L3 Informatique • Algorithmique avancée

Travaux dirigés 3

Parcours de graphes : cycles

Problèmes considères (Cours du Prof. Etienne Birmelé)

- **Chinese Postman problem**
  Un postier cherche l'itinéraire le plus court pour accomplir sa tournée, sachant qu'il doit parcourir toutes les rues dont il a la charge au moins une fois, et revenir à son point de départ.
- **Travelling salesman problem** *problème du voyageur de commerce*
  Un agent commercial doit faire le tour des clients dont il a la charge et revenir à son point de départ, tout en minimisant un certain coût (euros, temps, kilomètres...)

Les graphes de la figure 3.1 contiennent-ils

→ **Un chemin eulérien,**

→ **Un cycle eulérien,**

→ **Un chemin hamiltonien,**

→ **Un cycle hamiltonien.**

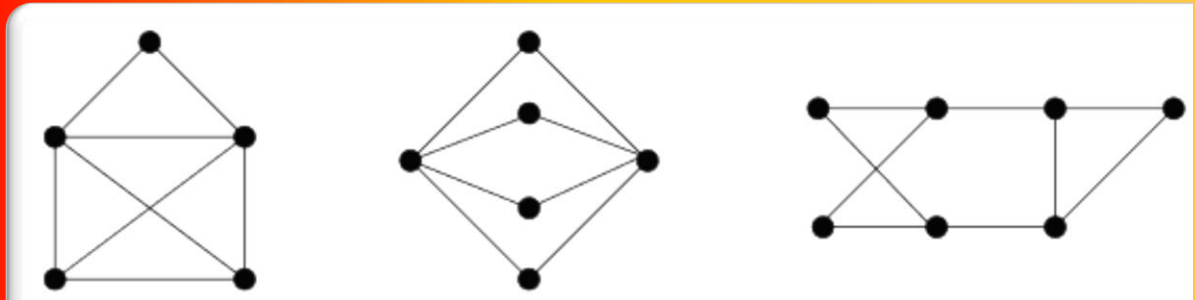
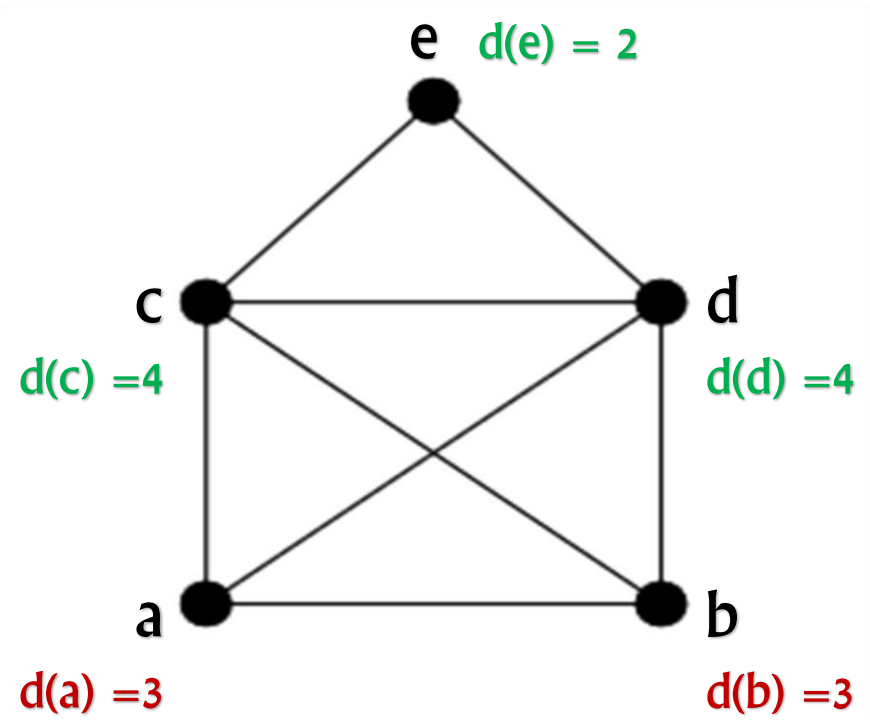


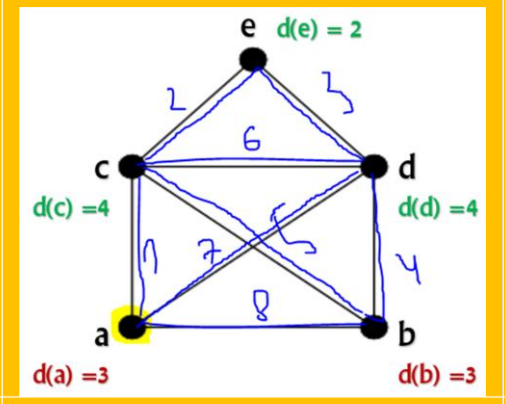
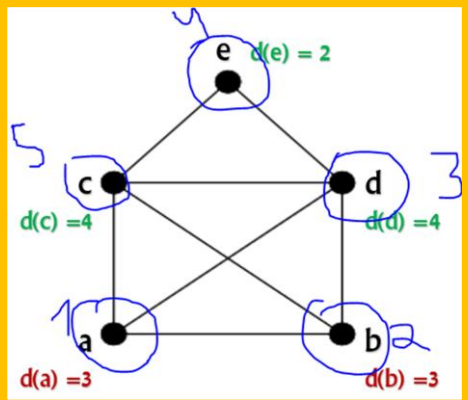
FIGURE 3.1 –

