets sont numérotés selon l'ordre de leur visite par un parcours en profondeur. Les deux dessins représentent l'état des ensembles `BLANC`, `GRIS` et `NOIR` à deux instants successifs $t < t'$. A l'instant t, la file `GRIS` est égale à $(4, 5, 6)$ (figure de gauche). A l'instant t', une fois son premier sommet retiré et ses voisins blancs visités, `GRIS` est égale à $(5, 6, 7)$ (figure de droite). Les traits pleins représentent les arêtes du graphe de liaison.

Fait 18 Lors de l'exécution de `ParcoursLargeur2` sur un graphe G et un sommet s, la file `GRIS` est une séquence de sommets de la forme $(s_1, \ldots, s_l)$ telle que :

1. pour tout entier $i \in [1, l-1]$, on a : $d[s_i] \leq d[s_{i+1}]$.
2. $d[s_l] \leq d[s_1] + 1$.

preuve :
Démontrons qu'à tout instant la file `GRIS` $:= (s_1, \ldots, s_l)$ vérifie la propriété $\mathcal{P} := d[s_l] \leq d[s_1] + 1 \bigwedge \forall i \in [1, l-1], d[s_i] \leq d[s_{i+1}]$. La preuve s'obtient par récurrence en observant que

```
fonction ParcoursLargeur2(G: graphe, s : sommet)
                        :(tableau V_G → ℕ,tableau V_G → V_G)

    couleur ← tableau à indices V_G initialisé à blanc ;
    d       ← tableau à indices V_G initialisé à +∞ ;
    père    ← tableau à indices V_G initialisé à NULL  ;

    couleur[s] ← gris ;
    GRIS ← enfiler(fileVide(),s) ;
    d[s] ← 0 ;

    tantque non(estVide(GRIS)) faire
        u ← premier(GRIS) ;
        pour tout sommet v successeur de u faire
            si couleur(v) = blanc
                 couleur[v] ← gris ;
                 GRIS ← enfiler(GRIS,v) ;
                 d[v] ← d[u] + 1 ;
                 père[v] ← u ;
        GRIS ← défiler(GRIS) ;
        couleur[u] ← noir ;

    retourner(d,père) ;
```

FIG. 6.2 – Parcours en largeur(bis)

la file avant l'éxécution de la boucle `tantque` ne contient qu'un élément et donc vérifie $\mathcal{P}$ et en remarquant en outre que toute autre opération modifiant la file `GRIS` est soit la suppression du premier élément, opération qui préserve à l'évidence $\mathcal{P}$, soit l'ajout en fin de liste d'un élément associé selon d à la valeur $d(s_1)+1$, autre opération qui préserve clairement $\mathcal{P}$. □

Fait 19 L'algorithme `ParcoursLargeur2` est correct (c.a.d résoud le problème des plus courtes distances à source le somemt s).

preuve :
La preuve sera réalisée en TD. □

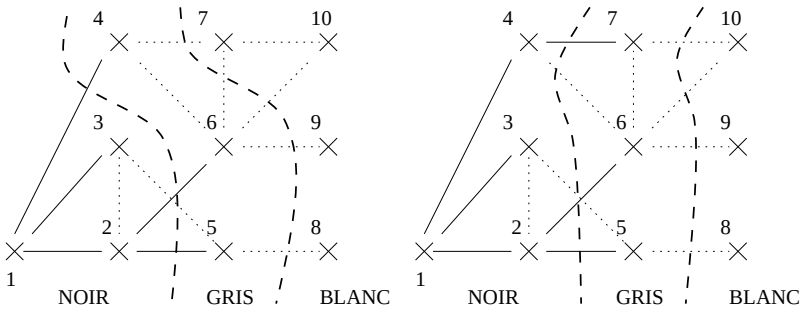

FIG. 6.3 – Parcours en largeur

Chapitre 7

Parcours en profondeur

Le nom "parcours en profondeur" exprime une opposition à celui en largeur. Dans un précédent chapitre, nous avons observé que le parcours en largeur visite un premier sommet s, puis la couche des sommets à distance 1 de s, puis celle des sommets à distance 2 et ainsi de suite. Le parcours en profondeur ne respecte pas ces couches mais les traverse : il aura tendance à s'éloigner le plus profondément possible de ce premier sommet s.

De façon identique au parcours en largeur nous utiliserons trois ensembles :

`BLANC :` l'ensemble des sommets non découverts.

`GRIS :` l'ensemble des sommets découverts mais dont certains des successeurs n'ont pas éventuellement été découverts.

`NOIR :` l'ensemble des sommets découverts dont tous les successeurs ont été découverts.

À la différence du parcours en largeur où l'on découvrait tous les successeurs blancs du sommet colorié le plus tardivement en gris (implémentation de l'ensemble `GRIS` à l'aide d'une file), lors du parcours en profondeur le sommet gris dont on découvre les successeurs blancs est le sommet le plus récemment colorié en gris : en d'autres termes, on implémente l'ensemble `GRIS` à l'aide d'une pile. Une première définition de l'algorithme de parcours en profondeur peut être celle-ci :

```
procédure visiterSommetProfondeur(G:graphe,u : sommet)

    couleur[u] ← gris ;

    pour chaque sommet v successeur de u faire
        si couleur[v]=blanc alors
            visiterSommetProfondeur(G,v);
    couleur[u] ← noir ;
```

7.1 Définition

À l'image du parcours en profondeur dans les arbres binaires (parcourir un arbre c'est parcourir son sous-arbre gauche puis son sous arbre droit ; tout est dit !), la définition la plus

simple d'un parcours en profondeur est assurément récursive. La pile `GRIS` mentionnée en introduction est alors cachée par la pile d'appel utilisée lors de ces appels récursifs. C'est cette définition que nous présenterons ici. La version itérative sera vue en TD. Cet algorithme utilise plusieurs registres globaux :

- un entier `temps` incrémenté à chacune de ses lectures (`u :=++v` signifie `v := v + 1 ; u := v`).
- ainsi que quatre tableaux `d`, `f`, `p`, `couleur` indexés par les sommets indiquant pour tout sommet x respectivement :

  `d` la date de la coloration en gris de x.

  `f` la date de la coloration en noir de x.

  `père` le sommet grâce auquel x a été découvert.

  `couleur` la couleur de x ($\in \{$`blanc`, `gris`, `noir`$\}$).

Afin de simplifier les définitions, les tableaux `couleur`, `père`, `d` et `f` ainsi que la variable temporelle `temps` sont considérés comme des variables globales. L'algorithme de parcours en profondeur a pour définition :

```
procédure parcoursProfondeur(G:graphe)
    temps ← 0 ;

    couleur ← tableau à indices V_G initialisé à blanc ;
    père    ← tableau à indices V_G initialisé à NULL  ;
    d       ← tableau à indices V_G initialisé à 0 ;
    f       ← tableau à indices V_G initialisé à 0 ;

    pour chaque sommet u faire
        si couleur(u) = blanc alors
            visiterSommetProfondeur2(G, u) ;
procédure visiterSommetProfondeur2(G:graphe,u : sommet)
début
    couleur[u] ← gris ;
    d[u] ← ++ temps ;

    pour chaque sommet v successeur de u faire
        si couleur[v]=blanc alors
            père[v] ← u ;
            visiterSommetProfondeur2(G, v);
    couleur[u] ← noir ;
    f[u] ← ++ temps ;
fin
```

Nous appellerons graphe de liaison induit par un tel parcours le sous graphe partiel de G engendré par les arcs d'extrémités de la forme (`pere`$(s), s$) où s désigne un sommet. Si il s'agit d'un graphe non orienté, il s'agit du graphe partiel engendré par les arêtes de la forme $\{$`pere`$(s), s\}$ où s désigne un sommet.

Exemple 17 Soit G le graphe représenté sur la figure ci-dessous. Ses arêtes sont représentées par des traits en pointillés. Sur la figure de gauche est représenté l'état d'avancement d'un parcours en profondeur à la date 10, sur la figure de droite l'état de ce même parcours à la date 11. Pour chaque sommet $s \in [1, 10]$ est indiqué son père, la date $d(s)$ de découverte de ce sommet, c'est à dire sa date de coloriage en gris, ainsi que la date $f(s)$ de fin de traitement de ce sommet, c'est à dire sa date de coloriage en noir. Par exemple, au sommet 9 est associé le sommet $\texttt{pere}(9) := 4$ et l'intervalle $[d(9), f(9)] := [5, 6]$.

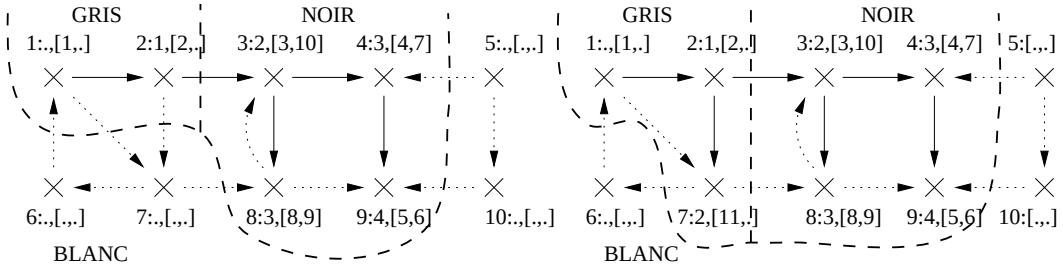

FIG. 7.1 – Parcours en profondeur aux dates 10 et 11.

7.2 Premières propriétés

Fait 20 Soient G un graphe et (s, t) un arc de G. Lors du parcours en profondeur, à aucune date les sommets s et t ne peuvent être coloriés respectivement en noir et blanc. En d'autres termes, on a : $d(t) < f(s)$.

preuve :
Conséquence immédiate de la définition de `parcoursProfondeur`. □

Fait 21 Le graphe de liaison induit par le parcours en profondeur d'un graphe est une forêt (une union disjointe d'arborescences si G est orienté et une union disjointe d'arbres si G est non orienté) qui est un graphe partiel de G.

preuve :
Considérons le graphe de liaison L induit par un parcours en profondeur dans un graphe orienté G. Conséquence immédiate de la définition du graphe de liaison, tout sommet de L possède au plus un arc entrant. De plus, un sommet s est successeur d'un sommet t si s est associé une date strictement supérieure (par la fonction d) à celle associée à t. Ce qui interdit la présence de tout cycle. Ce graphe est donc un forêt et est par construction un graphe partiel du graphe parcouru.

