

Notes de cours Algorithmique de graphes :
Magistère Informatique, Cachan

Michel Habib,
email: habib@liafa.jussieu.fr
<http://www.liafa.jussieu.fr/~habib>

19 décembre 2011

2

1

1. Merci de me faire part de toute remarque, concernant ce texte, erreurs, passages trop rapides ...

Chapitre 1

Introduction

Nous noterons un graphe $G = (X, E)$, où X est un ensemble **fini** de sommets et $E \subseteq X^2$ un ensemble d'arêtes. Une arête est donc constituée de deux sommets. Les graphes non orientés considérés dans ce cours seront, sauf mention contraire, **simples** : sans boucle (arête de type xx) et sans arête multiple (plusieurs arêtes entre deux sommets).

$H = (Y, F)$ est un **sous-graphe induit** de $G = (X, E)$, si $Y \subseteq X$ et si $F = \{e \in E \mid \text{les deux extrémités de } e \text{ sont dans } Y\}$.

$H = (Y, F)$ est un **sous-graphe partiel** de $G = (X, E)$, si $Y \subseteq X$ et si $F \subseteq E \cap Y^2$.

En général $|X| = n$ et $|E| = m$. Le **degré** d'un sommet noté $d(x)$ est le nombre d'arêtes adjacentes à x . Un sommet **pendant** dans un graphe, est un sommet x tel que : $d(x) = 1$.

Une **chaîne** de longueur k est un graphe :

$P = (\{x_0, x_1, \dots, x_k\}, \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\})$ ayant $k + 1$ sommets et k arêtes, noté P_k . Ce graphe s'appelle une chaîne de longueur k joignant x_0 à x_k .

On confond souvent la chaîne avec l'une des deux séquences : $[x_0, x_1, \dots, x_k]$ ou $[x_k, x_{k-1}, \dots, x_0]$. Un graphe réduit à un sommet et sans arête est une chaîne de longueur 0.

Un **cycle** de longueur k est un graphe :

$C = (\{x_1, \dots, x_k\}, \{x_1x_2, x_2x_3, \dots, x_{k-1}x_k, x_kx_1\})$ ayant k sommets et k arêtes, noté C_k . On confond souvent le cycle avec l'une des séquences : $[x_1, x_2, \dots, x_k, x_1]$ (à une permutation circulaire près).

Avec de telles définitions, une chaîne (resp. un cycle) ne passe qu'une fois par un sommet, cela correspond à une chaîne (resp. cycle) élémentaire chez

certain auteurs.

On appelle **marche** dans $G = (X, E)$ une séquence non nulle $M = x_0e_1x_1e_2 \dots e_kx_k$ alternant sommets et arêtes telle que :

$\forall 1 \leq i \leq k$, les extrémités de e_i sont x_{i-1} et x_i . On dit que la marche est de longueur k et qu'elle joint x_0 à x_k .

Lorsque $x_0 = x_k$ on dit que la marche est **fermée** ou un **pseudocycle**. Si le graphe est simple, nous pouvons omettre les arêtes et noter la marche par $[x_0, x_1, \dots, x_k]$.

Lorsque toutes les arêtes de la marche sont différentes on dit que la marche est un **sentier**¹, lorsque tous les sommets sont distincts les sommets et les arêtes de la marche constituent un graphe partiel de G qui est une chaîne.

Proposition 1 *S'il existe une marche de longueur finie joignant x à y dans G , alors il existe une chaîne joignant x à y dans G , en outre les sommets et les arêtes de cette chaîne appartiennent à la marche.*

Preuve: Comme il existe au moins une marche de longueur finie dans G joignant x à y , parmi toutes les marches de longueur finie, il existe au moins une de longueur minimum joignant x à y dans G . Notons M , cette marche. Si elle n'est pas élémentaire, on peut extraire une marche de longueur strictement plus petite, d'où la contradiction. \square

N.B. Cette preuve repose sur la finitude de la marche, car si M est de longueur infinie, il n'est pas sûr que M' la marche extraite soit de longueur strictement inférieure.

Proposition 2 *S'il existe un pseudo-cycle de longueur finie dans G , alors il existe un cycle de longueur finie dans G .*

Un graphe $G = (X, E)$ est **connexe**, si $\forall x, y \in X$, il existe une chaîne allant de x à y .

Proposition 3 *On considère un graphe $G = (X, E)$ connexe, si $\forall x \in X$, $d(x) \leq 2$, alors G est soit une chaîne, soit un cycle.*

Preuve: Si G n'a aucune arête, vu l'hypothèse de connexité, G est réduit à un sommet et est donc une chaîne de longueur 0. Sinon G possède au moins une

1. Ces définitions de chaînes, marches, sentiers et autres cycles ne sont pas standard dans la littérature

arête, donc au moins une chaîne et soit C une chaîne de longueur maximale de G . Seules les extrémités x, y de la chaîne peuvent être adjacentes à des arêtes n'appartenant pas à la chaîne (car les autres sommets de la chaîne sont déjà de degré 2). Mais si ces arêtes ont une extrémité hors de la chaîne, celle-ci ne serait pas de longueur maximale. Donc soit $d(x) = d(y) = 1$ et G est une chaîne, soit $xy \in E$ et G est un cycle. \square

Théorème 1 *Les 6 conditions suivantes sont équivalentes et caractérisent les arbres :*

1. G est connexe minimal (si on enlève une arête, le graphe n'est plus connexe)
2. G est sans cycle maximal (si on ajoute une arête, le graphe admet un cycle)
3. G est connexe sans cycle
4. G est connexe avec $n - 1$ arêtes
5. G est sans cycle avec $n - 1$ arêtes
6. $\forall x, y \in X$, il existe une chaîne unique joignant x à y .

Preuve: Il est facile de vérifier que les conditions 1, 2, 3 et 6 sont équivalentes.

Montrons l'équivalence avec les conditions 4 et 5. Pour ce faire nous allons commencer par deux lemmes.

Lemme 1 *Un graphe connexe ayant $n - 1$ arêtes possède nécessairement un sommet pendant.*

Preuve: G étant connexe, $\forall x \in X$, $d(x) \geq 1$ (il n'y a pas de sommet isolé). S'il n'existe pas de sommet pendant, alors $\forall x \in X$, $d(x) \geq 2$ et donc $\sum_{x \in X} d(x) \geq 2n > 2m = 2n - 2$, d'où la contradiction. \square

Lemme 2 *Un graphe G sans cycle, ayant au moins une arête, possède nécessairement un sommet pendant.*

Preuve: Il existe au moins une chaîne dans G . Considérons une chaîne $[x = x_1, \dots, x_k = y]$ de longueur maximale.

Si x n'est pas un sommet pendant alors il admet un autre voisin z différent de x_2 . Si $z \in \{x_3, \dots, x_k\}$, alors l'arbre possède un cycle ce qui est exclu.

Donc $z \notin \{x_2, \dots, x_k\}$, mais alors la chaîne choisie n'était pas maximale.
 \square

Revenons à la preuve du théorème. Le premier lemme permet de montrer par induction 4 implique 3.

En effet d'après ce lemme un graphe G , vérifiant la condition 4, admet au moins un sommet pendant x . Si l'on considère le graphe $G' = G - x$. G' est sans cycle ssi G est sans cycle. En outre G' a un sommet et une arête de moins que G et vérifie donc aussi la condition 4. On peut donc réitérer le procédé jusqu'à arriver sur le graphe vide réduit à un sommet qui est évidemment sans cycle. Donc 4 implique 3.

Un raisonnement analogue basé sur le lemme 2 permet de montrer 5 implique 3.

Montrons maintenant 3 implique 4 et 5. Le seul graphe à un sommet est connexe et sans cycle et il a bien $n - 1 = 0$ arête.

Supposons maintenant vraie jusqu'à l'ordre $n - 1$ que la condition 3 implique 4 et 5. On considère un graphe $G = (X, E)$ ayant n sommets. Soit $a \in X$. $G - a$ possède $k = d(a) \geq 1$ composantes connexes $G_i = (X_i, E_i)$ qui sont nécessairement connexes et sans cycle, vérifiant ainsi la condition 3. L'hypothèse d'induction sur les graphes G_i donne :

$$\begin{aligned} |E_i| &= |X_i| - 1. \text{ D'où l'on déduit :} \\ |E| &= \sum_{1 \leq i \leq k} |E_i| + k = \sum_{1 \leq i \leq k} (|X_i| - 1) + k = \sum_{1 \leq i \leq k} |X_i| = |X| - 1. \quad \square, \end{aligned}$$

Corollaire 1 *Tout arbre non réduit à un sommet admet au moins deux sommets pendants (sommet de degré 1)*

En outre lorsqu'on supprime un sommet pendant à un arbre, le graphe obtenu est encore un arbre, ce qui nous donne un schéma récursif simple pour manipuler les arbres.

1.1 Exercices

1. Montrer qu'un graphe non orienté sans boucle ni arête multiple possède au moins deux sommets de même degré.
2. Quelle est la structure d'un graphe G dont tous les sommets ont un degré égal à 2 ?
3. Montrer que dans tout graphe non orienté, il y a un nombre pair de sommets de degré impair.

4. Considérons un graphe complet dont les arêtes sont colorées avec deux couleurs. Montrer qu'il existe toujours un arbre recouvrant monocoloré.
5. Montrer le théorème d'Euler : un graphe connexe admet un cycle Eulerien (i.e. un cycle qui passe une fois et une seule par chaque arête) ssi tous ses degrés sont pairs.
6. Proposer un algorithme en $O(n + m)$ qui calcule un tel cycle Eulerien lorsqu'il existe.

1.2 Dualité cycle-cocycle

Nous allons maintenant exhiber une propriété d'échange ainsi qu'une dualité.

Définition 1 *Un cocycle $\theta(A)$ (ou une coupe $(A, X-A)$) d'un graphe $G = (X, E)$ est un ensemble d'arêtes incidentes à un sous ensemble $A \subset X$, i.e. l'ensemble des arêtes n'ayant qu'une extrémité dans A l'autre extrémité étant dans $X-A$.*

Par définition $\theta(A) = \theta(X - A)$.

Lemme 3 *Si un cycle μ et un cocycle θ ont une arête en commun, ils en ont au moins une deuxième.*

Il est possible de préciser un peu le résultat : un cycle et un cocycle ont en commun un nombre pair d'arêtes.

Lemme 4 *Soit $G = (X, E)$ un graphe connexe et $T = (X, F)$ un arbre recouvrant de G , pour toute arête $e \in E - F$ il existe une arête $f \in F$ telle que $T' = T - f + e$ soit un arbre recouvrant de G .*

Une arête d'un graphe est appelée un **isthme** lorsque sa suppression disconnecte le graphe.

Lemme 5 *Soit $G = (X, E)$ un graphe connexe et $T = (X, F)$ un arbre recouvrant de G , pour toute arête $f \in F$, si $G - f$ est toujours connexe (i.e. f n'est pas un isthme de G), il existe une arête $e \in E - F$ telle que $T' = T - f + e$ soit un arbre recouvrant de G .*

Dans les deux cas, on dit alors avoir procédé à un échange. Il est facile d'en déduire que si T_1 et T_2 sont deux arbres recouvrants de G , il existe une suite finie d'échanges qui permet de passer de T_1 à T_2 .

1.3 Espaces vectoriels des cycles-cocycles

À chaque cycle (resp. cocycle) on associe le vecteur caractéristique de ses arêtes, un vecteur de $\{0, 1\}^m$.

On considère les coefficients dans $Z/2Z$.

Il est facile de vérifier que : $\theta(A) = \sum_{x \in A} \theta(x)$

Ce qui permet de définir la somme de cocycles :

$$\theta(A) + \theta(B) = \theta(A \Delta B)$$

Nous avons ainsi un sous espace vectoriel des cocycles.

Lorsqu'on considère la somme de cycles, les arêtes communes disparaissent, cependant la définition de la somme de cycles quelconques, nous oblige à considérer comme cycle une union disjointe de cycles. À ce prix on peut définir le sous espace vectoriel des cycles.

On vérifie bien que :

\forall cycle μ et \forall cocycle $\theta(A)$, on a bien

$\mu \cdot \theta(A) = 0$. Les deux sous-espaces sont donc orthogonaux.

