Notes de cours Option Info MP/MP* : Graphes

Citer des applications des graphes.

Vocabulaire

Graphes non orientés

Définition: Un graphe non orienté G est un couple (S, A) où S est un ensemble fini dont les éléments sont appelés sommets (ou nœuds), et A un ensemble de paires de sommets appelées arêtes (ou segments).

Remarque: Il y a au plus une arête entre deux sommets (pas de multi-arêtes). On ne peut pas avoir une arête d'un sommet vers lui-même.

Définition: On dit que deux sommets sont adjacents s'il existe une arête entre les deux. Le degré $d(x)$ d'un sommet x est le nombre de sommets adjacents à x.

Définition: Un graphe pondéré est un graphe où l'on attribue à chaque arête un nombre appelé poids.

Graphes orientés

Définition: Un graphe orienté G est un couple (S, A) où S est un ensemble fini (les sommets), et A un sous-ensemble de $S \times S \setminus \{(x, x), x \in S\}$ (les arêtes ou arcs).

Définition: Si $G = (S, A)$ est un graphe orienté et $a = (x, y) \in A$, on dit que x est l'origine et y l'extrémité de a.

Définitions: Si $G = (S, A)$ est un graphe orienté et $x \in S$, on appelle:
-degré sortant de x, noté $d^+(x)$, le nombre de sommets y tels que $(x, y) \in A$.
-degré entrant de x, noté $d^-(x)$, le nombre de sommets y tels que $(y, x) \in A$.
-degré de x, noté $d(x)$, la somme $d^+(x) + d^-(x)$.

Exercice: Montrer que si $G = (S, A)$ est un graphe orienté, alors:

$$\sum_{x \in S} d^+(x) = \sum_{x \in S} d^-(x).$$

Chemin dans un graphe

Définition: Soit $G = (S, A)$ un graphe (orienté ou non). Un chemin dans G est une suite finie de sommets $(s_0, \ldots, s_n)$ telle que pour tout i entre 0 et $n-1$, $(s_i, s_{i+1}) \in A$. Dans ces conditions, on dit que s_0 est l'origine du chemin et s_n son extrémité. On dit qu'un sommet e est *accessible* à partir d'un sommet o si et seulement si $o = e$ ou s'il existe un chemin dans le graphe d'origine o et d'extrémité e. On note $o \to e$.

Composantes connexes/fortement connexes

Définition: Soit G un graphe non orienté. La relation $\rightarrow$ définit une relation d'équivalence *(le montrer, rappeler la définition)* sur l'ensemble des sommets du graphe. Les classes d'équivalence sont les composantes connexes du graphe. On dit que G est connexe s'il n'y a qu'une seule classe d'équivalence.

Question: Si G est orienté, la relation $\rightarrow$ est-elle toujours une relation d'équivalence ?

Définition: Dans le cas d'un graphe orienté, on définit une autre relation d'équivalence : $u \simeq v$ ssi $u \rightarrow v$ et $v \rightarrow u$. Les classes d'équivalence sont les composantes fortement connexes du graphe. On dit que G est fortement connexe s'il n'y a qu'une seule classe d'équivalence.

Remarque: On parle parfois aussi de composantes connexes d'un graphe orienté en considérant celles du graphe non orienté associé (en "supprimant les flèches").

On verra plus loin des algorithmes pour calculer les composantes connexes/fortement connexes.

Implémentation des graphes en Caml

Il existe deux méthodes d'implémentation des graphes:
-Les listes d'adjacence
-La matrice d'adjacence

On suppose par la suite que G possède n sommets numérotés de 0 à $n-1$.

Listes d'adjacence

On utilise un tableau `t` de longueur n tel que pour i entre 0 et $n-1$, `t.(i)` contient la liste des extrémités des arêtes d'origine i.

Si le graphe est pondéré, la liste contient des couples (extrêmité, poids).

