université de BORDEAUX

Année 2021-2022 — DS novembre 2022

Épreuve de : Recherche Opérationnelle
Date : 12 novembre 2021 — **Durée :** 1h30
Épreuve de : O. Baudon — **Nb. pages : 5**

Une attention toute particulière sera portée à la présentation et à la rédaction de la copie.

Dans tous les exercices, on désignera par $V(G)$ et $E(G)$ respectivement l'ensemble des sommets et l'ensemble des arêtes d'un graphe G. Les variables n et m désigneront respectivement le nombre de sommets et d'arêtes.

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 nom.

Les algorithmes utilisés dans les exercices sont disponibles à la fin du sujet.

1 Structures de données et complexité

Dans un graphe simple orienté G, on appelle ***puits universel*** tout sommet v tel que :
— pour tout sommet $u \neq v$, il existe un arc uv ;
— v n'a pas de successeur.

1.1) Dessiner un graphe à 4 sommets possédant un puits universel. Quel est le nombre maximum de puits universels que peut contenir un graphe ? (Justifier votre réponse).

1.2) Pour chacune des deux représentations :

1. listes de successeurs,
2. matrice d'adjacence,

écrire un algorithme pour déterminer si le graphe contient un puits universel et en étudier la complexité. *Attention de présenter les algorithmes de façon lisible !*

Solution Q1
Le graphe à 4 sommets avec un puits universel est dessiné ci-dessous dans la Figure 1. Le puits est le sommet étiqueté *puits*.
Un graphe ne peut contenir qu'au plus un puits universel. En effet, tous les autres sommets doivent avoir ce sommet comme successeur et un puits universel n'a pas de successeur.

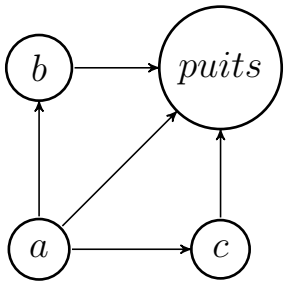

FIGURE 1 – Graphe avec un puits

Solution Q2

La fonction `Puits` renvoie `Faux` si le graphe ne possède pas de puits universel, ou le sommet puits sinon.

Pour les deux représentations, l'algorithme consiste à sélectionner un sommet candidat n'ayant aucun successeur, puis à vérifier que tous les autres sommets sont bien des prédécesseurs de ce candidat.

Version avec liste de successeurs (G est un graphe représenté par ses listes d'adjacence) :

```
0   Puits(G) :
1   candidat <- Nil
2   pour tout sommet v de V(G) faire
3           si Adj(v) = ∅ alors
4                   candidat <- v
5                   break
6   si candidat == Nil alors
7           retourner Faux
8   pour tout sommet v de V(G) faire
9           si v != candidat alors
10                  pred <- Faux
11                  pour tout sommet u de Adj(v) faire
12                          si u = candidat alors
13                                  pred <- Vrai
14                                  break
15                  si pred = Faux alors
16                          retourner Faux
17  retourner candidat
```

Version avec matrice d'adjacence (A est une matrice d'adjacence de taille $n \times n$) :

```
0   Puits(A) :
1   pour i de 1 à n faire
2           candidat <- i
3           pour j de 1 à n faire
4                   si A[i, j] != 0
5                           candidat <- -1
6                           break
7           si candidat != -1 alors
8                   break
9   si candidat = -1 alors
10          retourner Faux
11  pour i de 1 à n faire
12          si i != candidat et A[i, candidat] = 0 alors
13                  retourner Faux
14  retourner candidat
```

La complexité de la version avec les listes d'adjacences est en $O(n+m)$. En effet la boucle 2-5 est en $O(n)$, les boucles 8-16 et 11-16 combinées sont en $O(n+m)$.
La complexité de la version avec la matrice d'adjacence est en $O(n^2)$. En effet la boucle 1-8 est en $O(n^2)$, la boucle 11-13 est en $O(n)$.