Théorème 2 *On considère un graphe $G = (X, E)$ connexe, la dimension de l'espace vectoriel des cycles (resp. cocycles) est $m - n + 1$ (resp. $n - 1$).*

Preuve: Soit $T = (X, F)$ un arbre recouvrant de G . Pour $\forall e \in E - F$, $T + e$ contient un cycle unique μ_e . Ceci nous permet de construire un ensemble de $m - n + 1$ cycles indépendants, car ils ont tous une arête qui les distingue de tous les autres. Donc $\dim(\text{Cycles}) \geq m - n + 1$.

De même pour $\forall f \in F$, $T - f$ admet deux composantes connexes, et il existe donc un cocycle unique de G , noté θ_f . On peut ainsi construire $n - 1$ cocycles indépendants. Donc $\dim(\text{Cocycles}) \geq n - 1$.

Comme les espaces vectoriels sont orthogonaux,

$$\dim(\text{Cycles}) + \dim(\text{Cocycles}) \leq m, \text{ d'où le résultat annoncé. } \square$$

On appelle base fondamentale de cycles d'un graphe, une base obtenue à l'aide d'un arbre recouvrant. Il est facile de voir en considérant le graphe 3-soleil, qu'il existe des bases de cycles qui ne sont pas fondamentales.

Ces bases de cycles sont utiles en chimie organique pour l'analyse des fonctions associées à une molécule.

Chapitre 2

Arbre recouvrant de poids minimum et algorithmes gloutons

On considère, un graphe non orienté connexe $G = (X, E)$ et une fonction de poids $\omega : E \rightarrow R_+$. Et l'on cherche donc un un sous-graphe $T = (X, F)$ connexe dont la somme des poids des arêtes est minimale.

Comme application on peut imaginer que le graphe G représente le réseau des rues d'une ville que l'on cherche à cabler. Il faut donc trouver un graphe **connexe** (chaque sommet doit avoir accès au réseau) et de coût minimum. En effet le coût d'installation d'un câble peut dépendre de la rue choisie (faire passer un câble sous un tramway ou un monument historique peut coûter cher) c'est ce que modélise la valuation ω .

Il est facile de voir, comme la valuation ω est positive qu'une solution optimale d'un tel problème est nécessairement un arbre (graphe connexe minimal, cf. le chapitre précédent). Ce problème est donc appelé **le problème de l'arbre recouvrant de poids minimum**.

Ce problème admet une solution, car l'ensemble des arbres recouvrants est un ensemble fini, donc il admet un élément de coût minimal. Tout le problème c'est de le trouver sans énumérer tous les arbres recouvrants car il peut y en avoir beaucoup (ex : n^{n-2} pour un graphe complet, d'après le célèbre théorème de Cayley publié en 1889).

Pour ce faire on va construire un schéma d'algorithme, basé sur une bicoloration en rouge ou bleu des arêtes de G . Cette présentation très élégante est due à Tarjan [33].

Au départ les arêtes ne sont pas colorées et pour les colorier (de manière définitive) nous allons utiliser les deux règles suivantes :

Règle Bleue : Choisir un cocycle θ qui ne contient pas d'arête bleue, colorier en bleu une arête non colorée de coût minimal de θ .

Règle Rouge : Choisir un cycle μ qui ne contient pas d'arête rouge, colorier en rouge une arête non colorée de coût maximal de μ

On remarquera que ces règles sont duales l'une de l'autre, par l'échange cycle-cocycle, bleu-rouge et minimum-maximum.

Arbre recouvrant de poids minimum

tant que une règle est applicable faire

└ l'appliquer

Théorème 3 *Si G est connexe, l'algorithme précédent calcule un arbre recouvrant de poids minimum.*

Preuve:

Le graphe étant fini, il existe un nombre fini d'arbres recouvrants, parmi ceux-ci il en existe un de poids minimum, notons le $T = (X, F)$.

Nous allons prouver que l'invariant suivant est maintenu dans l'algorithme :

Invariant principal : Il existe un arbre de poids minimum qui contient toutes les arêtes bleues et aucune rouge.

La preuve se fait par récurrence sur le nombre d'application des règles. Initialement l'invariant est trivialement vrai.

Montrons tout d'abord que les arêtes bleues forment une forêt de G (i.e. un graphe sans cycle). La preuve se fait par induction sur le nombre d'applications de la règle bleue. Au départ il n'y a pas d'arêtes bleues donc pas de cycle et supposons que l'application de la i ème règle bleue sur l'arête $ab \in \theta$ introduise un cycle entièrement bleu μ . Mais d'après le lemme 3 θ et μ ont une autre arête en commun, d'où la contradiction.

Supposons l'invariant principal vrai lorsque k arêtes ont déjà été coloriées par application des ces deux règles, il existe donc un arbre $T = (X, F)$ recouvrant de poids minimal qui contient toutes les arêtes bleues et aucune rouge.

Considérons une nouvelle application, par exemple de la règle bleue. On colore donc en bleu une arête f d'un cocycle θ . Dans le cas où $f \in F$ alors l'invariant est encore trivialement vérifié. Supposons donc le contraire, f n'appartient pas à F . Le graphe $T = (X, F \cup \{f\})$ admet donc un cycle μ . f appartient à μ et θ , or un cycle et un cocycle ont au moins deux arêtes en

commun, soit g une telle arête. Nécessairement $g \in \theta$ et d'après l'application de la règle bleue $\omega(f) \leq \omega(g)$.

Cependant $T' = (X, F \cup \{f\} - g)$ est aussi un arbre recouvrant (échange des arêtes f et g , on note $T' = T + f - g$). Nous avons $\omega(T') = \omega(T) + \omega(f) - \omega(g)$, la minimalité de T impose donc que $\omega(f) = \omega(g)$ et donc $\omega(T') = \omega(T)$, et l'arbre T' permet de montrer que l'invariant est encore vrai.

Le raisonnement est similaire dans le cas d'une application de la règle rouge.

Examinons ce que se passe lorsqu'on ne peut plus appliquer de règle. Supposons que les arêtes bleues forment un graphe partiel ayant plusieurs composantes connexes sur les ensembles de sommets X_1, \dots, X_k , avec $k \geq 2$. Considérons le cocycle engendré par X_1 , $\theta(X_1)$ qui ne contient par définition aucune arête bleue. Mais est-il possible que toutes ses arêtes soient rouges? Montrons le contraire. L'invariant étant toujours vrai, il existe un arbre de poids minimal $T_{final} = (X, F_{final})$ qui contient toutes les arêtes bleues et aucune rouge. Cet arbre utilise au moins une arête de $\theta(X_1)$, et cette arête ne peut donc être rouge, donc il existe au moins une arête de $\theta(X_1)$ incolore. On peut appliquer sur $\theta(X_1)$ la règle bleue et colorier en bleu l'arête incolore de poids minimal, d'où la contradiction.

Ceci implique qu'à la fin les arêtes bleues forment un graphe partiel connexe de G lorsque celui-ci est connexe. Ce graphe est nécessairement un arbre d'après l'invariant principal car les arêtes bleues sont incluses dans un arbre de poids minimum.

Montrons maintenant que toutes les arêtes sont bien coloriées. Pour ce faire supposons qu'il existe une arête $g \in E$ non coloriée. Nécessairement $g \notin F$, mais alors il existe un cycle unique dans $T + g$ sur lequel nous allons pouvoir appliquer la règle rouge, contradiction.

Conclusions : l'algorithme s'arrête donc et lorsqu'aucune règle n'est applicable, toutes les arêtes sont coloriées. Les arêtes bleues constituent un arbre recouvrant de poids minimum et les arêtes rouges un coarbre (un coarbre est par définition le complémentaire d'un arbre recouvrant de G).

□

Corollaire 2 *On peut arrêter l'algorithme après $n - 1$ applications de la règle bleue (resp. $m - n + 1$ applications de la règle rouge).*

Preuve: D'après l'invariant démontré ci-dessus, les $n - 1$ arêtes obtenues après $n - 1$ applications de la règle bleue sont incluses dans un arbre optimal.

Mais vu la caractérisation des arbres, ces $n-1$ arêtes constituent une solution, i.e. un arbre optimal. On peut donc colorer en rouge toutes les arêtes encore incolores du graphe.

De même à l'issue $m - n + 1$ applications de la règle rouge, nous avons $m - n + 1$ arêtes colorées en rouge. Ces arêtes appartiennent à $m - n + 1$ cycles différents et d'après l'invariant prouvé ci-dessus, il existe un arbre optimal qui ne contient aucune de ces arêtes. Cet arbre optimal est nécessairement constitué des $n - 1$ autres arêtes de G . On peut donc colorer en bleu toutes les arêtes encore incolores du graphe. \square

L'algorithme générique proposé est dit **glouton** car il ne remet jamais en question la coloration d'une arête. Il existe plusieurs mises en application différentes de cet algorithme générique. Les plus célèbres sont dues à Kruskal [23] et à Prim [28].

Dans tous les algorithmes présentés ci-après pour la recherche d'un arbre recouvrant de poids minimum, le graphe initial G est supposé connexe.

2.0.1 Kruskal

Algorithme de Kruskal

Trier les arêtes de G par poids croissant en une liste L

$F \leftarrow \emptyset$

tant que $|F| < n - 1$ **et** $L \neq \emptyset$ **faire**

$e \leftarrow Premier(L)$	
Retrait(L, e); si $T = (X, F \cup \{e\})$ <i>n'a pas de cycle</i> alors	
└─ $F \leftarrow F \cup \{e\}$	

Cet algorithme revient à utiliser $n-1$ fois la règle bleue lorsqu'on ajoute une arête dans F , et un autant de fois la règle rouge que l'on détecte la présence d'un cycle en ajoutant l'arête e . En outre il est facile de montrer qu'à chaque itération la propriété : F est sans cycle est maintenue, et donc à la fin F est un arbre recouvrant de G .

Il n'est donc pas nécessaire de colorier explicitement les arêtes non considérées (les arêtes de L à la fin de l'algorithme) car elles sont nécessairement rouges et ne peuvent intervenir dans un arbre de poids minimal (cf. l'invariant).

Dans cet algorithme le point difficile pour la complexité reste la constitution de la liste triée des arêtes. Pour la suite il suffit d'utiliser une structure de données de type Union-Find pour gérer les composantes connexes du graphe $H = (X, F)$.

2.0.2 Prim

L'algorithme de Prim revient aussi à toujours appliquer la règle bleue, en travaillant toujours sur un seul cocycle que l'on maintient au cours de l'algorithme. En outre la propriété F engendre un graphe connexe est maintenue, et à la fin F est nécessairement un arbre recouvrant de G .

Algorithme de Prim $F \leftarrow \emptyset$ $A \leftarrow \{x_0\}$ **tant que** $|F| < n - 1$ **faire**

Calculer l'arête $e = (a, x)$ (avec $a \in A$) de coût minimal de $\theta(A)$
$F \leftarrow F \cup \{e\}$
$A \leftarrow A \cup \{x\}$

Pour l'implémentation de cet algorithme, il suffit d'utiliser une structure de données de tas afin de gérer dynamiquement la liste des arêtes du cycle courant $\omega(A)$. À chaque étape on insère au plus $d(x)$ arêtes dans le tas.

2.0.3 Boruvka

Une variante due à Boruvka (1926) d'après [20], est d'autant plus intéressante qu'elle précédait historiquement celles de Kruskal et de Prim.

Algorithme de Boruvka $F \leftarrow \emptyset$ **tant que** G *n'est pas trivial* **faire**

Détruire les boucles de G
Remplacer les arêtes multiples entre deux sommets, par une seule dont le poids est le minimum des poids de ces arêtes multiples
pour $\forall x \in G$ faire
trouver l'arête e_x de poids minimum adjacente à x
$F \leftarrow F \cup \{e_x\}$
Contracter e_x

Montrer que cet algorithme se prête aisément à la parallélisation.

