

Dernière mise à jour	Informatique	Denis DEFAUCHY – Site web
22/03/2023	11 – Bases des graphes	Cours

Informatique

11

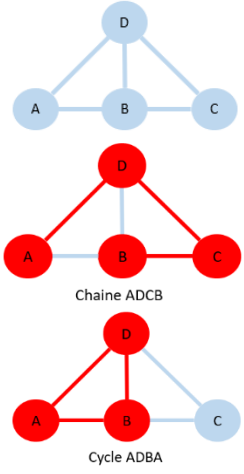
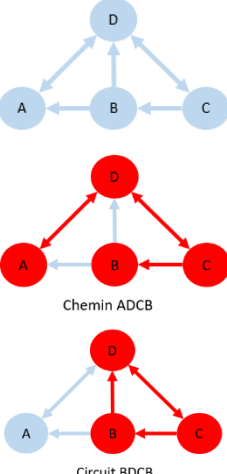
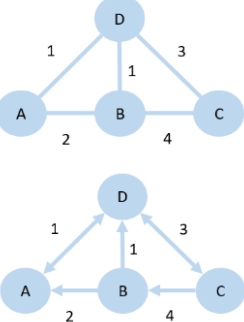
Bases des graphes

Résumé

Dernière mise à jour	Informatique	Denis DEFAUCHY – Site web
22/03/2023	11 – Bases des graphes	Cours

Bases des Graphes

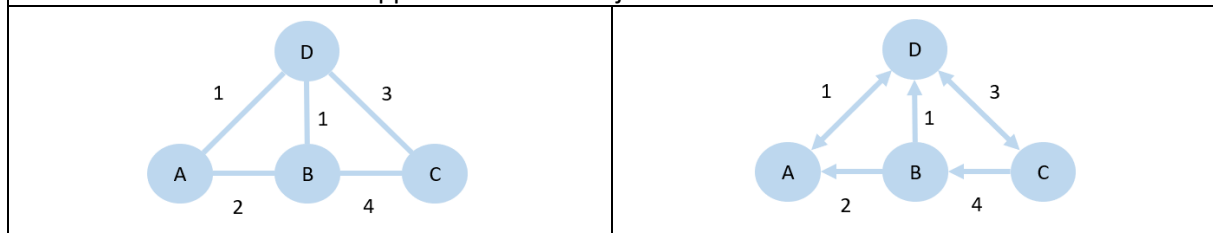
Définitions	
Graphe	Ensemble d'éléments (sommets) et de liens (arêtes, arcs) entre eux
D'ordre n	Composé de n sommets
Non orienté	Composé d'arêtes (parcours dans les 2 sens)
Orienté	Composé d'arcs (flèches) orientés
Sommets adjacents	Reliés par une arête ou un arc
Sommet isolé	Sommet relié à aucun autre
Degré d'un sommet $d(s)$	Nombre d'arêtes/arcs (quel que soit leur sens) auxquels il est relié
Degré entrant $d_-(s)$ /sortant $d_+(s)$	Nombre d'arcs arrivant/partant d'un sommet $d(s) = d_-(s) + d_+(s)$
Graphe complet	Tous les sommets sont adjacents entre eux
Chaîne/Chemin de longueur n	Succession de n arêtes/arcs telle que l'extrémité de chacune est l'origine de la suivante, sauf la dernière
Graphe non orienté connexe	Il existe une chaîne entre n'importe quelle paire de sommets distincts du graphe
Chaîne/Chemin fermé	Chaîne/un chemin qui possède le même sommet de départ et d'arrivée
Cycle/Circuit	Chaîne/Chemin fermé(e) est composé(e) d'arêtes/arcs distincts
Graphe pondéré	Graphe pour lequel chaque arête/arc possède un poids
Propriété	N Nombre d'arcs/sommets $\sum d(s_i) = 2N$
Notations	
Arêtes	$A = \{\{s_i, s_j\}, s_i \in S, s_j \in S\}$
Arcs	$A = \{(s_i, s_j), s_i \in S, s_j \in S\}$
Graphe non pondéré	$G = (S, A)$
Graphe pondéré	$G = (S, A, \omega)$ Pondération des arêtes/arcs : $\omega(s_i, s_j)$

Exemples		
Graphe non orienté d'ordre 4	Graphe orienté	Graphe pondéré
 <p>Chaine ADCB</p> <p>Cycle ADDBA</p>	 <p>Chemin ADCB</p> <p>Circuit BDCB</p>	

Dernière mise à jour	Informatique	Denis DEFAUCHY – Site web
22/03/2023	11 – Bases des graphes	Cours