Matrice d'adjacence

On utilise une matrice (tableau à deux dimensions) de booléens `m` tel pour tous i, j entre 0 et $n-1$, `m.(i).(j)` contient `true` si (i, j) est une arête du graphe et `false` sinon.

Dans le cas d'un graphe pondéré, on met le poids de l'arête et une valeur spéciale (`max_int` ou `min_int` par exemple) s'il n'y a pas d'arête.

Remarque: Dans le cas d'un graphe non orienté, chaque arête apparaît deux fois : si (i, j) est une arête alors (j, i) aussi. En particulier la matrice d'adjacence est .....

Comparaison des deux méthodes

Citer les avantages/inconvénients de chaque implémentation.

Certains problèmes se traitent plus facilement avec les matrices d'adjacence, d'autres avec les listes d'adjacence, on choisira donc à chaque fois la meilleure représentation.

À savoir faire en Caml (voir TP):
-Passer d'une représentation à une autre.
-Pour chacune des deux représentations : ajouter ou supprimer une arête (attention : pour les graphes non orientés, chaque arête est représentée deux fois).
-Supprimer un sommet (supprimer toutes les arêtes passant par ce sommet).
-Rendre non orienté un graphe orienté.

Parcours dans un graphe

Un parcours de graphe est un algorithme consistant à explorer les sommets du graphe de proche en proche à partir d'un sommet initial. Pour éviter de "tourner en rond", il est nécessaire de marquer les sommets déjà visités.

Il y a deux types de parcours qui nous intéresseront:

-Le parcours en profondeur (ou DFS pour "Depth First Search") dans lequel on parcourt les sommets par "branches", on suit chaque branche jusqu'à être bloqué puis on revient en arrière ;

-Le parcours en largeur (ou BFS pour "Breadth First Search") dans lequel on parcourt les sommets par "niveaux" de distance croissante par rapport à l'origine ;

Solution avec une pile/file

On adopte la représentation par ..... On utilise:

-Une pile pour le parcours en .....

-Une file pour le parcours en .....

Rappeler les opérations élémentaires sur les piles/files et leur implémentation en Caml.

Le principe est le suivant *(à coder en TP)*:

-On insère le sommet initial dans la pile/file.

-Tant que la pile/file est non vide:

-On fait sortir un élément n de la pile/file.

-Si n n'est pas encore marqué:

-On marque n et on l'ajoute à la liste des sommets parcourus.

-On fait rentrer tous ses voisins dans la pile/file.

Voir exemple au tableau.

Complexité (en fonction du nombre n de sommets et du nombre p d'arêtes): .....

Applications des parcours

Recherche des composantes connexes/fortement connexes

Pour calculer la composante connexe d'un élément du graphe (si celui-ci est non orienté), il suffit de lancer un parcours en partant de ce sommet. En particulier le graphe est connexe ssi .....

Pour les composantes fortement connexes d'un graphe orienté G, on doit aussi considérer le graphe inverse G^{-1} (où on inverse toutes les arêtes) et lancer un parcours dans chacun des deux graphes à partir de x. La composante fortement connexe de x est ..... des deux listes obtenues.

Détection de cycles

Expliquer comment détecter la présence d'un cycle à l'aide d'un parcours.

Plus court chemin dans un graphe non pondéré

Le parcours en ..... est particulièrement adapté à la recherche du plus court chemin entre deux sommets dans un graphe non pondéré (où toutes les arêtes sont supposées de longueur 1).

Chemin de poids minimal dans un graphe pondéré

Définition: Étant donné un graphe (orienté ou non) pondéré G et un chemin c dans ce graphe, le poids de c est la somme des poids des arêtes qui le composent.

Propriété: Si le graphe ne possède pas de cycle de poids négatif (en particulier si les poids sont tous positifs) et que s_1 et s_2 sont deux sommets de G dans la même composante connexe, alors il existe un chemin de poids minimal entre s_1 et s_2.

Remarque: Le résultat précédent devient faux si le graphe comporte des cycles de poids négatif, expliquer pourquoi.