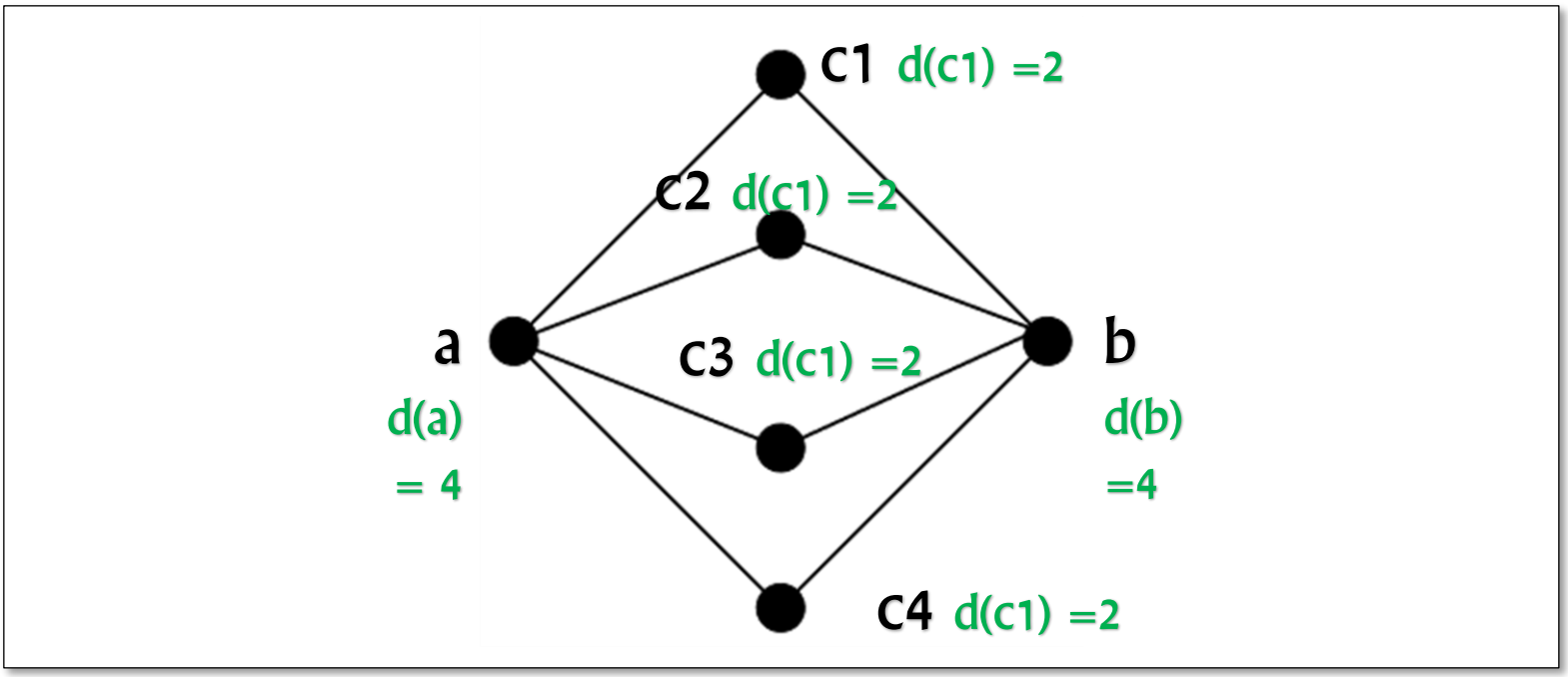
Exercice 3.1

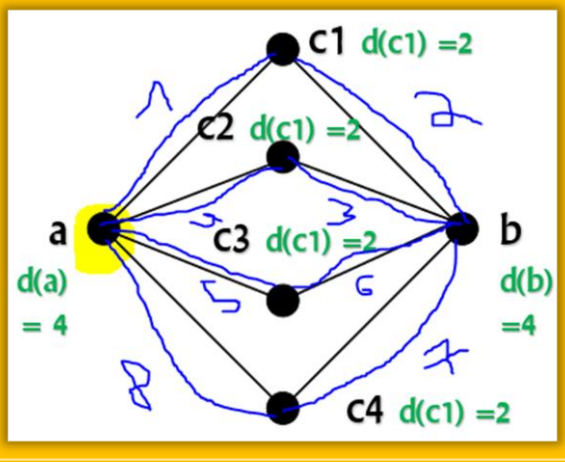
Définitions # 1 [Cours du Prof. Etienne Birmelé]		
Cycle	**Chemin**	**Marche**
Un cycle de longueur k est une suite de k arêtes (ui,ui+1) tels que u0=uk et tous les ui sont disjoints	**Un chemin** de longueur k entre deux sommets u et v est une suite de k arêtes (ui,ui+1) tels que u0=u, uk=v et tous les ui sont disjoint.	Si les arêtes et les sommets ne sont pas tous disjoints, on parle de **marche** entre u et v.

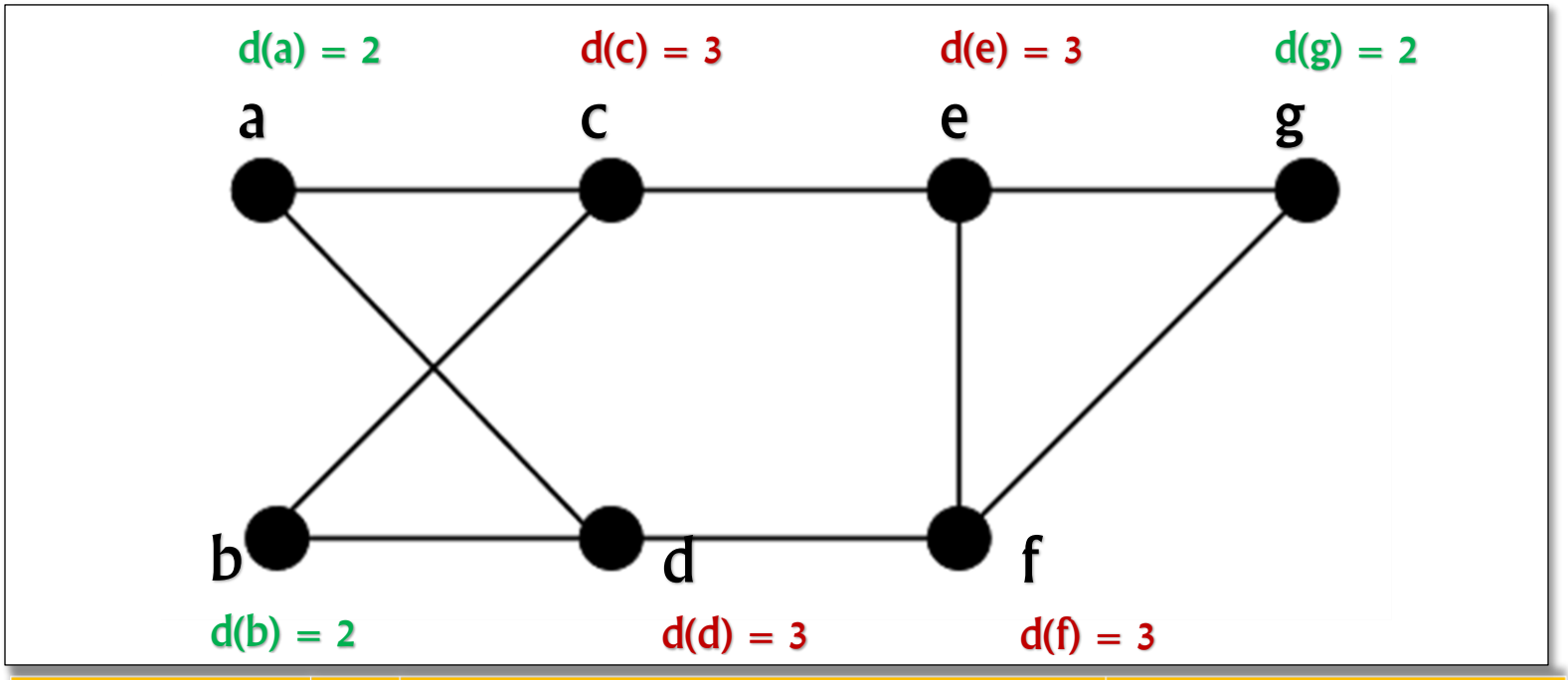
Définitions # 2 [Cours du Prof. Etienne Birmelé]		
	Eulérien	**Hamiltonien**
Graphe	Un **graphe** connexe est **eulérien** si et seulement si tous ses sommets sont de degré pair	Un **graphe est dit hamiltonien** s'il admet un circuit hamiltonien.
Cycle	Un **circuit eulérien** dans un graphe est une marche empruntant chaque arête exactement une fois et terminant en son point de départ. Un **graphe eulérien** est un graphe admettant un circuit eulerien	Un **cycle hamiltonien** est un cycle qui passe par tous les sommets d'un graphe.
Chemin	Un **parcours eulérien** est une marche couvrant toutes les arêtes en les empruntant de façon unique (on supprime simplement la condition de retour au point de départ) Un graphe contient un **parcours eulérien** si et seulement si au plus deux de ses sommets ont un degré impair.	Un **chemin hamiltonien** d'un graphe orienté ou non orienté est un chemin qui passe par tous les sommets une fois et une seule [Wikipedia]

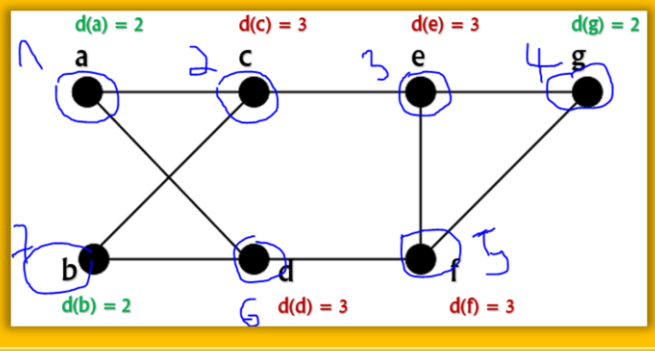


chemin eulérien	+	a c e d b c d a b est un chemin eulérien.  	Un **parcours eulérien** est une marche couvrant toutes les arêtes en les empruntant de façon unique (on supprime simplement la condition de retour au point de départ) Un graphe contient un **parcours eulérien** si et seulement si au plus deux de ses sommets ont un degré impair.
cycle eulérien	-	car il y a des sommets de degré impair.	Un **graphe** connexe est **eulérien** si et seulement si tous ses sommets sont de degré pair . Un **graphe eulérien** est un graphe admettant un circuit eulerien
chemin hamiltonien	+	abdec est un cycle hamiltonien et donc aussi un chemin hamiltonien	Un **cycle hamiltonien** est un cycle qui passe par tous les sommets d'un graphe.
cycle hamiltonien	+	 	Un **chemin hamiltonien** d'un graphe orienté ou non orienté est un chemin qui passe par tous les sommets une fois et une seule



chemin eulérien	+	a c1 b c2 a c3 b c4 a est un cycle eulérien et donc un chemin eulérien	Un **parcours eulérien** est une marche couvrant toutes les arêtes en les empruntant de façon unique (on supprime simplement la condition de retour au point de départ)
cycle eulérien	+		Un **circuit eulérien** dans un graphe est une marche empruntant chaque arête exactement une fois et terminant en son point de départ. Un **graphe eulérien** est un graphe admettant un circuit eulerien
chemin hamiltonien	-	Ne contient ni chemin ni cycle hamiltonien. En effet, passer par c1, c2 et c3 oblige à passer deux fois par a ou par b.	Un **chemin hamiltonien** d'un graphe orienté ou non orienté est un chemin qui passe par tous les sommets une fois et une seule
cycle hamiltonien	-		Un **cycle hamiltonien** est un cycle qui passe par tous les sommets d'un graphe.