Matrices d'adjacence

A tout graphe G d'ordre p , de sommets notés s_i , on peut associer une matrice carrée d'ordre p : $M = (m_{ij})$ où m_{ij} est le nombre d'arcs/arêtes reliant les sommets s_i à s_j . Cette matrice est appelée matrice d'adjacence associée à G



		Destination			
		A	B	C	D
PROVENANCE	A	0	1	0	1
	B	1	0	1	1
	C	0	1	0	1
	D	1	1	1	0

		Destination			
		A	B	C	D
PROVENANCE	A	0	0	0	1
	B	1	0	0	1
	C	0	1	0	1
	D	1	0	1	0

Matrices des distances

		Destination			
		A	B	C	D
PROVENANCE	A	0	2	0	1
	B	2	0	4	1
	C	0	4	0	3
	D	1	1	3	0

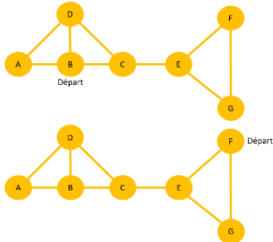
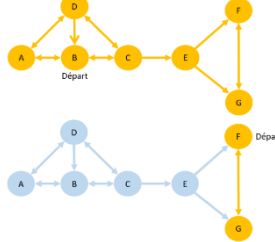
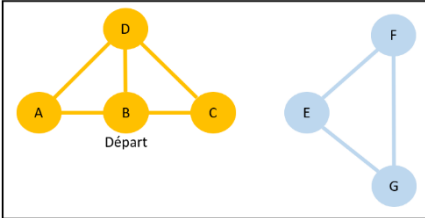
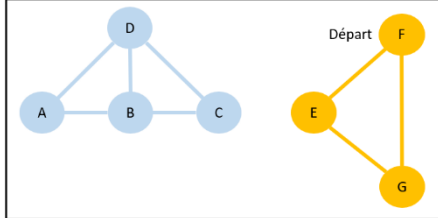
		Destination			
		A	B	C	D
PROVENANCE	A	0	0	0	1
	B	2	0	0	1
	C	0	4	0	3
	D	1	0	3	0

On introduit alors la notion de **poids d'une chaîne**, comme somme de toutes les pondérations des arêtes/arcs composant la chaîne (ex : dans le graphe orienté, le poids de la chaîne ADCB vaut 8)

Liste d'adjacence

Sous forme de liste	Sous forme de dictionnaire
<pre>>>> S = ["A", "B", "C", "D"] >>> L = [{"B", "D"}, {"A", "C", "D"}, {"B", "D"}, {"A", "B", "C"}] >>> s = "B" >>> i = S.index(s) >>> i 1 >>> L[i] ['A', 'C', 'D']</pre>	<pre>>>> D = {"A": ["B", "D"], "B": ["A", "C", "D"], "C": ["B", "D"], "D": ["A", "B", "C"]} >>> D["B"] ['A', 'C', 'D'] Avec des pondérations : >>> D["B"] [['A', 2], ['C', 4], ['D', 1]]</pre>

Dernière mise à jour	Informatique	Denis DEFAUCHY – Site web
22/03/2023	11 – Bases des graphes	Cours

Parcours d'un graphe	
Un outil aux opérations de complexité $O(1)$ pour créer des files (et des piles)	<pre> from collections import deque f = deque() # Création une « dèque » vide f.append(x) # Ajoute l'objet x à droite f.appendleft(x) # Ajoute l'objet x à gauche x = f.pop() # Enlève l'élément à droite x = f.popleft() # Enlève l'élément à gauche t = len(f) # Nombre d'éléments de f </pre>
Parcours en largeur	<p>Mettre le nœud de départ dans une file et le marquer comme visité</p> <p>Tant que la file n'est pas vide :</p> <ul style="list-style-type: none"> - Retirer le premier sommet de la file pour le traiter - Mettre tous ses voisins non visités à la fin de la file et les marquer visités
Parcours en profondeur	<p>Fonction récursive d'exploration Explorer(s) :</p> <ul style="list-style-type: none"> - Marquer le sommet s comme visité <ul style="list-style-type: none"> o Pour tout voisin v de s non marqué : <ul style="list-style-type: none"> ▪ Explorer(v) <p>Le parcours total est alors réalisé ainsi :</p> <ul style="list-style-type: none"> - Pour tout sommet s non marqué du graphe : <ul style="list-style-type: none"> o Explorer(s)
Remarques	<p>Que le graphe soit orienté ou non, le parcours en largeur au départ de s / une seule exécution de la fonction Explorer(s) du parcours en profondeur permet de trouver tous les sommets accessibles depuis s (sommets en orange ci-dessous)</p>   <p>Pour un graphe non orienté connexe, on obtient alors tous les sommets de G (et donc tous les sommets accessibles depuis s).</p>
	<p>Un parcours en largeur depuis s / une seule exécution de Explorer(s) pour un graphe non orienté permet de déterminer les composantes connexes (sous graphe connexe), c'est-à-dire toutes les stations reliées ensemble.</p>  
	<p>Si le résultat du parcours ne contient pas toutes les stations du graphe, le graphe n'est pas connexe.</p>
	<p>Selon l'ordre de sélection des voisins, les parcours peuvent se comporter quelque peu différemment.</p>
	<p>Dans le cas du parcours en largeur, comme les sommets sont explorés par distance croissante au sommet de départ, on trouve le plus court chemin au sens « nombre de sommets » depuis le départ. L'algorithme de Dijkstra abordé plus tard dans ce cours peut être vu comme une extension de cette méthode pour les graphes pondérés</p>