2.1 Quelques variantes

Kruskal à l'envers

Trier les arêtes de G par poids décroissant en une liste L

$F \leftarrow E$

tant que $|F| > n - 1$ **et** $L \neq \emptyset$ **faire**

$\left[\begin{array}{l} e \leftarrow \text{Premier}(L) \\ \text{Retrait}(L, e); \text{ si } T = (X, F - \{e\}) \text{ est connexe alors} \\ \quad \left[F \leftarrow F - \{e\} \right] \end{array} \right.$

L'algorithme "Kruskal à l'envers" est intéressant à utiliser lorsque le nombre d'arêtes du graphe G est très proche de $n-1$, car le nombre de passages dans la boucle **tant que** est borné par $|E| - n + 1$.

Kruskal-dans-le-désordre

Construire une liste L avec les arêtes de G

$F \leftarrow \emptyset$

tant que $L \neq \emptyset$ **faire**

$\left[\begin{array}{l} e \leftarrow \text{Premier}(L) \\ \text{Retrait}(L, e) \\ \text{si } T = (X, F \cup \{e\}) \text{ n'a pas de cycle alors} \\ \quad \left[F \leftarrow F \cup \{e\} \right] \\ \text{sinon} \\ \quad \left[\begin{array}{l} \text{si } \omega(e) < \omega(f), \text{ où } f \text{ est une arête de poids maximum du cycle} \\ \text{de } F \cup \{e\} \text{ alors} \\ \quad \left[F \leftarrow (F \cup \{e\}) - \{f\} \right] \end{array} \right] \end{array} \right.$

L'algorithme **Kruskal-dans-le-désordre** permet d'éviter la phase initiale de tri des arêtes du graphe suivant leur poids.

Kruskal dual $F \leftarrow E$ **tant que** $|F| > n - 1$ **faire**

Choisir un cycle μ de $T = (X, F)$
Soit e une arête de poids maximal de μ
$F \leftarrow F - e$

Cet algorithme peut aussi se voir comme une sorte d'analogie de Prim mais basé sur les cycles, pour l'implémenter il faut bien choisir les cycles.

2.1.1 Ordre lexicographique sur les arbres

L'ensemble des arbres recouvrants d'un graphe peuvent être muni d'un ordre partiel \preceq , défini comme suit :

À chaque arbre T on associe le mot $m(T)$ constitué des poids des arêtes de l'arbre classées par ordre croissant.

Nous pouvons alors définir $T \preceq T'$ ssi $m(T) \leq_{lex} m(T')$.

L'ordre défini ci-dessus n'est qu'un ordre partiel, car un même mot peut-être associé à deux arbres, qui ont alors le même poids.

Il est facile de constater que l'algorithme de Kruskal construit un arbre minimal pour l'ordre \preceq . Cependant nous avons un résultat beaucoup plus fort.

Propriété 1 *Un arbre T est de poids minimum ssi il est minimal pour l'ordre \preceq .*

Cette dernière formulation montre que l'arbre obtenu ne minimise pas que la somme des poids.

Corollaire 3 *L'arbre de poids minimum est aussi celui qui minimise le produit des poids des arêtes.*

Pour quelle autres fonctions que la somme et le produit, le résultat du corollaire est-il valide ?

2.1.2 Exercices

1. Proposer un algorithme qui calcule un deuxième arbre recouvrant de poids minimum.

2. Maintenance dynamique d'un arbre recouvrant de poids minimum. Ceci peut servir lorsqu'une arête ou liaison est cassée, ou encore lorsque sa valuation augmente.
3. * Ecrire un algorithme qui vérifie qu'un arbre donné est bien un arbre recouvrant de poids minimum.
4. Peut-on minimiser aussi le diamètre de l'arbre recouvrant de poids minimum. On commencera par montrer qu'il existe des solutions optimales ayant des diamètres différents.
5. Ecrire un algorithme qui calcule s'il existe un arbre recouvrant de poids minimum de G dans lequel deux sommets fixés a et b sont pendants dans l'arbre. (i.e. $d_T(a) = d_T(b) = 1$).
6. Dans le cas où il n'existe pas un tel arbre de poids minimum, écrire un algorithme qui calcule le meilleur arbre recouvrant (s'il en existe un) vérifiant cette contrainte sur a et b .
7. Mêmes questions lorsqu'on remplace les sommets a et b , par un ensemble de sommets $S \subseteq X$.
8. Rappelez la condition pour qu'il existe plusieurs arbres recouvrants de poids minimum. Dans un tel cas écrire un algorithme qui calcule parmi ces arbres recouvrants de poids minimum, celui dont l'arête de plus fort poids est minimale.
9. * Comment minimiser le nombre de sommets pendants d'un arbre recouvrant de poids minimum ?

Chapitre 3

Graphes orientés

3.1 Définitions de base sur les graphes orientés

Nous noterons un graphe orienté $G = (X, U)$, où X est un ensemble **fini** de sommets et $U \subseteq X^2$ un ensemble d'arcs. Un arc (x, y) est donc constituée de deux sommets : une origine x et une extrémité terminale y . Les graphes orientés considérés dans ce cours seront, sauf mention contraire, **simples** : sans boucle (arête de type xx) et sans arc multiple (plusieurs arcs entre deux sommets).

$H = (Y, V)$ est un **sous-graphe induit** de $G = (X, U)$, si $Y \subseteq X$ et si $V = \{(x, y) \in U \mid x, y \in Y\}$.

$H = (Y, V)$ est un **sous-graphe partiel** de $G = (X, U)$, si $Y \subseteq X$ et si $V \subseteq U \cap Y^2$.

En général $|X| = n$ et $|E| = m$. Le **degré** d'un sommet noté $d^+(x)$ (resp. $d^-(x)$) est le nombre d'arcs sortant (resp. rentrant) de x . Un sommet **pendant** dans un graphe, est un sommet x tel que : $d(x) = 1$.

Une **chemin** de longueur k est un graphe :

$P = (\{x_0, x_1, \dots, x_k\}, \{(x_0, x_1), (x_1, x_2), \dots, (x_{k-1}, x_k)\})$ ayant $k + 1$ sommets et k arcs, noté P_k . Ce graphe s'appelle un chemin de longueur k joignant x_0 à x_k .

On confond souvent le chemin avec la séquence : $[x_0, x_1, \dots, x_k]$. Un graphe réduit à un sommet et sans arc est une chemin de longueur 0.

Un **circuit** de longueur k est un graphe :

$C = (\{x_1, \dots, x_k\}, \{(x_1, x_2), (x_2, x_3), \dots, (x_{k-1}, x_k), (x_k, x_1)\})$ ayant k sommets et k arcs, noté C_k . On confond souvent le circuit avec l'une des sé-

quences : $[x_1, x_2, \dots, x_k, x_1]$ (à une permutation circulaire près).

Avec de telles définitions, un chemin (resp. un circuit) ne passe qu'une fois par un sommet, cela correspond à une chaîne (resp. circuit) élémentaire chez certains auteurs.

On appelle **marche orientée** dans $G = (X, U)$ une séquence non nulle $M = x_0 u_1 x_1 u_2 \dots u_k x_k$ alternant sommets et arcs telle que :

$\forall 1 \leq i \leq k$, l'origine de u_i est x_{i-1} et son extrémité terminale x_i . On dit que la marche orientée est de longueur k et qu'elle joint x_0 à x_k .

Lorsque $x_0 = x_k$ on dit que la marche est **fermée** ou un **pseudocycle**. Si le graphe est simple, nous pouvons omettre les arcs et noter la marche par $[x_0, x_1, \dots, x_k]$.

Lorsque tous les arcs de la marche orientée sont différents on dit que la marche est une **piste**¹, lorsque tous les sommets sont distincts la marche est un chemin.

Pour les graphes orientés, les choses se compliquent un peu, car nous retrouvons les objets du chapitre précédents dès que l'on considère le graphe non orienté sous-jacent, sur lequel on peut identifier des chaînes, cycles et autres marches.

Proposition 4 *S'il existe une marche orientée finie joignant x à y dans G , alors il existe un chemin fini joignant x à y dans G , en outre les sommets et les arcs de ce chemin appartiennent à la marche.*

Preuve: Soit M la marche orientée de G joignant x à y . Il existe donc une marche orientée de longueur minimum joignant x à y dans G . Notons M' , cette marche orientée. Si elle n'est pas élémentaire, on peut extraire une marche orientée de longueur strictement plus petite, d'où la contradiction. \square

N.B. Cette preuve repose sur la finitude du graphe, car si M est de longueur infinie, il n'est pas sûr que M' la marche extraite soit de longueur strictement inférieure.

3.2 Arborescences

Une **racine** d'un graphe orienté $G = (X, U)$ est un sommet $r \in X$ tel que $\forall x \in X, x \neq r$, il existe un chemin de r à x .

1. Ces définitions de chemins, marches, pistes et autres circuits ne sont pas standard dans la littérature.

Théorème 4 *Pour un graphe orienté $G = (X, U)$ avec $|X| \geq 2$, les 6 conditions suivantes sont équivalentes et caractérisent les arborescences :*

1. G est sans cycle et admet une racine
2. G possède une racine et a $n - 1$ arcs
3. G possède une racine r et $\forall x \in X$, il existe un chemin unique de r à x
4. G possède une racine et est minimal avec cette propriété (tout retrait d'un arc de G supprime la propriété)
5. G est connexe et il existe un sommet $r \in X$, avec $d^-(r) = 0$ et $\forall x \in X$, $x \neq r$, $d^-(x) = 1$
6. G est sans cycle et il existe un sommet $r \in X$, avec $d^-(r) = 0$ et $\forall x \in X$, $x \neq r$, $d^-(x) = 1$

Preuve: (2) \rightarrow (3) :

L'existence d'une racine implique la connexité de G , le nombre d'arcs permet de conclure que le graphe non orienté sous-jacent est un arbre. D'après le théorème sur les arbres, $\forall x \in X$, $x \neq r$ le chemin de r à x est nécessairement unique.

(3) \rightarrow (4) : trivial.

(4) \rightarrow (5) :

□

Ainsi dans une arborescence, tout sommet sauf la racine admet un prédécesseur unique ($d^-(x) = 1$) et donc on peut représenter une arborescence à l'aide de la fonction Parent : $X \rightarrow X$.

3.3 Exercices :

1. On appelle graphe fonctionnel un graphe orienté qui vérifie : $\forall x \in G$, $d^+(x) = 1$. Quelle est la structure de ces graphes ?
2. On considère un graphe G orienté connexe tel que : $\forall x \in G$ $d^+(x) = d^-(x)$, montrer que G est fortement connexe. Que peut-on dire d'autre sur la structure de G ?

N.B. La finitude du graphe est indispensable pour ce résultat. Si l'on considère \mathcal{Z} muni de l'ordre strict pour lequel ($\forall x$, $d^-(x) = d^+(x) = 1$), le résultat est faux.

Chapitre 4

Parcours de graphes

4.1 Parcours générique

Un algorithme de parcours dans un graphe peut se formaliser à l'aide de deux ensembles de sommets OUVERTS et FERMES, le premier contenant les sommets à explorer et le deuxième les sommets déjà explorés. L'exploration d'un sommet consistant à examiner tous les voisins d'un sommet : visite de tous les arcs sortants du sommet.

ParcoursGénérique(G, x_0)

Données: un graphe orienté $G = (X, U)$, un sommet $x_0 \in X$

Résultat: une arborescence de chemins issus de x_0

$OUVERTS \leftarrow \{x_0\}$

$FERMES \leftarrow \emptyset$

$Parent(x_0) \leftarrow NIL$

$\forall y \neq x_0, Parent(y) \leftarrow y$

tant que $OUVERTS \neq \emptyset$ **faire**

$z \leftarrow Chois(OUVERTS)$
 $Ajout(z, FERMES)$
 Explorer(z)
 $Retrait(z, OUVERTS)$

Explorer(z)
pour *Tous les voisins y de z faire*
 si $y \in FERMES$ **alors**
 └ Ne rien faire
 si $y \in OUVERTS$ **alors**
 └ Ne rien faire
 sinon
 └ $Ajout(y, OUVERTS)$
 └ $Parent(y) \leftarrow z$

Théorème 5 *L'algorithme précédent calcule en $O(n + m)$ une arborescence de racine x_0 recouvrant l'ensemble des sommets atteignables à partir de x_0 .*

Preuve: Il est facile de vérifier que l'algorithme précédent vérifie bien les invariants suivants :

Invariants :

- La fonction Parent définit une arborescence de racine x_0 ,
- $\forall x \in OUVERTS$, il existe un chemin de x_0 à x .