La preuve dans le cas non orienté est similaire. □

Fait 22 Soit G un graphe et s et t deux sommets de G. Lors de l'exécution de `parcoursProfondeur`, les deux intervalles d'entiers $I := [d(s), f(s)]$ et $J := [d(t), f(t)]$ vérifient l'une des trois assertions suivantes :

- I et J sont disjoints.
- I contient J.
- J contient I.

preuve :
Soient s et t deux sommets distincts d'un graphe G. Clairement $d(s)$ et $d(t)$ sont distincts. Quitte à renommer s et t, on peut supposer $d(s) < d(t)$. Notons $I := [d(s), f(s)]$ et $J := [d(t), f(t)]$. Plusieurs cas se présentent :

1. $f(s) < d(t)$.
   I et J sont disjoints.
2. $d(t) < f(s)$.
   Ceci signifie que t a été découvert alors que s était encore gris, le retour de l'appel de fonction `visiterSommetProfondeur2` sur l'entrée t est antérieur à celui sur l'entrée s. On en déduit $f(t) < f(s)$ et donc $I \supset J$.

□

Nous dirons qu'un sommet s est *ascendant* d'un sommet t dans une arborescence (ou dans un forêt) si il existe un chemin allant de s à t (on autorise de plus $s = t$).

Théorème 23 (Théorème des Intervalles) Soient G un graphe et s et t deux de ses sommets. Lors de l'exécution de `parcoursProfondeur` les deux assertions suivantes sont équivalentes :

1. s est un ascendant de t dans le graphe de liaison.
2. $[d(s), f(s)]$ contient $[d(t), f(t)]$.

preuve :

(1.) ⇒ (2.)
Si t est un fils de s dans le graphe de liaison, il vient clairement $d(s) < d(t)$ et $f(t) < f(s)$. Cette propriété s'étend par transitivité au cas où t est un descendant de s.

– Avant de démontrer la réciproque, démontrons préalablement que pour tous sommets s et t vérifiant $[d(s), f(s)] \supset [d(t), f(t)]$ et tels qu'aucun autre sommet u vérifie $[d(s), f(s)] \supset [d(u), f(u)] \supset [d(t), f(t)]$, s est père de t dans le graphe de liaison.

  Soient s et t deux tels sommets. Par définition de `parcoursProfondeur`, les fils $s_1, \ldots, s_l$ de s dans le graphe de liaison sont tels que :
  – $d(s_1) = d(s) + 1$.
  – $d(s_i) = f(s_{i-1}) + 1$, pour tout $i \in [2, l]$.
  – $f(s) = d(s_l) + 1$.
  Par hypothèse, $[d(t), f(t)]$ n'est strictement contenu dans aucun des intervalles de la forme $[d(s_i), f(s_i)]$ avec $i \in [1, l]$. Or nécessairement, $[d(t), f(t)]$ intersecte au moins l'un de ces intervalles. Notons s_k le sommet de plus petit indice tel que $[d(t), f(t)]$ intersecte $[d(s_k), f(s_k)]$. Conséquence du Fait 22, $[d(t), f(t)] \supseteq [d(s_k), f(s_k)]$. Ce qui nécessite $d(t) = d(s_k)$ et $s = t_k$.

(2.) $\Rightarrow$ (1.)

Soient deux sommets s et t avec $[d(s), f(s)] \supseteq [d(t), f(t)]$. Si $[d(s), f(s)] = [d(t), f(t)]$, on a $s = t$: s est un ascendant de t ! Supposons le cas contraire. Conséquence du Fait 22, il existe une suite de sommets $(s_1, \ldots, s_l)$ de longueur quelque entier $l \in \mathbb{N}^*$ avec $s_1 = s$ et $s_l = t$ tel que pour tout entier $i \in [1, l[$, on ait
– $[d(s_i), f(s_i)] \supseteq [d(s_{i+1}), f(s_{i+1})]$ et
– aucun sommet u autre que s_{i+1} vérifie : $[d(s_i), f(s_i)] \supset [d(u), f(u)] \supset [d(t_{i+1}), f(t_{i+1})]$.
Conséquence de la remarque préalable, pour tout entier $i \in [1, l-1]$ s_i est le père de s_{i+1} dans le graphe de liaison. Ainsi, $s = s_1$ est un ascendant de $s_l = t$.

□

Théorème 24 (Théorème du chemin blanc) Soient G un graphe et s et t deux de ses sommets. Lors de l'exécution de `parcoursProfondeur`, les deux assertions suivantes sont équivalentes :

1. t est un descendant de s dans le graphe liaison.
2. à la date $d(s)$ où est découvert s, il existe un chemin allant de s à t composé uniquement de sommets blancs (excepté s).

preuve :
Notons L le graphe de liaison. Soient s et t deux sommets de G.

(1.) $\Rightarrow$ (2.)

Supposons que t est un descendant de s dans L. Soit un chemin élémentaire $(s_1, s_2, \ldots, s_l)$ de s à t dans le graphe de liaison (on a : $s + s_1$ et $s_l = t$). Pour tout entier $i \in [2, l]$, s_{i-1} est père de s_i, on en déduit $d(s_{i-1}) < d(s_i$ (c.a.d s_i est blanc à la date $d(s_{i-1})$). On déduit que pour tout $i \in [2, l]$, $d(s) < d(s_i)$. En d'autres termes, le chemin $(s_1, \ldots, s_l)$ de s à t du graphe de liaison est un chemin du graphe à sommets tous blancs à la date $d(s)$.

(2.) $\Rightarrow$ (1.)

supposons l'existence d'un chemin c allant de s à t à sommets tous blancs à la date $d(s)$ (excepté s).
En d'autres termes, tout sommet u de ce chemin vérifie : $d(s) \leq d(u)$. Supposons qu'il existe un sommet v de ce chemin vérifiant $d(v) > f(s)$. De tous les sommets vérifiant $d(v) > f(s)$, choisissons celui le plus proche de s sur le chemin c. Clairement $v \neq s$. Soit u le sommet qui précède v sur le chemin c. Clairement, on a : $d(u) < f(s)$. La double inégalité $d(s) \leq d(u) < f(s)$ et le Fait 22 entraînent $f(u) \leq f(s)$ et donc $f(u) < d(v)$. Or d'après le Fait 20, l'existence d'un arc (u, v) et l'inégalité $f(u) < d(v)$ sont contradictoires.

Ainsi, tout sommet v de c vérifie : $d(v) < f(s)$ et donc $d(s) \leq d(v) < f(s)$. En particulier l'extrémité t. Le Fait 20 et le Théorème des Intervalles entraînent s ascendant de t dans le graphe de liaison.

□

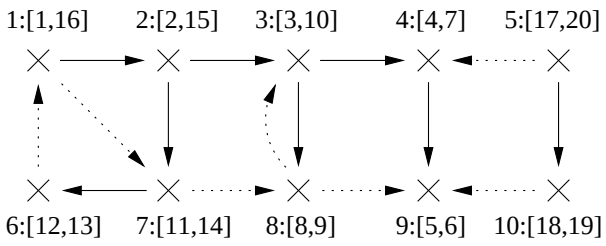

FIG. 7.2 – Parcours en profondeur

Classification des arcs

Le parcours en profondeur d'un graphe G partitionne son ensemble d'arcs en quatre ensembles ainsi définis. Notant L le graphe de liaison induit, un arc allant d'un sommet s à un sommet t est un :

arc de liaison si il appartient à L.

arc retour si s est un descendant de t dans L.

arc avant si s est un ascendant de t dans L et si l'arc n'est ni une boucle ni un arc de liaison.

arc couvrant tout autre arc.

Exemple 18 Sur la figure gauche de l'exemple 19, est représenté le graphe de liaison issu d'un parcours en profondeur. On observe que :
- les arcs de liaison sont : $(1,2), (2,3), (3,4), (4,9), (3,8), (2,7), (7,6), (5,10)$.
- les arcs de retour sont : $(6,1), (8,3)$.
- l'unique arc avant est : $(1,7)$.
- les arcs couvrant sont : $(7,8), (8,9), (5,4), (10,9)$.

Exemple 19 Sur la figure de gauche est représenté un graphe G (dont les arcs sont dessinés en pointillés ou en traits pleins) et le graphe de liaison (arcs en traits pleins) issu d'un parcours en profondeur.

7.3 Première application : reconnaissance de graphes acycliques

On s'intéresse ici à déceler la présence de cycles c'est à dire plus précisément de cycles simples ayant au moins un arc.

L'algorithme décidant si un graphe est sans cycle est une variante immédiate du parcours en profondeur. Sa correction est basée sur le fait suivant :

Fait 25 Pour tout graphe G (orienté ou non), les deux assertions suivantes sont équivalentes :

1. G contient un cycle simple.
2. tout parcours en profondeur de G induit un arc de retour.

preuve :

(2.) $\Rightarrow$ (1.)
Clairement si e est un arc retour allant de s à t, s est par définition un descendant de t dans le graphe de liaison L induit par le parcours. Il existe donc un chemin simple dans L et donc dans G allant de t à s et qui ne contient pas e. Si on le concatène au chemin (s, e, t), on obtient un cycle simple d'extrémité t.

(1.) $\Rightarrow$ (2.)
Soit $c = (s_0, e_1, \ldots, s_{l-1}, e_l, s_0)$ un cycle simple où pour tout $i \in [0, l]$ (resp. $\in [1, l]$), le terme s_i (resp. e_i) désigne un sommet (resp. un arc ou une arête) de G. Quitte à considérer un sous-cycle de c, on peut supposer c élémentaire. Soit s' le sommet du cycle découvert le plus tôt (dont la valeur $d(s)$ est minimale). Sans perte de généralité, on peut supposer $s' = s_0$. A la date $d(s_0)$, tous les autres sommets de c sont blancs, aussi d'après le Théorème du chemin blanc, s_{l-1} est un descendant de s_0 dans le graphe de liaison. L'arc e_l est donc par définition un arc de retour.

□

7.4 Deuxième application : tri topologique

Comment numéroter des sommets de façon à ce que pour tout arc (s, t), s ait un plus petit numéro que t : ce problème a pour nom le "tri topologique" et est ainsi défini :

Définition 6 Un *tri topologique* d'un graphe orienté G est une numérotation φ des sommets de G tels que pour tous sommets s et t on ait :

$$(s, t) \in E_G \Rightarrow \varphi(s) < \varphi(t)$$

Clairement tout graphe n'admet pas un tri topologique. Propriété remarquable, la possession d'un tel tri est une caractérisation de l'acyclicité, comme nous l'indiquent les deux faits suivants :

Fait 26 Pour tout graphe orienté acyclique G et pour tout parcours en profondeur de G la fonction qui à chacun des n sommets s de G associe l'entier $n - f(s)$ est un tri topologique.

preuve :
Soit G un graphe orienté acyclique, n son nombre de sommets, L le graphe de liaison généré par un parcours en profondeur et f la fonction définit lors de ce parcours. Démontrons que la fonction qui à tout sommet s associe $n - f(s)$ est un tri topologique en établissant pour tout arc de s à t l'inégalité $f(t) < f(s)$.

Soit e un arc allant de s à t. L'absence de boucle entraîne $s \neq t$ et donc $f(s) \neq f(t)$ et $d(s) \neq d(t)$. Deux cas se présentent :

1. $d(s) < d(t)$.
   D'après le théorème du chemin blanc, t est un descendant de s dans L. Le Théorème des Intervalles entraîne : $f(t) < f(s)$.

