Quelques algorithmes sur les graphes

Dans ce TD nous étudions une des structures fondamentales de l'informatique : les graphes. On commence par détailler les différentes manières de représenter un graphe en caml, et on les compare à travers quelques algorithmes simples. Dans la troisième partie on étudie les algorithmes de parcours en *profondeur* et en *largeur* d'un arbre, qui sont à la base de la plupart des algorithmes sur les graphes, fournissant une manière pratique de parcourir tous les noeuds d'un graphe. Finalement, on s'intéressera à un algorithme de construction de circuits eulériens dans un graphe. Cette dernière partie, plus difficile, est directement inspirée d'une partie du sujet d'info ENS 2003.

1 Représentations

Définition 1 (Graphe orienté)**.** *Un* graphe orienté *G est la donnée d'un couple (V, E), où V est un ensemble fini de sommets et $E \subset V \times V$ une relation binaire sur V : l'ensemble des arêtes. Pour représenter graphiquement un graphe, on dessine les sommets comme des cercles, et les arêtes comme des flêches allant d'un sommet à un autre.*

Définition 2 (Graphe non orienté)**.** *Un* graphe non orienté *est un un graphe pour lequel la relation binaire E est symétrique et irréflexive (symétrique pour qu'il n'y aie pas d'orientation et irréflexive pour éviter les boucles d'un sommet sur lui-même). On peut ainsi voir E comme un ensemble de paires plutôt que de couples.*

Pour représenter un graphe il faut donc spécifier à la fois son ensemble de sommets et son ensemble d'arêtes. Comme on peut toujours se ramener à ce que l'ensemble des sommets soit l'ensemble $\{1, \ldots, n\}$, où $n = |V|$, on spécifiera simplement le nombre de sommets par leur nombre n et l'on définit

```
type sommet == int ; ;
```

La première représentation d'un graphe est par sa *matrice d'adjacence*. Si $G = (V, E)$ est un graphe, on note A la matrice de taille $n \times n$ à coefficients dans $\{0, 1\}$ définie par $A_{i,j} = 1$ si $(i, j) \in E$ et $A_{i,j} = 0$ sinon. La matrice d'adjacence d'un graphe non orienté est donc une matrice symétrique. Le type correspondant est

```
type Agraph = {Ataille : int ; matadj : int vect vect} ; ;
```

Une deuxième façon de représenter un graphe est par l'ensemble de ses *listes d'adjacence* : à chaque sommet i on associe la liste $l_i = \{j \in V, (i, j) \in E\}$; le graphe est alors spécifié par un vecteur contenant les listes d'adjacence de chaque sommet.

```
type Lgraph = {Ltaille : int ; listeadj : sommet list vect} ; ;
```

2 Premiers algorithmes

Définition 3 (taille)**.** *La taille d'un graphe $G = (V, E)$ est $t(G) = |V| + |E|$. C'est le paramètre qui sert habituellement à quantifier la complexité d'un algorithme de graphes (bien qu'il soit parfois utile d'étudier la complexité en fonction des deux paramètres $n = |V|$ et $m = |E|$).*

Question 1. *Écrire des procédures* `Ltaille : Lgraphe -> int` *et* `Ataille : Agraphe -> int` *renvoyant la taille d'un graphe donné dans chacune des deux représentations.*

Définition 4 (degré)**.** *Dans un graphe non orienté, le degré d'un sommet est le nombre d'arêtes qui lui sont incidentes. Dans le cas d'un graphe orienté, on distingue le degré entrant (nombre d'arêtes qui arrivent à ce sommet) du degré sortant (nombre d'arêtes qui en partent).*

Question 2. *Écrire des procédures* `{A,L}pred : {A,L}graphe -> sommet -> sommet list` *et* `{A,L}succ : {A,L}graphe -> sommet -> sommet list` *qui, étant donné un graphe orienté (V, E) et un sommet i, renvoient respectivement la liste des sommets j tels que* $(i, j) \in E$ *et celle des sommets j tels que* $(j, i) \in E$*. Comparer leurs complexités.*