2 Plus courts chemins

Un étudiant du Master Miage de l'Université de Bordeaux, désirant faire un séjour linguistique, décide de se rendre en Suède. Après avoir fait le tour de quelques compagnies, il a recensé plusieurs connexions aériennes lui permettant d'aller de Bordeaux à Stockholm. Il les a représentées à l'aide du graphe suivant (Figure 2) :

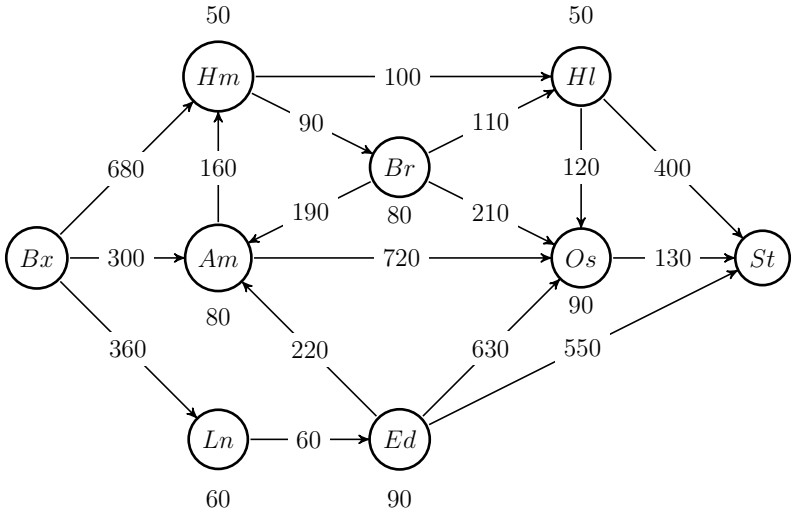

FIGURE 2 – Voyage

où Bx = Bordeaux, Hm = Hambourg, Am = Amsterdam, Ln = Londres, Ed = Edimbourg, Br = Berlin, Hl = Helsinki, Os = Oslo, St = Stockholm.

Cependant, les horaires des vols sont tels que l'étudiant est obligé de passer la nuit dans chacune des villes où il fait escale.

Les valeurs sur les arcs correspondent aux prix en euros pour les vols, et les valeurs à coté des sommets représentent le prix en euros à payer pour passer la nuit dans un hôtel de la ville correspondante.

2.1) On désire trouver une solution optimale pour ce problème. Quelles modifications faut-il appliquer au graphe pour pouvoir utiliser l'algorithme de Dijkstra ?

2.2) Calculer la solution optimale. On donnera en particulier la suite des valeurs prises par la variable `pivot` et pour chacune de ces valeurs, les modifications des valeurs du tableau `d` associées.

Solution Q1

Il existe deux possibilités :

— on peut remplacer chaque sommet s ayant un coût d'hôtel par deux sommets s_1 et s_2 reliés par un arc avec comme poids le prix de l'hôtel. Les arcs entrants sur le sommet original s seront entrants sur le sommet s_1 et les arcs sortants du sommet s seront sortants du sommet s_2.
Le graphe obtenu est décrit dans la Figure 3 ci-dessous.

— Une autre solution est, pour chaque sommet s représentant une ville avec un coût d'hôtel, de rajouter le coût de l'hôtel au poids de chaque arc sortant de s.
Le graphe obtenu est décrit dans la Figure 4 ci-dessous.

