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

Devoir Surveillé du 5 novembre 2018
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

On considère le graphe G_1 ci-dessous :

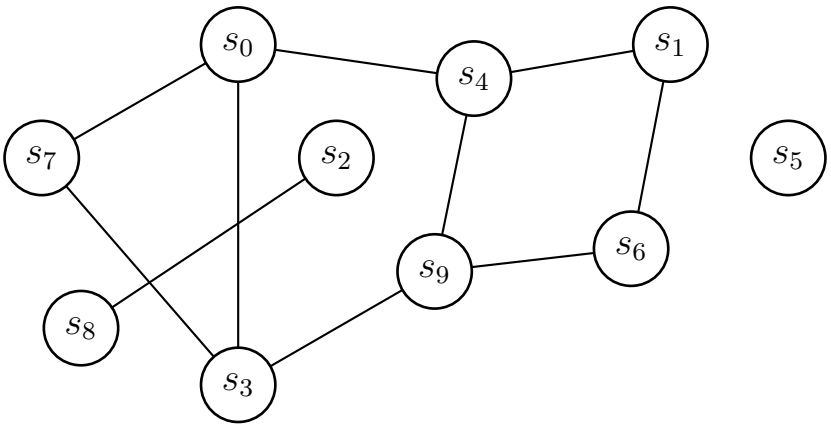

FIGURE 1 – Le graphe G_1

1. Donner sa représentation par listes d'adjacence.
2. Donner sa représentation par matrice d'adjacence.
3. Appliquer à G_1 l'algorithme de *parcours en largeur* à partir du sommet s_7. En recopiant et remplissant les tableaux ci-dessous, donner l'ordre d'entrée des sommets dans la file `F` et le contenu final des tableaux `d[]` et `pere[]`.

sommet `u`	`d[u]`	`pere[u]`
s_0		
s_1		
s_2		
s_3		
s_4		
s_5		
s_6		
s_7		
s_8		
s_9		

file `F`	s_7, ...

4. Un graphe est bicoloriable si ses sommets peuvent être coloriés avec deux couleurs, de telle sorte que deux sommets voisins ne soient pas de la même couleur. Est-ce que le graphe G_1 est bicoloriable ? Justifiez votre réponse.

Exercice 2

Soit $G = (V, E)$ un graphe non orienté de sommets V et d'arêtes E. Soit $e = \{u, v\}$ une arête reliant des sommets u et v distincts.

La *contraction* d'une arête $e = uv$, $u \neq v$ est une opération qui transforme G en un autre graphe de la manière suivante : d'abord, l'arête e est supprimée, puis, les sommets u et v sont identifiés (fusionnés).

Par exemple, la contraction d'une arête (n'importe quelle) d'une chaîne élémentaire de longueur k produit une chaîne élémentaire de longueur $k-1$ ($k \geq 1$) ; pareil, la contraction d'une arête d'un cycle élémentaire de longueur k produit un cycle élémentaire de longueur $k-1$ ($k \geq 2$).

1. Si on contracte une arête d'un graphe simple, le graphe obtenu est toujours simple ? Justifier.
2. Montrer que si on contracte une arête d'un graphe connexe, alors le graphe obtenu est connexe.
3. Montrer que si on contracte une arête d'un arbre, alors le graphe obtenu est un arbre.
4. On apelle la *maille* d'un graphe, la taille de son plus petit cycle élémentaire. Montrer que si on contracte une arête d'un graphe, alors la maille diminue au plus de 1.
5. Supposons que le degré maximum de G est d. Quelle est la plus grand valeur possible du degré maximum d'un graphe obtenu par une contraction d'une arête de G ?

Exercice 3

L'objectif de cet exercice est de concevoir un algorithme efficace qui calcule la longueur d'un plus long chemin dans un graphe orienté sans circuit à l'aide du parcours en profondeur.

Tous les graphes de cet exercice sont orientés et sans circuit.

Soit G un graphe orienté sans circuit. Notons $t(G)$ la longueur d'un plus long chemin dans G. Pour un sommet u, notons $t(u)$ la longueur d'un plus long chemin ayant u comme sommet de départ. Observez que

$$t(G) = \max_{u \in V(G)} t(u).$$

1. Montrer que dans un graphe orienté sans circuit, tout chemin est élémentaire.
2. Montrer que $t(u) = 0$ si et seulement si u est un sommet puits (sommet sans successeur).
3. Notons $N^+(u)$ l'ensemble des voisins sortants (successeurs) d'un sommet u. Montrer que

$$t(u) = 1 + \max_{v \in N^+(u)} t(v)$$

dès que $N^+(u)$ est non-vide.

(Rappel de cours) Il est connu qu'à l'issue d'un parcours en profondeur d'un graphe orienté sans circuit il n'y a aucun arc de retour. Par conséquent, pour tout arc uv d'un graphe orienté sans circuit, à l'issue d'un parcours en profondeur on a $f(u) > f(v)$ (où $f(x)$ désigne la date de la fin de visite d'un sommet x).

4. Montrer que pour chaque sommet u, le calcul de $t(u)$ ne dépend que de valeurs $t(v)$ telles que $f(v) < f(u)$.
5. Proposer une modification de l'algorithme `PP(G)` (et de la fonction `Visiter_PP(u)`) permettant de calculer les valeurs $t(u)$ pendant un parcours en profondeur d'un graphe orienté sans circuit à l'aide de la formule de la question 3.
   Expliciter les endroits où ajouter / modifier des lignes.
6. Quelle est la complexité de votre algorithme ? Justifier.

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
```

```
TRI_TOPOLOGIQUE(G)
  1/ appeler PP(G) pour calculer les temps de fin de traitement f[v]
     pour chaque sommet v
  2/ chaque fois que le traitement d'un sommet s'achève, l'insérer
     au début d'une liste chaînée
  3/ retourner la liste chaînée des sommets
```