2. $d(t) < d(s)$.
   Supposer $f(s) < f(t)$ entraînerait d'après le Théorème des Intervalles que s est descendant de t dans L et donc l'existence d'un chemin de t à s dans G. Concaténé au chemin (s, e, t), on obtiendrait un cycle. Contradiction. On a donc : $f(t) \leq f(s)$.

□

Fait 27 Les deux assertions suivantes sont équivalentes :

1. G est acyclique.
2. G admet un tri topologique.

preuve :

(1.) $\Rightarrow$ (2.)
   Conséquence immédiate du Fait 26.

(2.) $\Rightarrow$ (1.)
   La possession par G d'un tri topologique f et d'un cycle simple $(s_0, e_1, \ldots, e_l, s_l)$, que l'on peut supposer sans perte de généralité élémentaire, entraîne $f(s_0) < f(s_{l-1}) < f(s_0)$ et donc $f(s_0) \neq f(s_0)$. Contradiction.

□

7.5 Troisième application : calcul des composantes fortement connexes

Le troisième problème admettant une solution par parcours en profondeur est celui du calcul des composantes fortement connexe. Calculer dans un graphe non orienté ses composantes connexes est assez immédiat : tout parcours, qu'il soit en largeur ou en profondeur, suffit. Dans le cas non orienté, ceci est plus délicat. Quand on exécute un parcours en profondeur, Il est facile d'observer que toute composante fortement connexe est contenue dans une arborescence. Inversement, chacune des arborescences n'est pas nécessairement fortement connexe (voir exemple 20).

Pour obtenir une correspondance exacte entre ces composantes et ces arborescences, on exécute la fonction récursive `visiterSommetProfondeur2` sur le graphe transposé de G et ce à partir de sommets initialement blancs particuliers distingués lors d'un premier parcours en profondeur.

Le graphe "transposé" d'un graphe orienté G est obtenu en inversant chaque arc : si G est représenté par une matrice d'adjacence M, le transposé de G est représenté par la matrice transposée de M. D'où son nom ! Une définition plus formelle suit :

Définition 7 Le *transposé* d'un graphe orienté $G = (V, E, f)$ est le graphe orienté (V, E, g) où g associe à tout arc e le couple (s, t) défini par $(t, s) = f(e)$.

Considérons l'algorithme suivant portant sur un graphe orienté G. Son fonctionnement est illustré par l'exemple qui suit. Sa correction est étudiée en TD.

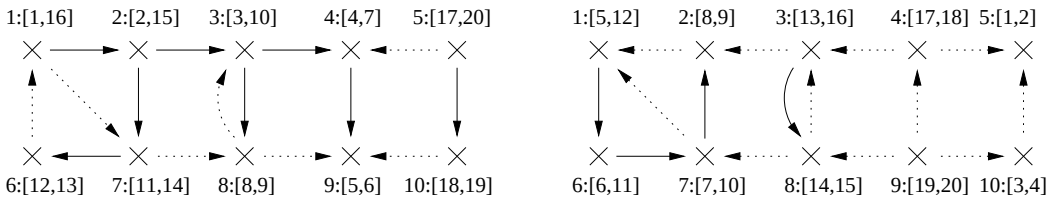

FIG. 7.3 – Calcul des composantes fortement connexes

```
fonction composantesFrtmtConnexes(G:graphe):ensemble de parties de V_G
début
    exécuter parcoursProfondeur sur G
        et mémoriser la fonction temps f: V_G → ℕ

    calculer le graphe transposé ^tG de G ;

    exécuter parcoursProfondeur sur ^tG
        en choisissant dans la boucle principale
        un nouveau sommet u blanc de valeur f maximale ;

    retourner l'ensemble des arborescences obtenues lors
        du dernier parcours en profondeur ;
fin
```

Exemple 20 Sur la figure de gauche est représenté un graphe G (dont les arcs sont dessinés en pointillés ou en traits pleins) et le graphe de liaison (arcs en traits pleins) issu d'un parcours en profondeur. Sur la figure de gauche est représenté le graphe transposé de G et le graphe de liaison (arcs en traits pleins) issu d'un second parcours en profondeur. Lors de ce second parcours, la condition de l'algorithme `composantesFrtmtConnexes` portant sur le choix d'une nouvelle racine d'une nouvelle arborescence est respectée. Chacune des arborescences est une composante fortement connexe de G (les composantes fortement connexes de G sont $\{1,2,6,7\}, \{3,8\}, \{4\}, \{9\}, \{5\}, \{10\}$).

Le rôle du premier parcours en profondeur est de singulariser dans chaque composante connexe de G un sommet : l'unique sommet x à vérifier l'équation : $x = \phi(x)$. Définissons dès à présent la fonction ϕ :

Notation 8 Soit G un graphe et f la fonction temps induite par un parcours en profondeur de G. Pour tout sommet x de G, nous notons $\phi(x)$ le sommet de G accessible à partir de x et de valeur f maximale.

Fait 28 Reprenant les notations précédentes, pour tous sommets x et y de G, les deux assertions suivantes sont équivalentes :

1. x et y appartiennent à la même composante fortement connexe.
2. $\phi(x) = \phi(y)$.

preuve :
La preuve sera réalisée en TD. □

Fait 29 Chacune des arborescences retournées par `composantesFtmtConnexes` sur une entrée G est une composante fortement connexe de G et a pour racine un sommet invariant par ϕ.

preuve :
La preuve sera réalisée en TD. □

Chapitre 8

Le problème du flot maximal

Un système dans lequel un matériau s'écoule, tel l'eau ou l'électricité, peut être modélisé à l'aide d'un graphe. Une question naturelle se pose : quelle est la capacité maximale de ce système ?

Ce problème est connu sous le nom de flot maximal et admet plusieurs solutions algorithmiques efficaces. Nous en présenterons ici quelques unes.

Les graphes considérés ici, sont sauf mention contraire, simples et orientés.

8.1 Définitions

Définition 9 Un *réseau* est un 5-uplet $G = (V, E, c, s, t)$ noté $(V_G, E_G, c_G, s_G, t_G)$ où :
- (V, E) est un graphe simple orienté où E est supposé être $E = V \times V$.
- c associe à chaque arc e sa *capacité*, c.a.d un réel positif ou nul noté $c(e)$.
- s est un sommet appelé la *source*.
- t est un sommet distinct de s appelé le *puits*.

Un *flot* de G est une fonction qui associe à chaque arc $(u, v) \in E$ un réel appelé le flot réel de u à v et tel que :

contrainte de capacité : pour tout arc $(u, v) \in E$, on a : $f(u, v) \leq c(u, v)$.

symétrie : pour tout arc $(u, v) \in E$, on a : $f(u, v) = -f(v, u)$.

conservation du flot : tout sommet $u \in V - \{s, t\}$ vérifie : $\sum_{v \in V} f(u, v) = 0$.

La *valeur* du flot f, notée $|f|$, est la quantité $\sum_{v \in V} f(s, v)$.

Le problème du flot maximal consiste à calculer pour tout réseau un flot de valeur maximale.

Dans la suite de ce chapitre, nous noterons *flot net positif d'un sommet* u la somme des flots nets positifs sortant de u c'est à dire plus formellement : $\sum_{v \in V, f(u,v)>0} f(u, v)$.

Afin d'établir la correction des algorithmes résolvant le problème du flot maximal, nous établissons ci-dessous un lemme portant sur le flot entre deux ensembles de sommets X et Y. Cette quantité définie pour tout couple d'ensembles de sommets X et Y est notée $f(X, Y)$ et est égale à :

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)$$

Lemme 30 Soit f un flot sur un réseau (V, E, c, s, t). Pour toutes parties de sommets X, Y et Z on a :

1. $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ si $X \cap Y = \emptyset$.
2. $f(X, Y \cup Z) = f(X, Y) + f(X, Z)$ si $Y \cap Z = \emptyset$.
3. $f(X, Y) = -f(Y, X)$.
4. $f(X, X) = 0$.

preuve :
Soient f un flot défini sur un réseau et trois parties de sommets X, Y et Z.

- $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ si $X \cap Y = \emptyset$.
  Conséquence immédiate de la définition de $f(X, Y)$.
- $f(X, Y \cup Z) = f(X, Y) + f(X, Z)$ si $Y \cap Z = \emptyset$.
  Conséquence immédiate de la définition et de l'égalité $\sum_{x \in X} \sum_{y \in Y \cup Z} f(x, y) = \sum_{y \in Y \cup Z} \sum_{x \in X} f(x, y)$.
- $f(X, Y) = -f(Y, X)$.
  Conséquence des deux propriétés précédentes et de l'égalité $f(u, v) = -f(v, u)$ trivialement vérifiée par tout couple $(u, v) \in V^2$.
- $f(X, X) = 0$.
  Conséquence de la propriété précédente, on a $f(X, X) = -f(X, X)$ et donc $f(X, X) = 0$.

□

8.2 Propriétés

8.2.1 Réseau résiduel

La propriété qui suit indique que tout flot de valeur non nulle est, si il n'est pas maximal, un début de réponse. En effet on peut définir à partir de ce flot f défini sur un réseau G un nouveau réseau G' "plus simple" pour lequel tout flot maximal f' permettra de définir le flot maximal $f' + f$ sur G.

Définition 10 La *capacité résiduelle* d'un réseau (V, E, c, s, t) induit par un flot f est la fonction notée c_f qui associe à tout arc $(u, v) \in E$ le réel positif ou nul $c(u, v) - f(u, v)$. Le *réseau résiduel* d'un réseau (V, E, c, s, t) induit par un flot f est le réseau (V, E, c_f, s, t).

Lemme 31

Si f est un flot sur un réseau G et

si g est un flot sur le réseau résiduel de G induit par f,

alors $f + g$ est un flot de G de valeur $|f + g| = |f| + |g|$.

preuve :
Soient f un flot sur un réseau $G = (V, E, c, s, t)$ et g un flot sur le réseau résiduel de G induit par f. Démontrons que la fonction $h := f + g$ est un flot de G de valeur $|f| + |g|$:

- h vérifie la contrainte de capacité.
  Par définition, pour tout arc $e \in E$ on a : $c_f(e) = c(e) - f(e)$ et $g(e) \leq c_f(e)$ et donc $h(e) = f(e) + g(e) \leq f(e) + (c(e) - f(e)) \leq c(e)$.
- h vérifie la symétrie.
  La somme de deux fonctions symétriques est clairement symétrique.
- h conserve le flot.
  Soit un sommet u autre que la source et le puits de G. La quantité $\sum_{v \in V} h(u, v)$ est égale à $\sum_{v \in V} f(u, v) + \sum_{v \in V} g(u, v) = 0 + 0 = 0$.
- la valeur de h est $|f| + |g|$. Par définition, $|h|$ est égale à $\sum_{v \in V} g(s, v)$ c'est à dire $\sum_{v \in V} f(s, v) + \sum_{v \in V} g(s, v) = |f| + |g|$.

□

8.2.2 Chemin améliorant

Définir un flot peut se réaliser simplement en choisissant dans le réseau un chemin de s à t et en prenant pour valeur la capacité minimale des arcs de ce chemin.

