TD n° 7 - Parcours

Exercice 1. *Graphes bipartis*

Un graphe non orienté $G = (V, E)$ est dit *biparti* s'il existe une partition de V en deux indépendants X et Y, autrement toutes les arêtes $xy \in E$ ont une extremité $x \in X$ et l'autre extremité $y \in Y$. Une autre manière d'exprimer cette définition est de dire que ce sont les graphes 2-colorables (une couleur = un indépendant). On note parfois un tel graphe $G = (X, Y, E)$.

1. Montrer qu'un graphe est biparti si et seulement si il n'admet aucun cycle de longueur impaire (König, 1936).

2. Décrire un algorithme linéaire qui reconnaît si un graphe donné par ses listes d'adjacences est biparti.

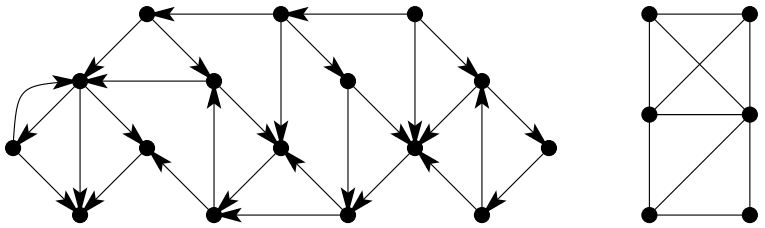
Exercice 2. *Parcours en profondeur (DFS)*

Les fonctions ci-dessous implémentent de manière récursive les parcours en profondeur, avec une double numérotation pour chaque sommet v où $\texttt{pre}[v]$ désigne le début de la visite de v et $\texttt{post}[v]$ désigne la fin de la visite de v.

DFS(v) : routine d'exploration
1. $\texttt{pre}[v] \leftarrow t \leftarrow t + 1$;
2. pour tout w voisin de v sans aucun numéro, faire DFS(w) ;
3. $\texttt{post}[v] \leftarrow t \leftarrow t + 1$;

DFS(G) : parcours de tout le graphe
1. $\forall v$, $\texttt{pre}[v] \leftarrow \texttt{nil}$, $\texttt{post}[v] \leftarrow \texttt{nil}$; $t \leftarrow 0$;
2. Tant qu'il existe un sommet s non numéroté, DFS(s)

1. Exécuter cet algorithme de parcours en calculant les numérotations $\texttt{pre}[.]$ et $\texttt{post}[.]$ sur les deux exemples ci-dessous.



2. Montrer que chaque appel DFS(v) va numéroter tous les sommets non numérotés, accessibles par un chemin non numéroté depuis v. Quelle est la complexité de DFS(G), G étant donné par ses listes d'adjacence ?

3. Montrer les propriétés suivantes de la double numérotation DFS :

(1) les intervalles $[\texttt{pre}[v], \texttt{post}[v]]$ sont soit imbriqués, soit disjoints,

(2) $\texttt{pre}[v] < \texttt{pre}[u] < \texttt{post}[u] < \texttt{post}[v]$ ssi v ancêtre de u dans un des arbres de parcours de G,

(3) il n'existe pas d'arc uv tel que $\texttt{pre}[u] < \texttt{post}[u] < \texttt{pre}[v] < \texttt{post}[v]$.

4. Montrer qu'un graphe orienté G est sans cycle (DAG, Directed Acyclic Graph) ssi il n'existe aucun arc retour dans un DFS de G. En déduire un algorithme linéaire de reconnaissance des DAG.

5. L'algorithme suivant calcule les composantes fortement connexes d'un graphe orienté :

Algorithme de Kosaraju/Sharir
1- DFS(G) en calculant la numérotation `post`
2- DFS(G^{-1}) en choisissant sources selon `post` $\searrow$
3- Sortie : comp. fortement connexes = arbres de ce dernier parcours

Quel est la complexité de cet algorithme ? Prouver sa correction.

Exercice 3. *Graphes pour 2-SAT*

Le problème 2-SAT consiste à déterminer la satisfiabilité d'une formule booléenne en *Forme Normale Conjonctive d'ordre 2* (2-FNC). Une telle formule est une conjonction de clauses, chaque clause étant elle-même constituée d'une disjonction de deux littéraux. Un littéral dans une formule booléenne est une variable ou sa négation. Exemple de formule 2-FNC : $(x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$. Dans cet exemple, la formule est satisfiable : il suffit de prendre l'assignation suivante $x_1 =$ `vrai`, $x_2 =$ `faux` et x_3 quelconque.
En remarquant que la clause $\ell_1 \vee \ell_2$, où ℓ_1 et ℓ_2 sont deux littéraux est équivalente à la formule $\neg\ell_1 \to \ell_2$ et à la formule $\neg\ell_2 \to \ell_1$, on associe à chaque formule 2-FNC sur les variables $x_1, \ldots, x_n$ un graphe orienté $G = (V, E)$ où $V = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ et $E = \{(\neg\ell_1, \ell_2), (\neg\ell_2, \ell_1) \mid \ell_1 \vee \ell_2$ est une clause de la formule$\}$.

1. Dessiner les graphes correspondants aux formules suivantes :
- $(\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee \neg x_2)$
- $(\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_4)$

2. Etant donnée une formule 2-FNC, trouver une condition nécessaire et suffisante sur son graphe pour qu'elle soit satisfiable.

3. En déduire que 2-SAT $\in$ P, ainsi qu'un algorithme efficace qui trouve une assignation des variables qui satisfait la formule si celle-ci est satisfiable.