chemin eulérien	-	car il y a plus que 2 sommets de degré impair.	Un graphe contient un **parcours eulérien** si et seulement si au plus deux de ses sommets ont un degré impair.
cycle eulérien	-	car il y a des sommets de degré impair.	Un **graphe eulérien** est un graphe admettant un circuit eulerien. Un graphe connexe est eulérien si et seulement si tous ses sommets sont de degré pair
chemin hamiltonien	+	a c e g f d b est un chemin hamiltonien 	Un **chemin hamiltonien** d'un graphe orienté ou non orienté est un chemin qui passe par tous les sommets une fois et une seule
cycle hamiltonien	-	un cycle hamiltonien contiennent forcément la séquence c e g f d mais on ne peut pas y inclure a et b de façon satisfaisante et donc le graphe n'a pas de cycle hamiltonien.	Un **cycle hamiltonien** est un cycle qui passe par tous les sommets d'un graphe.

Résoudre le problème du postier sur le graphe de la figure 3.2

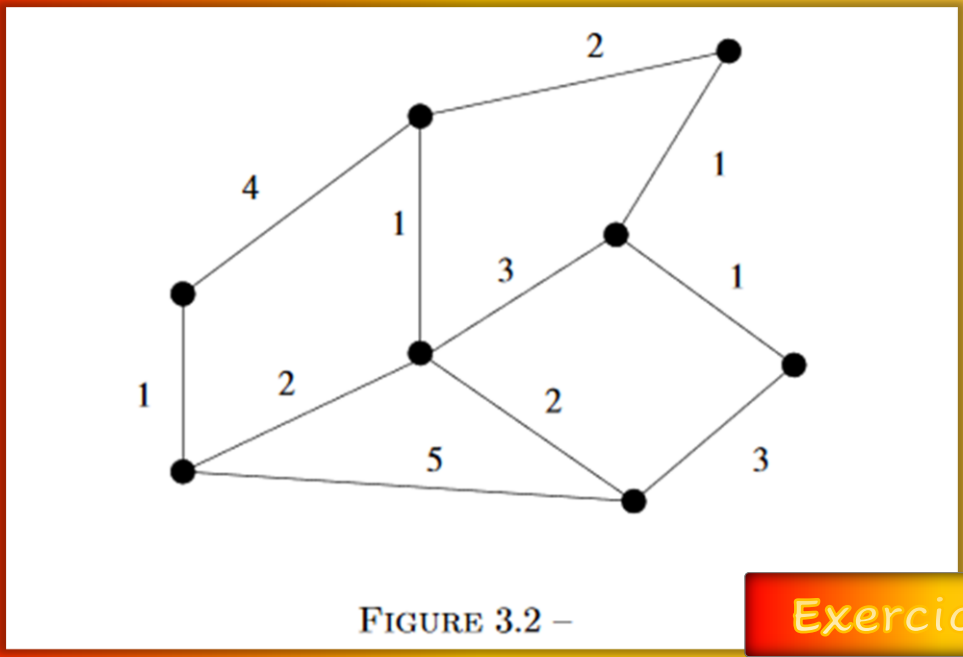


FIGURE 3.2 –

Exercice 3.2

Chinese Postman problem – *le problème du postier*

Un postier cherche l'itinéraire le plus court pour accomplir sa tournée, sachant qu'il doit parcourir toutes les rues dont il a la charge au moins une fois, et revenir à son point de départ.

Recherche de cycles couvrant de poids minimum dans un graphe arête-valué : Le postier doit couvrir toutes les arêtes

Résolution du Chinese Postman Problem

1. Enumérer l'ensemble S des sommets de degrés impair.
2. Pour toute paire de sommets de S, chercher un chemin de longueur minimale entre ces deux sommets (Algorithme de Dijkstra)
3. Créer un graphe complet H avec $V(H) = S$ dans lequel les arêtes sont valués par le poids du plus court chemin précédent.
4. Chercher un couplage parfait de poids minimal dans ce graphe. Un couplage est un ensemble d'arêtes tel que tout sommet soit incident à exactement une arête. Ce problème peut être résolu en temps polynomial par un algorithme de programmation linéeaire. (hors cadre de ce cours)
5. Doubler les arêtes le long des chemins correspondant à ce couplage minimal
6. Appliquer l'algorithme de Fleury sur le graphe résultant.

1. Enumérer l'ensemble S des sommets de degrés impair.

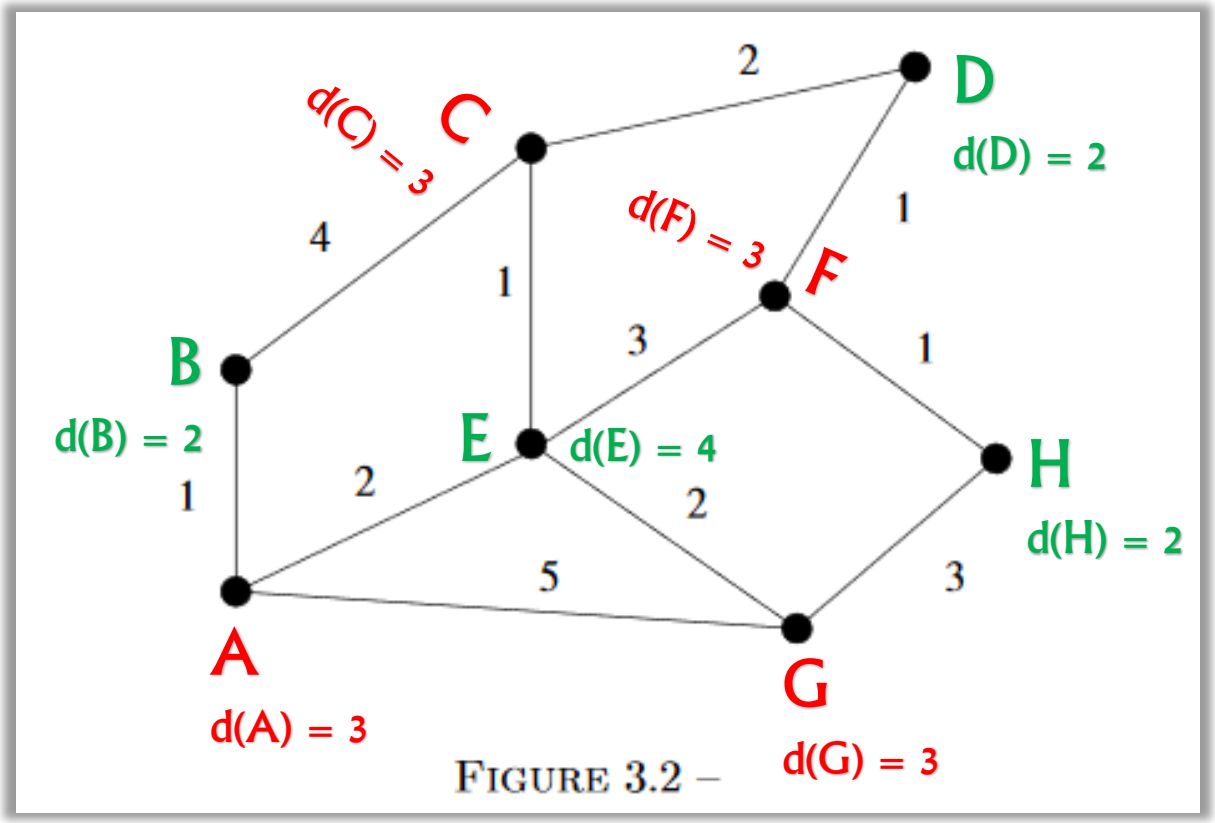


FIGURE 3.2 –

Les sommets de degré impair sont A C F G.