Définition 11 Soit $G = (V, E, c, s, t)$ un réseau et p un chemin élémentaire dans G de s à t. La *capacité* de p est le minimum des capacités des arcs que possède p et est noté $c(p)$. Le *flot induit par p* est la fonction notée f_p qui associe à tout arc $(u, v) \in V^2$ la quantité définie par :

- $c(p)$ si (u, v) appartient à p.
- $-c(p)$ si (v, u) appartient à p.
- 0 sinon.

Un chemin p allant de s à t *améliore* un flot f de G si la capacité de p dans le réseau résiduel de G induit par f est > 0.

Lemme 32 La fonction f_p induit par un chemin élémentaire p de la source au puits dans un réseau est un flot de valeur $c(p)$.

preuve :
Soient $G = (V, E, c, s, t)$ un réseau et p un chemin élémentaire de s à t. Observons tout d'abord le bien fondé de la définition précédente en remarquant que puisque s et t sont distincts et puisque p est élémentaire, deux occurrences de deux sommets quelconques de p sont nécessairement distincts et donc que deux arcs de la forme (u, v) et (v, u) ne peuvent pas apparaître simultanément dans p. Il vient :

- f_p vérifie la contrainte de capacité.
  Trivialement le flot réel de tout arc est inférieur à sa capacité.
- f_p vérifie la symétrie.
  Conséquence triviale de la définition.
- f_p conserve le flot.
  Soit u un sommet de $V - \{s, t\}$. Si u n'appartient pas à p, tout arc incident à u a un flot nul. La somme de ces flots est donc nulle. Sinon, u est nécessairement un sommet interne de p. Ainsi, il existe deux arcs de la forme (x, u) et (u, y) appartenant à p. Tout autre arc incident à u est de flot nul. On en déduit : $\sum_{v \in V} f_p(u, v)) = f_p(u, x) + f_p(u, y) = -c(p) + c(p) = 0$.

– la valeur de f_p est $c(p)$.
L'unique arc de p incident à s est de la forme (s, u) avec $u \in V$. Ainsi, $|f_p| = f_p(s, u) = c(p)$.

□

8.2.3 MaxiFlot & MiniCoupe

Nous établissons ici un joli "minimax theorem" qui caractérise un entier maximal à vérifier une certaine propriété (ici le flot maximum) comme étant minimal à en vérifier une seconde (ici la capacité minimale des coupes). Ce genre de résultat est à remarquer car il augure souvent favorablement la possibilité de calculer efficacement un tel entier. Ce que nous démontrerons dans la section suivante.

Définition 12 Une *coupe* dans un réseau $G = (V, E, c, s, t)$ est un couple d'ensembles de sommets de la forme $(X, V - X)$ avec $X \subseteq V$ tel que $s \in X$ et $t \in Y$. Sa *capacité*, notée $c(X, Y)$, est la somme $\sum_{x \in X, y \in Y} c(x, y)$. Une coupe est *minimale* si sa capacité est au plus égale à la capacité de toute coupe de ce réseau. Le flot d'une coupe (X, Y) relativement à un flot f est la quantité $f(X, Y)$.

Lemme 33 Tout flot f et toute coupe (X, Y) d'un même réseau de capacité c vérifient :

$$|f| = f(X, Y) \leq c(X, Y)$$

preuve :
Soient f un flot et une coupe $(X, V - X)$ dans un réseau $G = (V, E, c, s, t)$. L'inégalité $f(X, Y) \leq c(X, Y)$ est une conséquence de l'inégalité $f(e) \leq c(e)$ vérifiée par tout arc e. D'après le Lemme 30, $f(X, V - X)$ est égal successivement à :
– $f(X, V) - f(X, X)$
– $f(\{s\}, V) + f(X \backslash s, V)$
– $f(\{s\}, V) + \sum_{x \in X \backslash s} f(x, V) = |f| + \sum_{x \in X \backslash x} 0 = |f|$.

□

Théorème 34 Soit f un flot dans un réseau G. Les quatre assertions suivantes sont équivalentes :

1. f est un flot maximal.
2. f n'admet aucun chemin améliorant.
3. il existe une coupe dans le réseau résiduel induit par f de capacité nulle.
4. il existe une coupe (X, Y) de capacité $c_G(X, Y)$ égale au flot $f(X, Y)$.

preuve :
Soient $G = (V, E, c, s, t)$ un réseau, f un flot et H le réseau résiduel de G et f. Il vient :

4. $\Leftrightarrow$ 3. Conséquence immédiate de la définition du réseau résiduel de G H induit par f, pour toute coupe (X, Y) de G et donc de H, on a : $c_H(X, Y) = c_G(X, Y) - f(X, Y)$. Ce qui suffit à conclure.

3. $\Rightarrow$ 1. Du simple fait que tout arc (u, v) a un flot $f(u, v)$ au plus égal à sa capacité $c(u, v)$. On en déduit que toute coupe a un flot au plus égal à sa capacité. Or tout flot a une valeur égale au flot d'une quelconque coupe (Lemme 33). Ainsi, la valeur de tout flot est au plus la capacité d'une quelconque des coupes. En d'autres termes si un flot f et une coupe (X, Y) sont tels que $|f| = c(X, Y)$, alors f est un flot maximal.

1 $\Rightarrow$ 2. Si p est un chemin améliorant du flot f, le Lemme 32 indique que le flot f_p est de valeur strictement positive. Le Lemme 31 indique que $f + f_p$ est un flot de G de valeur $|f| + |f_p|$ strictement supérieur à celle de f. et donc, que f n'est pas maximal. Ainsi, si f est maximal, il n'est amélioré par aucun chemin.

2 $\Rightarrow$ 3. Supposons que f n'admet aucun chemin améliorant. Soit X l'ensemble des sommets de G accessibles à partir de s en utilisant des chemins à arcs de capacité résiduelle $c_H(u, v) > 0$. Conséquence de la définition d'un chemin améliorant le puits t n'appartient pas à X. De plus tout arc $(x, y) \in X \times V - X$ est de capacité résiduelle nulle c'est à dire vérifie $f(x, y) = c_H(x, y)$. Ainsi, la coupe (X, Y) a une capacité nulle dans H (on a : $c_H(X, Y) = 0$).

□

Une formulation plus ramassée du précédent théorème est la suivante :

Corollaire 35 (MaxiFlot & MiniCoupe) Pour tout réseau, la valeur maximale des flots est égale à la capacité minimale des coupes.

Exemple 21 Sur la figure 8.1, sont représentés un réseau R (dessin d'en haut), deux flots (dessins du milieu) et deux réseaux résiduels (dessins d'en bas). Sur chacun des réseaux, les arcs de capacité 0 ne sont pas représentés. De façon analogue, pour chacun des flots, les arcs de flot 0 ne sont pas représentés. Le flot de gauche fg est induit par le chemin $(a, b, d, ef,)$ et a pour valeur 2. Le flot de droite fd est maximal et a pour valeur 5. En bas de la figure, sont dessinés les deux réseaux résiduels Rg et Rd de R induits respectivement par le flot de gauche et le flot de droite. On observe que Rg contient un chemin à capacité > 0 de la source a au puits t : fg n ést donc pas maximal. Alors que Rd ne contient pas un tel chemin : fd est donc maximal. On observe en outre que l'une des coupes de capacité minimale dans R (resp. Rd) est $(\{a, b, d\}, \{c, e, f\})$: sa capacité est 5 (resp. 0).

8.3 Deux solutions au problème du flot maximal

8.3.1 Ford Fulkerson

Dans cette section, nous présenterons au problème `flotMaximal` une solution algorithmique qui a pour nom `FordFulkerson` et qui est d'efini sur la Figure 8.2.

Corollaire 36 L'algorithme `FordFulkerson` est correct.

preuve :
Conséquence immédiate du Théorème 34. □

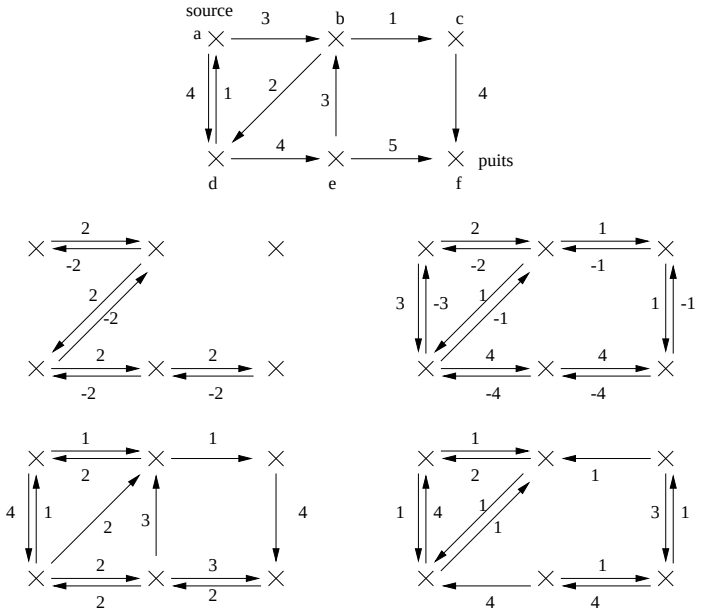

FIG. 8.1 – Un réseau, des flots et des réseaux résiduels

```
fonction FordFulkersonRec(G = (V, E, c, s, t) : réseau) : flot ;

    si il n'existe aucun chemin de s à t de capacité > 0 faire
        retourner 0 ;              %0 représente le flot nul
    sinon
        calculer un chemin p élémentaire de s à t de capacité > 0;
        f ← flotInduit(G,p) ;
        H ← réseauRésiduel(G ,f) ;
        retourner f + FordFulkersonRec(H) ;
```

FIG. 8.2 – l'Algorithme `FordFulkerson`

remarque 1 Attention ! Si l'algorithme est prouvé correct, cette correction n'est en un sens que partielle. L'algorithme peut ne pas terminer ! Voir l'Exemple 22.

Cependant, il est facile de démontrer que pour les réseaux à capacités entières ou rationnelles le programme termine. Cependant si il termine, ce même algorithme sur des réseaux à capacités entières a une complexité en temps exponentielle. Voir l'Exemple 22.

Exemple 22 Pour chaque quadruplet de réels positifs (c_1, c_2, c_3, c_4) notons $R(c_1, c_2, c_3, c_4)$ le réseau défini par l'ensemble des sommets $V := \{g0, g1, g2, g3, g4, d1, d2, d3, d4, d0\}$, par la source $g0$, par le puits $d0$ et par la fonction de capacité c qui associe à chaque arc $e \in V^2$ le réel :

- respectivement égal à c_1, c_2, c_3 ou c_4 si l'arc est respectivement égal à $(g1, d1)$, $(g2, d2)$, $(g3, d3)$ ou à $(g4, d4)$.
- $+\infty$ si l'arc a ses deux extrémités dans $\{g0, g1, g2, g3, g4\}$ ou ses deux extrémités dans $\{d0, d1, d2, d3, d4\}$.