Question 3. *En déduire des procédures* `{A,L}degre : {A,L}graphe -> sommet -> int * int` *qui renvoient le degré entrant et le degré sortant d'un sommet donné d'un graphe, dans les deux représentations. Donner la complexité, en fonction de la taille du graphe et du degré du noeud considéré, de chacune de ces deux procédures dans le pire cas. Quelle est leur complexité en moyenne ?*

Question 4. *Écrire des procédures* `{A,L}neigbour : {A,L}graphe -> sommet -> sommet -> bool` *telles que* `{A,L}neighbour i j` *renvoie true si et seulement si* $(i, j) \in E$*. Comparer leurs complexités.*

Définition 5 (carré d'un graphe)**.** *Le carré d'un graphe orienté* $G = (V, E)$ *est le graphe* $G^2 = (V, E^2)$ *tel que* $(u, v) \in E^2$ *si et seulement s'il existe* $w \in V$ *tel que* $(u, w) \in E$ *et* $(w, v) \in E$*. C'est-à-dire qu'il y a une arête dans le carré si, et seulement si, il y a un chemin de longueur exactement deux dans le graphe de départ.*

Question 5. *Écrire des procédures* `{A,L}carre : {A,L}graphe -> {A,L}graphe` *qui calculent le carré d'un graphe. Comparer leurs complexités.*

Comme vous avez pu le constater, chacune des deux représentations a ses avantages et ses inconvénients. Un des inconvénients majeurs de la représentation par matrice d'adjacence est sa taille, qui est $O(n^2)$ quel que soit le nombre d'arêtes de E, ce qui est beaucoup trop si le graphe est peu dense. Le plus gros inconvénient de la représentation par liste d'adjacence est qu'il est relativement coûteux de tester si deux sommets donnés sont reliés par une arête, ce que l'on a souvent besoin de faire en pratique.

On préférera souvent la représentation par listes d'adjacence si l'on est en train d'implémenter un algorithme de complexité linéaire ou quasi-linéaire, alors que si l'on considère un algorithme à la complexité quadratique ou plus, les différences entres les deux représentations sont moins cruciales.

3 Parcours d'un graphe

On choisit dans cette section la représentation d'un graphe sous forme de liste d'adjacence. On va étudier deux manières différentes de parcourir l'ensemble des sommets d'un graphe. On verra ensuite des applications algorithmiques de chacun de ces deux parcours.

3.1 Parcours en largeur

Le parcours en largeur d'un graphe permet un parcours du graphe en s'aidant d'une file contenant la liste des noeuds à visiter, et construite au fur et à mesure. Il procède comme suit :

- Visiter le noeud de départ
- Retirer le premier noeud de la file, et le visiter
- Ajouter tous ses voisins non encore visités *à la fin* de la file
- Si la file n'est pas vide retourner en 2

Question 6. *Écrire une procédure* `largeur : Lgraphe -> sommet -> sommet list` *qui renvoie une liste contenant l'ensemble des sommets du graphe, dans l'ordre d'un parcours en largeur commençant en un sommet passé en argument. On poura utiliser la procédure @ de concaténation de deux listes. On pourra également s'aider d'un vecteur auxiliaire pour marquer les sommets déjà visités.*

3.2 Parcours en profondeur

Le parcours en profondeur d'un graphe permet un parcours de l'ensemble des sommets du graphe. Il s'aide également d'une file contenant la liste des noeuds à visiter et procède comme suit :

- Visiter le noeud de départ
- Retirer le premier noeud de la file, et le visiter
- Ajouter tous ses voisins non encore visités *au début* de la file
- Si la file n'est pas vide retourner en 2

Question 7. *Écrire une procédure* `profondeur : Lgraphe -> sommet -> sommet list` *qui renvoie une liste contenant l'ensemble des sommets du graphe, dans l'ordre d'un parcours en profondeur commençant en un sommet donné en argument. On poura utiliser la procédure @ de concaténation de deux listes.*