$S = (A, C, F, G)$

2. Pour toute paire de sommets de S, chercher un chemin de longueur minimale entre ces deux sommets (Algorithme de Dijkstra)

On peut faire le calcul directement (à la place d'utiliser l'algorithme de Dijkstra).

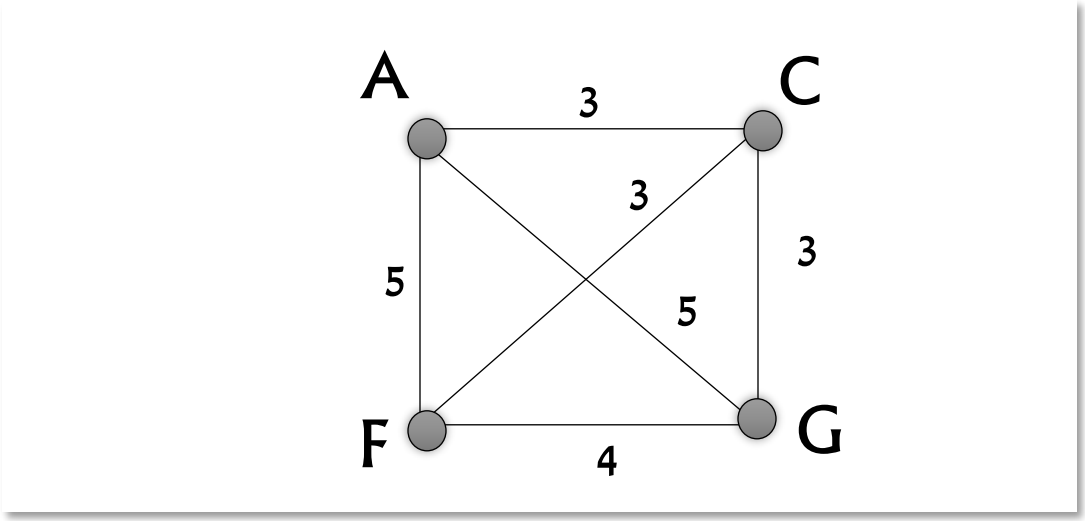
On va regarder l'ensemble des sommets qui apparaisse dans S, donc A, C, F et G, et donner le chemin de longueur minimal entre chaque 2 sommets, donc – la distance. Donc on prend les sommets 2 a 2 .

Par exemple la distance entre A et F est égal a 5, parce que lorsque qu'on part de A on fait 2 jusqu'a E + 3 entre E et F,

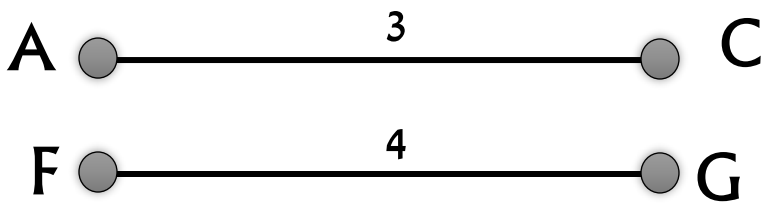
Calcul des distances :

$\boldsymbol{distance(A,F) = 5}$
$\boldsymbol{distance(A,C) = 2 + 1 = 3}$
$\boldsymbol{distance(A,G) = 5}$
$\boldsymbol{distance(C,F) = 2 + 1 = 3}$
$\boldsymbol{distance(C,G) = 1 + 2 = 3}$
$\boldsymbol{distance(F,G) = 1 + 3 = 4}$

3. Créer un graphe complet H avec $V(H) = S$ dans lequel les arêtes sont valués par le poids du plus court chemin précédent.



4. Chercher un couplage parfait de poids minimal dans ce graphe. Un couplage est un ensemble d'arêtes tel que tout sommet soit incident à exactement une arête. Ce problème peut être résolu en temps polynomial par un algorithme de programmation linéeaire. (hors cadre de ce cours)



5. Doubler les arêtes le long des chemins correspondant à ce couplage minimal

On va sur le graphe doubler les arêtes qui relie de C à A, donc on va regarder le chemin dont on parle, et on va ajouter les 2 arêtes suivantes :

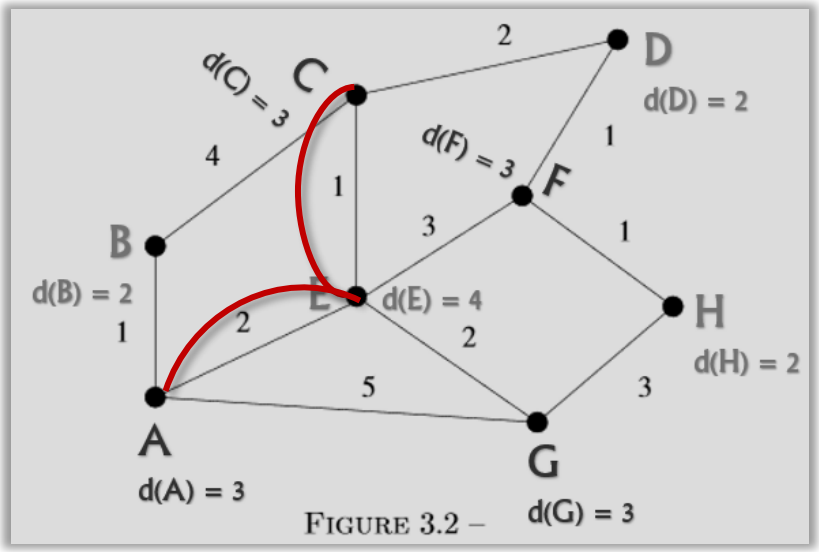


FIGURE 3.2 –

Donc on a l'arête qui existe déjà et là on la double.
On va faire pareil pour F et G : on double les arêtes qui relie F à G par le chemin le plus court.

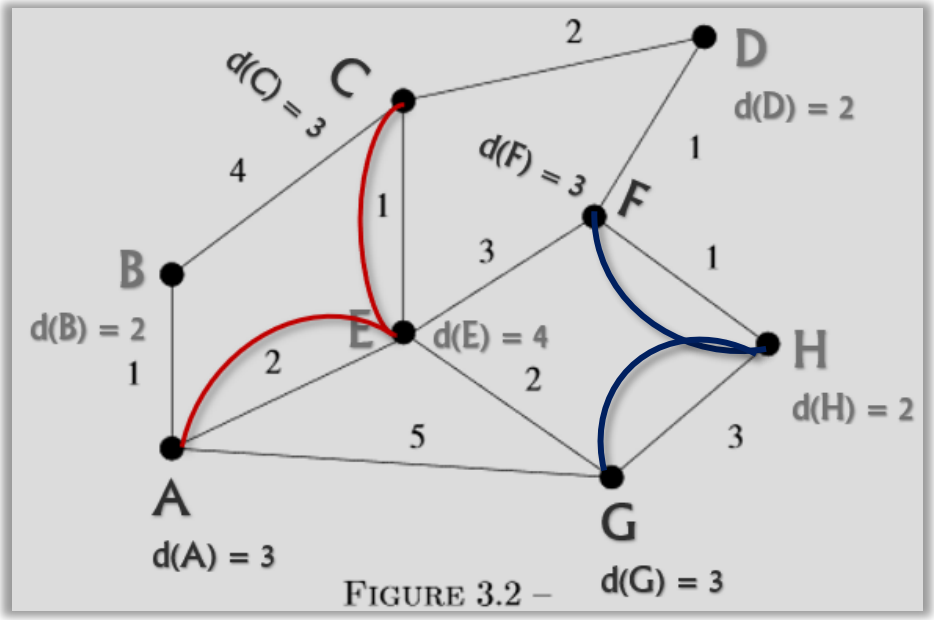


FIGURE 3.2 –

6. Appliquer l'algorithme de Fleury sur le graphe résultant.

Ensuite, à partir de ce graphe obtenu, on applique l'algorithme de Fleury.

