Algorithmique des Graphes

1. Définitions

- **Un graphe orienté** est caractérisé par un ensemble S de sommets un ensemble A d'arcs ayant une origine et une terminaison qui sont des sommets de S

  Graphe orienté G = < S, A > S est un ensemble fini de sommets
  A est un ensemble fini de couples : Arcs
- **Un graphe non orienté** est caractérisé par un ensemble S de sommets, un ensemble A d'arêtes ayant deux extrémités qui sont des sommets de S

  Graphe non orienté G = < S, A > S est un ensemble fini de sommets
  A est un ensemble fini de couples : Arêtes
- **Un graphe peut être valué :** ses arcs ou ses arêtes portent une valeur (poids)

  Exemple d'application
  - Villes avec des routes, routeurs dans un réseau avec des liaisons,
  - planification de travaux, circuits électriques
- **Un Chemin** dans un graphe orienté, est une suite d'arcs, l'origine d'un arc étant la terminaison de l'arc précédent
- **Une chaîne** dans un graphe non orienté, est une suite d'arêtes, l'extrémité d'une arête étant l'extrémité de l'arête précédente
- **La longueur d'un chemin** est son nombre d'arcs ou la somme des valuations de ses arcs si le graphe est valué
- **Un Circuit** est un chemin qui a une origine et une terminaison confondues
- **Un graphe non orienté est Connexe** s'il existe une chaîne entre deux sommets quelconques
- **La composante connexe** d'un sommet S dans un graphe non connexe, est l'ensemble des sommets qui sont reliés à s par au moins une chaîne
- **Un graphe orienté est non connexe** s'il existe deux sommets non liés par aucun chemin
- **Un graphe orienté est faiblement connexe** s'il existe toujours un chemin entre deux sommets mais sans tenir compte de l'orientation des arcs
- **Un graphe orienté est fortement connexe** si deux quelconques sommets sont toujours extrémités initiale et terminale de chemins les unissant
- **Le demi-degré extérieur d'un sommet S** (successeurs de S) (resp. **intérieur -** prédécesseurs de S) d'un sommet S dans un graphe orienté, est le nombre d'arcs ayant leur extrémité initiale (resp. terminale) en S. On note les successeurs de S par $d^+(S)$; les prédecesseurs de S par $d^-(S)$
- **Le degré d'un sommet** S est le nombre d'arcs (resp. arêtes) dont il est une extrémité.
  $d(S) = d^+(S) + d^-(S)$

2. Représentation des graphes

- Par **Matrice d'Adjacence:** Matrice de booléens
- Par **liste d'adjacence :** Tableau contenant pour chaque sommet, la liste de ses successeurs
- Si le graphe est valué (pondéré), les poids sont représentés.

3. Le parcours de graphes

Le parcours d'un graphe à partir d'un sommet S permet de visiter tous les sommets qui peuvent être atteints à partir de S

3.1 Parcours en largeur

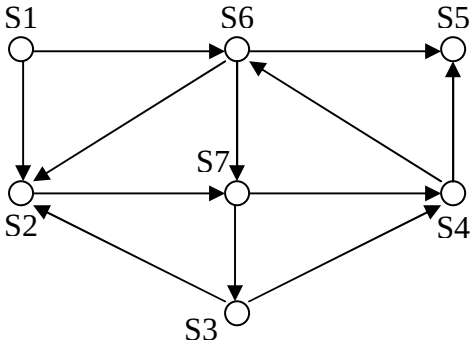
- On part d'un sommet d'origine s
- On visite tous les successeurs de s avant de visiter les autres descendants de s de la façon suivante :
  - On visite d'abord tous les sommets qui sont à la distance 1 de s
  - Puis ceux qui sont à la distance 2 de s
  - Puis ceux qui sont à la distance 3 de s
  - …… etc.….

