Cours d'Algorithmique et Complexité

Algorithmique des graphes

Catalin Dima

Rappel de la notion de graphe

Définition

Un graphe est une paire $G = (V, E)$ où :

- V est un ensemble fini de noeuds.
- $E \subseteq V \times V$ est l'ensemble des arètes.

- Boucles : $(v, v) \in E$.
- Par défaut, les graphes sont orientés : il se peut que $(v_1, v_2) \in E$ mais $(v_2, v_1) \notin E$.
- Chemins : suite d'arètes $(v_1, v_2), (v_2, v_3), \ldots$.
  - Il faut pas avoir des arètes de sens contraires !
- Graphe non-orienté : si $(v_1, v_2) \in E$ alors $(v_2, v_1) \in E$.
- Graphe bipartite :
  - $V = V_1 \cup V_2$ avec $V_1 \cap V_2$
  - Et $E \subseteq E_1 \times E_2 \cup E_2 \times E_1$.
- Généralisation : graphe pondéré $G = (V, E)$ où $E : V \times V \to \mathbb{N} \cup \{\infty\}$.
  - $E(v_1, v_2)$: poids de l'arète $E(v_1, v_2)$.
  - $E(v_1, v_2) = \infty$: arète absente.
  - Assez souvent on considère implicitement $E(v, v) = 0$ (boucles).

Applications

Énormement !

- Modélisation des routes/distances/connexions d'un pays/région.
- Modélisation des contraintes (transitives ou pas).
- Connexions dans divers types réseaux.
- Applications en chimie, physique, biologie, etc. etc.
- Modèles de représentation de donnés en informatique.

Représentation des graphes

- Matrice d'adjacence :
  - $V = \{0, \ldots, n-1\}$.
  - $E =$ matrice binaire $n \times n$.
  - Si G graphe pondéré alors $E =$ matrice entière étendue (élément supplémentaire représentant ∞).
  - Matrice rare : assez souvent $card(E) \ll n^2$.
- Liste d'adjacence :
  - $E : V \to 2^V$, c.à.d. E est une table contenant des listes (chaînées ou tabulaires) de noeuds, $E(v) =$ la liste des successeurs de v.
  - Éventuellement, pour faciliter certains types d'algorithmes, on rajoute aussi la table *inverse* : $E'(v) =$ prédécesseurs de v.
  - Adaptation aux graphes podérés : liste d'enregistrements (successeur, poids-de-l'arète).
- En fonction du problème, la représentation "liste de successeurs" peut être remplacée par une des différentes structures de données vues.
- En supplément : attributs des sommets :
  - Nom de la ville, coordonnées géographiques, etc.

Parcours des graphes en largeur

- “Franchir” chaque noeud du graphe, **en partant d’un noeud donné** x et dans l’ordre du plus court chemin les reliant à x.
- À la différence des arbres, on peut franchir un noeud y à partir de x par plusieurs chemins de longueurs égales ou différentes.
- Lorsqu’un noeud a déjà été franchi, il est marqué pour que lors du prochain franchissement on l’ignore.
- Le parcours en largeur peut s’implémenter à l’aide une file (FIFO) – à chaque itération :
  - On dé-file le premier noeud dans la file,
  - On rajoute ses successeur dans la file,
  - Et on “traite” le noeud dé-filé (afficher sa valeur, calculer qqchose avec celle-ci).
  - Puis on recommence.
- Nécessité d’avoir avec un champs supplémentaire (couleur) pour chaque noeud :
  - Blanc : noeud non-franchi.
  - Gris : noeud franchi (découvert) mais pas encore traité.
  - Noir : noeud traité.
  - Seuls les noeuds blancs sont traités.
  - On n’enfile pas les successeurs des noeuds noirs.
- Le parcours en largeur construit implicitement un arbre de couverture.

Parcours de graphes en largeur et plus court chemin

- L'algorithme de parcours de graphes en largeur peut être utilisé pour construire le plus court chemin entre le noeud donné x et chaque autre noeud.
- Il suffit de créer un champs supplémentaire dans chaque noeud qui pointera vers l'ancètre du noeud dans le plus court chemin.
- Alors, à chaque coup, après avoir dé-filé x et lorsqu'on rajoute (certains de) ses successeurs dans la file, on instancie le champs supplémentaire de chaque noeud enfilé avec un pointeur vers x.

Parcours des graphes en profondeur

- Franchir chaque noeud du graphe en partant d'un noeud donné, en essayant à chaque fois d'de s'éloigner encore plus de x.
- Correspond à un parcours utilisant une pile (LIFO) – à chaque itération :
  - On dépile le sommet de la pile.
  - On rajoute ses successeurs dans la pile.
  - Et on traite le noeud dépilé.
  - Puis on recommence.
- Toujours besoin du champs supplémentaire (couleur), avec la même sémantique.
- On peut aussi rajouter d'autres champs supplémentaires :
  - Date de premier franchissement.
  - Date de dernier franchissement.