Dernière mise à jour	Informatique	Denis DEFAUCHY – Site web
22/03/2023	11 – Bases des graphes	Cours

Plus court chemin d'un graphe pondéré	
Dijkstra	<p>Soient un graphe pondéré $G(S, A, \omega)$ et les sommets de début s_{deb} et de fin s_{fin}. L'algorithme se déroule ainsi :</p> <ul style="list-style-type: none"> - $P = \emptyset$ ou $Q = S$ - Initialisation de <i>Predecesseur</i> - $d[s] = +\infty \forall s \in S$ - $d[s_{deb}] = 0$ - Tant qu'il existe <u>un sommet hors de P ou un sommet dans Q</u> <ul style="list-style-type: none"> o Choisir s_i hors de P tel que $d[s_i] = \min(d[s_k]), s_k \in \bar{P}$ o $P \leftarrow P \cup s_i$ ou $Q \leftarrow Q - s_i$ o Pour chaque sommet $v_j \in \bar{P}$ ou $v_j \in Q$ voisin de s_i (ie. $\omega(s_i, v_j) \neq 0$) <ul style="list-style-type: none"> ▪ Si $d[s_i] + \omega(s_i, v_j) < d[v_j]$ <ul style="list-style-type: none"> • $d[v_j] = d[s_i] + \omega(s_i, v_j)$ • $Predecesseur[v_j] = s_i$ <p>Fin Pour</p> <p>Fin Tant que</p> <ul style="list-style-type: none"> - Si $d[s_{fin}] \neq \infty$ <ul style="list-style-type: none"> o Le plus court chemin vaut $d[s_{fin}]$ o Remonter le chemin à l'envers par succession des prédécesseurs de s_{fin} à s_{deb}
	Amélioration via condition « Tant que $s_i \neq s_{fin}$ » si une exécution doit donner le résultat au plus vite. Sinon, on trouve tous les chemins depuis s_{deb} , à utiliser ensuite quel que soit s_{fin} sans relancer l'algorithme
	Travailler avec P est légèrement plus intéressant qu'avec Q si on ajoute la condition $s_i \neq s_{fin}$ vis-à-vis de la recherche du minimum car P est vide au départ et se remplit, contrairement à Q
	Amélioration via condition « Tant que $d[s_i] \neq \infty$ » si s_{fin} ne peut être atteint
	La recherche « Pour chaque sommet [...] voisin de s_i » peut être réalisée au préalable en créant une liste d'adjacente, ou alors dans la boucle de résolution à l'aide d'une matrice d'adjacence. Quel que soit l'endroit où elle est placée, elle présentera un même cout de calculs.
	Le choix de s (premier ou dernier minimum), et l'inégalité stricte ou non du test $d[s_i] + \omega(s_i, v_j) < d[v_j]$ peuvent changer la solution retenue (à distances finales égales).
	Il est possible de réaliser une version de l'algorithme de Dijkstra qui améliore son temps d'exécution avec un objet contenant les seuls sommets voisins des sommets déjà traités sans aucun sommet déjà traité. On recherche alors le minimum dans un objet toujours petit (contrairement à \bar{P} ou Q). Encore mieux, on peut ne pas filtrer les sommets déjà traités lors de la mise à jour des voisins.
A* A star A étoile	Heuristique $h(s)$ telle que : $f(s) = g(s) + h(s)$ $g(s)$ le coût réel du chemin optimal partant du sommet initial jusqu'à s $h(s)$ le coût estimé du reste du chemin partant de s jusqu'à un état satisfaisant du but
	Exemple de chemin sur une image : $f(s) = d(s) + d'(s, s_{fin})$ Avec pour heuristique $d'(s, s_{fin})$ la distance à vol d'oiseau entre s et s_{fin}
	Changer la ligne de Dijkstra : Choisir s hors de P tel que $d[s] = \min(d[s_i]), s_i \in \bar{P}$ Par Choisir s hors de P tel que $d[s] = \min(f[s_i]), s_i \in \bar{P}$