Algorithme de principe

Procédure Largeur (données : le graphe ; nombre de sommets ; sommet d'origine A)
Début
{On utilise une file d'attente}
Initialiser la file ;
Traiter A ; ***Marquer*** A ; *Enfiler* (A, file) ;
Tant que la file est non vide **faire début**
    X := *Défiler* (file) ;
    Pour tout sommet S successeur à X et non marqué **faire début**
        Traiter(S); ***Marquer*** (S); ***Enfiler(***S, file***)*** ;
    FIN
FIN
FIN.

Exp. : Parcours en profondeur à partir du sommet S1 des graphes G1, G2, G3

G1

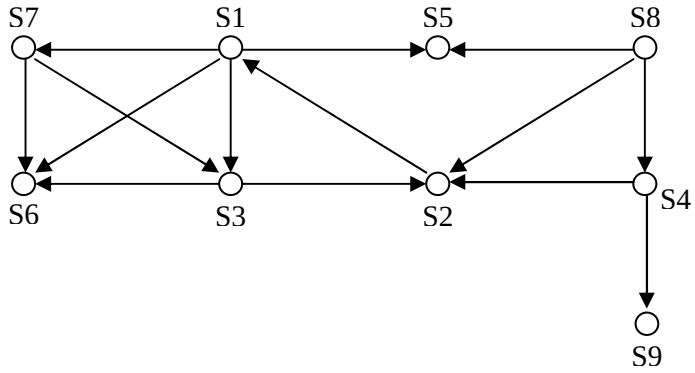


Listes des successeurs :
S1 : S2 , S6
S2 : S7
S3 : S2, S4
S4 : S5, S6
S5 : Ø
S6 : S2, S5, S7
S7 : S3, S4

Etats successifs de la file pour G1

(tête) **FILE** (Queue)	S1	Φ	S2	S2 S6	S6	S6 S7	Φ	S7	S7 S5	Φ	S5	S5 S3	S5 S3 S4	S3 S4	S4	Φ
Ordre de Parcours	S1		S2	S6		S7			S5			S3	S4			

G2

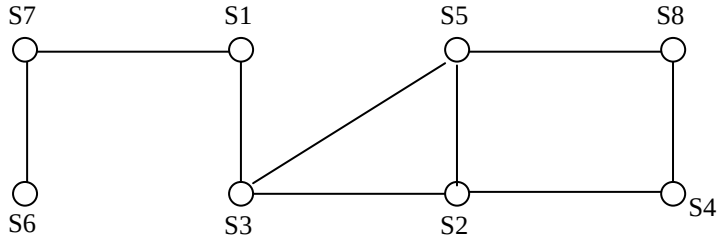


Listes des successeurs :
S1 : S3, S5, S6, S7
S2 : S1
S3 : S2, S6
S4 : S2, S9
S5 : Ø
S6 : Ø
S7 : S3, S6
S8 : S2, S4, S5
S9 : Ø

Etats successifs de la file pour G2

File	
Ordre de Parcours	

G3



Listes des successeurs :
S1 : S3, S7
S2 : S3, S4, S5
S3 : S1, S2, S5
S4 : S2, S8
S5 : S2, S3, S8
S6 : S7
S7 : S1, S6

Etats successifs de la file pour G3

File	S1	S3	S3 S7	S7 S2	S7 S2 S5	S2 S5 S6	S5 S6 S4	S6 S4 S8	S4 S8	S8	Φ
Marqué	S1	S3	S7	S2	S5	S6	S4	S8			

3.2 Parcours *en profondeur*

- On part d'un sommet d'origine S, on le marque
- On suit un chemin à partir de S, aussi loin que possible, en marquant les sommets au fur et à mesure qu'on les rencontre
- Lorsqu'on est en fin de chemin, on revient au dernier choix fait, et on prend une autre direction

Algorithme de principe

Procédure Profondeur (données : le graphe ; nombre de sommets ; sommet d'origine A)
Début
{On utilise une pile}
Initialiser la pile ;
Traiter(A) ; ***Marquer(*** A) ; ***Empiler(*** A, Pile) ;
Tant que la pile est non vide **faire début**
    Tant qu'il y a un sommet S successeur à S*ommet* (Pile) et non marqué **faire début**
        Traiter (S) ; ***Marquer(*** S); ***Empiler*** (S) ;
    FIN
Dépiler(Pile);
FIN
FIN.