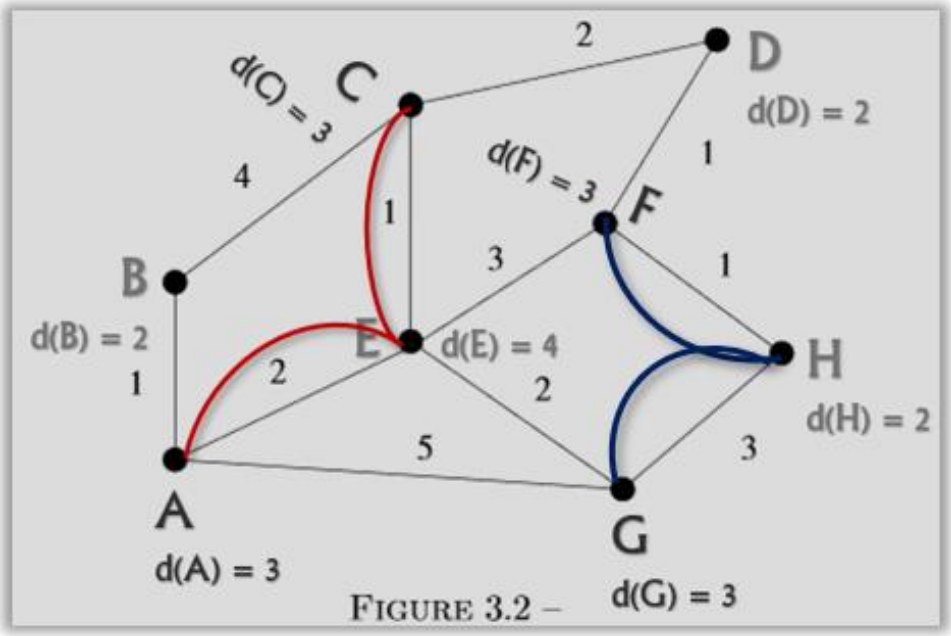


FIGURE 3.2 –

Algorithme de Fleury

`https://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/`

1. Choisir un sommet v_0 et poser $W_0 = \emptyset$.
2. Supposons que le parcours (liste d'arête) $W_i = e_1 \dots e_m$ a été contruit. Soit v_m le dernier sommet de ce parcours.
3. Si il existe des arêtes adjacentes à v_m non-encore utilisées, on en choisit une appelée e_{m+1}. Sauf s'il n'y a pas le choix, on ne choisit pas e_{m+1} comme étant un pont dans $G_m = G - \{e_1, \dots, e_m\}$. On pose $W_{m+1} = W_m \cup e_{m+1}$.
4. Sinon, on arrête et on retourne W_m

On notera que quand on efface une arête après l'avoir sélectionné on retire également les points qui deviendraient isolés. On notera également que si aucune degré impair, on peut partir de n'importe quel sommet et si deux degrés impairs on part de l'un de ces deux sommets.

1. Choisir un sommet v_0 et poser $W_0 = \emptyset$.

On peut choisir A comme point de départ,

et ensuite on a un ensemble W_0, dans cette ensemble on va mettre les arêtes qu'on choisit.

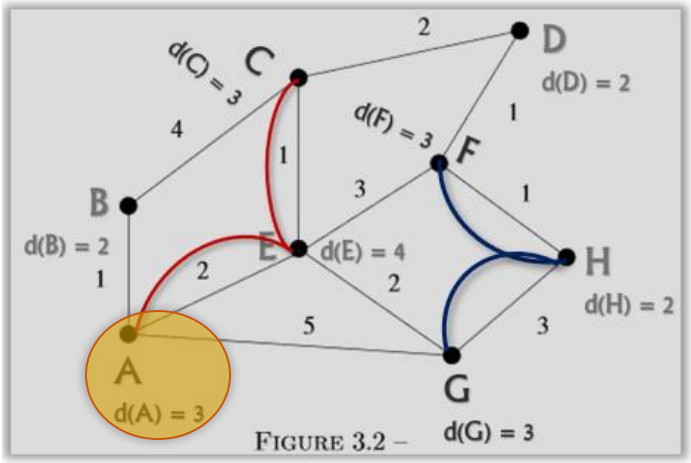


FIGURE 3.2 –

En partant d'un sommet on va choisir l'une des arêtes, mes la condition qu'on doit respecter est que l'arête qu'on va choisir ne sépare pas le graphe on plusieurs composantes connexes. Donc on va choisir une arête et l'objectif est que l'arête qu'on choisi ne coupe pas le graphe.

Ont choisi une arête et enfaite ce qu'on va faire c'est la supprimer pour montrer qu'on va plus la sélectionner par la suite,

Donc la première arête a supprimé est :

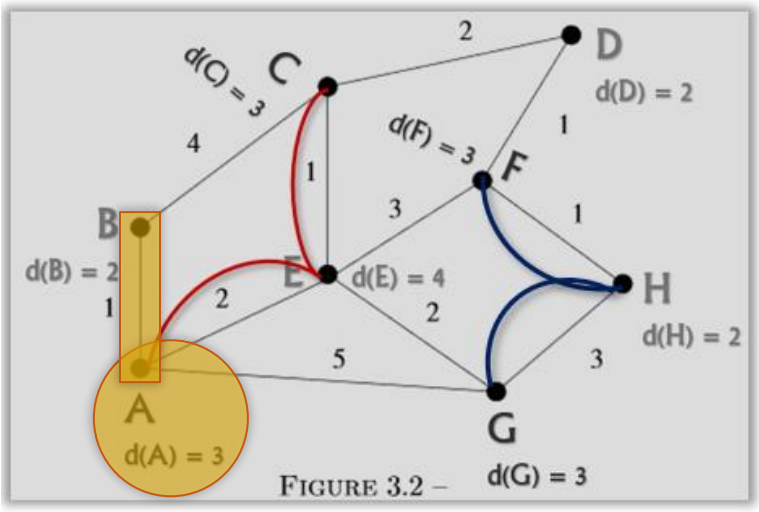


FIGURE 3.2 –

On peut la choisir parce qu'en supprimons cette arête on n'est pas en train de séparer, de créerez 2 composantes connexes, le graphe qu'on obtient en supprimons cette arrête reste connexe,

Ensuite le sommet courant sur laquelle on est, c'est le sommet B,

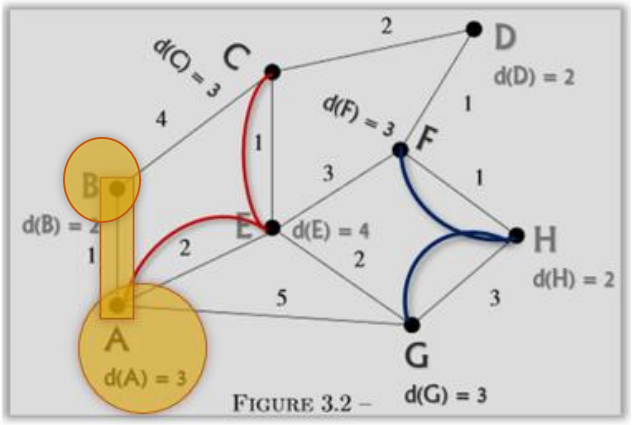


FIGURE 3.2 –

Donc là, la seul possibilité qu'on a c'est de choisir l'arête BC.

