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 :

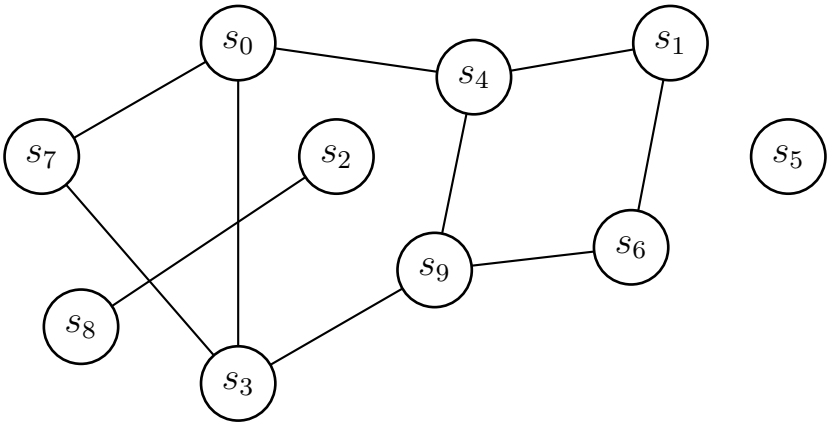

FIGURE 1 – Le graphe G_1

1. Donner sa représentation par listes d'adjacence.

sommet u	Adjacences
s_0	s_3, s_4, s_7
s_1	s_4, s_6
s_2	s_8
s_3	s_0, s_7, s_9
s_4	s_0, s_1, s_9
s_5	
s_6	s_1, s_9
s_7	s_0, s_3
s_8	s_2
s_9	s_3, s_4, s_6

2. Donner sa représentation par matrice d'adjacence.

$M(i,j)$	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
s_0	0	0	0	1	1	0	0	1	0	0
s_1	0	0	0	0	1	0	1	0	0	0
s_2	0	0	0	0	0	0	0	0	1	0
s_3	1	0	0	0	0	0	0	1	0	1
s_4	1	1	0	0	0	0	0	0	0	1
s_5	0	0	0	0	0	0	0	0	0	0
s_6	0	1	0	0	0	0	0	0	0	1
s_7	1	0	0	1	0	0	0	0	0	0
s_8	0	0	1	0	0	0	0	0	0	0
s_9	0	0	0	1	1	0	1	0	0	0

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	1	s_7
s_1	3	s_4
s_2	∞	`nil`
s_3	1	s_7
s_4	2	s_0
s_5	∞	`nil`
s_6	3	s_9
s_7	0	`nil`
s_8	∞	`nil`
s_9	2	s_3

file `F`	$s_7, s_0, s_3, s_4, s_9, s_1, s_6$

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.

   Pour qu'un graphe soit bicoloriable, il faut qu'il existe une bicoloration de telle sorte que deux sommets adjacents ne soient pas de la même couleur. On a vu en TD que seul les graphes bipartis étaient bicoloriables. Nous avons démontré en TD que les graphes contenant un cycle élémentaire de taille impaire ne pouvaient pas être biparti et donc pas bicoloriable. Or G_1 contient un cycle élémentaire (s_0, s_3, s_7) de taille 3. Ainsi, G_1 ne peut pas être bicoloriable.

Exercice 2

Soit $G = (V, E)$ un graphe non orienté où V représente l'ensemble des sommets et E représente l'ensemble de arêtes du graphe. Soit $e = uv$ une arête reliant les deux sommets u et v avec u et v des sommets distincts.

La *contraction* d'une arête $e = uv$ (avec $u \neq v$) 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 laquelle) d'une chaîne élémentaire de longueur k produit une chaîne élémentaire de longueur $k-1$ ($k \geq 1$) ; idem, 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-il toujours simple ? Justifier ou donner un contre-exemple.

   Non, pas toujours. La contraction d'une arête d'un triangle (cycle élémentaire de longueur 3, qui est simple) produit un graphe contenant deux arêtes parallèles, donc pas simple.

2. Montrer que si on contracte une arête d'un graphe connexe, alors le graphe obtenu est encore connexe.

   Soit G un graphe connexe, soit $e = uv$ une arête de G. Notons $G' = G/e$ le graphe obtenu par la contraction de e.
   Soient x et y deux sommets de G. Puisque G est connexe, il existe une chaîne reliant x et y dans G. (D'après le cours, il existe aussi une chaîne élémentaire les reliant.) Si cette chaîne n'emprunte pas l'arête e, elle relie x et y dans G' également. Si elle contient l'arête e, il suffit de supprimer (toute occurrence de) e pour obtenir une chaîne reliant x et y dans G'. Observer que la justification reste valide même dans le cas où un des sommets x et y est u ou v.

3. Montrer que si on contracte une arête d'un arbre, alors le graphe obtenu est un arbre.

   D'après la question précédente, la contraction d'une arête d'un graphe connexe (en particulier, d'un arbre) produit un graphe connexe. Puisque le graphe donné est un arbre, son nombre d'arêtes est le nombre de sommets moins un. La contraction d'une arête fait diminuer le nombre de sommets et aussi le nombre d'arêtes de 1. La propriété reste valide pour le graphe obtenu également. Le graphe obtenu est connexe et il a un bon nombre d'arêtes, d'après un théorème du cours il est un arbre.

