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

Devoir Surveillé du 6 novembre 2024
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 $G = (V, E)$ un graphe orienté. On note par G^{-1} le graphe inversé de G, c'est-à-dire le graphe $G^{-1} = (V, E^{-1})$ ayant le même ensemble de sommets et dont l'ensemble des arcs E^{-1} est défini par :

$$(x, y) \in E^{-1} \iff (y, x) \in E$$

.

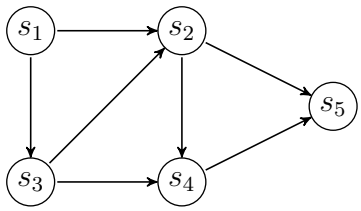

FIGURE 1 – Graphe orienté G_1.

1. Soit G_1 le graphe de la Figure 1. Dessiner le graphe G_1^{-1}.

   Le graphe inversé de G_1 est représenté par la Figure 2

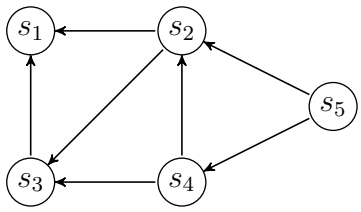

FIGURE 2 – Graphe orienté G_1^{-1}.

2. Donner la représentation de G_1^{-1} par ses listes de successeurs, puis par sa matrice d'adjacence.

   Pour la liste des successeurs, nous avons :
   s_1 :
   $s_2 : s_1, s_3$
   $s_3 : s_1$
   $s_4 : s_3, s_2$
   $s_5 : s_2, s_4$

Pour la matrice d'adjacence, nous avons :

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

3. Pour un graphe G orienté quelconque, représenté par ses listes de successeurs, proposer un algorithme pour calculer G^{-1}. Préciser la complexité de votre algorithme.

   L'algorithme est donné ci-dessous :

```
0   // Les listes d'adjacence de G' sont notées Adj'
1   Inverser(G)
2       V(G') <- V(G)
3       pour chaque sommet u de V(G) faire
4           Adj'(u) <- {}
5       pour chaque sommet u de V(G) faire
6           pour chaque sommet v de Adj(u) faire
7               Adj'(v) <- Adj'(v) U {u}
8       retourner G'
```

   Sa complexité est en $O(n+m)$.

4. Pour un graphe G orienté quelconque, représenté par sa matrice d'adjacence, proposer un algorithme pour calculer G^{-1}. Préciser la complexité de votre algorithme.

   L'algorithme est donné ci-dessous :

```
0   Inverser(A)
1       pour i de 1 à n-1
2           pour j de i+1 à n
3                   tmp <- A[i, j]
4                   A[i, j] <- A[j, i]
5                   A[j, i] <- tmp
```

   Sa complexité est en $O(n^2)$.

Exercice 2

Important : Pour cet exercice vous répondrez sur l'annexe en page 7 et vous détacherez cette annexe pour l'insérer dans votre copie.

Soit le graphe orienté G_2 représenté par la Figure 3 que vous trouverez en annexe page 7. Ce graphe G_2 a 8 sommets. Le nom des sommets n'est pas indiqué sur le graphe, mais ils seront nommés par les lettres `A`, `B`, `C`, `D`, `E`, `F`, `G` et `H`.

Le but de cet exercice est d'attribuer correctement un nom à chaque sommet, de telle sorte qu'à l'issu d'un parcours en profondeur (en suivant l'ordre lexicographique) il y ait des arcs de **liaisons**, au moins un arc **retour**, au moins un arc **avant** et au moins 2 arcs **transverses**.

N'hésitez pas à faire des essais sur votre brouillon. Lorsque vous aurez trouver les noms des sommets qui permettent de respecter les contraintes précédemment indiquées,

1. vous indiquerez sur le graphe G_2 (en écrivant sur l'annexe) le nom de chacun des sommets (à l'intérieur de chaque sommet).
2. à l'issu de votre parcours en profondeur (en respectant l'ordre lexicographique), vous colorierez sur le graphe G_2 en annexe
   - en rouge les arcs de liaisons,
   - en vert le ou les arcs de retours,
   - en bleu le ou les arc avants,
   - et vous laisserez en noir les arcs transverses (**il doit y en avoir au moins 2**).

   Si vous n'avez pas de couleurs, vous attacherez à chaque arc, à la place, une étiquette R, V et B pour rouge, vert et bleu.
3. vous indiquerez sur l'annexe de manière explicite au moins deux des arcs transverses obtenus à partir de votre parcours en profondeur.