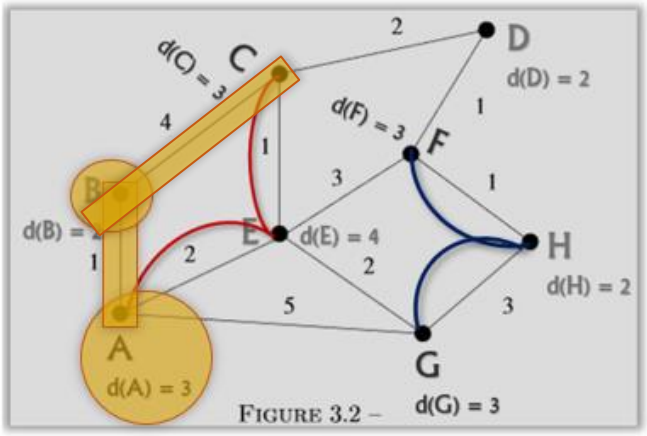


FIGURE 3.2 –

Arriver ici, on a la possibilité de choisir entre les arêtes rouges et grise, donc ont choisi une arête qui permet de ne pas séparer le graphe,

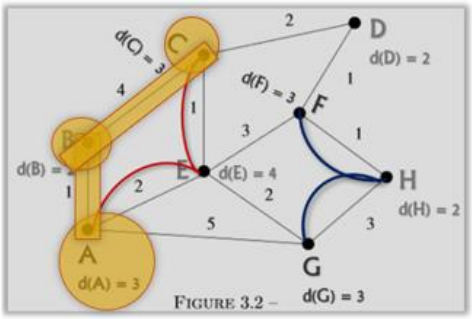


FIGURE 3.2 –

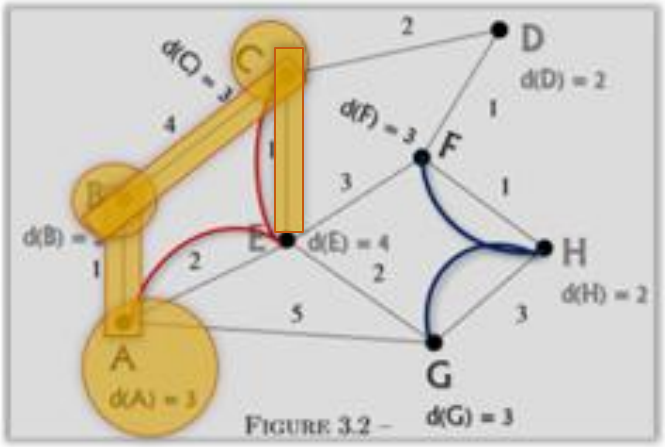


FIGURE 3.2 –

Et là c'est pareil, on a le sommet courant et ont choisi une arête qui permet de ne pas séparer en plusieurs composantes connexes, par exemple :