– 0 à tout autre arc.

Il est aisé d'observer qu'un tel réseau $R(c_1, c_2, c_3, c_4)$ a pour flot maximal un flot de valeur $c_1 + \ldots + c_4$.

Dans un tel réseau tout chemin élémentaire de la source au puits de capacité non nulle non nulle peut se définir (à une équivalence près) par la donnée des arcs appartenant à ce chemin et ayant une extrémité dans $\{g1, g2, g3, g4\}$ et une seconde extrémité dans $\{d1, d2, d3, d4\}$, c'est à dire par la donnée de quatre valeurs (p_1, p_2, p_3, p_4) dans $\{1, 0, 1\}$ à somme égale à 1 (Pour chaque entier $i \in \{1, 2, 3, 4\}$, -1 indique que l'arc (di, gi) appartient au chemin p, 1 indique que l'arc (gi, di) appartient au chemin p et 0 indique que ni l'arc (di, gi) ni l'arc (gi, di) n'appartient au chemin p). Un tel chemin est noté $P(p_1, p_2, p_3, p_4)$.

Il est aisé d'observer que tout chemin $P(p_1, p_2, p_3, p_4)$ de la source au puits a d'une part pour capacité dans le réseau $R(c_1, c_2, c_3, c_4)$ le minimum m des valeurs positives parmi $p_1 \cdot c_1, \ldots, p_4 \cdot c_4$ et d'autre part qu'il induit comme réseau résiduel le réseau $R(c_1 - p_1 \cdot m, \ldots, c_4 - p_4 \cdot m)$. Pour simplifier certaines notations, le réseau résiduel induit par un réseau $R(\ldots)$ et un chemin $P(\ldots)$ sera noté $R(\ldots) : P(\ldots)$.

Ainsi si l'on considère l'unique réel $\varphi > 1$ à vérifier l'équation de Fibonacci $\varphi - 1 = \frac{1}{\varphi}$ et si l'on considère le réseau $\mathbf{R} := R(0, 1, \frac{1}{\varphi}, \frac{1}{\varphi^2})$, on obtient les égalités suivantes :

– $R(0, 1, \frac{1}{\varphi}, \frac{1}{\varphi^2}) : (-1, 1, 1, 0) = R(\frac{1}{\varphi}, \frac{1}{\varphi^2}, 0, \frac{1}{\varphi^2})$.

– $R(\frac{1}{\varphi}, \frac{1}{\varphi^2}, 0, \frac{1}{\varphi^2}) : (1, 1, -1, 0) = R(\frac{1}{\varphi^3}, 0, \frac{1}{\varphi^2}, \frac{1}{\varphi^2})$.

– $R(\frac{1}{\varphi^3}, 0, \frac{1}{\varphi^2}, \frac{1}{\varphi^2}) : (1, -1, 1, 0) = R(0, \frac{1}{\varphi^3}, \frac{1}{\varphi^4}, \frac{1}{\varphi^2})$ isomorphe à $\frac{1}{\varphi^2} \cdot \mathbf{R}$.

  Ici, le produit scalaire associe à tout scalaire $\alpha > 0$ et à tout réseau $R(c_1, \ldots, c_4)$ le réseau $R(\alpha \cdot c_1, \ldots, \alpha \cdot c_4)$. Pour l'isorphisme de $\frac{1}{\varphi^2} \cdot \mathbf{R}$ il est facile d'observer que deux réseaux de la forme $R(a, b, c, d)$ et $R(A, B, C, D)$ sont isomorphes si et seulement si les multi ensembles $[a, b, c, d]$ et $[A, B, C, D]$ sont égaux (un multiensemble est un ensemble où l'on compte le nombre d'apparition des éléments).

Conséquence de l'isomorphisme de $(((\mathbf{R} : (1, 1, -1, 0)) : (1, -1, 1, 0)) : (1, -1, 1, 0)$ et de $\frac{1}{\varphi^2} \cdot \mathbf{R}$ et du fait que pour tout scalaire $\alpha > 0$, tout réseau R et tout chemin p on a :

$$(\alpha \cdot R) : p = \alpha \cdot (R : p)$$

on en déduit que tout réseau de la forme $\frac{1}{\varphi^i} \cdot \mathbf{R}$ avec i entier naturel peut se transformer en trois passages de l'algorithme en le réseau $\frac{1}{\varphi^{i+2}} \cdot \mathbf{R}$. Ainsi, une éxécution de `FordFulkerson` peut produire l'ensemble infini de tous les réseaux de la forme $(\frac{1}{\varphi^2})^i \cdot \mathbf{R}$ avec $i \geq 0$. En conclusion, `FordFulkerson` ne termine pas.

remarque 2 Il existe plusieurs façons équivalentes d'écrire de façon itérative l'algorithme `FordFulkerson`. La Figure 8.3 en définit une.

8.3.2 Une amélioration de Ford Fulkerson qui termine

Dans la définition de `FordFulkerson`, aucune précision n'a été indiqué sur la façon de calculer un chemin de s à t de capacité non nul. Cette question est pourtant cruciale : sur des graphes à valeurs réelles, l'algorithme ne termine pas.

```
fonction FordFulkerson(G = (V, E, c, s, t) : réseau) : flot ;

    fmax ← 0 ;

    tantque il existe un chemin de s à t de capacité > 0 faire
        calculer un chemin p élémentaire de s à t de capacité > 0;
        f ← flotInduit(G,p) ;
        G ← réseauRésiduel(G ,f) ;
        fmax ← fmax + f ;

    retourner fmax ;
```

FIG. 8.3 – l'Algorithme FordFulkerson en version itérative

Cette grande faiblesse de `FordFulkerson` peut être facilement corrigée : en effet, si l'on utilise un simple parcours en largeur à partir de s pour calculer un plus court chemin de s à t (sans considérer les étiquettes des arcs), on obtient un algorithme qui termine et donc de bien meilleure complexité en temps dans le pire des cas.

Cet algorithme est le suivant :

```
fonction EdmondKarpsRec(G = (V, E, c, s, t) : réseau) : flot ;
    si il n'existe aucun chemin de s à t de capacité > 0 faire
        retourner 0 ;              %0 représente le flot nul
    sinon
        calculer un plus court chemin p de s à t de capacité > 0;
        f ← flotInduit(G,p) ;
        H ← réseauRésiduel(G ,f) ;
        retourner f + EdmondKarpsRec(H) ;
fin
```

Lors des deux prochains faits, pour tout réseau $G = (V, E, c, s, t)$, nous noterons $G_{\neq 0}$ le graphe orienté obtenu à partir de (V, E) en supprimant tout arc de capacité nulle et conformément à des notations présentées antérieurement $\delta_H(s, t)$ désignera la longueur d'un plus court chemin de G allant de s à t ou l'élément $+\infty$ si il n'existe pas de chemin de s à t.

Fait 37 Soient G un réseau, s le sommet source de G, p un plus court chemin de s à t de capacité non nulle et H le réseau résiduel de G induit par le flot induit par p. Tout sommet x de V vérifie : $\delta_{G_{\neq 0}}(s, x) \leq \delta_{H_{\neq 0}}(s, x)$.

preuve :
Soient G un réseau, s le sommet source de G, p un plus court chemin de s à t de capacité non nulle et H le réseau résiduel de G induit par le flot induit par p. Afin de simplifier les notations, $\delta_{G_{\neq 0}}$ sera noté δ_G, $\delta_{H_{\neq 0}}$ sera noté δ_H.

Supposons qu'il existe un sommet x_0 tel que : $\delta_G(s, x_0) > \delta_H(s, x_0)$. Parmi tous ceux vérifiant cette inéquation, choisissons-en un de valeur $\delta_H(s, x_0)$ minimale. Ainsi :

$$\forall y \in V \delta_H(s, y) < \delta_G(s, y) \Rightarrow \delta_H(s, x_0) \leq \delta_H(s, y) \tag{8.1}$$

La stricte inégalité $\delta_H(s,x_0) < \delta_G(s,x_0)$ entraine $\delta_H(s,x_0) \neq +\infty$. On en déduit l'existence d'un chemin dans H de s à x_0. La claire équivalence $\delta_H(s,x) = 0 \Leftrightarrow s = x$ pour tout sommet $x \in V$ entraine $x_0 \neq s$. Ainsi, il existe un plus court chemin p dans H de s à x_0 de longueur non nulle. Soit u le sommet qui précède x_0 sur ce chemin p. Le chemin de s à u extrait de p est un plus court chemin de longueur $\delta_H(s,x_0) - 1$. On en déduit :

$$\delta_H(s,u) = \delta_H(s,x_0) - 1 \tag{8.2}$$

et donc d'après l'équation 8.1 :

$$\delta_G(s,u) \leq \delta_H(s,u) \tag{8.3}$$

Supposer que l'arc (u,x_0) soit de capacité non nulle dans G entraine $(u,x_0) \in E(G_{\neq 0})$, puis $\delta_G(s,x_0) \leq \delta_G(s,u) + 1$ et donc d'après l'égalité 8.2 et l'inégalité 8.3 ceci entraine : $\delta_G(s,x_0) \leq \delta_H(s,x_0)$. Contradiction. On en déduit donc :

$$c_G(u,x_0) = 0 \tag{8.4}$$

L'égalité ci-dessus et l'inégalité $c_H(u,x_0) \neq 0$ nécessite que (x_0,u) appartienne au plus court chemin p dans G de s à t. On en déduit :

$$\delta_G(s,u) = \delta_G(s,x_0) - 1 \tag{8.5}$$

et donc $\delta_H(s,x_0) = \delta_H(s,u)+1 \geq \delta_G(s,u)+1 > \delta_G(s,x_0)$. Contradiction. Ainsi, tout sommet $x \in V$ vérifie $\delta_G(s,x) \leq \delta_H(s,x)$. □

Soient un réseau G et un chemin p de la source au puits. Un arc e est dit *critique* relativement à G et p si il appartient au chemin et si sa capacité est celle du chemin.

Fait 38 Si p est un plus court chemin dans un réseau G d'extrémité initiale s et de capacité > 0 et si (u,v) est un arc de p, alors on a :

$$\delta_{G\neq 0}(s,u) \in [0,|V(G)|-1], \delta_{G\neq 0}(s,v) \in [0,|V(G)|-1], \delta_{G\neq 0}(s,v) = \delta_{G\neq 0}(s,u) + 1$$

preuve :
Simple conséquence du fait que le chemin extrait de p allant de s à u (resp. v) est un plus court chemin de $G_{\neq 0}$ et donc a pour longueur $\delta_{G_{\neq 0}}(s,u) \in [0,|V(G)|-1]$ (resp. $\delta_{G_{\neq 0}}(s,v) \in [0,|V(G)|-1]$). □

Fait 39 Lors de l'éxécution de `EdmondKarpsRec` sur une entrée G, tout arc (u,v) est au plus $\frac{n}{2}$ fois critique où n désigne le nombre de sommets de G.

preuve :
Soit $G := (V,E,c,s,t)$ un réseau ayant n sommets. Notons $p_1,\ldots,p_l$ les différents plus courts chemins calculés lors des différents appels récursifs de `EdmondKarpsRec` et $G_1,\ldots,G_{l+1}$ les différents réseaux calculés lors de ces mêmes appels : on a $G = G_1$ et pour tout $i \in [1,l-1]$, G_{i+1} est le réseau résiduel de G_i et du flot induit par p_i.