Ainsi à la fin du parcours, l'ensemble des sommets FERMES est égal à l'ensemble des sommets atteignables dans le graphe G à partir de x_0 . Cet ensemble est décrit à l'aide de la fonction Parent. Un sommet est exploré au plus une fois et donc l'algorithme est en $O(n + m)$ si le graphe est représenté par ses listes d'adjacence. \square

Ce parcours s'applique aux graphes non orientés et orientés. Le déroulement d'un parcours permet d'ordonner (numéroter) les sommets d'un graphe. Ces ordres sont liés à la gestion de l'ensemble des sommets OUVERTS et à la fonction de choix du prochain sommet à explorer. Cet ensemble peut être géré comme une pile (Parcours en profondeur) ou une File (Parcours en largeur).

Lorsque tous les sommets du graphe ne sont atteignables à partir de x_0 si l'on veut parcourir tout le graphe, il suffit d'ajouter les instructions suivantes et de représenter l'ensemble des sommets FERMES à l'aide d'un tableau de booléens :

pour tous les $v \in X$ faire
 └ $Ferme(x) \leftarrow Faux$
pour tous les $v \in X$ faire
 └ **si** $Ferme(x) = Faux$ **alors**
 └ $ParcoursGénérique(G, x)$

Dans ce cas l'algorithme calcule une forêt d'arborescences recouvrante de G .

4.1.1 Parcours en largeur

On obtient un parcours en largeur dès que l'on gère l'ensemble des sommets OUVERTS comme une file (premier entré, premier sorti).

4.2 Parcours en profondeur

Défini en premier par Tremeaux 1882 puis par Tarry 1895 pour des problèmes de parcours dans les labyrinthes. Redécouvert en 1972 par Tarjan et Hopcroft afin de concevoir des algorithmes de résolution des problèmes de recherche dans un graphe : le calcul des composantes fortement connexes, celui des composantes 2-arêtes connexes, le test de planarité ... On obtient un parcours en profondeur dès que l'on gère l'ensemble des sommets OUVERTS comme une pile (dernier entré, premier sorti). Cette gestion d'une pile se prête à une écriture récursive du programme, comme suit :

```

DFS(G) :
  forall v ∈ X do
    ⊥ Ferme(x) ← Faux
  forall v ∈ X do
    ⊥ si Ferme(x) = Faux alors
      ⊥ Explorer(G, x)
  Explorer(G, x) :
  Ferme(x) ← Vrai ;
  pre(x) ;
  forall xy ∈ U do
    ⊥ si Ferme(y) = Faux alors
      ⊥ Explorer(G, y)
  post(x) ;

```

Et les deux fonctions suivantes utilisant deux variables : comptpre et comptpost étant initialisées à 1.

```

pre(x) :
pre(x) ← comptpre ;
comptpre ← comptpre + 1 ;

post(x) :
post(x) ← comptpost ;
comptpost ← comptpost + 1 ;

```

En utilisant les propriétés des numérotations $pre(x)$ et $post(x)$, il est possible de reconstituer le fonctionnement de la pile qui a permis de gérer l'ensemble de sommets OUVERTS. Ce parcours a permis d'écrire des algorithmes linéaires (en $O(n + m)$) pour la recherche des composantes fortement

connexes d'un graphe orienté, de recherche des composantes 2-arête connexes d'un graphe non orienté, ainsi qu'un algorithme qu'un test de planarité.

4.2.1 Un algorithme simple pour le calcul des composantes fortement connexes

Théorème 6 *Si $post(a) < post(b)$ et $ab \in U$, alors il existe nécessairement un chemin monotone décroissant de b à a dans G , qui n'utilise que des sommets de $[a, b]_{post}$.*

Preuve: Si le parcours en profondeur avait utilisé l'arc ab , nécessairement le traitement de b se serait terminé avant le traitement de a et nous aurions $post(b) < post(a)$.

Ceci implique que lorsque qu'on termine le traitement du sommet a , le sommet b a déjà été visité et comme son travail n'est pas achevé, a est un descendant de b . En outre les sommets entre b et a de ce chemin (N.B. ce chemin est dans l'arborescence associée au parcours) appartiennent nécessairement à $[a, b]_{post}$, d'où le résultat. \square

Dans ce cas a et b appartiennent à la même composante fortement connexe de G .

C'est peut-être une caractérisation des parcours en profondeur ?

Considérons le mot γ associé au parcours en profondeur dans lequel chaque sommet x apparaît exactement deux fois, la première lorsqu'il est exploré $pre(x)$ et la dernière $der(x)$ lorsqu'on dépile x (la fin de l'exploration).

On va construire sur γ une famille d'intervalles $\mathcal{J}(G)$:

1. tous les intervalles $[x, x]$ dont les extrémités sont les deux occurrences du sommet x .
2. À chaque arc $ab \in U$ tel que $post(a) < post(b)$, on associe l'intervalle $[der(a), der(b)]$

Deux intervalles I, J se rencontrent si soit

1. ils se chevauchent (i.e. $I \cap J \neq \emptyset$ et si $I \not\subseteq J$ et $J \not\subseteq I$).
2. l'un est inclus dans l'autre, mais ils ont une extrémité en commun

N.B. L'intersection peut être réduite à un sommet.

Lemme 6 *Les sommets extrémités de deux intervalles qui se rencontrent appartiennent à la même composante fortement connexe de G .*

Preuve:

Il est bien connu que les composantes de chevauchement d'une famille d'ensembles, ordonnées par inclusion constituent une arborescence, appelée l'arborescence de chevauchement.

Théorème 7 *Les composantes fortement connexes de G correspondent aux composantes de chevauchement de la famille $\mathcal{J}(\mathcal{G})$.*

Preuve: Il est facile de vérifier que les composantes fortement connexes de G sont des intervalles de l'ordre *post*.

Considérons deux intervalles de retour $[a, b]$ et $[c, d]$ qui se chevauchent. On supposera : $a <_{post} c <_{post} b <_{post} d$.

Dans l'arbre du parcours en profondeur a est un descendant de b et comme c est dépilé entre les deux, nécessairement c est aussi un descendant de b . Il existe donc un chemin dans l'arbre du parcours de d vers a et un chemin $[a, b, c, d]$ dans G .

... \square Afin de calculer ces composantes il suffit d'exécuter un petit automate qui analyse ces intervalles comme un mot de parenthèses.

Définition 2 *Une **extension linéaire** d'un graphe sans circuit $G = (X, U)$ est un ordre total sur les sommets de G , noté τ , vérifiant :*

$$\forall x, y \in X, xy \in U \text{ implique } x \leq_{\tau} y.$$

Certains auteurs appellent une extension linéaire un **tri topologique** ou encore **un ordre total compatible**.

Corollaire 4 *Si G est un graphe sans circuit, alors le parcours en profondeur appliqué sur G vérifie la propriété : L'ordre inverse de dépilement (noté $post^d$) est une extension linéaire de G .*

Une variante :

Corollaire 5 *$post^d$ est une extension linéaire de G/P le graphe G quotienté par la partition en composantes fortement connexes de G .*

4.2.1.1 Une partition des arcs de G

Comme tous les parcours, un parcours en profondeur permet de construire une forêt d'arborescences. On distingue trois types d'arcs : les arcs de retour, les arcs traversiers et enfin les arcs de transitivité de la forêt d'arborescences. Les ordres pre et $post$ permettent de les distinguer, comme l'indiquent les trois équivalences suivantes :

- xy arc de retour ssi $pre(y) \leq pre(x)$ and $post(x) \leq post(y)$
- xy arc traversier ssi $pre(y) \leq pre(x)$ and $post(y) \leq post(x)$
- xy arc de transitivité de la forêt d'arborescences ssi $pre(x) \leq pre(y)$ and $post(y) \leq post(x)$
- Le cas où $pre(x) \leq pre(y)$ and $post(x) \leq post(y)$ étant impossible par le principe du parcours en profondeur.

Ces trois types d'arcs partitionnent les arcs du graphe.

Preuve: Considérons deux sommets $x, y \in X$ et supposons $xy \in U$. Deux cas sont possibles :

1. $pre(x) < pre(y)$. Mais alors l'exploration de y se terminera donc avant celle de x et nous avons bien : $post^d(x) < post^d(y)$.
2. $pre(y) < pre(x)$. Comme il n'existe pas de chemin de y à x , car sinon G aurait un circuit, nous avons que l'exploration de y se terminera avant celle de x . Et donc $post^d(x) < post^d(y)$.

□

Ce théorème peut aussi s'écrire comme suit :

Théorème 8 *Après un parcours en profondeur sur un graphe G les 3 propriétés suivantes sont équivalentes :*

- i** G est sans circuit
- ii** Il n'y a pas d'arc de retour
- iii** $post^d$ est une extension linéaire de G

Le parcours en profondeur permet donc de calculer une extension linéaire d'un graphe sans circuit. Il existe d'autres algorithmes, le plus classique étant celui basé sur la remarque que tout graphe sans circuit admet une source (i.e. un sommet tel que $d^-(x) = 0$). Si on modifie un parcours générique comme suit :

ExtensionLineaire(G, S)

Données: Un graphe orienté $G = (X, U)$ sans circuit, S l'ensemble des sources de G

Résultat: Une numérotation des sommets

$OUVERTS \leftarrow \{S\}$;

$FERMES \leftarrow \emptyset$;

$compteur \leftarrow 1$;

tant que $OUVERTS \neq \emptyset$ **faire**

$z \leftarrow Chois(OUVERTS)$
$numero(z) \leftarrow compteur$;
$compteur \leftarrow compteur + 1$;
$Ajout(z, FERMES)$
Explorer(z)
$Retrait(z, OUVERTS)$

Explorer(z)

pour Tous les successeurs y de z **faire**

$d^-(y) \leftarrow d^-(y) - 1$;
si $d^-(y) = 0$ alors
└─ $Ajout(y, OUVERTS)$

Cette procédure fournit une numérotation des sommets qui est une extension linéaire de G . Si l'on gère l'ensemble des sommets $OUVERTS$ comme une File, l'extension linéaire obtenue est appelée numérotation par niveau, ou par rang.

4.2.1.2 Algorithme de Tarjan 1972

Il suffit d'ajouter deux structures de données un tableau de booléens Pile définissant la présence d'un sommet dans une pile appelée Resultat de transformer la fonction d'exploration du parcours en profondeur comme suit :

```

DFS( $G$ ) :
  comptpre  $\leftarrow$  1;
  Resultat  $\leftarrow$   $\emptyset$ ;
  forall  $x \in X$  do
     $\lfloor$  Ferme( $x$ )  $\leftarrow$  Faux; Pile( $x$ ) = Faux
  forall  $x \in X$  do
     $\lfloor$  si Ferme( $x$ ) = Faux alors
       $\lfloor$  Explorer( $G, x$ )
  Explorer( $G, x$ ) :
    Empiler(Resultat,  $x$ )
    Pile( $x$ ) = Vrai
    Ferme( $x$ )  $\leftarrow$  Vrai;
    pre( $x$ )  $\leftarrow$  comptpre; comptpre  $\leftarrow$  comptpre + 1;
    racine( $x$ )  $\leftarrow$  pre( $x$ );
    forall  $xy \in U$  do
       $\lfloor$  si Ferme( $y$ ) = Faux alors
         $\lfloor$  Explorer( $G, y$ );
         $\lfloor$  racine( $x$ )  $\leftarrow$  min{racine( $x$ ), racine( $y$ )};
      sinon
         $\lfloor$  si Pile( $y$ ) = Vrai alors
           $\lfloor$  racine( $x$ )  $\leftarrow$  min{racine( $x$ ), racine( $y$ )}
    si racine( $x$ ) = pre( $x$ ) alors
       $\lfloor$  Dépiler Resultat jusqu'à  $x$  inclus et écrire ces sommets;
       $\lfloor$  mettre à jour le tableau Pile

```

Les composantes fortement connexes se repèrent récursivement avec le test $racine(x) = pre(x)$ à la fin de l'exploration de x . Plus précisément voici les invariants placés lorsqu'on a fini l'exploration d'un sommet x .