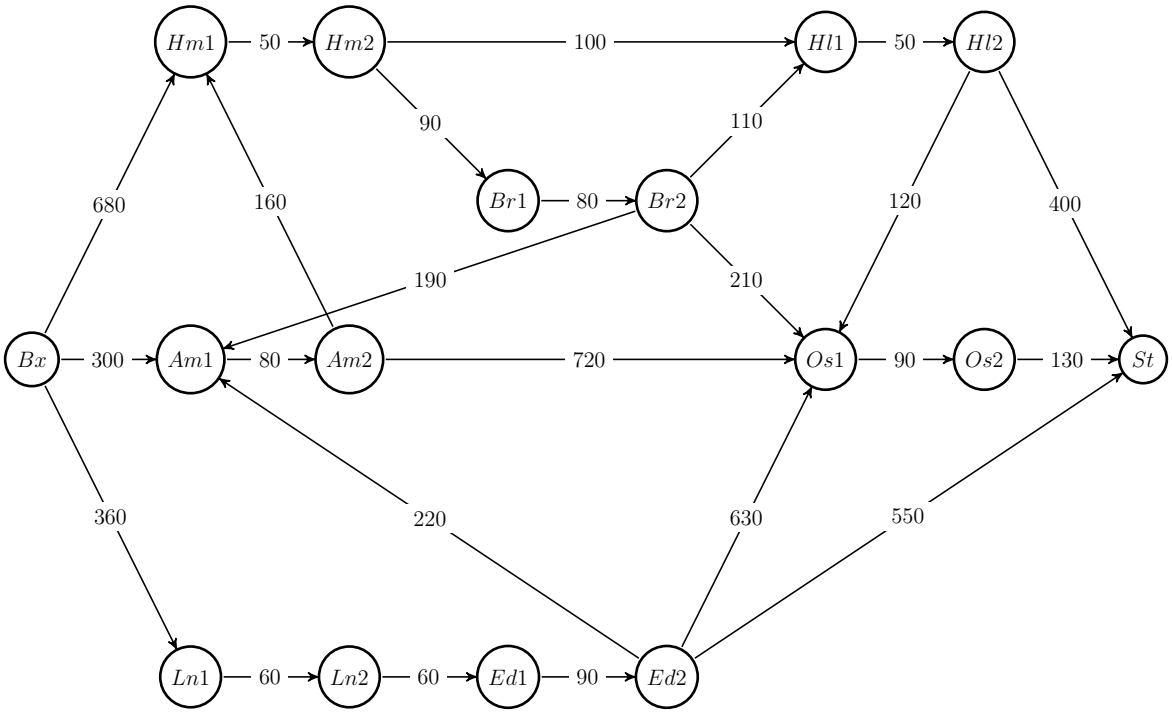

FIGURE 3 – Voyage modifié - version 1

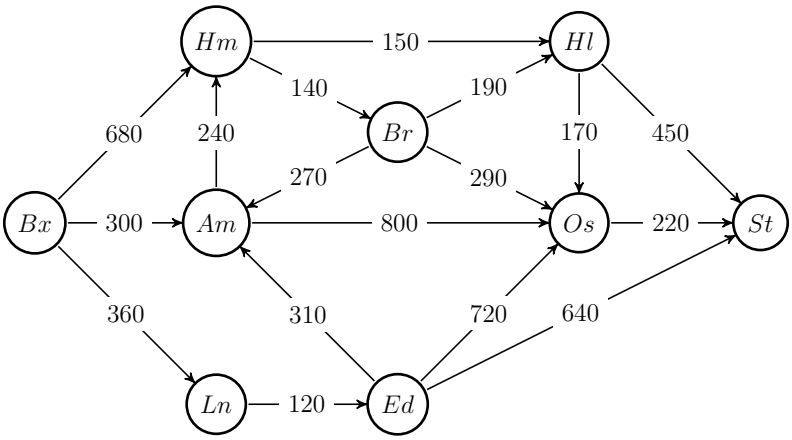

FIGURE 4 – Voyage modifié - version 2

Solution Q2

On va utiliser Dijkstra sur une des deux modifications proposées. Le tableau ci-dessous utilise la 2ème version.

Pivot	*Bx*	*Hm*	*Am*	*Ln*	*Br*	*Ed*	*Hl*	*Os*	*St*
	0	∞	∞	∞	∞	∞	∞	∞	∞
Bx/0	*X*	680	300	360					
Am/300	*X*	540	*X*					1100	
Ln/360	*X*		*X*	*X*		480			
Ed/480	*X*		*X*	*X*		*X*			1120
Hm/540	*X*	*X*	*X*	*X*	680	*X*	690		
Br/680	*X*	*X*	*X*	*X*	*X*	*X*		970	
Hl/690	*X*	*X*	*X*	*X*	*X*	*X*	*X*	860	
Os/860	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	1080
St/1080	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*

Ce qui nous donne l'arborescence des plus courts chemins de la Figure 5.

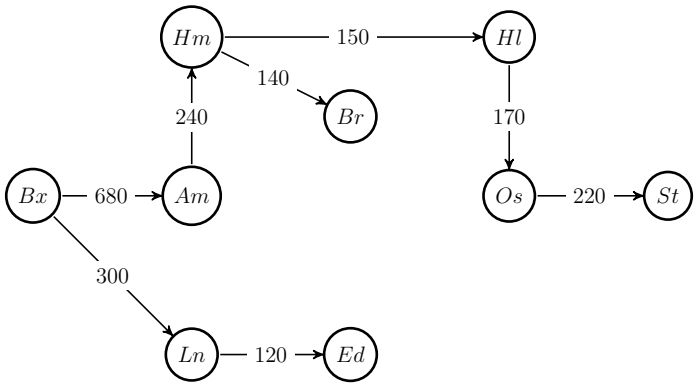

FIGURE 5 – Voyages les moins chers à partir de Bordeaux

Le chemin le moins cher pour relier Bordeaux à Stockholm est donc : Bordeaux, Amsterdam, Hambourg, Helsinki, Oslo, Stockholm pour un coût total de 1080 euros.