Soit $(u,v) \in V^2$ un couple de sommets. Soit I l'ensemble des indices i pour lesquels (u,v) est critique relativement à (G_i,p_i). Pour conclure et démontrer l'inégalité $|I| \leq \frac{n}{2}$, il

nous suffit en conséquence de l'appartenance $\delta_{G_i}(s,v) \in [1,n]$ pour tout $i \in I$ (Fait 38) de démontrer pour tous indices $i_0 < j_0$ appartenant à I l'inégalité :

$$\delta_{G_{i_0}}(s,v) + 2 \leq \delta_{G_{j_0}}(s,v) \tag{8.6}$$

Soient deux indices $i_0 < j_0$ appartenant à I. Conséquence du Fait 38, il vient :

$$\delta_{G_{i_0}}(s,v) = \delta_{G_{i_0}}(s,u) + 1 \tag{8.7}$$

$$\delta_{G_{j_0}}(s,v) = \delta_{G_{j_0}}(s,u) + 1 \tag{8.8}$$

Dans le réseau, G_{i_0+1} l'arc (u,v) a une capacité nulle, or dans le réseau G_{j_0}, ce même arc a une capacité non nulle. Soit k_0 le plus petit entier $> i_0$ pour lequel (u,v) a une capacité non nulle dans G_{k_0+1}. L'arc (u,v) de capacité nulle dans G_{k_0} ne peut appartenir à p_{k_0}. Nécessairement, l'arc (v,u) appartient à p_{k_0}, qui est par construction un plus court chemin de G_{k_0} de s à t. On en déduit :

$$\delta_{G_{k_0}}(s,u) = \delta_{G_{k_0}}(s,v) + 1 \tag{8.9}$$

Les inégalités $\delta_{G_{i_0}}(s,u) \leq \delta_{G_{k_0}}(s,u) \leq \delta_{G_{j_0}}(s,u)$, $\delta_{G_{i_0}}(s,v) \leq \delta_{G_{k_0}}(s,v) \leq \delta_{G_{j_0}}(s,v)$ (Fait 37) et les inégalités 8.7, 8.8 et 8.9 entrainent l'inégalité 8.6. □

Corollaire 40 Le nombre d'appels récursifs de `EdmondKarpsRec` sur une entrée G est au plus $n \cdot m$ où n désigne le nombre de sommets de G et m le nombre d'arcs de capacité $\neq 0$.

preuve :
Conséquence directe du fait précédent, du fait que lors de tout appel récursif nécessite au moins un arc critique (ceux qui appartiennent au chemin et qui ont pour capacité la capacité du chemin) et que tout arc critique (u,v) lors d'un des appels est initialement ou de capacité non nulle ou son arc opposé est de capacité non nulle (le nombre de tels arcs étant au plus $m \cdot 2$). □

Lemme 41 L'algorithme `EdmondKarpsRec` est correct et a pour complexité en temps dans le pire des cas $O(n \cdot m^2)$ où m est le nombre d'arcs à capacité non nulle du réseau et n son nombre de sommets.

preuve :
La correction de l'algorithme `EdmondKarpsRec` est une conséquence de la correction de `FordFulkersonRec` et du fait que tout réseau admet un chemin de s à t de capacité > 0 si et seulement il admet un plus court chemin de s à t de capacité > 0 !

Pour obtenir une complexité en $O(n \cdot m^2)$ avec m le nombre d'arcs à capacité non nulle, nous ne pouvons pas représenter chacun des arcs de E qui par définition est V^2 et est donc de taille n^2. Aussi, pour chaque réseau G nous ne considérerons que les arcs (x,y) vérifiant $c(x,y) \neq 0 \wedge c(y,x) \neq 0$ et nous noterons m_G leur cardinalité. Il est facile d'observer que tout réseau résiduel H d'un réseau G a même nombre de sommets et vérifie : $m_H \leq m_G$ (en fait on a : $m_H = m_G$) et donc est moins complexe en taille que G.

Il est aisé de choisir une bonne structure de données permettant d'obtenir une complexité dans le pire des cas de chaque instruction de `EdmondKarpsRec` (autre que l'appel récursif !) en $O(m)$:

1. l'extraction d'un plus court chemin de s à t se fait par un parcours en largeur dont la complexité en temps est $O(m)$. Il est suffisant pour cela de représenter le graphe par un tableau de listes de successeurs.
2. le calcul du réseau résiduel d'un réseau et d'un flot-chemin se fait en $O(m)$. Il est suffisant pour cela d'avoir une structure de données permettant de modifier la capacité de tout arc en temps constant.

Conséquence du Fait 40, le nombre d'appels récursifs est au plus $n \cdot m$, la complexité en temps dans le pire des cas est $O(n \cdot m^2)$. □

8.4 Théorèmes de Menger

Nous conclurons cette section en présentant trois théorèmes célèbres : les "Théorèmes de Menger". Nous les présentons ici comme corollaires du Théorème "MaxiFlot MiniCoupe". Chacun de ces théorèmes établit pour tous sommets $s \neq t$ l'égalité de deux entiers :

- le plus petit nombre d'"objets" qu'il faut retirer pour déconnecter s de t.
- le plus grand nombre de chemins "objet disjoint" connectant s à t.

Fait remarquable, cette propriété concerne des graphes orientés ou non et des "séparateurs" formés de sommets, d'arcs ou d'arêtes.

Définition 13 Un ensemble S d'arcs (resp. d'arêtes, de sommets) d'un graphe *sépare* un sommet s d'un sommet t si il ne contient ni s ni t et si tout chemin de G allant de s à t contient un élément de S. Un tel ensemble est dit *séparateur*.

Cette définition en appelle d'autres, qui permettent de quantifier le nombre de sommets, d'arcs ou d'arêtes nécessaires à déconnecter le graphe, c'est à dire à séparer deux sommets.

Définition 14 Soit k un entier. Un graphe G est :

k-arc-connexe si tout ensemble d'arcs séparateur est de cardinalité $\geq k$.

k-arête-connexe si tout ensemble d'arêtes séparateur est de cardinalité $\geq k$.

k-connexe si tout ensemble de sommets séparateur est de cardinalité $\geq k$.

remarque

Dans la littérature, les définitions de k-connexité, k-arc-connexité ou de k-arête-connexité peuvent quelque peu différer. On impose parfois à tout graphe 2-arc-connexe ou 2-arête-connexe de posséder au moins deux sommets ; on impose à tout graphe k-connexe de posséder au moins $k+1$ sommets.

L'intérêt de ces définitions est, par exemple, qu'un graphe avec un seul sommet n'est pas 10-arc-connexe, 10-arête-connexe ou 10-connexe (ce qui est le cas avec nos définitions).

Les définitions choisies ici l'ont été pour leur simplicité. L'autre intérêt est l'extrême simplicité avec laquelle on caractérise ces notions de connexité. C'est l'objet des sections suivantes.

8.4.1 Caractérisation de la k-arc-connexité

Théorème 42 Pour tout graphe orienté G et pour tous sommets distincts s et t, les deux entiers suivants sont égaux :
- la cardinalité minimale d'un ensemble d'arcs séparant s de t.
- le plus grand nombre de chemins de s à t deux à deux arcs-disjoints.

preuve :
La preuve s'établit en construisant pour tous sommets distincts s et t un réseau à partir de G de source s et de puits t et en appliquant le théorème MiniCoupe=MaxiFlot. Cette construction est étudiée en Travaux Dirigés. □

Ce théorème peut se reformuler en utilisant la notion de k-arc-connexité :

Corollaire 43 Pour tout graphe orienté G et tout entier k, les deux assertions suivantes sont équivalentes :
- G est k-arc-connexe.
- pour tous sommets distincts s et t, il existe au moins k chemins de s à t deux à deux arcs-disjoints.

preuve :
Conséquence assez immédiate du Théorème 42. □

8.4.2 Caractérisation de la k-arête-connexité

Théorème 44 Pour tout graphe orienté G et pour tous sommets distincts s et t, les deux entiers suivants sont égaux :
- la cardinalité minimale d'un ensemble d'arêtes séparant s de t.
- le plus grand nombre de chemins de s à t deux à deux arêtes-disjoints.

preuve :
Une preuve possible est réalisée en utilisant l'égalité du Théorème 42 et en transformant le graphe non orienté G en un graphe orienté $o(G)$ en remplaçant toute arête par deux arcs opposés. Cette construction est étudiée en Travaux Dirigés. □

Ce théorème peut se reformuler en utilisant la notion de k-arête-connexité :

Corollaire 45 Pour tout graphe orienté G et tout entier k, les deux assertions suivantes sont équivalentes :

1. G est k-arête-connexe.
2. pour tous sommets distincts s et t, il existe au moins k chemins de s à t deux à deux arcs-disjoints.

preuve :
Conséquence assez immédiate du Théorème 44. □

8.4.3 Caractérisation de la k-connexité

Théorème 46 Pour tout graphe orienté ou non G et pour tous sommets distincts et non adjacents s et t, les deux entiers suivants sont égaux :

- la cardinalité minimale d'un ensemble de sommets séparant s de t.
- le plus grand nombre de chemins de s à t deux à deux sommets-disjoints.

preuve :
Une preuve possible (parmi d'autres éventuellement plus directes) utilise l'égalité du Théorème 42 et transforme le graphe G en un graphe orienté $f(G)$ en remplaçant tout sommet s par deux sommets $(s, 0)$ et $(s, 1)$, le premier extrémité terminale de tous les arcs entrants dans s, le second extrémité initiale de tous les arcs sortants de s. Cette construction est étudiée en Travaux Dirigés. □

Ce théorème peut se reformuler en utilisant la notion de k-connexité :

Corollaire 47 Pour tout graphe orienté ou non G et tout entier k, les deux assertions suivantes sont équivalentes :

1. G est k-connexe.
2. pour tous sommets distincts et non adjacents s et t, il existe au moins k sommets de s à t deux à deux sommets disjoints (exceptés, naturellement, les extrémités s et t).

preuve :
Conséquence assez immédiate du Théorème 46. □

Chapitre 9

Couplage

Le problème posé à l'aviation anglaise durant la bataille d'angleterre était de pouvoir former un nombre maximum de couples de pilotes parmi un ensemble de pilotes venant des quatre coins du monde. On confiait à deux pilotes les commandes d'un avion à l'unique condition qu'ils partageaient une même langue. Un grand nombre de pilotes parlaient plusieurs langues.

Ce problème se formalise aisément en un problème de graphes, qui a pour nom le problème du couplage maximum.

