Licence Informatique — **4TIN501U – Algorithmique et structures de données 2**
Université de Bordeaux — **Aucun document autorisé**

Devoir Surveillé du 16 novembre 2016
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

Proposer un algorithme qui, pour tout graphe non orienté G, retourne `VRAI` si et seulement si G possède au moins un sommet de degré 1, dans les deux cas suivants :

1. G est représenté par ***matrice d'adjacence*** `A` ;
2. G est représenté par ***listes de successeurs*** `Adj`.

Dans les deux cas, donner la complexité asymptotique dans le pire des cas de votre solution en la justifiant.

Exercice 2

La ***distance*** $\text{dist}(u, v)$ entre deux sommets u et v dans un graphe non orienté G est la taille d'une plus courte chaîne reliant u et v. La ***taille d'une chaîne*** est le nombre d'arêtes composant cette chaîne. En particulier, $\text{dist}(u, v) = 0$ si $u = v$ et $\text{dist}(u, v) = \infty$ s'il n'y a aucune chaîne entre u et v.

L'excentricité d'un sommet v d'un graphe non orienté $G = (V, E)$, notée $\text{exc}(v)$, est le maximum des distances entre v et les autres sommets du graphe (∞ si G n'est pas connexe).

1. Modifier l'algorithme de **parcours en largeur** pour calculer l'excentricité d'un sommet et appliquer votre algorithme au sommet s_0 du graphe G_1 ci-dessous.
2. Quelle est la complexité de votre algorithme (justifier) ?

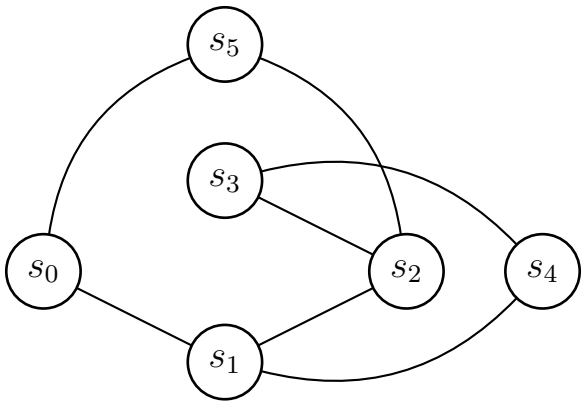

FIGURE 1 – Graphe G_1

Exercice 3

1. Appliquer l'algorithme de **parcours en profondeur** `PP(G)` au graphe G_2 de la FIG.2 à partir du sommet s_0 et classifier les arcs (*arcs de liaison*, ...) dans les différentes catégories possibles. Un arc sera noté par son sommet initial suivi de son sommet d'arrivé, le tout entre parenthèse. Par exemple l'arc entre s_8 et s_{10} sera noté (s_8, s_{10}). Vous pouvez également recopier le graphe et colorier les différentes catégories d'arcs (n'oubliez pas d'indiquer en légende la signification des couleurs).

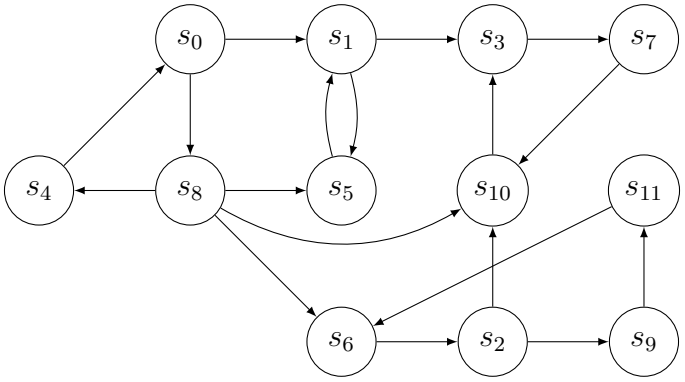

FIGURE 2 – Graphe orienté G_2.

2. En utilisant l'algorithme `CFC`, déterminer les composantes fortement connexes de G_2. En plus des composantes obtenues, vous donnerez l'ordre des sommets utilisé lors du second parcours servant à déterminer les différentes composantes fortement connexes.
3. Donner le graphe des composantes fortement connexes.
4. Combien d'arcs au minimum faut-il ajouter à G_2 pour ne plus avoir qu'une seule composante fortement connexe ? Donner un exemple.

Exercice 4

Dans cet exercice la fonction $\text{dist}(u, v)$ dénote la distance entre les sommets u et v dans un graphe non orienté G, tel que définie au début de l'exercice 2.

1. Soit G un graphe connexe, soient $u, v, w \in V(G)$. Montrer que
$$\text{dist}(u, v) \leq \text{dist}(u, w) + \text{dist}(w, v).$$
2. Soit G un graphe connexe, soient $u, v_1, v_2 \in V(G)$. Montrer que si v_1 et v_2 sont adjacents, alors
$$|\text{dist}(u, v_1) - \text{dist}(u, v_2)| \leq 1.$$
3. Soit G un graphe connexe. Montrer que G contient un cycle impair (un cycle ayant un nombre impair d'arêtes) si et seulement si quelque soit $u \in V(G)$ il existe deux sommets v_1 et v_2 adjacents tels que
$$\text{dist}(u, v_1) = \text{dist}(u, v_2).$$

Rappel :

```
0   PL(G,s)
1     pour chaque sommet u de X[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
```

```
0   PP(G)
1     pour chaque sommet u de X faire
2         couleur[u] <- BLANC
3         pere[u] <- nil
4     temps <- 0
5     pour chaque sommet u de X 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   CFC(G)
1     Exécuter PP(G) et trier les sommets selon un ordre décroissant de f[]
2     Calculer t(G), le graphe transposé de G, en renversant chaque arc de G
3     Exécuter PP(t(G)) en respectant l'ordre obtenu en 1
4     Retourner les arborescences obtenues comme composantes fortement connexes de G
```