TP 3— Graphes et parcours de graphes

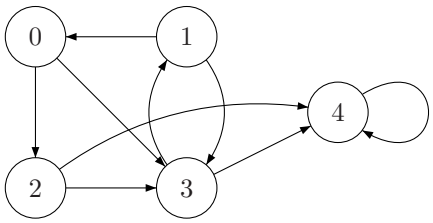
http://www.liafa.jussieu.fr/~nath/tp3/tp3.pdf

Les questions sont les bienvenues et peuvent être envoyées à nathanael.fijalkow@gmail.com.

L'objectif de ce TP est d'introduire un objet fondamental en informatique, aussi bien théorique que pratique, les graphes. On explore les différentes options pour manipuler un graphe en programmation, puis on introduit les parcours en largeur et en profondeur, et deux applications : la détection de circuit et le calcul de composantes connexes dans le cas non-orienté.

1 Graphes

Intuitivement, un graphe est un ensemble de points reliés par des flèches. Vous devez vous représenter un graphe par un dessin, comme celui-ci:



Un graphe est la donnée $G = (V, E)$ où V est l'ensemble des sommets (V pour vertex) et $E \subseteq V \times V$ est l'ensemble des arcs (E pour edge).

Il y a plusieurs distinctions à faire. En général, les graphes que l'on considére sont **finis** (c'est-à-dire qu'ils ont un ensemble de sommets $\{0, \dots, n-1\}$ pour un certain $n \in \mathbb{N}$) ; dans ce TP, les graphes seront finis. On notera $n = |V|$ le nombre de sommets et $m = |E|$ le nombre d'arcs.

Les graphes que l'on considère sont **simples** : pour un couple de sommets (u, v), il y a au plus un arc entre u et v. Ainsi on peut représenter l'ensemble des arcs par une relation $E \subseteq V \times V$, telle que $(u, v) \in E$ s'il y a un arc entre u et v.

Les graphes sont soit **orientés**, ou ("directed" en anglais), soit **non-orientés** ("undirected" en anglais).

Dans le cas orienté, les arcs ont une origine et un but. Un arc est donc un couple ordonné (u, v) où u et v sont des sommets. Le graphe ci-dessus est orienté.

Dans le cas non-orienté, les arcs sont juste des liens entre deux sommets, on ne se soucie donc pas de l'ordre du couple. On parle d'arête et non plus d'arc. *Le cas non-orienté est un cas particulier du cas orienté*, où $(u, v) \in E$ si et seulement si $(v, u) \in E$. Dans ce TP, les graphes sont orientés.

2 Représentation d'un graphe

Pour représenter un graphe on peut utiliser la matrice d'adjacence M, de taille $n \times n$, qui vérifie :

$$M_{u,v} = \begin{cases} 1 & \text{si } (u, v) \in E \\ 0 & \text{sinon} \end{cases}$$

▷ **Question 1.** Que représente la transposée de M? Et l'égalité entre M et sa transposée? ◁

▷ **Question 2.** On munit $\{0, 1\}$ des lois $\oplus$ et $\otimes$, données par :

$\oplus$	0	1
0	0	1
1	1	1

$\otimes$	0	1
0	0	0
1	0	1

Interpréter ces lois par des opérations logiques. On considère l'ensemble des matrices sur $\{0, 1\}$ muni de ces deux lois. Attention, ce n'est pas un anneau, c'est un semi-anneau (il n'y a pas d'inverse pour la loi additive).

Notons $D = I \oplus M$, où I est la matrice identité de taille $n \times n$. Définir une multiplication de matrices, puis expliquer comment on peut définir D^k (attention, il y a une subtilité). Interpréter D, puis D^2, puis D^n en termes de chemin. En déduire un algorithme qui détermine s'il existe un chemin entre deux sommets u et v donnés, en calculant la suite $D, D^2, D^3, \dots, D^n$. Améliorer cet algorithme en calculant $D, D^2, D^4, \dots, D^{2^k}$. Démontrer sa correction et estimer sa complexité. ◁

Pour représenter un graphe on peut aussi utiliser des listes d'adjacence : Succ_u est la liste des successeurs (immédiats) de u, Pred_u est la liste des prédecesseurs (immédiats) de u. On conserve ces listes dans un tableau de taille n.