3 Application du parcours en profondeur

L'objectif de cet exercice est de fournir un algorithme de recherche d'un chemin hamiltonien dans un tournoi.

Un *tournoi* est un graphe simple complet orienté. Autrement dit, si T est un tournoi et u et v deux sommets de T, alors $(u, v) \in E(T) \Leftrightarrow (v, u) \notin E(T)$.

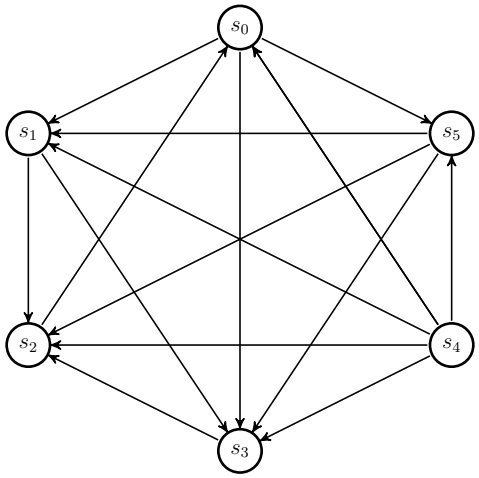

FIGURE 6 – Exemple de tournoi à 6 sommets

Un *chemin hamiltonien* est un chemin qui passe par tous les sommets du graphe une

et une seule fois.
Théorème : tout tournoi possède un chemin hamitonien. Par exemple, dans le tournoi de la Figure 6 : $s_4, s_3, s_2, s_0, s_5, s_1$ est un chemin hamiltonien.

3.1) Appliquer l'algorithme de parcours en profondeur `PP` au tournoi T de la Figure 6. On conviendra que dans les listes d'adjacence, les sommets sont rangés dans l'ordre des indices et également que la boucle principale de `PP` (lignes 5 à 7) considère les sommets dans l'ordre des indices. Pour chaque sommet, donner les valeurs `d`, `f`, `pere` retournées par l'algorithme en respectant l'ordre de visite des sommets. Ordonner les sommets de T dans l'ordre inverse de leur fin de visite. Que remarquez-vous ?

On rappelle que les arcs obtenus lors d'un parcours en profondeur peuvent être de 4 types :

- arcs de liaison : entre un sommet et ses fils dans l'arborescence ;
- arcs de retour (ou arrière) : entre un sommet et l'un de ses ancêtres ($\neq$ de son père) dans l'arborescence ;
- arcs avant : entre un sommet et l'un de ses descendants ($\neq$ de ses fils) dans l'arborescence ;
- arcs transverses : entre deux sommets sans relation dans l'arborescence.

3.2) Lorsque l'on range les sommets dans l'ordre inverse de leur fin de visite, quels sont les types d'arcs reliant deux sommets consécutifs ? Montrer que si les sommets sont rangés dans l'ordre inverse de leur fin de visite, alors si le sommet v suit immédiatement le sommet u dans cet ordre, l'arc entre ces deux sommets est orienté de u vers v.

3.3) Modifier l'algorithme de parcours en profondeur pour afficher les sommets dans un ordre induisant un chemin hamiltonien. Quelle est la complexité de l'algorithme obtenu ?

3.4) Démontrer le théorème donné en début d'exercice : tout tournoi possède un chemin hamiltonien. *Indication : montrer qu'un chemin de longueur maximale contient tous les sommets.*

Solution Q1

Les arborescences obtenues lors du parcours en profondeur du tournoi T sont donnée par la Figure 7. Les valeurs de `d`, `f` et `pere` sont données par le tableau ci-dessous :

v	s_0	s_1	s_2	s_3	s_4	s_5
`d(v)`	1	2	3	5	11	8
`f(v)`	10	7	4	6	12	9
`pere(v)`	`NIL`	s_0	s_1	s_1	`NIL`	s_0

L'ordre inverse de fin de visite est : $s_4, s_0, s_5, s_1, s_3, s_2$. Cet ordre correspond à un chemin hamiltonien dans le tournoi de la Figure 6.