Définition: Dans la même situation que précédemment, on définit la distance $d(s_1, s_2)$ de s_1 à s_2 comme le poids d'un chemin de poids minimal entre s_1 et s_2 s'ils sont dans la même composante connexe (en particulier elle est nulle si $s_1 = s_2$), et infinie sinon.

Problématique: Étant donné un graphe pondéré G sans cycle de poids négatif, et deux sommets s_1 et s_2, on cherche à déterminer:
- La distance entre s_1 et s_2
- Un chemin de poids minimal entre s_1 et s_2

On suppose dans la suite que **tous les poids sont positifs**.

L'algorithme de Floyd-Warshall

Objectif: Calculer toutes les distances entre deux sommets quelconques, le résultat sera une matrice de distances.

Principe: On augmente peu à peu le nombre de sommets intermédiaires possibles (programmation .....).

Numérotons les sommets du graphe $s_0, \dots, s_{n-1}$. Pour i, j entre 0 à $n-1$ et k entre 0 et n, on note $d_k(i, j)$ le poids d'un chemin minimal entre i et j en passant uniquement par des sommets intermédiaires entre 0 et $k-1$ si un tel chemin existe et $+\infty$ sinon. On a la formule de récurrence:

$$d_{k+1}(i, j) = \text{.....}$$

Preuve de la correction: On montre par récurrence qu'à chaque étape, les distances sont minimales parmi les chemins ne passant que par les sommets intermédiaires actuels.

Implémentation: On utilise la représentation du graphe sous forme de ..... . On utilise `max_int` (correspond à $+\infty$) dans le cas où il n'y a pas d'arête.

Complexité: .....

Remarque/Question: Comme il n'y a pas de poids négatif, $d_k(i, k) = d_{k+1}(i, k)$ et $d_k(k, j) = d_{k+1}(k+1, j)$, donc l'ordre de traitement des cases d'indice $(i, j), (i, k)$ et (k, j) n'importe pas, comment peut-on améliorer la complexité spatiale ?

Code (à faire en TP): On commence par réécrire la somme de deux entiers en incluant l'infini (`max_int`).

L'algorithme de Dijkstra

Objectif: Calculer toutes les distances par rapport à un sommet d'origine s_0.

Principe: On parcourt le graphe suivant une liste de sommets situés à distance croissante de s_0 (programmation .....). On stocke dans un tableau d les distances à s_0. Initialement tout le monde est à une distance $+\infty$ (représenté par `np.inf` en Python) et on commence par traiter le sommet $s = s_0$. À chaque étape, en traitant le sommet s_k on met à jour les distances de ses voisins à l'aide de la formule:

$$d(s) \leftarrow min(d(s), d(s_k) + w(s_k, s)),$$

puis on cherche le nouveau sommet à distance minimale parmi ceux non encore traités.

Preuve de la correction: Comme avant, on montre par récurrence qu'à chaque étape, les distances sont minimales parmi les chemins ne passant que par les sommets déjà visités.

Implémentation: On adoptera donc la représentation des graphes par ..... Il nous faut aussi un moyen de représenter les sommets déjà visités, adapté à la recherche du sommet à distance minimale à chaque étape. Deux solutions *(à coder en TP)*:

- deux tableaux : un tableau de booléens de sommets déjà visités et un autre contenant les distances
- une file de priorité, implémentée par ..... (*rappeler le principe et les opérations élémentaires*)

Complexité:

- avec les deux tableaux : .....
- avec la file de priorité : .....

Compléments (HP)

Graphes eulériens et hamiltoniens

Définition: Étant donné un graphe non orienté $G = (S, A)$ connexe on dit que G est :

- eulérien s'il existe un circuit (c'est-à-dire un chemin fermé) qui passe une et une seule fois par chaque arête
- semi-eulérien s'il existe un chemin (pas forcément fermé) qui passe une et une seule fois par chaque arête
- hamiltonien s'il existe un circuit qui passe une et une seule fois par chaque sommet (sans compter le dernier sommet qui est égal au premier)
- semi-hamiltonien s'il existe un chemin qui passe une et une seule fois par chaque sommet

On dispose d'un algorithme pour détecter si un graphe est eulérien. Remarquons déjà plusieurs propriétés sur les graphes eulériens/semi-eulériens:

Propriété 1:

1. Si on enlève une arête à un graphe eulérien, on obtient un graphe .....
2. En ajoutant une arête bien choisie à un graphe semi-eulérien, on obtient un graphe .....