▷ **Question 3.** Écrire une fonction `de_succ_a_pred` qui étant donné le tableau contenant la liste des successeurs, construit le tableau contenant la liste des prédecesseurs. ◁

▷ **Question 4.** Écrire une fonction `de_pred_a_succ` qui étant donné le tableau contenant la liste des prédecesseurs, construit le tableau contenant la liste des successeurs. ◁

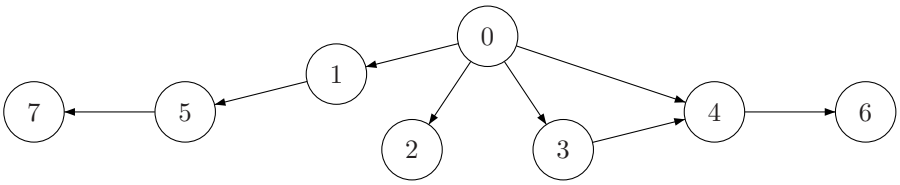
▷ **Question 5.** Quelles sont les complexités de `de_succ_a_pred` et `de_pred_a_succ`? Elles devraient être $O(m+n)$. ◁

3 Parcours de graphes

Considérons un graphe $G = (V, E)$, la bordure $\Gamma(T)$ d'une partie T de V est le sous-ensemble des sommets de $V \setminus T$ qui sont les extrémités d'un arc dont l'origine est dans T.
Un parcours de G depuis s est une liste des sommets L de G telle que :

- chaque sommet de G apparaît une fois et une seule dans L;
- chaque sommet de L (sauf le premier) appartient à la bordure du sous-ensemble des sommets placés avant lui dans L, si cette bordure est non vide.

Il y a essentiellement deux paradigmes pour parcourir un graphe : en largeur et en profondeur. Le parcours en largeur consiste, à partir d'un sommet, à explorer tous les voisins, puis à continuer le parcours à partir de chaque voisin. À l'inverse, le parcours en profondeur explore chaque branche les unes après les autres. Par exemple, sur le graphe suivant, que l'on parcourt depuis le sommet 0 :



en largeur : 0; 1; 2; 3; 4; 5; 6; 7.
en profondeur : 0; 1; 5; 7; 2; 3; 4; 6.

▷ **Question 6.** Le code ci-contre retourne sous forme de liste un parcours du graphe à `n` sommets décrit par `tab_succ` depuis le sommet `sommet_depart`. Quel est ce parcours? Comment modifier une ligne de ce code pour obtenir l'autre parcours? ◁

```
let parcours n tab_succ sommet_depart =
let tab_non_parcourus = make_vect n true in
 let rec boucle liste_a_traiter = match liste_a_traiter with
   | [] -> []
   | sommet_courant::suite
   when tab_non_parcourus.(sommet_courant)->
     tab_non_parcourus.(sommet_courant) <- false ;
     let liste_a_traiter = tab_succ.(sommet_courant) @ suite in
     sommet_courant::(boucle liste_a_traiter)
   | _::suite -> boucle suite
in boucle [sommet_depart] ;;
```

▷ **Question 7.** Combien y a-t-il d'appels à la fonction `boucle`? ◁

▷ **Question 8.** Dans les deux cas, on utilise la concaténation de liste `@`: quel est le coût en opérations élémentaires? Quelles sont les structures de données adaptées dans chacun des cas pour maintenir `liste_a_traiter`? ◁

4 Applications

On va utiliser le parcours en profondeur pour obtenir des informations sur le graphe.

4.1 Calcul des composantes connexes dans un graphe non-orienté

On se place dans le cas des graphes **non-orientés**. Un sous-graphe G' de G est un graphe (V', E') tel que V' est un sous-ensemble de V et E' l'ensemble des arêtes de G ayant ses deux extrémités dans V'.

Une composante connexe de G est un sous-graphe G' de G tels que pour tous sommets u et v de G' il existe un chemin de u a v dans G', G' étant maximal pour cette propriété. Une composante connexe de G est donc entièrement décrite par la liste de ses sommets.