Exp. : Parcourir les graphe G1, G2, G3 à partir des sommets S1

Etats successifs de la pile pour G1

PILE	S1	S3 S1	S2 S3 S1	S3 S1	S6 S3 S1	S3 S1	S1	S5 S1	S1	S7 S1	S1	Φ	S4	S9 S4	S4	Φ	S8	Φ
Ordre de Parcours	S1	S3	S2		S6			S5		S7			S4	S9			S8	

Etats successifs de la pile pour G2

PILE	
Ordre de Parcours	

Etats successifs de la pile pour G3

PILE	
Ordre de Parcours	

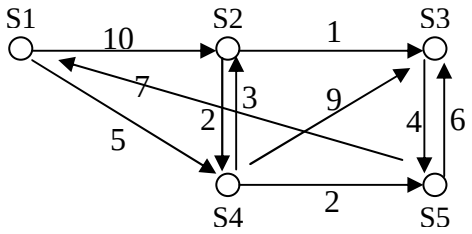
4. Calcul du plus court chemin (algorithme de Dijkstra)

Algorithme de Dijkstra (algorithme de principe)

{***Distance*** (S, T) représente la distance du sommet **S** au sommet **T**}
{ *Poids* (T, U) représente le poids de l'arc (T, U) }

Pour tout sommet **T** **Faire**
    Distance(*S, T*) := une valeur strictement plus grande que tous les poids du graphe;
Distance (S, S) := 0
Tant qu'il reste des sommets non marqués **faire début**
    Choisir un sommet **T** non marqué tel que ***Distance***(S, T) soit minimale ;
    Marquer **T ;**
        Pour tout successeur **U** du sommet **T faire** {choisir la distance min}
            Si ***Distance*** (S, U) > ***Distance*** (S, T) + ***Poids*** (T, U)
            Alors ***Distance*** (S, U) := ***Distance*** (S, T) + ***Poids*** (T, U)
Fin

Exp1. : Trouver les plus courts chemins entre le sommet S1 et tous les autres sommets du graphe en déroulant l'algorithme de Dijkstra sur le graphe suivant :



On utilise les structures de données suivantes :

- **TabCout** : Tableau des distances entre S1 et les sommets Si
- **Smin** : Sommet qui donne la distance min
- **succSmin** : Successeurs du sommet qui donne la distance min
- **TabChemins** : Tableau pour retrouver la trace des plus courts chemins

TabCouts					**Smin**	**succSmin**	**TabChemins**				
1	2	3	4	5			1	2	3	4	5
0	+∞	+∞	+∞	+∞	S1	S2, S4	-	-	-	-	-

S2 : min (+∞, S1S2+S2S2) =10
S4 : min (+∞, S1S4+S4S4)= 5

1	2	3	4	5			1	2	3	4	5
0	10	+∞	5	+∞	S4	S2, S3, S5	S1	S1	S1	S1	S1

S2 : min (10, S1S4 + S4S2)= 8
S3 : min (+∞, S1S4 + S4S3)=14
S5 : min (+∞ , S1S4 + S4S5)= 7

1	2	3	4	5			1	2	3	4	5
0	8	14	5	7	S5	S1, S3	S1	S4	S4	S1	S4

S3 : min (14, S1S5+S5S3)= 7 + 6= 13

1	2	3	4	5			1	2	3	4	5
0	8	13	5	7	S2	S3, S4	S1	S4	S5	S1	S4

S3 : min (13, S1S2+S2S3)= 8+1=9

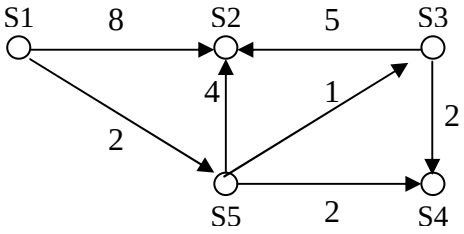
1	2	3	4	5			1	2	3	4	5
0	8	9	5	7	S3	S5	S1	S4	S2	S1	S4