4. On appelle 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.

   Si on contracte une arête n'appartenant à aucun cycle élémentaire, la longueur des cycles du graphe donné ne change pas, la maille non plus. Si on contracte une arête appartenant à un cycle (ou plusieurs), la longueur de chaque cycle la contenant diminue de 1. Par définition, la maille est soit conservée telle quelle (si aucun cycle minimal n'est touché) soit diminue de 1 (si au moins un cycle minimal contient l'arête contractée).

5. Supposons que le degré maximum de G soit Δ. Quelle est la plus grand valeur possible du degré maximum d'un graphe obtenu par une contraction d'une arête de G ?

   Si l'arête contractée relie deux sommets de degré Δ, dans le graphe obtenu le nouveau sommet correspondant à la fusion des extrémités de l'arête contractée est de degré $2\Delta - 2$. Les autres sommets gardent leurs degrés. La valeur recherchée est donc $2\Delta - 2$.

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, $V(G)$ représente l'ensemble des sommets du graphe. 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.

Par l'absurde. Supposons qu'il existe un chemin non-élémentaire. Alors, dans ce chemin, au moins un sommet se répète. Le sous-chemin entre deux occurrences d'un même sommet est un circuit, par définition. Absurde !

2. Montrer que $t(u) = 0$ si et seulement si u est un sommet puits (sommet sans successeur).

   Si $t(u) = 0$, alors le chemin le plus long à partir de u est de longueur 0, c-à-d chemin trivial sans arc. Il n'y a donc par d'arc sortant de u.
   Si u est un sommet puits, alors le seul chemin qui peut commencer à u est le chemin trivial, donc $t(u) = 0$.

3. Notons $N^+(u)$ l'ensemble des voisins sortants (successeurs) d'un sommet u. Montrer que

$$t(u) = 1 + \max_{v \in N^+(u)} t(v), \text{ dès que } N^+(u) \text{ est non-vide.}$$

   D'après la question précédente, dès que $N^+(u)$ est non-vide, on a $t(u) \geq 1$.
   Soit C un chemin ayant u comme sommet de départ. Soit v le successeur de u le long de C. Évidemment, v est un voisin sortant de u et le chemin $C \setminus u$ (obtenu à partir de C en omettant le sommet u) est un chemin ayant v comme sommet de départ, alors il est de longueur au maximum $t(v)$.
   En considérant tous les chemins non-triviaux ayant u comme sommet de départ, on obtient

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

   D'une autre part, un chemin le plus long ayant u comme sommet de départ (qui est de longueur $t(u)$), disons C_O, est parmi les chemins considérés. En plus, si on omet u, on obtient un chemin le plus long ayant comme sommet de départ un des successeurs de u. On a donc bien l'égalité.

Rappel de cours : *À l'issue d'un parcours en profondeur d'un graphe orienté sans circuit il n'y a aucun arc retour. Ainsi, pour tout arc uv d'un graphe orienté sans circuit 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 $f(v) < f(u)$.

   Si u n'a pas d'arc sortant, on a $t(u) = 0$ (et il n'y a pas de calcul à faire). Si u a des voisins sortants, on peut utiliser la formule de la question précédente pour calculer $t(u)$ à partir des valeurs $t(v)$ des voisins sortant de u. Or, d'après cours, pour tout successeur v de u on a $f(v) < f(u)$.

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.

   On initialise `t[u]` à 0 pour tout sommet de G en ajoutant une ligne supplémentaire après ligne 2 de `PP(G)`.
   Puis, pendant la visite d'un sommet u, pour tout successeur v de u, quelle que soit sa couleur, après avoir exécuté la visite de v, on remplace la valeur actuelle de $t(u)$ par $1+t(v)$ si cette dernière dépasse la précédente. On ajoute alors une ligne suivante à l'intérieur de la boucle de ligne 3 de `Visiter_PP(u)`, mais pas conditionnée par la ligne 4, c-à-d entre 6 et 7 :

```
t[u] <- max(t[u],1+t[v])
```

Après avoir parcouru le graphe entier, on recherche et retourne le maximum du tableau `t[]`, ce qui se fait par une simple boucle à la fin de `PP(G)`.

Alternativement, on peut calculer le maximum au fur et à mesure, en le gardant dans une variable globale (à détailler).

6. Quelle est la complexité de votre algorithme ? Justifier.

   La complexité est inchangée par rapport à celle d'un parcours en profondeur, c-à-d $O(m+n)$. En effet, en plus d'un parcours qui lui-même est de complexité $O(m+n)$, l'initialisation du tableau $t[\,]$ se fait en $O(n)$ (il y a une seule instruction élémentaire par sommet), puis, des mises à jours se font en $O(m)$ (il y a, pour chaque arc, une addition, une comparaison et éventuellement une affectation à exécuter) ; enfin, le calcul du maximum d'un tableau de longueur n se fait en $O(n)$.

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