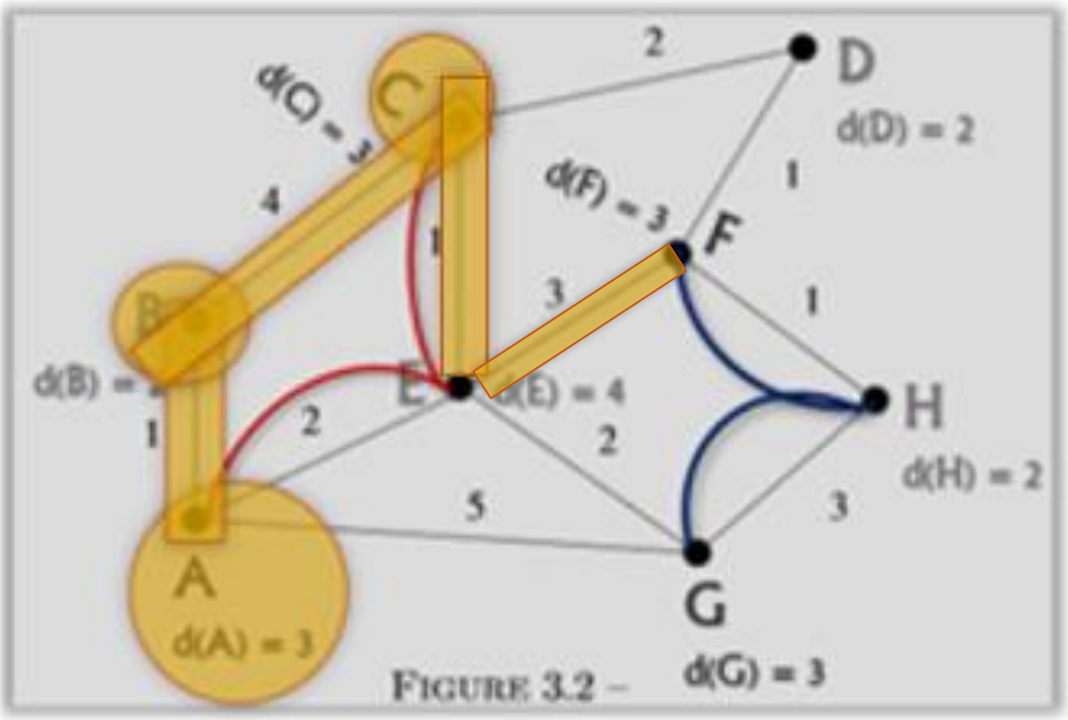


FIGURE 3.2 –

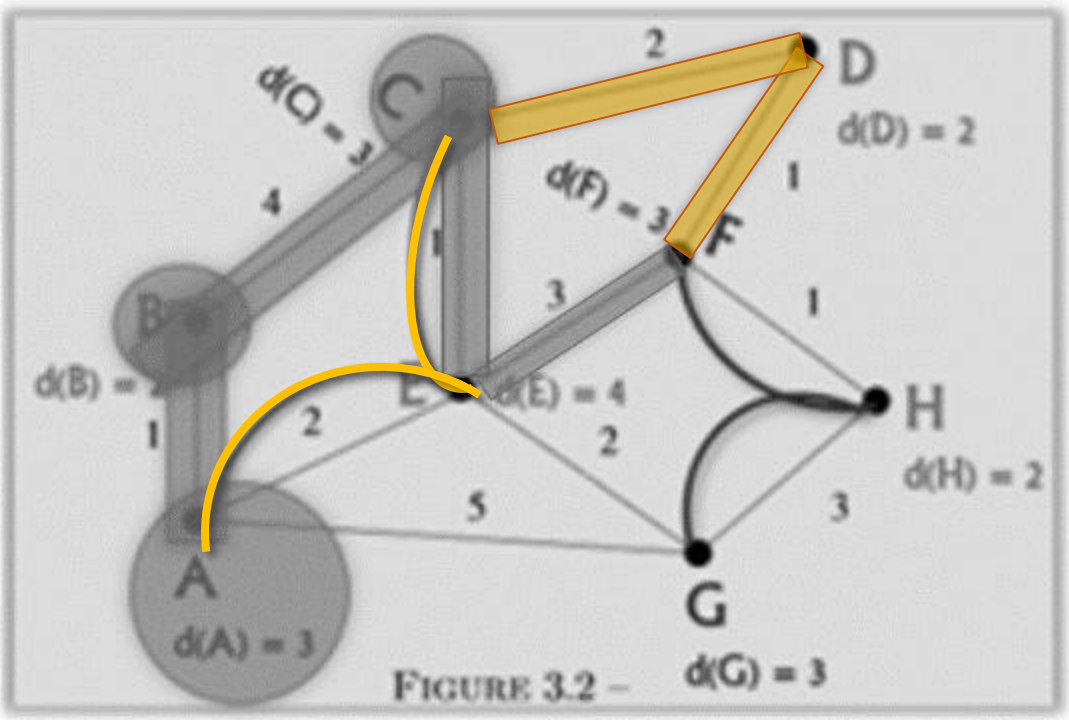


FIGURE 3.2 –

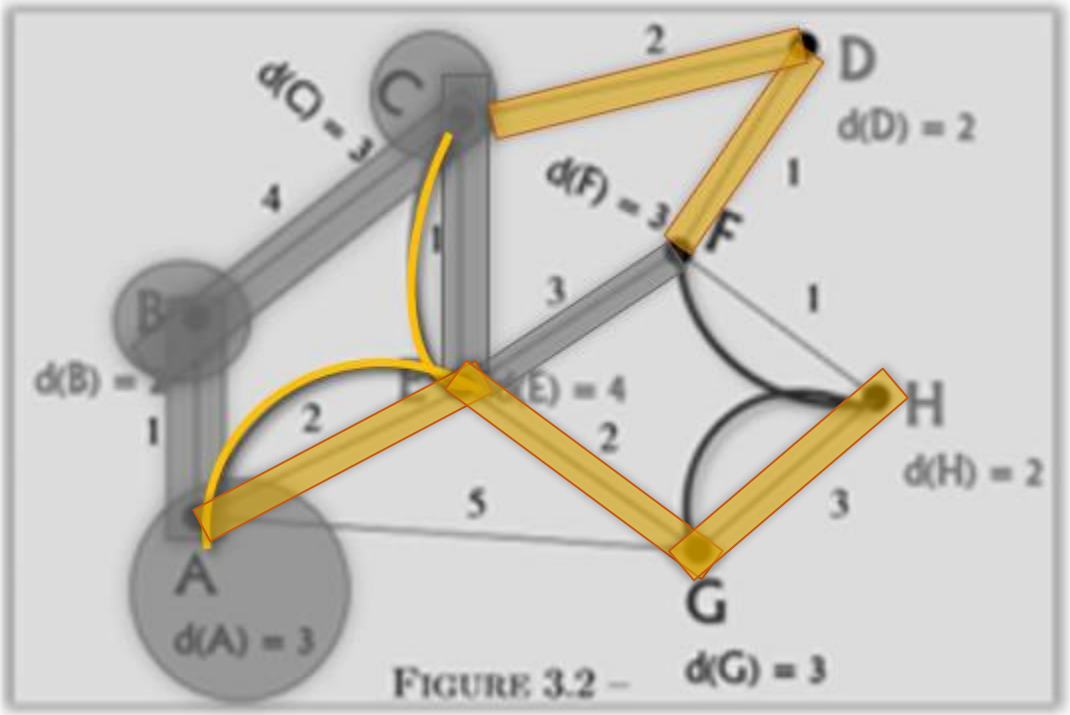


FIGURE 3.2 –

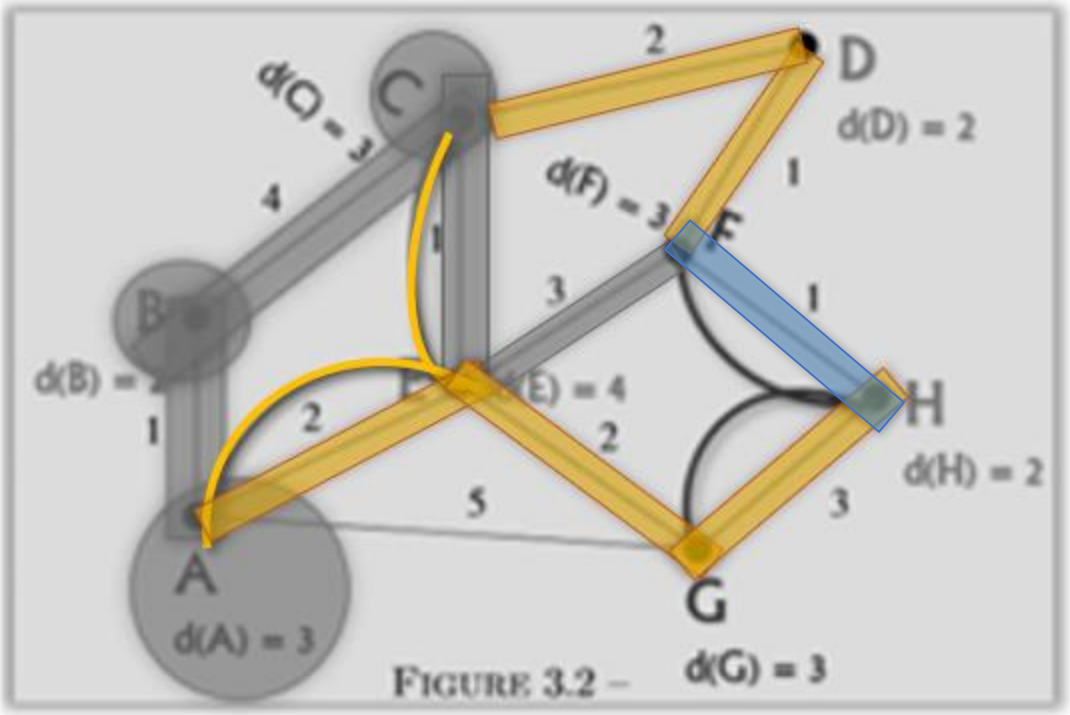


FIGURE 3.2 –

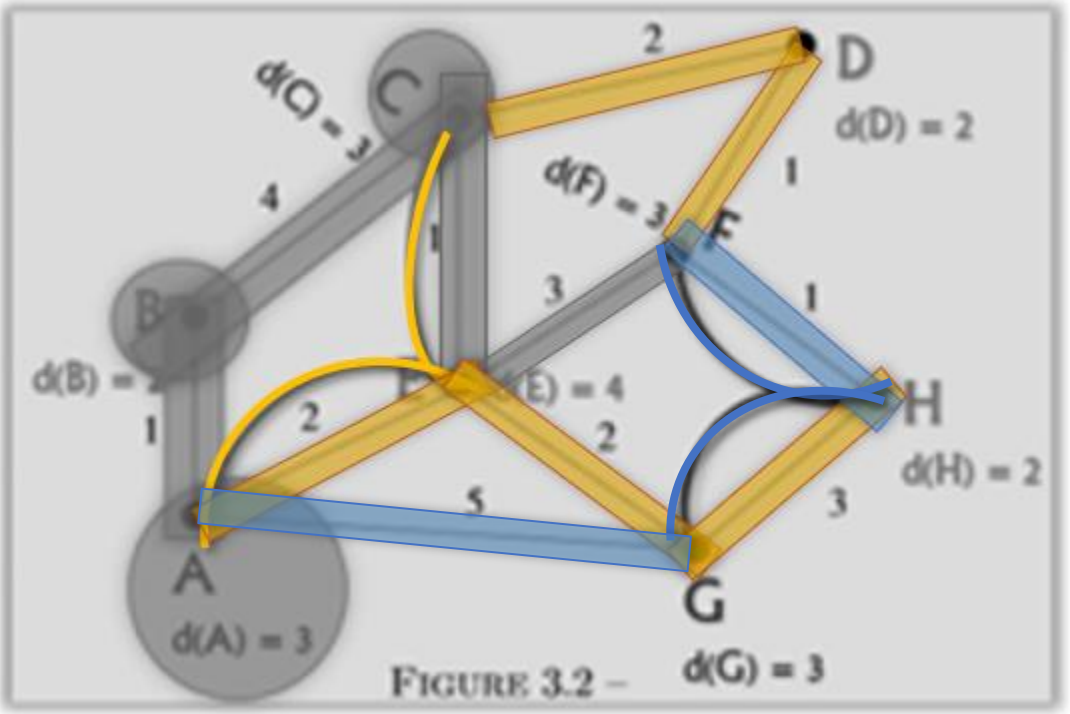


FIGURE 3.2 –

On obtient par exemple,

A B C E F D C E A E G H F H G A.

Il existe beaucoup d'autres solutions mais elles sont toutes de poids égal.

(somme des poids des arêtes du graphe de départ + 3 + 4).

Et donc là ça nous donne un cycle.

Un **circuit eulérien** dans un graphe est une marche empruntant chaque arête exactement une fois et terminant en son point de départ. Un **graphe eulérien** est un graphe admettant un circuit eulerien

[Cours du Prof. Etienne Birmelé]

Algorithme de Fleury

il existe un algorithme simple, du a Fleury (1883), qui trouve un tour eulérien dans un graphe connexe pair quelconque G.

L'Algorithme de Fleury construit un tel tour de G en marquant un parcours sous la condition qu'à chaque étape, une arête séparatrice du sous-graphe non-marqué F est prise seulement s'il n'y a pas d'alternative.

Théorème 3.4 *Si G est un graphe connexe pair, la marche W retournée par l'algorithme de Fleury est un tour eulérien de G.*

Théorie des Graphes

J.A. Bondy et U.S.R. Murty
Traduit de l'anglais par F. Havet

Pour résoudre le problème du voyageur de commerce, on propose l'heuristique (résolution approchée)_suivante :

Etape 1 construire un arbre couvrant de poids minimal ;
Etape 2 parcourir les sommets de cet arbre comme un parcours en profondeur.

Question 1

Exercice 3.3

Quelle est la complexité de cet algorithme ?

Travelling salesman problem *problème du voyageur de commerce*
Un agent commercial doit faire le tour des clients dont il a la charge et revenir à son point de départ, tout en minimisant un certain coût (euros, temps, kilomètres…)

Recherche de cycles couvrant de poids minimum dans un graphe arête-valué : le postier doit couvrir toutes les arêtes, alors que le voyageur doit 'seulement' couvrir tous les sommets.

Cette différence rend ces deux problèmes très différents : celui du postier peut être résolu de façon exacte, alors que celui de l'agent ne peut faire l'objet que d'une résolution approchée (heuristique).