Pour retrouver le chemin le plus court entre S1 et un sommet Si, on utilise le tableau TabChemins :
Exp. : chemin entre S1 et S3 :

TabChemins [3]=S2
TabChemins [2]=S4
TabChemins [4]= S1
Donc le chemin le plus court est S1 S4 S2 S3 avec un coût de tabCout[3]= 9

Résultats :
S2 : S1-S4-S2 = 8
S3 : S1-S4-S2-S3 = 9
S4 : S1-S4 = 5
S5 : S1-S4-S5 = 7

Exp 2. :



	TabCouts					**Smin**	**succSmin**	**TabChemins**				
	1	2	3	4	5			1	2	3	4	5
	0	+∞	+∞	+∞	+∞	S1	S2, S5	-	-	-	-	-

S2 : min (+∞, S1S2) =8
S5 : min (+∞, S1S5)= 2

	1	2	3	4	5							
	0	8	+∞	+∞	2	S5	S2, S3, S4	S1	S1	S1	S1	S1

S2 : min (+∞, S1S5+S5S2) =6
S3 : min (+∞, S1S5+S5S3)= 3
S4 : min (+∞, S1S5+S5S4)= 4

	1	2	3	4	5							
	0	6	3	4	2	S3	S2, S4	S1	S5	S5	S5	S1

S2 : min (6, S1S3+S3S2) = min (6, 3+5)=6
S4 : min (4, S1S3+S5S2)= 4

	1	2	3	4	5							
	0	6	3	4	2	S4	Φ	S1	S5	S5	S5	S1

	1	2	3	4	5							
	0	6	3	4	2	S2	Φ	S1	S5	S5	S5	S1

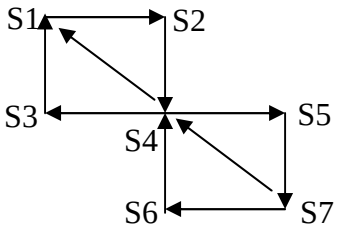
Résultats :
S2 : S1 S5 S2 = 6
S3 : S1 S5 S3 = 3
S4 : S1 S5 S4 = 4
S5 : S1 S5 = 2

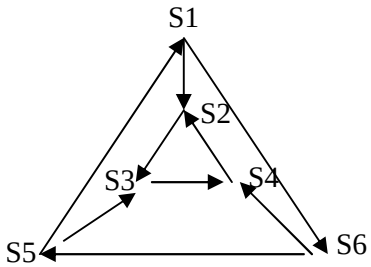
5. Connexité

5.1 Définitions

- **Graphe orienté fortement connexe :** pour toute paire ordonnée de sommets distincts (*u, v*), il existe un chemin de u vers un chemin de *v* vers *u*.
- **Composante fortement connexe d'un graphe orienté :** un sous graphe fortement connexe qui n'est pas strictement contenu dans un autre sous graphe fortement connexe. Tous les sommets d'une même composante sont mutuellement accessibles, et deux sommets de deux composantes connexes distnctes ne peuvent être joints par aucune chaîne.

Exp.





G1 fortement connexe
G1 contient une seule composante connexe

G2 n'est pas fortement connexe
2 composantes fortement connexes : S1-S5-S6 ; S2-S3-S4

- **Graphe non orienté connexe :** pour toute paire ordonnée de sommets distincts (*u, v*), il existe une chaîne reliant *u* et *v*.
- **Composante connexe d'un graphe non orienté :** un sous graphe connexe qui n'est pas strictement contenu dans un autre sous graphe connexe.

- **Application de la notion de connexité :** La vérification de la connexité des graphes est un problème pratique majeur. La détermination des composantes connexes peut permettre de :
- vérifier si deux points dans un réseau électrique ou téléphoniques sont reliés entre eux
- vérifier l'accessibilité entre des régions dans un réseau routier lors d'une catastrophe naturelle,….