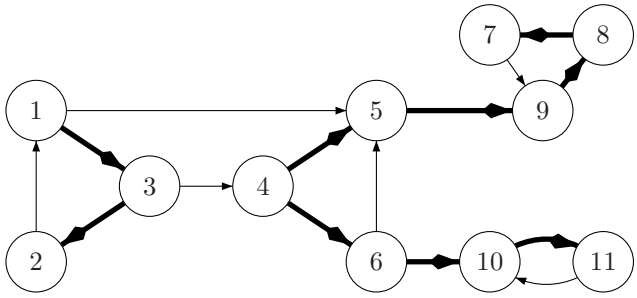
▷ **Question 9.** Écrire une fonction `calcul_composantes_connexes` qui étant donné un graphe non-orienté, retourne la liste de ses composantes connexes. ◁

4.2 Détection de circuit dans un graphe orienté

On revient au cas (général) des graphes **orientés**.

Soit $G = (V, E)$ un graphe et $L = [x_1; x_2; \ldots; x_n]$ un parcours quelconque de G. On associe à L le sous-graphe $\mathcal{F}(L)$ dont les sommets sont tous les sommets de G et les arcs sont choisis ainsi : pour chaque x_j, s'il y a un arc $(x_i, x_j) \in E$ pour un certain $i < j$, alors on choisit un tel arc.

▷ **Question 10.** Montrer que $\mathcal{F}(L)$ est une forêt couvrante, c'est-à-dire un ensemble d'arbres disjoints couvrant le graphe. ◁



Notons $r(x)$ le rang d'un sommet x dans L: le rang de x_i est i. En dehors des arcs de $\mathcal{F}(L)$, le parcours L permet de distinguer trois autres classes d'arcs dans G.

- Un arc (x, y) est arrière si y est un ascendant de x dans $\mathcal{F}(L)$;
- Un arc (x, y) est avant si x est un ascendant de y dans $\mathcal{F}(L)$;
- Un arc (x, y) est transverse si ses deux extrémités appartiennent à deux arbres différents, ou si x et y ont un ancêtre commun z dans $\mathcal{F}(L)$ distinct de x et de y.

Dans l'exemple, pour le parcours $[4; 5; 9; 8; 7; 6; 10; 11; 1; 3; 2]$, les arcs $(7, 9)$, $(11, 10)$ et $(2, 1)$ sont arrières, les arcs $(6, 5)$, $(1, 5)$ et $(3, 4)$ sont transverses et il n'y a pas d'arc avant.

▷ **Question 11.** Montrer que si (x, y) est un arc transverse pour un parcours en profondeur L, alors $r(y) < r(x)$. ◁

▷ **Question 12.** Soit L un parcours en profondeur fixé. Montrer que G est sans circuit si et seulement s'il n'existe pas d'arc arrière. En déduire un algorithme de complexité $O(n + m)$ qui détecte si un graphe possède un circuit. ◁

5 Questions difficiles

▷ **Question 13.** Revenons sur les calculs de distances. On a défini une structure algébrique : le semi-anneau $(\{0, 1\}, \oplus, \otimes)$, on a fait du calcul matriciel, puis on a interprété les matrices obtenues en termes de

chemins. Considérons maintenant que les arcs sont valués, par une fonction $v : E \to \mathbb{R}$. Comment calculer le poids minimal d'un chemin de u à v? Le poids maximal? La longueur des p plus courts chemins? ◁

▷ **Question 14.** Il y a une troisième application du parcours en profondeur : le calcul des composantes fortement connexes dans un graphe orienté. On définit la notion de connexité forte de la même manière que dans le cas non-orienté, mais ici l'existence d'un chemin de u vers v n'implique pas l'existence d'un chemin de v vers u. Trouver un algorithme en $O(m + n)$ qui calcule les composantes fortement connexes d'un graphe orienté, en utilisant deux parcours en profondeur successifs. En cas de manque d'inspiration, se reporter au cours d'algorithmique de Beauquier, Berstel et Chrétienne, à partir de la page 130 (PDF disponible sur la page du cours). ◁

▷ **Question 15.** Dessiner un chameau. ◁