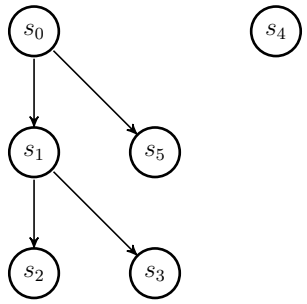

FIGURE 7 – Forêt couvrante de T obtenue par PP

Solution Q2
Si v est le successeur de u dans l'ordre inverse de fin de visite, l'arc entre u et v sera soit un arc de liaison, soit un arc transverse.
Ça ne peut pas être un arc retour car dans ce cas $f(u) < f(v)$. Donc u sera classé après v.
Ça ne peut pas être un arc avant car dans ce cas, il existe au moins un sommet w (par exemple le père de v dans l'arbre de liaison) tel que $f(u) > f(w) > f(v)$ et donc w serait situé entre u et v dans l'ordre inverse de fin de visite.

Soit u et v deux sommets successifs dans l'ordre inverse de fin de visite. On a $f(u) > f(v)$.
Si $d(u) < d(v)$, on est dans le cas d'un arc de liaison avec $d(u) < d(v) < f(v) < f(u)$ et dans ce cas, l'arc entre u et v est un arc de liaison de u vers v.
Si $d(u) > d(v)$, on est dans le cas d'un arc transverse avec $d(v) < f(v) < d(u) < f(u)$ et dans ce cas, l'arc entre u et v est un arc transverse de u vers v.
Donc dans les deux cas (arc de liaison ou arc transverse), l'arc entre u et v sera orienté de u vers v.

Solution Q3
Il suffit d'afficher les sommets dans l'ordre inverse de fin de visite. Pour cela on utilise une pile où le premier élément empilé sera le premier sommet pour lequel l'appel de `Visiter_PP` se termine (il aura donc la plus petite valeur de `f[]`) et le dernier sommet empilé sera celui avec la plus grande valeur de `f[]`. À la fin on affiche les éléments de la pile dans l'ordre de dépilement. Aussi bien les opérations d'empilement que de dépilement s'effectuent en temps constant.

On rajoute donc :
- une ligne au début de `PP(G)` :
```
0 P <- Pile_Vide()
```
- une ligne à la fin de `Visiter_PP(u)` :
```
9 Empiler(P, u)
```
- une ligne à la fin de `PP(G)` :
```
8 Afficher(P)
```

Les instructions rajoutées sont en temps constant pour les deux premières et en $O(n)$ pour l'affichage de la pile. La complexité sera donc la même que pour le parcours en profondeur, soit $O(n + m)$.

Solution Q4

Soit T un tournoi et $P = v_1, v_2, \ldots, v_k$ un chemin de longueur maximum dans T.

Si P n'est pas un chemin hamiltonien, alors il existe un sommet u de $V(T)$ n'appartenant pas à P.

Considérons les arcs entre les sommets de P et le sommet u.

L'arc entre u et v_1 est forcément orienté de v_1 vers u, sinon $u, v_1, v_2, \ldots, v_k$ est un chemin plus long que P.

Si l'arc entre u et un sommet v_i de P est orienté de v_i vers u, alors il en sera de même pour l'arc entre le sommet v_{i+1} et u. Sinon $v_1, \ldots, v_i, u, v_{i+1}, \ldots, v_k$ est un chemin plus long que P.

Comme l'arc entre v_1 et u est orienté de v_1 vers u, il en sera donc de même pour tous les arcs entre les sommets de P et u. Donc l'arc entre v_k et u est orienté de v_k vers u et donc $v_1, \ldots, v_k, u$ est un chemin plus long que $P \rightarrow$ *Contradiction.*

4 Rappels

```
0   Dijkstra(G,w,s)
1     pour chaque sommet v de V(G) faire
2         d[v] <- infini
3         pere[v] <- nil
4         couleur[v] <- BLANC
5     d[s] <- 0
6     F <- FILE_PRIORITE(V(G), d)
7     tant que F non vide faire
8         pivot <- EXTRAIRE_MIN(F)
9         couleur[pivot] <- GRIS
10        pour tout arc e = (pivot, v) arc sortant de pivot faire
11            si couleur(v) = BLANC et d[v] > d[pivot] + w(e)
12                alors d[v]  <- d[pivot] + w(e)
13                      pere[v]  <- pivot
14        couleur[pivot] <- NOIR
```

```
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
```