Exercice 3

Dans cet exercice, le graphe (non-orienté) représente un réseau d'ordinateurs dans lequel on veut transmettre un message d'un sommet u à un sommet v d'une manière la plus efficace possible, en évitant de passer par un sommet x qui représente le lieu d'une attaque potentielle d'un ennemi cherchant à corrompre le message transmis.

1. Rappeler la définition de la distance $\text{dist}(u, v)$ entre deux sommets u et v dans un graphe (non-orienté) G.

   Soit G un graphe, soient u et v deux sommets de G. La *distance* entre u et v est la longueur minimum d'une chaîne reliant u et v. (Si une telle chaîne n'existe pas, la distance est infinie.)
2. Montrer que
   $$\text{dist}(u, x) + \text{dist}(x, v) \geq \text{dist}(u, v)$$
   pour tout triplet de sommets $u, v, x \in V(G)$ dans un graphe G.

   Par définition, il existe une chaîne C_1 reliant u et x de longueur $\text{dist}(u, x)$, et aussi une chaîne C_2 reliant x et v de longueur $\text{dist}(x, v)$. En les concaténant, on obtient une chaîne $C = C_1 + C_2$ reliant u et v, de longueur $\text{dist}(u, x) + \text{dist}(x, v)$. La distance entre u et v (étant définie comme la longueur minimum d'une chaîne les reliant) est donc inférieure ou égale à ceci.
3. Décider si la proposition suivante est vraie ou fausse. Si elle est vraie, donner une preuve ; si elle fausse, donner un contre-exemple.

   Si $\text{dist}(u, x) + \text{dist}(x, v) > \text{dist}(u, v)$, *alors aucune plus courte chaîne reliant* u *et* v *ne passe par* x.

   VRAI. Par l'absurde. Supposons qu'il existe un graphe G et qu'il existe trois sommets u, v, x de G tels que $\text{dist}(u, x) + \text{dist}(x, v) > \text{dist}(u, v)$ alors qu'il existe une plus courte chaîne C reliant u et v qui passe par x. On peut couper cette chaîne en deux sous-chaînes, C_1 reliant u et x et C_2 reliant x et v. Par définition de distance, on a
   $$|C_1| \geq \text{dist}(u, x) \quad \text{et} \quad |C_2| \geq \text{dist}(x, v).$$
   Or, $C_1 + C_2 = C$ est une plus courte chaîne reliant u et v, et donc
   $$\text{dist}(u, v) = |C| = |C_1| + |C_2| \geq \text{dist}(u, x) + \text{dist}(x, v) > \text{dist}(u, v)$$
   , ce qui est bien absurde.

4. Décider si la proposition suivante est vraie ou fausse. Si elle est vraie, donner une preuve ; si elle fausse, donner un contre-exemple.

   Si $\text{dist}(u,x) + \text{dist}(x,v) = \text{dist}(u,v)$, *alors il existe une plus courte chaîne reliant* u *et* v *passant par* x.

   VRAI. Soit C_1 une plus courte chaîne reliant u et x, de longueur $\text{dist}(u,x)$, soit C_2 une plus courte chaîne reliant x et v, de longueur $\text{dist}(x,v)$. Alors $C = C_1 + C_2$ (la chaîne obtenue en concaténant C_1 est C_2 et une chaîne reliant u et v passant par x de longueur $\text{dist}(u,x) + \text{dist}(x,v) = \text{dist}(u,v)$; sa longueur est égale à la distance entre u et v, il s'agit en effet d'une chaîne la plus courte.

5. Décider si la proposition suivante est vraie ou fausse. Si elle est vraie, donner une preuve ; si elle fausse, donner un contre-exemple.

   Si $\text{dist}(u,x) + \text{dist}(x,v) = \text{dist}(u,v)$, *alors toute plus courte chaîne reliant* u *et* v *passe par* x.

   FAUX. Il suffit de prendre comme exemple le graphe consistant d'un cycle $u-x-v-y-u$ de longueur quatre. Il existe deux plus courtes chaînes reliant u et v, mais une seule passe par x.

6. En utilisant l'algorithme `PL(G,s)` comme une boîte noire (donc sans la possibilité de modifier le code, tout en ayant accès aux résultats du calcul, à savoir, les tableaux `couleur[]`, `d[]`, et `pere[]`), proposer un algorithme efficace pour décider si le sommet x représente un danger potentiel, c'est à dire, décider s'il existe une plus courte chaîne reliant u et v qui passe par x. Préciser la complexité de votre algorithme.