Arbres sur les graphes

- On peut utiliser les algorithmes de parcours en largeur ou en profondeur pour générer des arbres qui contiennent exactement tous les noeuds du graphe.

Tri topologique

- Supposons qu'on a un ensemble de tâches à exécuter.
  - Construire une maison.
- Chaque tâche dépend d'un certain nombre d'autres tâches (couler la dalle du plafond avant couler les poteaux ?...).
- Cela nous génère un graphe de dépendance **acyclique** .
  - Une fois un mur érigé, on ne le refait pas ! enfin, on l'espère...
- Et on a une seule équipe qui peut travailler à la fois.
  - Ou un seul **processeur** qui peut exécuter les tâches !
- Comment lui donner les ordres pour que les tâches soient exécutées par cette seule équipe, tout en respectant les contraintes ?

Tri topologique

Étant donné un graphe $G = (V, E)$ acyclique, construire une liste L (table) qui contient tous les noeuds de V, avec la propriété suivante :

Si $L[i] = v_1$, $L[j] = v_2$ *et* $(v_1, v_2) \in E$ *alors* $i < j$.

Algorithme de tri topologique

Variante initiale :

1. On insère un noeud à la fin de la liste triée s'il n'a plus de prédécesseurs.
2. Chaque fois qu'on insère un noeud dans la liste triée, on l'élimine de la *liste des prédécesseurs* de tous ses successeurs.

Lemme

Un graphe orienté est acyclique si et seulement si la procédure précédente s'arrête avec un graphe pour lequel chaque noeud a une liste de prédécesseurs **vide**.

- Désavantage : à chaque itération il faut chercher, dans toute la liste des noeuds, un noeud qui n'a pas de prédécesseurs.
- Complexité de temps $\Theta(n^2)$!

Algorithme de tri topologique de complexité $\Theta(|V| + |A|)$

Utiliser le parcours en profondeur !

1. Appeler `parcours_profondeur` pour calculer les dates de fin de visite pour chaque sommet.
2. Chaque fois que le traitement d'un sommet se termine, l'insérer au début de la liste chaînée.
3. Retourner la liste chaînée des sommets.

Circuits spéciaux

Circuit eulerien dans un graphe non-orienté

Trouver (s'il existe !) un circuit qui passe exactement une fois par toutes les arètes d'un graphe.

- Propriété nécessaire et suffisante : chaque noeud est adjacent à un nombre pair d'arètes.
- Algorithme glouton de constructino d'un chemin eulerien (liste) :
  - À chaque instant, on choisit un successeur x du dernier noeud x inséré dans la liste.
  - On rajoute x à la liste.
  - On supprime l'arète (x, y).
  - Il se peut que y soit le premier noeud ! donc plus d'issue...
  - Alors on prend un autre noeud dans le chemin déjà construit et on rajoute là une autre boucle.
  - Jusqu'à ce qu'aucun noeud n'ait plus de successeur.

Circuits spéciaux (2)

Circuit hamiltonien

Trouver (s'il existe !) un circuit qui passe exactement une fois par tous les noeuds d'un graphe.

- Cette fois-ci plus d'algorithme "glouton" !
- Le parcours en profondeur peut nous donner une idée d'attaque, mais...
- ... chaque fois qu'on se rend compte que tous les successeurs d'un noeud sont noirs, il faut revenir sur ses pas, re-blanchir les noeuds et essayer une autre variante.
- Algorithme de type backtracking.

Composants fortement connexes

Définition

Soit $U \subseteq V$.

- Si pour toute paire de noeuds $u, v \in U$ il existe un chemin (orienté) reliant u à v et un deuxième chemin (orienté toujours) reliant v à u, alors on dit que U est **fortement connecté**.
- Une composante fortement connexe est un sous-ensemble de sommets U qui est fortement connecté et maximal (par rapport à cette propriété).
  - Donc pour tout $U' \supseteq U$, si U' est fortement connecté alors $U' = U$.

- *Ça s'appelle reviens* : dans un U qui est fortement connexe on a toujours des circuits qui visitent au moins une fois tous les noeuds.
- Pas forcements hamiltoniens !

Applications

- Langage infini d'un automate : recherche de composante fortement connexe atteignable et co-atteignable.
- Si le graphe est une représentation (réduite) d'une relation binaire ρ, on peut être intéressés de construire les composantes fortement connexes pour factoriser ρ et obtenir une relation d'ordre.