9.1 Définition

Les graphes que nous considérerons dans ce chapitre sont non orientés. Afin de simplifier l'étude, nous pourrons supposer qu'ils sont dépourvus de boucle.

Définition 15 Un *couplage* d'un graphe non orienté est un ensemble d'arêtes 2 à 2 non adjacentes et qui ne soient pas des boucles. Un sommet est *saturé* dans un couplage D si il est incident à l'une des arêtes de D. Un couplage est :

- *maximum* si il est de cardinalité maximale parmi tous les couplages.
- *parfait* si tout sommet est saturé.

Le problème du couplage maximum peut ainsi être formalisé :

`Couplage Maximum`
`ENTRÉE : un graphe non orienté` G
`SORTIE : un couplage maximum de` G

9.2 Première caractérisation d'un couplage maximum

Une condition suffisante pour qu'un couplage ne soit pas maximum est l'existence de deux sommets insaturés adjacents, voire l'existence de deux sommets insaturés reliés par un chemin alternant des arcs du couplage et des arcs n'appartenant pas à ce couplage. Une propriété remarquable des couplages est que cette condition est non seulement suffisante mais aussi nécessaire. Cette section est consacrée à cette caractérisation.

Définition 16 Un *chemin alterné* w relativement à un couplage K d'un graphe G est un chemin élémentaire à extrémités distinctes telle que pour toute arête e de w et pour toute arête f succédant immédiatement e sur w on ait $e \in K \Leftrightarrow f \notin K$. Un chemin alterné est *augmentant* relativement à K si ses deux extrémités sont insaturés par K.

Notation 17 La différence symétrique $(A - B) \cup (B - A)$ de deux ensembles A est B est notée $A \triangle B$.

Fait 48 Soit w un chemin alterné relativement à un couplage K d'un graphe G.

Si chaque extrémité de w est soit insaturé soit incidente à une arête de K appartenant à w,

alors $K \triangle E(w)$ est un couplage de G.

preuve :
Pour conclure, démontrons que tout sommet est incident à au plus une arête de $K \triangle E(w)$. Soit s un sommet de G. Plusieurs cas apparaissent :

- $s \notin V(w)$.
  Par hypothèse, s est incident à au plus une arête de K. Le sommet n'est incident à aucune arête de w, donc à aucune arête de $E(w) - K$ et est donc incident à au plus une arête de $K \triangle E(w)$.
- s est une extrémité de w.
  Par hypothèse, s est incident à au plus une arête de $E(w)$ et à aucune arête de $K - E(w)$. Le sommet s est donc incident à au plus une arête de $(E(w) - K) \cup (K - E(w))$.
- s est interne à w.
  Par hypothèse, s est incident à exactement une arête de $E(w) - K$, à exactement à une arête de $E(w) \cap K$ et donc à aucune arête de $K - E(w)$. Et donc à exactement à uen arête $K \triangle E(w)$.

□

Fait 49 Soit w un chemin alterné augmentant relativement à un couplage K d'un graphe G. L'ensemble $K \triangle E(w)$ est un couplage de cardinalité $|K \triangle E(w)| > |K|$.

preuve :
Conséquence du Fait 48, $K \triangle E(w)$ est un couplage. De plus il est facile d'observer que $|E(w) - K| > |E(w) \cap K|$. Ce qui entraine $|K \triangle E(w)| > |K|$. □

Fait 50 Si K et K' sont deux couplages d'un graphe G, alors chaque composante connexe de $K \triangle K'$ est de l'un des trois types suivant :

1. un sommet isolé.
2. un cycle élémentaire paire à arêtes alternativement dans K et K'.
3. un chemin élémentaire à arêtes alternativement dans K et K'.

preuve :
Clairement tout sommet de G est incident à au plus une arête de K et à au plus une arête de K' donc à au plus deux arêtes de $K \triangle K'$. Ainsi, toute composante connexe est un chemin élémentaire. K et K' sont des couplages, ce chemin alterne donc nécessairement les arêtes

de K et K', et si il est un cycle il est nécessairement de longueur paire. □

Théorème 51 Pour tout couplage K d'un graphe G, les deux assertions suivantes sont équivalentes :

1. K est maximum.
2. il n'existe aucun chemin alterné augmentant.

preuve :
Soit K un couplage d'un graphe $G = (V, E)$. Il vient :

(1.) $\Rightarrow$ (2.)
Si L est un chemin augmentant, l'ensemble $K \triangle L$ est un couplage de cardinalité $|K|+1$. Ainsi, K n'est pas maximum.

(2.) $\Rightarrow$ (1.)
Soit K' un couplage admettant aucun chemin alterné augmentant. Soit K un couplage maximum. Pour conclure il suffit de démontrer l'inégalité des cardinalités $|K| \geq |K'|$. Pour ce faire nous démontrerons l'inégalité $|K - K'| \geq |K' - K|$, en démontrons que chaque composante connexe U de $K \triangle K'$ possède au moins autant d'arêtes de $K - K'$ que d'arêtes de $K' - K$. D'après le Fait 50, trois cas apparaissent :

1. U est un sommet isolé.
   Conclusion immédiate.
2. U est un cycle paire.
   U est un chemin à arêtes alternativement dans $K - K'$ et $K' - K$. La conclusion est immédiate.
3. U est un chemin élémentaire.
   U est à arêtes alternativement dans $K - K'$ et $K' - K$, donc à arêtes alternativement dans K (resp. K') et dans $E - K$ (resp. $E - K'$). Si on suppose ce chemin de longueur impaire, ses deux extrémités sont incidentes soit à aucune arête de K soit à aucune arête de K'. Ainsi, un tel chemin est alterné augmentant soit relativement à K soit relativement à K'. Contradiction dans les deux cas. Ainsi, le chemin est de longueur paire. Le nombre d'arête de U appartenant à $K - K'$ est égal à celui appartenant à $K' - K$ et à U.

□

9.3 Caractérisation algorithmique

La solution algorithmique requiert la définition de ce qu'est un arbre alterné, extension aux arbres de la notion d'alternance définie sur les chemins.

Définition 18 Un *arbre alterné* selon un graphe $G = (V, E)$ et un couplage K de G est un triplet (U, D, r) tel que :
– (U, D) est un arbre qui est sous-graphe de G.

- r est un sommet de U appelé sa racine et est insaturé selon K.
- toute feuille de T est *paire*, c.a.d est à distance paire de r.
- tout sommet pair de T autre que la racine est adjacent a son père (impair) selon une arête de $D \cap K$.

Afin de simplifier les énoncés, un graphe (G, K) muni d'un de ses couplages sera appelé un *graphe couplé*, un graphe (G, K, T) muni d'un de ses couplages et d'un arbre alterné sera appelé un *arbre couplé arboré*.

Pour résoudre le problème `Couplage Maximum`, on utilise deux fonctions auxiliaires `CMC` et `CMCA` résolvant respectivement les deux problèmes suivants :

`Couplage Maximum d'un graphe couplé`
`ENTRÉE : un graphe couplé` (G, J)
`SORTIE : un couplage maximum` K `de` G `tel que` $K \neq J \Rightarrow |K| > |J|$

`Couplage Maximum d'un graphe couplé arboré`
`ENTRÉE : un graphe couplé arboré` (G, J, T)
`SORTIE : un couplage maximum` K `de` G `tel que` $K \neq J \Rightarrow |K| > |J|$

Le lien entre ses problèmes est immédiat :

- une définition de `CM` résolvant `Couplage Maximum` est simplement :

```
fonction CM(G:graphe):couplage

    retourner CMC(G,∅) ;
```

FIG. 9.1 – Algorithme `CM`

- une définition de `CMC` résolvant `Couplage Maximum2` est simplement :

```
fonction CMC((G,K):graphe couplé):couplage

    si il existe au plus un sommet insaturé selon K
        retourner K ;
    sinon
        r ← insaturé(G,K) ;
        T ← arbreAlterné1Sommet(r) ;
        retourner CMCA(G,K,T) ;
```

FIG. 9.2 – Algorithme `CMC`

L'écriture de `CMCA` nécessite quelques notations et définitions préalables :

Notation 19 Le graphe obtenu à partir d'un graphe G en fusionnant une partie de sommets $Y \subseteq V_G$ (voir Section 2.4.4) est noté dans cette section $G \bullet Y$. On suppose en outre que les boucles sont supprimées.

Le sous-graphe d'un graphe G induit par un ensemble de sommets $U \subseteq V_G$ est noté $G|U$. L'expression $G - U$ dénote le sous-graphe $G|(V - U)$.

Définition 20 (Orbite) Une *orbite* d'un graphe couplé arboré (G, K, T) est une arête e incidente à deux sommets pairs de T. Nous noterons :

G^e le graphe obtenu à partir de G en fusionnant tous les sommets de C (c.a.d $G \bullet C$).

K^e le couplage obtenu à partir de K en supprimant toutes les arêtes dont les deux extrémités appartiennent à C. Clairement K^e est un couplage de G^e.

T^e l'arbre obtenu à partir de T en fusionnant tous les sommets de C (c.a.d $T \bullet C$). Clairement T^e est un arbre alterné de (G^e, K^e).

où C désigne l'ensemble des sommets de l'unique circuit élémentaire du graphe T augmenté de l'arête e.

La définition de `CMCA` résolvant `Couplage Maximum3` est fournie par la Figure 9.3. Afin de faciliter l'écriture de cet algorithme, l'ensemble des sommets d'un graphe G habituellement noté V_G est noté ici $V(G)$.

```
fonction CMCA((G,K,T):graphe couplé arboré):couplage

    si E_T = E_G
        retourner K ;
    sinon si il existe une arête e = {i,j} ∈ E_G − E_T
              avec i ∈ V_T pair et j ∉ V_T
        si j est insaturé selon K        %chemin(T,i).(i,e,j) est
            L ← {e} ∪ E(chemin(T,i)) ;    %un chemin augmentant de (G,K)
            retourner CMC(G,K△L) ;
        sinon
           f ← uniqueArêteIncidenteDeK(G,K,j) ;
           T ← arbreAlternéAugmenté(T,e,f) ;
           retourner CMCA(G,K,T) ;
    sinon si il existe une orbite e relativement à (G,K,T)
        (G^e,K^e,T^e) ← fusionOrbite(G,K,T,e) ;
        K_1 ← CMCA(G^e,K^e,T^e) ;
        K_2 ← CM(circuit(T,e)-V(K_1)) ;
        si K_1 = K^e
            retourner K^e ∪ K_2
        sinon
            retourner CMC(G,K_1 ∪ K_2) ;
    sinon
        retourner (K ∩ E_T)∪CM(G − V(T)) ;
```

FIG. 9.3 – Définition de `CMCA`

La sémantique des routines utilisées par `CMCA` est :

- `chemin`(T, i) qui retourne l'unique chemin élémentaire de l'arbre T allant de la racine de T au sommet i.
- `uniqueAreteIncidenteDeK`(G, K, j) qui retourne l'unique arête de K incidente dans G au sommet j.