1. $\forall z$ tel que $pre(x) \leq pre(z)$ et $Pile(z) = Vrai$ nous avons : $racine(z) < pre(z)$
2. $\forall z$ tel que $Pile(z) = Vrai$ il existe un chemin de z au sommet y tel que $pre(y) = racine(z)$ n'utilisant que des sommets de Resultat.

Théorème 9 *Quand on trouve x t.q. $pre(x) = racine(x)$, $A = \{z \in X | pre(x) \leq pre(z) \text{ et } Pile(z) = Vrai\}$ constitue l'ensemble des sommets de la composante fortement connexe qui contient x .*

Preuve:

En utilisant les invariants il est facile de vérifier que $\forall z \in A$ il existe un chemin allant de z à x . Par ailleurs quand on a fini d'explorer x ce qui reste dans Resultat après x est constitué de descendants de x . Et donc ainsi $G(A)$ est fortement connexe.

La maximalité de A est aussi facile à vérifier.

4.2.1.3 Algorithme de Kosaraju 1978, Sharir 1981

1. Exécuter un parcours en profondeur sur G .
2. Faire un autre parcours en profondeur sur G^- avec pour ordre initial l'ordre *post*^d
3. Les arbres de la forêt recouvrante de G^- , sont les composantes fortement connexes de G

4.2.1.4 Implémentations

Dans un parcours en profondeur nous avons le choix de parcourir les successeurs d'un sommet dans un ordre qui peut soit provenir :

- d'un parcours précédent comme dans l'algorithme de Kosaraju et Sharir.
- d'un ordre explicite de préférence sur les liens (utilisé pour les algorithmes d'héritage en programmation objet)
- Un parcours en profondeur traverse chaque arête une fois, alors qu'un parcours en largeur examine tout le voisinage d'un sommet à la fois voisinage noté $N(x)$

4.2.1.5 Parcours en largeur lexicographique

Il suffit de prendre $T = \{1, 2, \dots, n\}^*$ (l'ensemble des mots construits avec les nombres 1,2 jusqu'à n), et de choisir comme sommet à explorer celui dont l'étiquette est maximale lexicographiquement.

Cet algorithme s'utilise sur les graphes non orientés.

La fonction Explorer devenant :

Explorer(z)*numero(z)* → *numerocourant* − 1 ;*numerocourant* → *numerocourant* − 1 ;**pour** *Tous les voisins y de z faire*

si <i>y</i> ∈ <i>FERMES</i> alors └ Ne rien faire
si <i>y</i> ∈ <i>OUVERTS</i> alors <i>d(y)</i> → <i>d(y)</i> • <i>numero(z)</i>
sinon <i>d(y)</i> → <i>d(y)</i> • <i>numero(z)</i> └ <i>Ajout(y, OUVERTS)</i>

le symbole "•" représentant l'opération de concaténation, on étiquette ainsi les sommets du graphe avec des mots sur $\{1, 2, \dots, n\}^*$.

La variable globale *numerocourant* doit être initialisée à *n* dans les initialisations de la procédure principale.

Chapitre 5

Algorithmes de plus courts chemins

5.1 Introduction

On considère un graphe $G = (X, U)$ orienté et l'on suppose les arcs munis d'étiquettes, i.e. une valuation $\omega : U \rightarrow R$.

Le problème de la recherche d'une plus court *chemin* dans un graphe orienté a de très nombreuses applications pratiques, telles :

- Routages de paquets dans les réseaux, routages de véhicules dans les réseaux de transports
- Diamètre d'un réseau de télécommunications (qualité de service)
- Problèmes de Transport
- Jeux (graphe dont les sommets sont les états du jeu et les arcs les transitions légales)
- Investissements, ordonnancements
- Navigation (routeurs de course au large : Vendée Globe Challenge, Route du Rhum, ...)

En général la valuation d'un chemin est la somme des valuations des arcs qui constituent le chemin.

5.2 Les différents problèmes

Pour un graphe $G = (X, U)$ orienté dont les arcs sont valués

- 1. Etant donné deux sommets a et b , trouver une plus courte marche allant de a jusqu'à b dans G .
- 2. Etant donné un sommet a trouver pour chaque $x \in X$ une plus courte marche allant de a jusqu'à x dans G .
- 3. Pour tout $x, y \in G$, trouver une plus courte marche allant de x jusqu'à y .

Dans l'énumération ci-dessus, la complexité du problème est croissante. En effet un algorithme qui résout le problème 3, permet de résoudre le problème 2 et un algorithme qui résout le problème 2 permet de résoudre le problème 1.

Cependant lorsque les valuations choisies sont de signe quelconque, la solution au problème 1 peut-être une marche infinie comprenant un circuit de valeur négative (circuit dit **absorbant**).

Exemple : $valuation(ij) = cout(ij) + taxe - subvention$

Il est tentant de transformer le problème en :

Etant donné deux sommets a et b , trouver le plus court chemin allant de a jusqu'à b dans G .

Mais alors le problème devient NP-difficile.

5.2.1 Algorithme de Dijkstra [17]

Algorithme de Dijkstra 1959

Données: un graphe orienté $G = (X, U)$, une fonction de coût
 $\omega : U \rightarrow \mathcal{R}^+$

Résultat: une arborescence de chemins issus de x_0

$OUVERTS \leftarrow \{x_0\}$

$FERMES \leftarrow \emptyset$

$Parent(x_0) \leftarrow NIL$

$\forall y \neq x_0, Parent(y) \leftarrow y$

$d(x_0) \leftarrow 0$

$\forall y \neq x_0, d(y) \leftarrow +\infty$

tant que $OUVERTS \neq \emptyset$ **faire**

Choisir un sommet $z \in OUVERTS$ tel que $d(z) = \min_{y \in OUVERTS} \{d(y)\}$ $Ajout(z, FERMES)$ Explorer(z) $Retrait(z, OUVERTS)$

```

Explorer(z)
pour Tous les voisins y de z faire
  | si y ∈ FERMES alors
  | | Ne rien faire
  | sinon
  | | si y ∈ OUVERTS alors
  | | | si d(z) + ω(z, y) < d(y) alors
  | | | | Parent(y) ← z
  | | | | d(y) ← d(z) + ω(z, y)
  | | | sinon
  | | | | Ajout(y, OUVERTS)
  | | | | Parent(y) ← z
  | | | | d(y) ← d(z) + ω(z, y)
  | |
  |

```

Théorème 10 Lorsque la valuation ω est à valeurs positives, l'algorithme calcule bien une arborescence des plus courts chemins issus de x_0 .

Preuve:

Considérons les invariants principaux de l'algorithme.

Invariants

1. $\forall x \in OUVERTS$, il existe un chemin μ de x_0 à x tel que, $d(x) = \omega(\mu)$
2. À la fin de la i ème exploration d'un sommet, $\forall x \in OUVERTS \cup FERMES$, $d(x)$ est égale à la valuation minimale d'un chemin de G de x_0 à x n'utilisant que des sommets $FERMES$ sauf éventuellement x .
3. $\forall u \in FERMES$ and $\forall v \in OUVERTS$, $d(u) \leq d(v)$.

L'invariant 1 se montre facilement par récurrence, montrons l'invariant 2 aussi par récurrence. Pour se faire indexons par i tous les objets de l'algorithme à la fin de l'exploration du i ème sommet.

À l'initialisation lorsque $i = 0$, l'invariant est trivialement vrai. Supposons le maintenant vrai jusqu'à l'exploration du $(i-1)$ ème sommet. Soit x le sommet exploré à la i ème itération.

Considérons $z \in OUVERTS_i \cup FERMES_i$, il y a plusieurs cas possibles.

- Soit $xz \notin U$, mais alors il n'existe pas de chemin joignant x_0 à z n'utilisant que des sommets de $FERMES_i$ sauf éventuellement z . Comme $d_i(z) = d_{i-1}(z)$, l'hypothèse de récurrence permet de conclure.

- Soit $xz \in U$ et $z \notin OUVERTS_{i-1} \cup FERMES_{i-1}$. Dans ce cas le seul chemin joignant x_0 à z n'utilisant que des sommets de $FERMES_i$ sauf éventuellement z passe par x et l'algorithme affecte bien à $d_i(z)$ la valeur de ce chemin.
- Soit $xz \in U$ et $z \in OUVERTS_{i-1}$. Quand le sommet x est fermé à la i ème étape, il existe un chemin joignant x_0 à z n'utilisant que des sommets de $FERMES_i$ sauf éventuellement z passant par x . L'algorithme prend en compte la valeur de ce chemin pour calculer $d_i(z)$, on peut conclure grâce à l'hypothèse de récurrence.
- Soit $xz \in U$ et $z \in FERMES_{i-1}$. Dans ce cas l'algorithme ne fait rien (i.e. $d_i(z) = d_{i-1}(z)$) montrons que c'est justifié. En effet supposons qu'il existe un chemin $\mu = [x_0, x_1, \dots, x_k, x_{k+1} = x, z]$ allant de x_0 à z tel que $\omega(\mu) < d_i(z)$.
Soit $j \in [0, k]$ l'indice du dernier sommet de μ qui ait été exploré avant que z ne soit exploré à l'étape h . Par hypothèse de récurrence $d_h(x_{j+1}) = \omega(\mu[x_0, x_{j+1}])$. Comme les valuations des arcs sont positives, on en déduit : $d_h(x_{j+1}) < d_h(z)$ ce qui contredit l'hypothèse sur j , car l'algorithme aurait du explorer x_{j+1} avant z .

□

5.2.2 Algorithme de Bellman - Ford

Données: un graphe orienté $G = (X, U)$, une fonction de coût
 $\omega : U \rightarrow \mathcal{R}$

Résultat: une arborescence des plus courts chemins issus de x_0 ou un circuit négatif

$Parent(x_0) \leftarrow NIL \ \forall y \neq x_0, \ Parent(y) \leftarrow y$
 $d(x_0) \leftarrow 0; \ \forall y \neq x_0, \ d(y) \leftarrow \text{infini}$

répéter

$\forall xy \in U$	
si $d(y) > d(x) + \omega(xy)$ alors	
└ $d(y) \leftarrow d(x) + \omega(xy)$ et $parent(y) \leftarrow x$	

jusqu'à $|X| - 1$ fois

pour chaque $xy \in U$ **faire**

si $d(y) > d(x) + \omega(xy)$ alors	
└ Il existe un circuit négatif	

- Complexité en $O(n.m)$
- Invariant :
 $\forall i, x,$
 $d_i(x)$ = la longueur minimum d'une marche orientée de s à x ayant au plus i arcs.
- A la fin de l'algorithme la fonction père représente une arborescence des plus courts chemins.

5.2.3 L'algorithme A*

Si la fonction choix fournit un sommet z vérifiant :

$d(z) + h(z) = \min_{y \in OUVERTS} \{d(y) + h(y)\}$ où $h(y)$ est une information "heuristique" sur la distance qui reste à parcourir.

et si l'instruction : "Ne rien faire" de l'algorithme de Dijkstra est remplacée par :

```

si  $d(z) + \omega(z, y) < d(y)$  alors
  |  $Parent(y) \leftarrow z$ 
  |  $d(y) \leftarrow d(z) + \omega(z, y)$ 
  |  $Ajout(y, OUVERTS); Retrait(y, FERMES)$ 

```

On suppose que cette fonction $h(y)$ est une information disponible en chaque sommet du graphe. L'usage de cet algorithme est adapté au cas où l'on cherche un plus court chemin d'un sommet x_0 à un sommet t . La valeur de $h(x)$ pour un sommet x donné, étant une estimation de la distance qu'il reste à parcourir pour aller de x à t .

Dans les deux instanciations précédentes, le goulot d'étranglement de complexité provient de la gestion de l'ensembles des OUVERTS. Il faut utiliser une structure de données qui permette le calcul du minimum en $O(\log n)$ ou mieux.