   Nous allons profiter des observations démontrées dans les questions 3 et 4. En fait, il suffit de détecter si $\text{dist}(u,x) + \text{dist}(x,v) = \text{dist}(u,v)$. Pour cela, on fait deux appels à `PL(G,s)`, à partir de u et puis à partir de x :

```
ExistePlusCourteChainePassantParX(G,u,v,x)
  PL(G,u)
  dist <- d[v]
  dist1 <- d[x]
  PL(G,x)
  dist2 <- d[v]
  return (dist = dist1+dist2)
```

   On exécute `PL(G,s)` (qui est de complexité $O(m+n)$) deux fois, la complexité reste donc $O(m+n)$.

7. Proposer une modification de l'algorithme `PL(G,s)` (cette fois-ci vous avez le droit à modifier le code de l'algorithme) pour décider si le sommet x représente un obstacle à éviter, c'est à dire si toute plus courte chaîne reliant u et v passe par x. Expliciter les lignes de code à ajouter / modifier / supprimer. Préciser la complexité de votre algorithme.

   Il suffit de calculer $\text{dist}(u,v)$ d'abord, et la comparer avec la distance entre u et v dans le graphe $G' = G \setminus x$. Au lieu de calculer le nouveau graphe, on peut simuler le fait qu'il est interdit de passer par x en le coloriant en noir avant de lancer le parcours :

```
ToutePlusCourteChainePasseParX(G,u,v,x)
  PL(G,u)
  dist <- d[v]
  PL'(G,u,x)
  dist2 <- d[v]
  return (dist != dist2)
```

avec dans `PL'(G,s,x)` on ajoute une ligne (par exemple entre les lignes 5 et 6)

```
5bis couleur[x] <- NOIR
```

On exécute `PL(G,s)` (qui est de complexité $O(m+n)$) deux fois, la complexité reste donc $O(m+n)$.

Rappels :

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

```
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 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
```

ANNEXE - VOUS DEVEZ DÉTACHER CETTE ANNEXE POUR L'INSÉRER DANS VOTRE COPIE
INDIQUEZ IMPÉRATIVEMENT VOTRE NOM ET PRÉNOM SUR CET ANNEXE

Nom : Prénom :

Rappel des contraintes à respecter : Vous devez indiquer un nom à chaque sommet, de telle sorte qu'à l'issu d'un parcours en profondeur (en suivant l'ordre lexicographique) il y ait des arcs de **liaisons** (coloriés en rouge), au moins un arc **retour** (coloriés en vert), au moins un arc **avant** (coloriés en bleu) et au moins 2 arcs **transverses** (coloriés en noir).

Vos sommets porterons nécessairement les noms `A`, `B`, `C`, `D`, `E`, `F`, `G` et `H`.

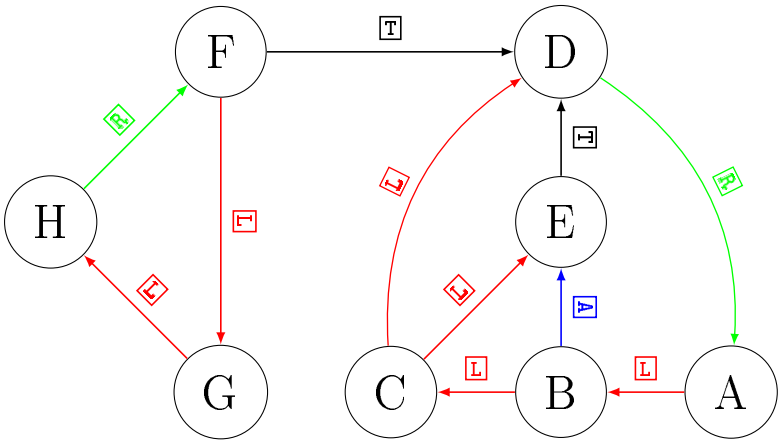

FIGURE 3 – Graphe orienté G_2.

Parmi les arcs transverses obtenus à l'issu de votre parcours en profondeur, on trouve au moins les deux arcs transverses suivants :

Arc transverse #1 : `(F,D)` Arc transverse #2 : `(E,D)`

(*Vous nommerez un arc à l'aide du nom des sommets à chacune de ses extrémités. Par exemple s'il y a un arc entre le sommet* `A` *et* `B`*, vous nommerez cet arc par* `(A,B)`.)

FIN.