5.2 Algorithme de recherche des composantes fortement connexes

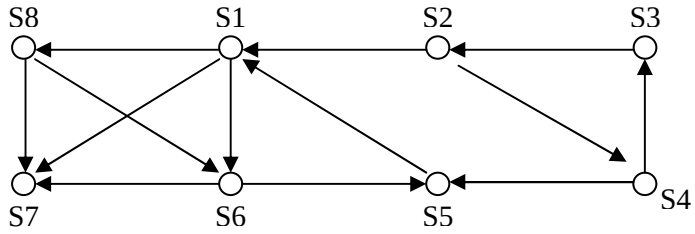
On utilise le parcours en profondeur du graphe G et de son inverse G^{-1} (obtenu à partir de G en inversant le sens de tous les arcs) selon le principe suivant :

1. Appeler le parcours en profondeur sur G, et numéroter les sommets par ordre de dépilement
2. Construire G^{-1}
3. Appeler le parcours en profondeur sur G^{-1} en commençant le parcours par le sommet ayant l'ordre de dépilement le plus élevé

Remarque : si tous les sommets ne sont pas marqués à la fin de cet appel, on recommence l'exécution de la procédure parcours avec le dernier sommet non maqué (ordre décroissant de dépilement)

4. Les arborescences obtenues dabs l'étape 3 sont les composantes fortement connexes

Exp.



Listes des successeurs :
S1 : S6, S8
S2 : S1, S4
S3 : S2
S4 : S3, S5
S5 : S1
S6 : S5, S7
S7 : Ø
S8 : S6, S7

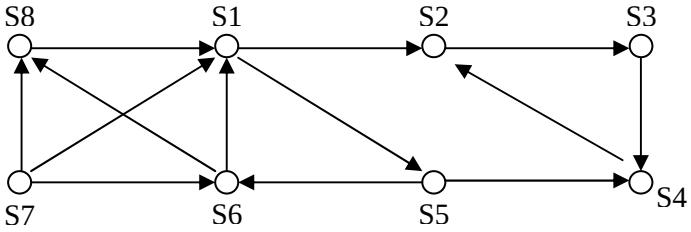
1) Parcours en profondeur sur G, et numéroter les sommets par ordre de dépilement

PILE	S1	S6 S1	S5 S6 S1	S6 S1	S7 S6 S1	S6 S1	S1	S8 S1	S1	∅	S2	S4 S2	S3 S4 S2	S4 S2	S2	∅		
Ordre de Parcours	S1	S6	S5	**S5**		**S7**	**S6**	S8	**S8**	**S1**	S2	S4	S3	**S3**	**S4**	**S2**		

ordreDepil : Tableau qui contient les ordres de dépilement :

1	2	3	4	5	6	7	8
S5	S7	S6	S8	S1	S3	S4	S2

2) Construire G^{-1}



Listes des successeurs :
S1 : S2 S5
S2 : S3
S3 : S4
S4 : S2
S5 : S4 S6
S6 : S1 S8
S7 : S1 S6 S8
S8 : S1

3) Parcours en profondeur sur G^{-1} en commençant le parcours par le sommet ayant l'ordre de dépilement le plus élevé

On effectue le parcours en profondeur sur G^{-1} en commençant par le sommet S2 :

PILE	S2	S3 S2	S4 S3 S2	S3 S2	S2		S1	S5 S1	S6 S5 S1	S8 S6 S5 S1	S6 S5 S1	S5 S1	S1		S7			
Ordre de Parcours	S2	S3	S4	**S4**	**S3**	**S2**	S1	S5	S6	S8	**S8**	**S6**	**S5**	**S1**	S7	**S7**		

indiceCompConnex : Tableau qui contient les indices des composantes connexes
indiceCompConnex[i] contient le numéro de la composante connexe à la quelle appartient le sommet Si

1	2	3	4	5	6	7	8
2	1	1	1	2	2	3	2