Propriété 2:

1. Dans un graphe eulérien, tous les sommets ont un degré .....
2. Dans un graphe semi-eulérien, il y a au plus deux sommets dont le degré est .....

Il se trouve qu'il y a une réciproque à cette propriété:

Propriété 3: Un graphe connexe dont tous les sommets ont un degré ..... est un graphe eulérien.

Preuve: Elle repose sur les 3 faits suivants:

- Un circuit eulérien est une union finie de circuits simples (ne passant pas deux fois par une même arête) disjoints (au niveau des arêtes)
- Si tous les degrés sont .... et non tous nuls, alors on peut construire un circuit simple.
- Si l'on retire du graphe toutes les arêtes d'un circuit, les degrés des sommets restent .....

On raisonne ensuite par l'absurde : supposons qu'il n'existe pas de circuit eulérien. Considérons un circuit simple maximal. Par ce qui précède, en retirant toutes les arêtes de ce cycle au graphe initial, on obtient un graphe dont tous les sommets sont encore ..... et non tous nuls car le circuit n'est pas eulérien (il reste des arêtes) donc on peut construire un autre circuit simple et fusionner les deux pour obtenir un circuit simple plus grand (car le graphe est connexe).

Signalons un problème historique dont la résolution a fait appel à cette propriété: le problème des 7 ponts de Koenigsberg (résolu par Euler, qui a donné son nom au circuit eulérien). Le problème du postier (trouver un circuit qui passe au moins une fois par chaque arête) a lui aussi été résolu en temps polynomial.

Cas d'un graphe orienté : Il existe un théorème analogue pour les graphes orientés : G est eulérien ssi il est fortement connexe et pour tout sommet, le degré entrant est égal au degré sortant.

Remarque: Contrairement aux graphes eulériens, il n'existe à ce jour pas de solution efficace permettant de déterminer si un graphe est hamiltonien ou non. Ce problème est dit NP-complet.

Exemples de problèmes liés : le problème du cavalier (qui a une solution en temps linéaire), le problème du voyageur de commerce (trouver un circuit hamiltonien de poids minimal dans un graphe pondéré : NP-complet).

D'autres problèmes difficiles (NP-complets) sur les graphes :

- Problème de la clique de taille maximale (une clique est un sous-ensemble de sommets tous reliés entre eux dans le graphe)

- Problème de couverture par sommets : trouver un ensemble minimal de sommets qui suffisent à couvrir toutes les arêtes (chaque arête est adjacente à au moins un élément de cet ensemble)

- Problème de coloriage : Associer à chaque sommet une couleur de sorte que deux sommets adjacents ne soient pas de la même couleur, et trouver le nombre de couleurs minimal (nombre chromatique). Déterminer si un graphe est 2-coloriable (coloriable avec 3 couleurs) peut être résolu en temps polynomial (comment ?), dès que $k \geqslant 3$, déterminer si un graphe est k-coloriable est un problème NP-complet.

Le problème 2-SAT

Rappel : soit F une formule sous FNC (nous avons vu que chaque formule est équivalente à une formule sous FNC), c'est-à-dire une conjonction (un ET) de clauses, une clause étant une disjonction (un OU) de littéraux (littéral = variable ou négation d'une variable) . On appelle clause d'ordre n une clause qui comporte au plus n littéraux, et FNC d'ordre n une FNC qui ne comporte que des clauses d'ordre n. Le problème n-SAT consiste à décider si une FNC d'ordre n est satisfiable. Pour $n \geqslant 3$, ce problème est NP-complet mais nous allons voir que pour $n = 2$ on peut le résoudre en temps polynomial à l'aide d'un graphe.