3.3 Applications

Définition 6. *Soit $G = (V, E)$ un graphe. Un* chemin *de G est une suite $v_1, \ldots, v_p$ de sommets du graphe tels que pour tout $1 \leq i \leq p-1$, $(v_i, v_{i+1}) \in E$. p est la* longueur *du chemin. Un* cycle *est un chemin tel que $v_p = v_1$.*

Question 8 (plus court chemin)**.** *Proposer un algorithme permettant de calculer la distance (i.e. la longueur du plus court chemin) entre deux sommets dans un graphe. Le programmer et donner sa complexité.*

Définition 7. *Un graphe orienté G est dit* fortement connexe *si, pour tous sommets $v, v' \in V$ de G, il existe un chemin de v à v'.*

Question 9 (connexité)**.** *Déduire de l'algorithme du plus court chemin une procédure* `connexe : Lgraphe -> bool` *permettant de tester si un graphe orienté* `g` *donné en entrée est fortement connexe. Quelle est sa complexité ? On fera mieux à la question 16.*

Question 10 (acyclicité)**.** *Proposer un algorithme permettant de tester si un graphe non orienté possède un cycle. Le programmer et donner sa complexité.*

4 Circuits Eulériens

Soit $G = (V, E)$ un graphe et $(i, j) \in E$. Si $\pi = v_1, \ldots, v_p$ est un chemin de G de longueur p, et $e = (v, v') \in E$ une arête, on note $|\pi|_e$ le nombre de fois où π passe par e, c'est-à-dire le nombre d'indices i tels que $(v_i, v_{i+1}) = e$.

Définition 8. *Un cycle π de G est dit eulérien si et seulement si π passe par tout arc de G exactement une fois, autrement dit si $|\pi|_e = 1$ pour tout $e \in E$.*

Question 11. *Montrer que si G a un circuit eulérien, alors G vérifie la* loi de Kirchoff *: tout sommet v de G a son degré entrant et son degré sortant égaux.*

Définition 9. *Un* sommet isolé *de G est un sommet de degré entrant et de degré sortant nuls.*

Question 12. *Montrer que si G a un circuit eulérien, et si G n'a pas de sommet isolé, alors G est fortement connexe.*

Question 13. *On considère un chemin π tel que $|\pi|_e \leq 1$ pour tout arc e de G, et maximal avec cette propriéte. Montrer que si G vérifie la loi de Kirchoff, alors π est un circuit.*

Question 14. *Si de plus $|\pi|_e = 0$ pour un arc e, et si G est fortement connexe, montrer qu'il existe un sommet v par lequel passe π, et un circuit π' de v à v passant par e, et qui ne passe que par des arcs par lesquels π ne passe pas.*

En déduire que π est eulérien. Ainsi, un graphe fortement connexe vérifiant la loi de Kirchoff a un circuit eulérien.

Question 15. *Écrire une procédure* `euler` *qui prend en entrée un graphe* `g` *fortement connexe (dans une représentation de votre choix), et renvoie un circuit eulérien de* `g`*, sous la forme d'une liste de sommets décrivant le circuit, s'il en a un, et la liste vide sinon.*

Question 16. *En déduire une procédure* `connexe` *permettant de teste si un graphe* `g` *est fortement connexe. Quelle est sa complexité ? La comparer à celle de l'algorithme de la question 9.*

De manière analogue aux circuits eulériens, on peut définir les circuit hamiltoniens : un circuit de G est hamiltonien s'il passe par chaque sommet exactement une fois. Malgré des définitions semblables, ces deux problèmes ne sont pas du tout de la même difficulté : nous venons de voir que le problème de la recherche d'un circuit eulérien pouvait être résolu en temps linéaire, alors que le problème de la recherche d'un circuit hamiltonien est NP-complet (il est très lié au célèbre problème du voyageur de commerce)...