- `arbreAlterneAugmente`(T, e, f) qui retourne l'arbre obtenu à partir de T en ajoutant les arêtes e et f.
- `fusionOrbite`(G, K, T, e) qui retourne le graphe couplé alterné induit par l'orbite e (voir Définition 20).
- `circuit`(T, e) qui retourne l'unique circuit de $T \cup \{e\}$.

9.3.1 Terminaison et Complexité en temps

Fait 52 Les algorithmes `CM`, `CMC` et `CMCA` terminent.

preuve :
Considérons les séquences passées en argument. Elles sont de la forme d'un graphe (G), d'un graphe couplé (G, K) ou d'un graphe couplé arboré (G, K, T). Notons $\prec$ la relation définie par : $(G_1, \ldots) \prec (G_2, \ldots)$ si :
- $|V_{G_1}| < |V_{G_2}|$
- ou si $(|V_{G_1}| = |V_{G_2}|$ et $|K_1| > |K_2|)$
- ou si $(|V_{G_1}| = |V_{G_2}|$ et $|K_1| = |K_2|$ et $|V_{T_1}| > |V_{T_2}|)$.

Il est aisé d'observer que $\prec$ est un ordre bien fondé (toute suite strictement décroissante est nécessairement finie) et que toute séquence $s_2 = (G_2, \ldots)$ de paramètres d'une fonction appelée dans la définition de `CMCA` est strictement inférieure selon $\prec$ à la séquence de paramètres passée en entrée de `CMCA`. Ainsi, `CMCA` termine. Et donc de même pour `CM` et `CMC`. □

Rassurons nous ! L'algorithme `CMCA` fait mieux que terminer, il a une complexité en temps dans le pire des cas en $O(n^3)$ où n désigne le nombre sommets. Les techniques pour implémenter les différentes structures et routines auxiliaires sont compliquées et sortent du cadre de ce cours. Ces implémentations ont été trouvées par Gabow en 1976 et Lawler en 1976 et sont accessibles à :
- Gabow H.N. 1976, An efficient implementation of Edmonds'algorithm for maximum matchings on graphs, J. of A.C.M., Vol. 23, n 2, pp. 120-130.
- Combinatorial otpimization : networks and matroïds, Holt, Rinehart and Winston.

9.3.2 Étude de la correction de `CMCA`

Fait 53 Soit un graphe couplé arboré (G, K, T). Pour toute arête $e = \{y, z\} \in K$ on a :

$$y \in V(T) \Leftrightarrow z \in V(T)$$

preuve :
Conséquence directe du fait que tout sommet de T est soit insaturé selon K soit est saturé et est incident à une arête de $K \cap E_T$ unique arête de K à être incidente à ce sommet. □

Fait 54 Soit e une orbite d'un graphe couplé alterné (G, K, T). On a :

Si $K - E(c)$ est un couplage maximum de G^e,

alors K est un couplage maximum de G.

où c désigne l'unique circuit de $T \cup \{e\}$.

preuve :
Soit e une orbite d'un graphe couplé alterné (G, K, T). Notons c l'unique circuit de $T \cup \{e\}$. Notons j_0 le sommet du circuit c le plus proche dans T de la racine de T, notée r. Sans perte de généralité, on peut supposer que le sommet de G^e issu de la fusion de $V(c)$ se nomme j_0.

Notons p l'unique chemin élémentaire de T allant de r à j_0. Du fait que T est alterné, p est un chemin alterné, d'après le Fait 48, l'ensemble $K' := K \triangle E(p)$ est un couplage de G. De plus, p n'étant pas augmentant (r est insaturé, j_0 est saturé), on a : $|K'| = |K|$ De même, p est un chemin alterné de $(G^e, K - E(c))$. Pour les mêmes raisons, $(K - E(c)) \triangle E(p)$, égal à $K' - E(c)$, est un couplage de G^e de cardinalité $|K - E(c)|$.

Ainsi, pour conclure, il suffit de démontrer que si K' n'est pas maximum dans G, $K' - E(c)$ ne l'est pas non plus dans G^e. Ce qui revient à démontrer, d'après le Théorème 51, que l'existence d'un chemin alterné augmentant dans (G, K') entraine l'existence d'un tel chemin dans $(G^e, K' - E(c))$. Supposons donc l'existence d'un tel chemin $w = (s_1, e_1, s_2, \ldots, e_l, s_{l+1})$ dans (G, K'). Deux cas apparaissent :

- aucun sommet de w n'appartient à $V(c)$.
  Clairement w est un chemin alterné augmentant de $(G^e, K' - E(c))$.
- un des sommets de w appartient à $V(c)$.
  Puisque T contient un unique sommet insaturé (la racine), au plus une des extrémités insaturés de w appartient à T et donc à $V(c)$. On peut donc supposer que l'extrémité initiale de w n'appartient pas à $V(c)$. Soit s_i le premier sommet de w appartenant à $V(c)$. On a : $i \neq 1$. D'après le Fait 53, e_{i-1} n'appartient pas à K. Le chemin $(s_1, e_1, s_2, \ldots, e_{i-1}, j_0)$ est un chemin alterné de $(G^e, K' - E(c))$ avec s_1 et j_0 insaturés. Il est donc augmentant.

□

Définition 21 Soit (G, K, T) un graphe couplé arboré. L'arbre T *complet* si aucune des assertions suivantes (apparaissant dans la définition de l'algorithme `CMCA` de la Figure 9.3) n'est vérifiée :

- $E_T = E_G$.
- il existe une arête $e = \{i, j\} \in E_G - E_T$ avec $i \in V(T)$ pair et $j \notin V(T)$.
- il existe une orbite e relativement à (G, K, T).

Fait 55 Soit un graphe couplé alterné (G, K, T). Si T est complet, alors les deux assertions suivantes sont équivalentes :

1. K est un couplage maximum de G.
2. $K - E_T$ est un couplage maximum de $G - V(T)$.

preuve :
Soit (G, K, T) graphe couplé alterné (G, K, T) avec T complet. Notons r la racine de T. K_1 l'ensemble $K \cap E_T$ et K_2 l'ensemble $K - E_T$, notons G_1 le graphe $G|V(T)$ et G_2 le graphe $G - V(T)$. Du fait que tout sommet de T est soit insaturé et est égal à la racine soit saturé et est incident à une arête de $K \cap E_T$ et donc à aucune autre arête de K, K_1 et K_2 sont deux couplages respectifs de G_1 et G_2. De plus on a :

(1.) $\Rightarrow$ (2.)

Démontrons la contraposée : $\neg(2.) \Rightarrow \neg(1.)$. Si K_2 n'est pas maximum, il existe un chemin p alterné augmentant relativement à (G_2, K_2). Clairement, p est un chemin de G (G_2 est sous graphe de G). Conséquence du Fait 53, aucune arête de K n'a une extrémité dans G_1 et une autre dans G_2, aussi tout sommet est insaturé dans (G_2, K_2) si et seulement si il est est insaturé dans (G, K) et appartient à $V - V(T)$. On en déduit que p est un chemin alterné augmentant dans (G, K). Ainsi, K n'est pas maximum.

(2.) $\Rightarrow$ (1.)

Démontrons la contraposée : $\neg(1.) \Rightarrow \neg(2.)$. Supposons K non maximum et démontrons K_2 non maximum.

Si K n'est pas maximum, il existe un chemin alterné augmentant p de (G, K). L'arbre T étant complet, au plus un sommet de T est insaturé. Aussi, l'une des extrémités j_0 de p est distincte de la racine r de T. Notons k_0 l'autre extrémité de p, que l'on peut supposer être sans perte de généralité la seconde extrémité de p. Plusieurs cas apparaissent :

- Aucun des sommets de p n'appartient à $V(T)$.
  Clairement, p est un chemin alterné augmentant de (G_2, K_2). K_2 n'est pas maximum.
- Le premier sommet x de p appartenant à $V(T)$ est pair.
  Le sommet y précédant x sur le chemin p existe (l'extrémité initiale est $j_o \notin V(T)$) et n'appartient pas par construction à $V(T)$. Ainsi, il existe une arête $\{y, x\} \in E$ avec x pair et $y \notin V(T)$. Donc T n'est pas complet. Contradiction.
- Le premier sommet x de p appartenant à $V(T)$ est impair.
  Notons i le dernier sommet de p appartenant à $V(T)$ tel que tous les sommets entre x et i dans p appartiennent à $V(T)$. Notons q le chemin extrait de p allant de x à i : q est un chemin alterné de G_1. La première arête de q appartient à K, car celle qui la précède dans p n'appartient pas à K (Fait 53). Toute arête de q a deux extrémités l'une paire, l'autre impaire :
  - soit cette arête appartient à K, et par définition ces deux extrémités sont l'une paire, l'autre impaire.
  - soit cette arête n'appartient pas à K, et ces deux extrémités sont l'une paire, l'autre impaire : autrement, l'arête serait une orbite relativement à (G, K, T) et T ne serait pas complet.

  Donc q est de la forme $(s_1, e_1, s_2, e_2, \ldots, s_l, e_l, s_{l+1})$ où :
  - $e_1, e_3, \ldots \in K$ et $e_2, e_4, \ldots \notin K$.
  - $s_1, s_3, \ldots$ sont impairs et $s_2, s_4, \ldots$ sont pairs.

  On en déduit que tout sommet de q est saturé selon K_1 et est donc saturé selon K. Ainsi l'extrémité du chemin alterné augmentant p, trivialement insaturé selon K ne peut appartenir à q et donc ne peut être le sommet i. Il est existe donc un sommet j succédant à i dans p. Trivialement j n'appartient pas à T. La complétude de p entraîne i impair. La non-appartenance $\{i, j\} \notin K$ entraîne que l'arête précédant i dans q appartient à K et entraîne donc i pair. Contradiction qui suffit à conclure.

□

Lemme 56 Les algorithmes `CM`, `CMC` et `CMCA` sont corrects.

preuve :
Conséquence directe des Faits 49, 54, 55. □

9.4 Variantes

Il existe un grand nombre de problèmes liés aux couplages. Citons-en un :

```
Couplage de poids maximum
Entrée : un graphe non orienté
Sortie: un couplage de poids maximum
```

Ici, on considère que les arêtes sont munies d'un poids positif ou négatif. Le poids d'un couplage est la somme des poids des arêtes. Ce problème est en fait une extension du problème du couplage maximum. En, effet pour résoudre ce dernier problème, il suffit de savoir résoudre le premier comme l'indique l'algorithme suivant qui résoud `Couplage maximum` en utilisant un oracle résolvant `Couplage de poids maximum` :

`fonction couplageMaximum(`G`: graphe):couplage`

$p \leftarrow$ `fonction associant à toute arête` $e \in V_G^2$ `le booléen` $e \in E_G$;
`retourner résoudreCouplagePoidsMaximum(`V_G, V_G^2, p`)` ;

Propriété remarquable, le problème plus général `Couplage de poids maximum` admet une solution algorithmique proche de `CM`, qui en est une extension.