Commençons par un lemme:

Lemme: La clause d'ordre 2 $(a \vee b)$ est équivalente à $(\neg a \Rightarrow b) \wedge (\neg b \Rightarrow a)$.

Il suffit de construire les tables de vérités associées pour s'en convaincre.

Étant donnée une formule F sous FNC d'ordre 2, on lui associe le graphe orienté $G = (S, A)$ construit comme suit:

- les sommets correspondent aux variables apparaissant dans la formule F ainsi que leurs négations

- pour chaque clause $(a \vee b)$ on ajoute les arêtes (orientées) $(\neg a, b)$ et $(\neg b, a)$.

On a alors le théorème suivant:

Théorème: F est satisfiable si et seulement si pour toute variable x, x et $\neg x$ ne sont pas dans la même composante fortement connexe de G.

Rappelons que l'on peut calculer les composantes fortement connexes à l'aide d'un double parcours (dans G et dans son graphe inverse G^{-1}) donc en temps ..... en la taille de G (qui est aussi ..... en la taille de la formule F).

La preuve du sens gauche-droite est immédiate : si F est satisfiable, considérons une valuation qui la satisfait. Si deux nœuds représentant des littéraux a et b sont dans la même composante fortement connexe de G, on a nécessairement $a \Leftrightarrow b$ par construction du graphe et le lemme. Comme $a \Leftrightarrow \neg a$ est toujours fausse, cela montre qu'elles ne sont pas dans la même composante fortement connexe.

Pour la réciproque : nous construisons une valuation qui satisfait F. Partons d'une remarque : étant donnée une variable x, comme x et $\neg x$ ne sont pas dans la même composante fortement connexe, on ne peut pas avoir à la fois un chemin de x vers $\neg x$ et un chemin de $\neg x$ vers x.

L'algorithme est le suivant:

- Tant qu'il reste des variables sans valeur affectée:
  - On choisit un littéral (une variable ou sa négation) x tel qu'il n'y ait pas de chemin de x vers $\neg x$ dans le graphe associé.
  - On met x à Vrai, et on affecte la valeur Vrai à tout littéral (si c'est la négation d'une variable, on attribue Faux à cette variable) descendant de x dans le graphe (en faisant un parcours)
  - On supprime du graphe les littéraux (et leurs négations) parcourus précédemment.

Exercice: Déterminer pour chacune des formules suivantes si elle est satisfiable, et le cas échéant donner une valuation la satisfaisant:
- $F_1 = (a \vee \neg b) \wedge (a \vee c) \wedge (\neg b \vee \neg c) \wedge (b \vee \neg c)$
- $F_2 = (a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b) \wedge (\neg b \vee c) \wedge (\neg a \vee \neg c)$

Arbres couvrants de poids minimal

Définition: Soit $G = (S, A)$ un graphe non orienté connexe. Un sous-graphe couvrant est un sous-graphe connexe de G qui passe par tous les sommets de G. Rappelons aussi qu'un arbre est un graphe connexe acyclique, et un arbre couvrant est alors un sous-graphe couvrant qui est un arbre.

Supposons de plus G pondéré avec des poids strictement positifs. On cherche maintenant un sous-graphe couvrant de poids minimal (cela a des applications par exemple dans la gestion des trafics maritimes/routiers). Signalons d'abord un résultat d'existence:

Théorème: Si G possède des poids strictement positifs, il possède un sous-graphe couvrant de poids minimal et ce dernier est un arbre.
Preuve: L'ensemble des sous-graphes couvrants est fini car G est fini, il est non vide car G est lui-même un sous-graphe couvrant, donc cet ensemble admet un élément de poids minimal. Si cet élément possède un cycle, comme tous les poids sont strictement positifs, on peut supprimer une arête du cycle en préservant la connexité du graphe et on obtient un sous-graphe couvrant de poids strictement inférieur, donc nécessairement un sous-graphe couvrant minimal est un arbre.

