Licence Informatique — **4TIN501U – Algorithmique des graphes**
Université de Bordeaux — **Aucun document autorisé**

Devoir Surveillé du 5 novembre 2019
durée 1h30

Dans tout le sujet, on conviendra que, dans les différentes structures de données modélisant les graphes, les sommets sont rangés dans l'ordre croissant de leur numéro.

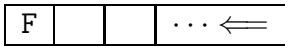
Exercice 1

Soit le graphe simple, non orienté $G = (V, E)$, donné par sa matrice d'incidence ci-dessous.

$$\begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{array} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

1. Dessiner le graphe G.
2. Quel est son degré minimum ? Son degré maximum ? Est-il connexe ?
3. Appliquer l'algorithme du ***parcours en largeur*** `PL`(G, s_0) pour calculer la distance entre le sommet s_0 et tous les autres sommets. Expliciter l'ordre dans lequel les sommets sont enfilés (et donc défilés) dans la file `F`, ainsi que les valeurs de `pere[`v`]` et `d[`v`]` pour tous les sommets dans un tableau. Vous prendrez soin de ranger dans l'entrée du tableau les sommets par ordre lexicographique (comme ci-après).

v	`pere[`v`]`	`d[`v`]`
s_0		
s_1		
$\vdots$	$\vdots$	$\vdots$
s_7		



Exercice 2

Pour chacune des propositions suivantes décider si elle est vraie ou fausse ; si elle est vraie, donner une preuve, si elle est fausse, donner un contre-exemple.

1. Soit u et v deux sommets (distincts) d'un graphe non-orienté tels que $dist(u, v) = k$ (pour un $k \geq 1$). Alors il existe un sommet w voisin de v tel que $dist(u, w) = k - 1$. Vrai ou faux ?
2. Soit u et v deux sommets (distincts) d'un graphe non-orienté. Supposons qu'il existe un sommet w voisin de v tel que $dist(u, w) = k - 1$ pour un k entier naturel. Alors on a dans ce cas là $dist(u, v) = k$. Vrai ou faux ?

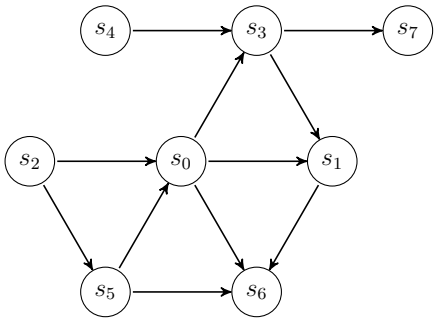
Exercice 3

1. Soit G un graphe non-orienté simple ayant n sommets et soit s un de ses sommets. Le graphe G étant représenté par matrice d'adjacence $A[i, j]$, proposer un algorithme qui détermine si s a au moins un voisin de degré 1. Quel est la complexité de votre algorithme dans le meilleurs des cas ? Dans le pire des cas ? Justifier votre réponse.
2. Soit G un graphe non-orienté non simple représenté par listes de successeurs $Adj[u]$. Proposer un algorithme qui détermine si tous les sommets du graphe sont origine (et extrémité) d'une boucle. Quel est la complexité de votre algorithme dans le meilleurs des cas ? Dans le pire des cas ? Justifier votre réponse.

Exercice 4

L'objectif de cet exercice est de concevoir un algorithme qui, étant donné un graphe orienté G sans circuit et deux sommets s et t de G, calcule le nombre de chemins de s à t en temps linéaire.

1. Soit G_1 le graphe dessiné ci-après. Expliciter tous les chemins de s_5 à s_6 (il y en a quatre).



2. Effectuer le parcours en profondeur de G_1 en explicitant, dans un tableau, les dates de début et de fin de visite, ainsi que le père pour chaque sommet de G_1.
3. Soit G un graphe orienté sans circuit quelconque. Soit t un sommet de G. Pour tout sommet u de G, notons $c[u]$ le nombre de chemins allant de u à t dans G. En particulier, on a $c[t] = 1$ et $c[x] = 0$ pour tout sommet x depuis lequel t n'est pas accessible. Montrer que pour tout sommet u on a
$$c[u] = \sum_{uv \in E(G)} c[v]$$
où la somme prend en compte tous les voisins sortants (les successeurs) v de u.
4. Calculer les valeurs des étiquettes $c[\,]$ pour les sommets du graphe G_1 et $t = s_6$.

Afin de répondre au problème cité au début de l'exercice, il suffit en fait de calculer $c[s]$. Nous allons quand même calculer $c[u]$ pour tout sommet de G, et cela pendant un (et un seul) parcours en profondeur. Rappelons que dans un graphe orienté G sans circuit, si uv est un arc, alors $f(v) < f(u)$ (où $f(x)$ désigne la date de fin de visite de x pour un sommet x de G). Ceci dit, observons que la formule de la question 3 peut être appliquée au moment de la fin de visite pour calculer $c[u]$, à condition que les valeurs de $c[v]$ pour ses voisins sortants soient déjà connues.

5. Modifier l'algorithme `PP(G)` en `NB_CHEMINS(G, s, t)` ainsi que la fonction `Visiter(u)` pour mettre en place le calcule de $c[u]$ pendant un parcours en profondeur. Expliciter les numéros de lignes à modifier et/où à ajouter au code existant. Ne pas oublier l'initialisation des étiquettes $c[\,]$.

FIN.

Rappel :

```
0   PP(G)
1     pour chaque sommet u de V faire
2         couleur[u] <- BLANC
3         pere[u] <- nil
4     temps <- 0
5     pour chaque sommet u de V faire
6         si couleur[u] = BLANC
7             alors Visiter_PP(u)
```

```
0   Visiter_PP(u)
1     couleur[u] <- GRIS
2     d[u] <- temps <- temps + 1
3     pour chaque v de Adj[u] faire
4         si couleur[v] = BLANC
5             alors pere[v] <- u
6                   Visiter_PP(v)
7     couleur[u] <- NOIR
8     f[u] <- temps <- temps + 1
```

```
0   PL(G,s)
1     pour chaque sommet u de V[G] \ {s} faire
2         couleur[u] <- BLANC
3         d[u] <- infini
4         pere[u] <- nil
5     couleur[s] <- GRIS
6     d[s] <- 0
7     pere[s] <- nil
8     Enfiler(F, s)
9     tant que non vide(F) faire
10        u <- tete(F)
11        pour chaque v de Adj(u) faire
12            si couleur[v] = BLANC
13                alors couleur[v] <- GRIS
14                      d[v] <- d[u] + 1
15                      pere[v] <- u
16                      Enfiler(F, v)
17        Defiler(F)
18        couleur[u] <- NOIR
```