5.2.3.1 Algorithme de Prim

L'algorithme de Prim que nous avons vu au chapitre sur les arbres recouvrants de poids minimum peut aussi se voir comme un algorithme de plus court chemin. Pour cela il suffit de remarquer que le graphe est non orienté, les arêtes sont valuées et si le sommet à explorer est celui qui vérifie :

$$d(z) = \min_{y \in OUVERTS} \{d(y)\}$$

et si l'exploration devient :

```

Explorer(z)
pour Tous les voisins  $y$  de  $z$  faire
  si  $y \in \text{FERMES}$  alors
    | Ne rien faire
  sinon
    | si  $y \in \text{OUVERTS}$  alors
      | | si  $\omega(z, y) < d(y)$  alors
        | | |  $\text{Parent}(y) \leftarrow z$ 
        | | |  $d(y) \leftarrow \omega(z, y)$ 
      | sinon
        | |  $\text{Ajout}(y, \text{OUVERTS})$ 
        | |  $\text{Parent}(y) \leftarrow z$ 
        | |  $d(y) \leftarrow \omega(z, y)$ 

```

L'algorithme ci-dessus calcule un arbre de poids minimum de G et c'est une implémentation de l'algorithme de Prim. En outre, il est facile de montrer que l'arborescence obtenue est celle des distances minimum (suivant la distance du max) issue de x_0 .

L'algorithme de Prim va donc visiter les sommets suivant un certain ordre σ . Par convention lorsque $xy \notin E$, on supposera $\omega(x, y) = \infty$

Définition 3 *Étant donné un ordre σ sur V vérifie la propriété Prim, si $\forall a, b, c \in V$, si $a <_{\sigma} b <_{\sigma} c$ et $\omega(a, b) > \omega(a, c)$, alors il existe un sommet d tel que $d <_{\sigma} b$ et $\omega(d, b) \leq \omega(a, c)$.*

Il est facile de vérifier qu'un ordre qui vérifie la condition Prim correspond à un parcours générique sur G .

Théorème 11 *Pour un graphe $G = (V, E)$, un ordre σ sur V peut-être obtenu à l'aide d'un parcours de type Prim de G ssi σ vérifie la propriété (Prim).*

Preuve: La condition est nécessaire, car si l'on considère le b' le prédécesseur de b dans σ , l'algorithme de Prim choisit une arête db du cocycle $\theta(b')$ de coût minimal et donc $\omega(d, b) \leq \omega(a, c)$.

Montrons que la condition est suffisante, et considérons la première différence x (en fonction de σ). Comme σ correspond à un parcours générique il

existe a , $a <_{\sigma} x$ avec $ax \in E$. Cependant comme c'est la première différence, il existe t , $t <_{\sigma} x$ et $y >_{\sigma} x$ avec $ty \in E$ tel que : $\omega(t, y) < \omega(a, x)$.

Mais en appliquant la condition sur le triplet t, x, y , il existe nécessairement t' , tel que $t <_{\sigma} x$ avec $\omega(t', x) \leq \omega(t, y)$. Ce qui contredit que x est la première différence. \square

5.3 Un cadre général

Tous les algorithmes de plus courts chemins précédemment présentés peuvent être vus comme des implémentations d'un algorithme générique, défini comme suit. On considère un graphe $G = (X, U)$ orienté et l'on suppose les arcs munis d'étiquettes, i.e. une valuation $\omega : U \rightarrow T$. où T est un ensemble muni d'une relation d'ordre total \ll , et de deux éléments distingués \top (resp. \perp) un unique élément maximal (resp. minimal).

On étend cette valuation aux chemins comme suit :

$$\mu = [x_1, \dots, x_k], \omega(\mu) = \oplus_{i=0}^{k-1} \omega(x_i, x_{i+1}).$$

La relation \oplus binaire associative sur T , étant interprétée suivant les exemples, comme :

- l'addition dans le cas usuel où T est l'ensemble des entiers, des rationnels ou des réels,
- le maximum,
- un produit ou la concaténation dans un monoïde,
- ...

La relation d'ordre total \ll s'interprétant comme :

- l'ordre usuel quand T est l'ensemble des entiers,
- l'ordre lexicographique lorsque T est un langage,
- ...

Algorithme de Parcours Générique**Données:** un graphe orienté $G = (X, U)$, une fonction de coût

$$\omega : U \rightarrow T$$

Résultat: une arborescence de chemins issus de x_0 $OUVERTS \leftarrow \{x_0\}$ $FERMES \leftarrow \emptyset$ $Parent(x_0) \leftarrow NIL$ $\forall y \neq x_0, Parent(y) \leftarrow y$ $d(x_0) \leftarrow \perp$ (l'élément minimal de T) $\forall y \neq x_0, d(y) \leftarrow \top$ (l'élément maximal de T)**tant que** $OUVERTS \neq \emptyset$ **faire** $z \leftarrow Chois(OUVERTS)$ Ajout($z, FERMES$) Explorer(z) Retrait($z, OUVERTS$)**Explorer(z)****pour** Tous les voisins y de z **faire** **si** $y \in FERMES$ **alors**

Ne rien faire

sinon **si** $y \in OUVERTS$ **alors** **si** $d(z) \oplus \omega(z, y) < d(y)$ **alors** $Parent(y) \leftarrow z$ $d(y) \leftarrow d(z) \oplus \omega(z, y)$ **sinon** Ajout($y, OUVERTS$) $Parent(y) \leftarrow z$ $d(y) \leftarrow d(z) \oplus \omega(z, y)$

5.4 Les graphes sans circuits

Supposons que les sommets soient numérotés selon une extension linéaire d'un graphe sans circuit G . Pour ce faire, nous pouvons utiliser soit un parcours en profondeur soit un parcours recherchant les sources.

```

pour  $i = 1$  à  $n$  faire
┌
  pour  $j = i + 1$  à  $n$  faire
  └  $d(j) = \min(d(j), d(i) + \omega(ij))$ 
    si modification de  $d(j)$  alors
    └  $Parent(j) \leftarrow i$ 

```

La complexité est en $O(n^2)$. Mais nous pouvons l'écrire de la façon suivante :

```

pour  $i = 1$  à  $n$  faire
└ Explorer( $i$ )

Explorer(i)
pour Tous les voisins  $j \geq i$  de  $i$  faire
└  $d(j) = \min(d(j), d(i) + \omega(ij))$ 
  si modification de  $d(j)$  alors
  └  $Parent(j) \leftarrow i$ 

```

Ce qui permet d'analyser d'obtenir la complexité : $\sum_{i=1}^n d(i) \in O(n + m)$.

Remarque : Si l'on remplace le minimum par maximum cet algorithme calcule une arborescence des plus longs chemins.

Chapitre 6

Fermeture transitive

- Dans cette section, nous considérons les graphes orientés comme des relations binaires, et l'on peut donc se poser la question de leur transitivité.
- Etant donné $G = (X, U)$ un graphe orienté, calculer $G^t = (X, U^t)$ le graphe de la plus petite relation transitive contenant U .

Propriété 2 $xy \in U^t$ ss'il existe un chemin de x à y dans G

Un premier algorithme en $O(n^3)$.

```
pour  $i = 1$  à  $n$  faire  
  pour  $j = 1$  à  $n$  faire  
    pour  $k = 1$  à  $n$  faire  
      si  $A[j, i] = 1$  et  $A[i, k] = 1$  alors  
         $A[j, k] \leftarrow 1$ 
```

Cet algorithme est connu sous le nom d'algorithme de Roy-Warshall.

L'invariant :

A la fin de la boucle en i ,

Pour tout $i' \leq i$, le graphe $G_{i'}$ n'admet pas de triplet du type $j i' k \in G_{i'}$ mais $j k \notin G_{i'}$.

6.1 Calcul via des produits de matrices

Soit A la matrice d'incidence de G .

- $A^2[i, j] = 1$ ss'il existe un chemin de longueur exactement 2 entre i et j dans le graphe.
- $(A + A^2)[i, j] = 1$ ss'il existe un chemin de longueur ≤ 2 entre i et j dans le graphe.
- Il suffit donc de calculer si l'on considère la fermeture réflexo-transitive (on ajoute toutes les boucles)
 $I + A + \dots + A^{n-1} = (I + A)^{n-1}$
- Calcul de X^n
- Le meilleur algorithme de produit de Matrices
- $O(n^\alpha \log n)$

Produit de matrices

- Considérons deux matrices carées (n, n) , A, B . Rappelons que le produit $C = A.B$ vérifie :
- $C[i, j] = \bigoplus_{k=1}^{k=n} A[i, k] \otimes B[k, j]$
- Les opérateurs \bigoplus et \otimes doivent être associatifs et distributifs.

6.2 Algèbres max, plus

Dans les applications ces opérateurs \bigoplus et \otimes , vont être surchargés comme suit :

- Existence d'un chemin, fermeture transitive
 Opérateurs logiques : $\bigoplus = \vee$, $\otimes = \wedge$
- Calcul du nombre de chemins :
 Opérateurs : $\bigoplus = +$ et $\otimes = *$ Dans les trois applications définies jusqu'à maintenant il faut partir d'une matrice A qui vérifie la convention $A[i, j] = a_{ij}$ la valuation de l'arc ij s'il existe et ∞ si ij n'est pas un arc de G .
- Nous pouvons aussi rentrer le calcul des plus courts chemins dans ce cadre.
 Opérateurs : $\bigoplus = \min$ et $\otimes = +$

Chapitre 7

Connexités

Un graphe G est **k-connexe** avec $k \geq 1$ si pour $\forall x, y \in G$, il existe k chaînes sommet-disjointes allant de x à y .

De même un graphe G est **k-arête-connexe** avec $k \geq 1$ si pour $\forall x, y \in G$, il existe k chaînes arête-disjointes allant de x à y .

Un point **d'articulation** (resp. un **isthme**) de G , est un sommet x (resp. une arête e) tel que $G - x$ (resp. $G - e$) est non-connexe.

Théorème 12 *Pour un graphe non orienté G connexe, les quatre conditions suivantes sont équivalentes :*

1. G est 2-connexe
2. G n'admet pas de point d'articulation
3. $\forall x, y, z \in G$, il existe une chaîne de x à y passant par z
4. $\forall x, y, z \in G$, il existe une chaîne de x à y évitant z

Chapitre 8

Flots dans les graphes

Algorithme de Recherche de chaîne améliorante

Données: un graphe orienté $G = (X, U)$, une fonction de capacité $c : U \rightarrow R_+$, deux sommets $s, t \in X$, un flot $\phi \geq 0$

Résultat: soit une chaîne améliorante de s à t , soit la maximalité de ϕ

$OUVERTS \leftarrow \{s\}$

$FERMES \leftarrow \emptyset$

tant que $OUVERTS \neq \emptyset$ **faire**

$x \leftarrow Chois(OUVERTS)$
 Explorer(x)
 Ajout(x, FERMES)
 Retrait(z, OUVERTS)

”Le flot ϕ est maximum”

Explorer(x)**pour** *Tous les successeurs y de x* **faire**

si	$y \notin FERMES \cup OUVERTS$ et $\phi(xy) < c(yx)$	alors
	si $y = t$ alors	
	STOP : "On a trouvé une chaîne améliorante de s à t "	
	sinon	
	Ajout(y , OUVERTS)	

pour *Tous les prédécesseurs y de x* **faire**

si	$y \notin FERMES \cup OUVERTS$ et $\phi(yx) > 0$	alors
	si $y = t$ alors	
	STOP "On a trouvé une chaîne améliorante de s à t "	
	sinon	
	Ajout(y , OUVERTS)	

Chapitre 9

Graphes planaires

9.1 Définitions de base

Les notions étudiées ci-après de planarité, sont parmi les notions les plus délicates à présenter de la théorie des graphes car elles utilisent de la topologie (notions de voisinage) de la géométrie (coordonnées, distances) et de la combinatoire (gestion de tous les associations possibles).

Cependant l'étude des graphes planaires est indispensable en imagerie, car toute image 2D peut être représentée par un graphe planaire.

Définition 4 *Un graphe $G = (X, E)$ non orienté est dit planaire s'il admet une représentation dans le plan dans laquelle les arêtes ne se croisent qu'aux sommets.*

Une représentation planaire d'un graphe est appelée un graphe planaire topologique ou plongement, on la notera $\phi(G)$. On posera $|X| = n$ et $|E| = m$.

Tous les graphes ne sont pas planaires, ainsi $K_{3,3}$ et K_5 ne le sont pas.

Un graphe planaire peut avoir plusieurs plongement planaires non isomorphes.