Nous allons voir deux algorithmes qui permettent de trouver un arbre couvrant de poids minimal en adoptant une stratégie gloutonne.

Algorithme de Prim:

On part d'un sommet initial s_0 quelconque, puis on ajoute à chaque étape nouveau sommet: parmi tous les sommets ne faisant pas encore partie de l'arbre, on choisit celui qui est relié à un des sommets de l'arbre actuel par une arête de poids minimal.

On notera par la suite $T = (V, E)$ l'arbre couvrant obtenu (les ensembles V et E évoluant au cours de l'algorithme).

Preuve de la correction: Par construction, T est bien un graphe connexe acyclique donc un arbre et il contient tous les sommets de G donc il est couvrant. Pour montrer qu'il est de poids minimal, on montre par récurrence sur $|V|$ qu'il existe un arbre couvrant de G de poids minimal contenant (V, E). A la fin de l'algorithme $V = S$ donc on a bien que T est de poids minimal.

Étude de la complexité: Notons $n = |S|$ et $p = |A|$. On implémente G par ..... A chaque étape, la recherche naïve d'une arête de poids minimal est en ..... donc l'algorithme est en ..... On peut cependant faire mieux en utilisant un tas représentant une file de priorité, la priorité étant la distance actuelle à l'arbre courant. Comme pour l'algorithme de Dijkstra, la construction du tas est en ..... et chaque mise à jour du tas en ..... Il y a une mise à jour par arête (quand on ajoute un sommet, on ne change la priorité que de ses voisins) donc la complexité est en .....

Algorithme de Kruskal:

Cette fois-ci, on ne cherche plus à préserver la connexité du sous-graphe à chaque étape. On part du graphe G' possédant les n sommets de G et aucune arête (il a donc n composantes connexes), et à chaque étape on ajoute une arête de poids minimal reliant deux composantes connexes de G'. On s'arrête quand le graphe G' est connexe (nécessairement c'est un arbre car on n'a pas créé de cycle au cours de l'algorithme). Il nous faut donc gérer une forêt (un ensemble d'arbres).

Preuve de la correction: Comme pour l'algorithme de Prim, on montre par récurrence qu'à chaque étape il existe un arbre couvrant de poids minimal contenant la forêt courante.

Complexité: Il nous faut d'abord trier les arêtes par ordre croissant, cela coûte ..... On peut ensuite gérer les différentes composantes connexes à l'aide d'une structure appelée "Union Find" (Find : renvoie le numéro de la classe d'équivalence d'un élément, permet de tester si deux éléments sont dans la même composante, Union : réalise l'union de deux composantes). En général cette structure consiste en une forêt (un arbre par composante connexe) où chaque élément contient une référence vers son père, et on équilibre les arbres à chaque union. Avec cette structure, on atteint une complexité de $O(p\log(p)) = O(p\log(n))$ car $p = O(n^2)$.

Graphes planaires

Définition: Un graphe est planaire si on peut le tracer dans le plan de sorte que deux arêtes ne se croisent pas.

Les graphes planaires vérifient plusieurs propriétés intéressantes:

- Théorème des 4 couleurs : il suffit de 4 couleurs pour colorier un graphe planaire.
- Formule d'Euler : si G est connexe et planaire, en notant F l'ensemble des faces (une face étant une composante connexe du dessin obtenu en traçant le graphe de manière planaire), on a la formule:

$$|S| - |A| + |F| = 2.$$

On la montre en considérant d'abord les arbres (une seule face car pas de cycle), en faisant une récurrence sur le nombre de noeuds, puis pour les graphes quelconques par récurrence sur le nombre de faces.

La formule d'Euler permet de montrer l'impossibilité de résoudre certains problèmes comme celui des 3 maisons.

Signalons enfin que la formule d'Euler est aussi valable pour les polyèdres convexes (cube, tétraèdre etc...). Pour la démontrer, on se ramène au cas des graphes planaires en "aplatissant" le solide.