Dans un plongement planaire on associe à chaque sommet un point du plan et les arêtes sont des courbes continues du plan "arcs de Jordan". Cependant pour l'usage que nous en ferons nous pouvons faire l'hypothèse que les arêtes sont représentées par des lignes polygonales (suite finie de segments de droites).

Commençons par quelques définitions de topologie :

Définition 5 U un ensemble de points de \mathbb{R}^2 est appelé un **ouvert**, s'il vérifie :

$$\forall p \in U, \exists \epsilon > 0 \text{ tel que } \forall p' \text{ vérifiant } d(p, p') \leq \epsilon \text{ alors } p' \in U.$$

Les deux définitions suivantes utilisent un plongement planaire $\phi(G)$ de G .

Définition 6 O un ouvert de \mathbb{R}^2 est appelé une **région** de \mathbb{R}^2 par rapport à $\phi(G)$, s'il vérifie :

$\forall p, p' \in O$, il existe une ligne polygonale qui va de p à p' sans croiser d'arête de $\phi(G)$.

Définition 7 Une région F maximale est appelée une **face** de $\phi(G)$.

La version restreinte du théorème de Jordan :

Théorème 13 Une ligne polygonale simple fermée partage le plan en 2 régions (l'intérieur et l'extérieur).

Soit F l'ensemble des faces de $\phi(G)$, on notera $|F| = f$ et $f = f_0 + 1$ où f_0 représente le nombre de faces finies.

Théorème 14 Si G est connexe, pour tout plongement planaire $\phi(G)$ de G vérifie la relation d'Euler :

$$\boxed{n - m + f = 2}$$

Preuve: La preuve se fait à l'aide d'une induction simple sur le nombre de sommets.

– Pour $n = 1$.

Lorsque $m = 0$, il n'y a qu'une face et la relation est vérifiée. On procède alors par induction sur le nombre d'arêtes en remarquant qu'un tel graphe est constitué de m boucles sur l'unique sommet. Si on considère une boucle, d'après le théorème précédent elle sépare le plan en deux. On peut donc enlever une boucle et cela revient à enlever exactement une face et donc cela marche par induction.

– Dans le cas où $n > 1$.

Comme G est connexe il admet au moins une arête $e = [a, b]$. On contracte cette arête, cela nous donne un nouveau graphe G' ayant une arête et un sommet de moins. Cette opération ne peut ni supprimer ni créer de face donc on peut conclure en utilisant l'hypothèse d'induction.

■

Il existe une autre preuve directe, basée sur la construction d'un arbre recouvrant T de $\phi(G)$. À chaque arête du coarbre de $\phi(G)$ on associe l'arête correspondante du dual. Cela nous permet de construire un arbre recouvrant de $\phi(G)^d$. Donc $m = n - 1 + f - 1$. Cette très jolie preuve est tirée de [4].

En conséquence le nombre de faces de $\phi(G)$ ne dépend pas du plongement de ϕ (ne dépend que des nombres de sommets et d'arêtes).

On peut aussi remarquer que les faces finies de G supposé connexe constituent une base de l'espace des cycles et l'on a :

$f_0 = m - n + 1 = \nu(G)$ formule du nombre cyclomatique d'un graphe, i.e. la dimension de l'espace vectoriel des cycles de G , ce qui engendre une autre preuve de la relation d'Euler.

Cette égalité n'implique pas que pour tout cycle μ de G , il existe un plongement $\phi(G)$ dans lequel ce cycle μ soit une face (pour s'en convaincre, il suffit de considérer un cycle à 4 sommets muni de deux diagonales K_4).

Remarque : En fait la relation d'Euler est valable pour un plongement sur une surface orientable S de genre $genre(S)$:

$$\boxed{n - m + f = 2 - 2genre(S)}$$

Rappelons que le genre de la sphère est 0, surface équivalente au plan, donc aussi de genre 0.

Corollaire 6 *Un polyèdre convexe P de R^3 ayant n sommets, f faces et m arêtes vérifie : $n - m + f = 2$.*

En effet on peut représenter sans croisement sur la sphère un tel polyèdre. En prenant une face quelconque du polyèdre comme face infinie on peut obtenir une représentation plane de P sur lequel on applique la formule d'Euler.

Cette formule n'est plus valable si le polyèdre admet des trous, il y a des contre-exemples.

9.2 Dualité

Définition 8 *A tout plongement planaire $\phi(G)$ d'un graphe $G = (X, E)$, on peut associer son dual, i.e. un autre graphe $\phi(G)^d$ dont les sommets sont les faces de $\phi(G)$ et l'adjacence entre faces est définie par les arêtes communes aux deux faces dans $\phi(G)$. À chaque arête commune aux deux faces, on associe une arête dans $\phi(G)^d$. Ainsi même si $\phi(G)$ est un graphe simple, $\phi(G)^d$ peut être un multigraphe.*

Une représentation planaire de $\phi(G)^d$ s'obtient aisément, il suffit de mettre un point au milieu de chaque face de $\phi(G)$ et de les relier à toutes les faces adjacentes.

Exemple célèbre : le diagramme de Voronoi [35] et la triangulation de Delaunay (Charles Eugène Delaunay, mathématicien Français 1816-1872), sont deux graphes planaires duaux définis à partir d'un ensemble de points.

Ainsi $\phi(G)^d$ admet f sommets et m arêtes (car une arête de $\phi(G)^d$ correspond exactement à une arête de $\phi(G)$).

Il est facile de vérifier que $(\phi(G)^d)^d$ est isomorphe à $\phi(G)$ si et seulement si G est connexe. En effet le passage au dual préserve la connexité, mais si un graphe admet plusieurs composantes connexes son plongement $\phi(G)$ aussi. Cependant $(\phi(G)^d)^d$ est connexe (grâce à la face infinie) et donc aussi $\phi(G)^d$ qui ne peut donc pas être isomorphe à $\phi(G)$.

Cependant il existe des graphes (par exemple un cycle avec deux arêtes pendantes) pour lesquels tous les plongements bien que combinatoirement isomorphes ne sont pas topologiquement équivalents (sur l'exemple du cycle, suivant que l'on mette les deux arêtes pendantes dans la même face ou non). Ces deux plongements même s'ils sont combinatoirement isomorphes, ne sont pas topologiquement équivalents.

Ainsi il existe des graphes planaires G possédant deux plongements $\phi(G)$ et $\phi'(G)$ tels que $\phi(G)^d$ et $\phi'(G)^d$ ne sont pas combinatoirement isomorphes.

En outre on peut montrer :

Un graphe n'admet qu'un plongement planaire (à un isomorphisme près préservant la topologie) s'il est 3-connexe. Dans un tel cas on peut vraiment parler du graphe dual G^d d'un graphe planaire G .

9.3 Cas particulier des graphes simples

On dit qu'un graphe G est *simple* s'il n'admet ni boucle ni arête multiple. Pour un graphe planaire connexe ayant $n \geq 4$ sommets, être simple implique que pour tout plongement les faces ont au moins trois arêtes :

$$\forall \alpha \in F, \text{degre}(\alpha) \geq 3$$

$$\text{d'où : } 2m = \sum_{\alpha \in F} \text{degre}(\alpha) \geq 3f,$$

$$\text{et donc } \boxed{f \leq \frac{2m}{3}}.$$

En reportant dans la formule d'Euler, on obtient :

$$m = n - 2 + f \leq n - 2 + \frac{2m}{3} \text{ d'où : } \boxed{m \leq 3n - 6}$$

De même on a $f \leq \frac{2m}{3} \leq 2/3(3n - 6)$, c'est à dire :

$$\boxed{f \leq 2n - 4}.$$

Lorsque $n = 3$ les deux inégalités sont encore vraies, bien que toute face ne soit pas un triangle (par exemple si $G = P_3$), donc nous pouvons conclure :

Théorème 15 *Pour un graphe planaire simple ayant $n \geq 3$ sommets, alors $m \leq 3n - 6$ et $f \leq 2n - 4$.*

On peut retenir que pour les graphes planaires simples $m, f \in O(n)$.

9.4 Exercices

1. On considère un graphe planaire G et un de ses plongement $\phi(G)$, montrer que pour toute face F de $\phi(G)$, il existe un plongement $\psi(G)$ dans lequel F est la face infinie.
2. Montrer en utilisant la formule d'Euler que $K_{3,3}$ et K_5 ne sont pas planaires. Proposer une autre preuve, sans utiliser la formule d'Euler.
3. $K_{3,3}$ et K_5 sont-ils plongeables planairement sur un tore, un ruban de Moebius ?
4. Montrer qu'un graphe planaire simple admet au moins un sommet de degré inférieur ou égal à 5.
En déduire qu'un ballon de football ne peut être uniquement réalisé par coutures de pièces de cuir hexagonales.
5. Déduire aussi de cette propriété un codage efficace en mémoire d'un graphe planaire. En déduire une borne sur le nombre de bits nécessaires pour coder un graphe planaire ayant n sommets.
6. Montrer que toute triangulation d'un polygone à n côtés utilise exactement $n - 3$ diagonales pour créer $n - 2$ triangles. Montrer que de plus le graphe dual (sans le sommet correspondant à la face infinie) est un arbre.
7. Les solides platoniciens. Montrer qu'il n'existe que 5 solides ou polyèdres qui soient réguliers de degré k et dont toute face soit un polygone régulier à d côtés.

9.5 Graphes planaires maximaux

Nous appellerons graphe planaire maximal un graphe simple planaire G , tel que l'ajout d'une arête lui fasse perdre cette planarité.

Il est facile de vérifier que dans un tel graphe toute face est un triangle (même la face infinie) et le nombre d'arêtes vérifie $m = 3n - 6$. En fait, nous avons les équivalences suivantes pour un graphe simple et planaire G :

G est maximal ssi toute face est un triangle ssi $m = 3n - 6$.

De tels graphes sont presque composés de 3 arbres recouvrants (ce qui ferait $3n-3$ arêtes) et il y a eu de nombreux travaux de recherche sur ce sujet.

9.5.1 Caractérisation des graphes planaires

Théorème 16 *Kuratowski*

Un graphe G est planaire ss'il ne contient pas $K_{3,3}$ ni K_5 comme subdivision.

Théorème 17 *Wagner*

Un graphe G est planaire ss'il ne contient pas $K_{3,3}$ ni K_5 comme mineur.

Ce théorème est plus fort que le précédent.

Chapitre 10

Recueil d'examens

10.1 2006–2007

10.1.1 Parcours de graphes

On notera $post(x)$ l'ordre de fin d'exploration des sommets engendré par un parcours en profondeur sur un graphe orienté $G = (X, U)$. À chaque composante fortement connexe C de G , on peut associer un intervalle $[premier(C), dernier(C)]$ qui correspond aux ordres respectivement des premier et dernier sommets de C dans l'ordre total $post$.

1. Montrer que si l'on considère deux composantes fortement connexes de G , les intervalles associés ne peuvent se chevaucher (i.e. ils sont soit inclus, soit disjoints).
2. Montrer que l'on peut donc associer un mot de parenthèses bien formées à un tel parcours.
3. En déduire un algorithme de calcul des composantes fortement connexes d'un graphe en deux étapes :
 - (a) Un parcours en profondeur (calcul de l'ordre $post$)
 - (b) Une analyse de $post$ qui permet l'extraction des composantes fortement connexes.

Détailler la deuxième étape (preuve, complexité).

4. En déduire une preuve de l'algorithme de Tarjan qui n'utilise qu'un seul parcours en profondeur.

10.1.2 Flots et chemins

1. Un flot ϕ est dit **complet** lorsque tout chemin allant de s à p admet un arc **saturé** (i.e. un arc u tel que $\phi(u) = c(u)$).
Un flot complet est-il un flot maximum ?
2. * Est-il possible de construire un exemple de graphe dont les arcs sont munis de capacités rationnelles, de manière à ce que l'algorithme de Ford-Fulkerson standard converge mais pas vers le flot maximum ?
3. On considère maintenant l'étape d'amélioration du flot à l'aide de l'algorithme de Ford-Fulkerson. Supposons que nous avons construit le graphe des écarts $G(\phi)$. Il s'agit maintenant de chercher dans ce graphe orienté un chemin de s à p . Si l'on veut trouver le chemin qui permet d'augmenter le plus le flot. On appellera un tel chemin un chemin améliorant optimal. Formaliser en termes de chemins cette recherche.
4. Proposer un algorithme de recherche d'un tel chemin améliorant optimal, en s'inspirant des algorithmes de plus courts chemins vus en cours. Invariants et preuve de l'algorithme proposé. Complexité de l'algorithme.
5. Que peut-on dire d'un algorithme qui utiliserait à chaque étape un chemin améliorant optimal, i.e. peut-on borner le nombre d'améliorations ?

10.1.3 Arbres de poids minimum

On appelle *Pseudo-arbres_k* la classe des graphes non orientés connexes $G = (X, E)$ vérifiant : $|E| \leq |X| + k$.

1. Analyser la complexité des algorithmes Prim, Kruskal sur cette classe de graphes.
2. Quel algorithme de calcul d'un arbre recouvrant de poids minimum vous paraît le meilleur sur ces graphes ?

On considère maintenant un graphe $G = (X, E)$ connexe muni d'une valuation $\omega : E \rightarrow N$.

1. Proposer un algorithme de calcul du deuxième arbre suivant l'ordre lexicographique sur les arbres.
2. coarbre de poids min max

10.2 2007-2008

10.2.1 Algorithme sur les arborescences

On considère une arborescence binaire saturée (chaque sommet non feuille a exactement deux fils) A de racine r muni de sa représentation classique chaque élément (sauf la racine) possède un pointeur sur son père. On supposera en outre que chaque sommet x de l'arborescence possède une variable $d^-(x)$ initialisée au nombre de fils de x dans l'arborescence.

Étant donné un ensemble F de feuilles de l'arborescence, il s'agit de vérifier que F correspond exactement à l'ensemble des feuilles d'une sous-arborescence de racine x où x est un sommet de l'arborescence.

L'algorithme a deux réponses possibles NON ou Oui : $F = A_x$

1. Donner deux exemples de données correspondant à ces deux réponses.
2. Écrire un algorithme qui vérifie cette propriété. La donnée étant la liste des éléments qui constituent F et l'arborescence A .

Piste : il existe un algorithme en $O(|F|)$.

N.B. On ne demande pas de vérifier que les sommets de F sont des feuilles de A , on considère que cela fait partie de la donnée.

10.2.2 Détection de cycles et circuits

1. Écrire et évaluer un algorithme qui vérifie si G possède un cycle.
2. Écrire et évaluer un algorithme qui calcule un cycle de G , s'il en existe un.
3. Les mêmes questions dans le cas orienté, en remplaçant cycle par circuit.
4. * Écrire un algorithme qui calcule la longueur du plus petit cycle sans corde de G .

10.2.3 Plus courts chemins

On considère dans ce qui suit un graphe orienté $G = (X, U)$ tel que $|X| = n$ et $|U| = m$ et l'on suppose G muni d'une valuation sur ses arcs, i.e. $\nu : U \rightarrow \mathbb{N}$ (valuation entière).

1. Étant donné une arborescence $A = (X, V)$ de racine r , proposer un algorithme qui vérifie que A est une arborescence des plus courts chemins issus de r de G .
2. Un ancien étudiant n'ayant pas assisté à tous les cours d'algo (comment diable est-ce possible?) a programmé le schéma d'algorithme suivant pour calculer les plus courts chemins issus du sommet r .
Initialisations : $d(r) = 0$, $d(x) = \nu(r, x)$ avec la convention $\nu(r, x) = C$, où C est une constante entière supérieure à $\sum_{u \in U} \nu(u)$ si l'arc (r, x) n'existe pas.
Tant qu'il existe un arc $u = (x, y) \in U$ tel que $d(y) > d(x) + \nu(u)$ alors faire $d(y) = d(x) + \nu(u)$.
De quels algorithmes s'est-il inspiré?
3. L'algorithme s'arrête-t-il et si oui évaluez la complexité de l'algorithme?
4. Corriger ou améliorer l'algorithme de l'étudiant.
5. On suppose maintenant que la valuation d'un chemin soit le produit des valuations des arcs qui le constituent. C.a.d si $\mu = [x_1, \dots, x_k]$ alors $\nu(\mu) = \prod_{1 \leq i \leq k-1} \nu((x_i, x_{i+1}))$. Donner un exemple d'usage d'une telle modélisation. Proposer un algorithme qui calcule les plus courts chemins issus d'un point r du graphe. L'évaluer.

10.2.4 Plus courts chemins (suite)

1. Considérons le graphe G réduit à une étoile, i.e. les sommets sont x_0, x_1, \dots, x_n et les seuls arcs sont de type x_0x_i de valuation positive a_i .
Comment se déroule l'algorithme de Dijkstra sur cet exemple pour calculer les plus courts chemins issus de x_0 ? Cet algorithme fait-il des calculs inutiles? Comment peut-on l'améliorer?
2. * On considère dans ce qui suit un graphe orienté $G = (X, U)$ tel que $|X| = n$ et $|U| = m$ et l'on suppose G muni d'une valuation sur ses arcs, i.e. $\nu : U \rightarrow \mathbb{N}$ (valuation entière).
Peut-on écrire un algorithme linéaire pour le calcul des plus courts chemins issus d'un sommet donné? Si oui expliquer le.
3. ** Et lorsque les valuations sont des réels positifs?

10.3 Parité de chemins **

On considère un graphe non orienté $G = (X, E)$, et deux sommets distincts $x, y \in X$.

1. Écrire un algorithme qui calcule s'il existe une chaîne (resp. une marche) de longueur paire de x à y
2. même question avec impaire.
3. * Mêmes questions avec chaîne (resp. marche) de longueur $i \in \{0, 1, 2\}$ modulo 3.
4. On appelle **paire d'amis** deux sommets x et y de G , tels que tous les chaînes sans corde joignant x à y soient de longueur paire. Écrire un algorithme de recherche d'une telle paire.

10.4 2008-2009

10.4.1 Plus courts chemins

Le chef du service R&D de votre nouvelle société vous propose un algorithme très simple pour calculer dans un graphe G une arborescence des plus courts chemins issus d'un sommet s ¹.

Si les valuations des arcs sont toutes positives appliquer l'algorithme de Dijkstra.

Sinon calculer la valuation minimale $-M$ du graphe. Ajouter $M + 1$ à toutes les valuations des arcs de G et appliquer l'algorithme de Dijkstra.

1. Comment expliquer à votre chef, que cela ne marche pas toujours, toutefois sans vous faire virer ?
2. Peut-on quand même tirer quelque chose de cette méthode ? Par exemple proposer des configurations pour lesquelles la méthode calcule une arborescence des plus courts chemins.
3. Peut-on améliorer cette méthode (i.e. en augmentant considérablement le nombre de graphes pour lesquels la méthode fonctionne) ?
4. Peut-on utiliser cette méthode pour obtenir des approximations ?
5. Votre Chef insiste, il demande un algorithme qui calcule une arborescence des plus courts chemins dans tous les cas, que lui proposez-vous ?

1. J'ai eu un problème de ce genre avec le centre de recherche EDF Clamart !

10.4.2 Arborescences

On considère une arborescence A finie. Les feuilles de A sont étiquetées par des éléments d'un ensemble X . A est représentée par la fonction père (tout élément sauf la racine admet un père unique), et l'on suppose en outre que chaque noeud de l'arborescence possède une variable qui contient le nombre de fils. On suppose qu'un pointeur permet d'associer à chaque élément de X la feuille qui lui correspond dans A .

1. Etant donné $S \subseteq X$ écrire un algorithme qui vérifie qu'il existe un noeud $a_s \in A$ tel que S corresponde à l'ensemble des feuilles de la sous-arborescence de A engendrée par a_s . Peut-on écrire un algorithme en $O(|S|)$?

10.4.3 Coupes minimales

1. Comment calculer un coupe minimale ayant un nombre minimum d'arcs (parmi les coupes minimales entre deux sommets s et t d'un graphe G), en utilisant un algorithme de flot maximum.
2. * On recherche maintenant une coupe minimale d'un graphe non orienté, i.e. un cocycle de poids minimum. Peut-on utiliser une variante de Kruskal ou Prim ?

10.4.4 Arbre de poids minimum

Parmi les arbres recouvrants de poids minimum d'un graphe connexe G , on recherche un arbre ayant un diamètre minimum (resp. un nombre minimum de feuilles). Le diamètre d'un graphe non orienté est la longueur de la plus longue chaîne sans corde.

* Que peut-on dire de ces problèmes ? Proposer un algorithme lorsque le problème est polynomial. []

Bibliographie

- [1] J. R. Abrial. *The B-book, assigning programs to meanings*. Cambridge University Press, 1996.
- [2] M. Agrawal, N. Kayal, and N. Saxena. Primes is in p. 2002.
- [3] A. V. Aho, I. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Welsey, 1983.
- [4] M. Aigner and G.M. Ziegler. *Proofs for the BOOK*. Springer-Verlag, 1999.
- [5] J.D. Boissonnat and M. Yvinec. Ediscience International, 1995.
- [6] J.A. Bondy and U.S.R Murty. *Graph Theory*. Springer Verlag, 2008.
- [7] P.B. Borwein. On the complexity of calculating factorials. *Journals of Algorithms*, 6 :376–380, 1985.
- [8] Catalan. Notes sur une équation aux différences finies. *J. Math. Pures et Appliquées*, 1838.
- [9] L. Comtet. *Analyse Combinatoire, volumes I et II*. Presses Universitaires de France, 1970.
- [10] S.A. Cook and R.A. Reckhow. Time bounded random machines. *Journal of Computer Science and Systems*, pages 354–375, 1973.
- [11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press. En français chez InterEditions.
- [12] J.P. D’Angelo and D.B. West. *Mathematical Thinking : problem solving and proofs*. Prentice-Hall, 1997.
- [13] S. Dasgupta, C. Papadimitriou, and U. Vazirani. *Algorithms*. Mc Graw Hill, 2006.
- [14] M.P. Delest and G. Viennot. Algebraic languages and polyominoes enumeration. *Theoretical Computer Science*, 1984.

- [15] R. Diestel. *Graph Theory*. Springer-verlag, 1997.
- [16] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, 22(6) :644–654, 1976.
- [17] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, (1) :269–271, 1959.
- [18] I. Dutour. *Grammaires d’objets : énumération, bijections et génération aléatoire*. PhD thesis, université Bordeaux, 1996.
- [19] P.C. Gilmore and R.E. Gomory. Sequencing a one-state-variable machine : a solvable case of the traveling salesman problem. *Operation Research*, pages 655–679, 1964.
- [20] R.L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1) :43–57, 1985.
- [21] D.E. Knuth. Big omicron and big omega and big theta. *Sigact news*, April–June :18–24, 1976.
- [22] C. Kozen. *The design and analysis of algorithms*. Texts and monographs in computer science. Springer-Verlag, 1991.
- [23] J.B. Kruskal. On the shortest spanning tree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*, pages 48–50, 1956.
- [24] M. Mares. The saga of minimum spanning trees. *Computer Science Review*, 2 :165–221, 2008.
- [25] B.M.E. Moret and H.D. Shapiro. *Algorithms from P to NP, volume I*. Benjamin Cummings Publishing Company, 1991.
- [26] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [27] G. Pick. Geometrisches zur zahlenlehre. *Sitzungsber lotos, Prague*, 1900.
- [28] R.C. Prim. Shortest connections networks and some generalizations. *Bell Syst. Tech. J.*, pages 1389–1401, 1957.
- [29] R. Rivest, A. Shamir, and Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1978.
- [30] J.E. Savage. *Models of Computation : exploring the power of computing*. Addison-Welsey, 1998.
- [31] D. Shasha and C. Lazere. *Out of their minds*. Copernicus, 1995.

- [32] I. Stewart. Belin, 1994.
- [33] R.E. Tarjan. *Data structures and network algorithms*, volume 44. SIAM Monography, 1983.
- [34] A. Troesch. Interpretation géométrique de l'algorithme d'euclide et reconnaissance de segments. *Theoretical Computer Science*, 1993.
- [35] G. Voronoi. *J. Reine Angew. Math.*, pages 198–287, 1908.
- [36] D. West. *Introduction to Graph Theory*. Prentice Hall, 1996.