Recherche d'un arbre couvrant de poids minimal	$o(m\ log\ m)$
Parcours en profondeur sur cet arbre	$o(m)$

Complexité en $o(m\ log\ m)$

BFS	**Breadth First Search**	Parcours en largeur	***O(m)***
DFS	**Depth First Search**	Parcours en profondeur	***O(m)***
Algorithme de Kruskal	Kruskal renvoie pour un graphe connexe, un arbre couvrant de poids minimal		***O(m log m)***
Algorithme de Prim	Prim renvoie pour un graphe connexe, un arbre couvrant de poids minimal.		***O(mn)*** pour un code naïf. Elle peut être abaissée en ***O(m log n)*** en codant à l'aide de tas binaires et en ***O(m + n log n)*** a l'aide de tas de Fibonacci

On note n(G) = |V | et m(G) = |E| , ou n et m quand il n'y a pas d'ambigüité.

Définitions (désignations des complexités courantes)

notation	*désignation*	*notation*	*désignation*
$\Theta(1)$	*constante*	$\Theta(n^2)$	*quadratique*
$\Theta(\log n)$	*logarithmique*	$\Theta(n^3)$	*cubique*
$\Theta(\sqrt{n})$	*racinaire*	$\Theta(n^k),\ k \in \mathbb{N}, k \geq 2$	*polynomiale*
$\Theta(n)$	*linéaire*	$\Theta(a^n),\ a > 1$	*exponentielle*
$\Theta(n \log n)$	*quasi-linéaire*	$\Theta(n!)$	*factorielle*

Pour résoudre le problème du voyageur de commerce, on propose l'heuristique (résolution approchée)_suivante :

Etape 1 construire un arbre couvrant de poids minimal ;

Etape 2 parcourir les sommets de cet arbre comme un parcours en profondeur.

Question 1

Exercice 3.3

Soit H le graphe induit par les arêtes utilisées par un parcours optimal.

Montrer que H contient un arbre couvrant.

En déduire que la solution proposée par l'heuristique est une 2-approximation, c'est-a-dire qu'elle est au pire de poids deux fois plus important que l'optimal.

Algorithmes d'approximation

- **Lorsqu'un problème d'optimisation est difficile, on sait qu'il est impossible de trouver un algorithme polynomial donnant une solution optimale.**
- Il est assez naturel de se demander si relâcher de manière contrôlée la contrainte d'optimalité pourrait permettre d'obtenir une complexité polynomiale.
- **En d'autres termes, on aimerait un algorithme rapide, retournant une solution non optimale mais dont on peut borner l'erreur.**

« On dit que A est un algorithme de $r - approximation$ »

Le ratio r donne la borne d'erreur de l'algorithme d'approximation par rapport à la solution optimale.

Par exemple

un algorithme de 2-approximation
a un ratio 2 et retournera toujours une solution au pire deux fois plus grande (ou plus petite) que la solution optimale.

Arbre

Un graphe **non-orienté** **connexe** et **acyclique** est appelé un arbre.

Connexité, graphe connexe (Définition) : Un graphe est connexe si,pour toute paire de sommets u et v de G, il existe un chemin entre u et v

Un graphe acyclique est un graphe ne contenant aucun cycle.

Dans un graphe non orienté, un cycle est une suite d'arêtes consécutives (chaine simple) dont les deux sommets extrémités sont identiques.

Arbre couvrant

Définition

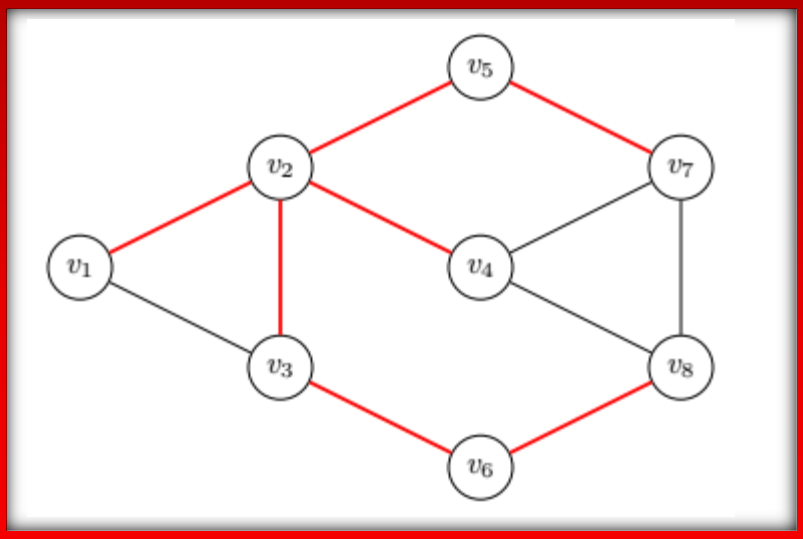
Soit G un graphe.

Un arbre couvrant de G

est un sous-graphe T de G tel que

→ T est un arbre

→ Et $V(T) = V(G)$.



Un arbre couvrant d'un graphe G est un sous-graphe de G qui est connexe, sans cycle et contient tous les sommets de G.

Etape 1 construire un arbre couvrant de poids minimal ;

Etape 2 parcourir les sommets de cet arbre comme un parcours en profondeur.

On construit un arbre qui est couvrant de poids minimal, et sur cette arbre ensuite on va parcourir les sommets avec un parcours en profondeur. Donc la solution apporter se fait a l'étape 2, cette a dire, parcourir l'arbre comme on le fait pour un parcours en profondeur et ca seurai le cheminement que suivrait le voyageur (= la solution apporter au problème du voyageur).

Soit H le graphe induit par les arêtes utilisées par un parcours optimal.

Montrer que H contient un arbre couvrant.

Soit H le parcours optimal.

Le parcours optimal = le parcours de l'ensemble des sommet avec un cout minimal (optimal). H est le graphe qu'on obtient si on arrive à résoudre le problème du voyageur de commerce. Cette a dire que c'est dans H qu'on retrouve les arêtes qu'on utilise pour résoudre le problème du voyageur.

H contient un arbre couvrant du graphe de départ G ?

Oui, parce que H, vu que c'est la solution du problème du voyageur, on retrouve tout les sommets du grange de départ G. Donc H contient un arbre couvrant de G (le graphe de départ).

H est un graphe connexe qui relie tous les sommets du graphe de départ. Tout arbre couvrant de H est un arbre couvrant du graphe de départ.

Soit T l'arbre couvrant de poids minimal déterminé par l'algorithme et U un arbre couvrant de H.

Alors, $w(T) \leq w(U)$ car T est minimal
et $w(U) \leq w(H)$ car U est un sous-graphe de H.

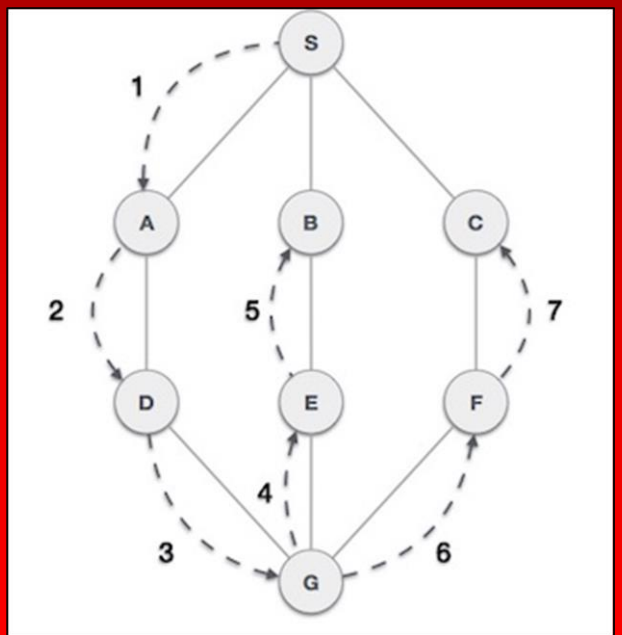
D'où $w(T) \leq w(H)$ → **Weight**

En déduire que la solution proposée par l'heuristique est une 2-approximation, c'est-a-dire qu'elle est au pire de poids deux fois plus important que l'optimal.

Suivre le parcours en profondeur le long de T, parcourt chaque arête deux fois (une fois en descendant, une fois en montant). Ce parcours pour le voyageur coute donc $2w(T)$.

Le cout est par conséquent inferieur a $2w(H)$
Càd égal à au plus deux fois le cout optimal.

DFS	Depth First Search	Parcours en profondeur



Soient G un graphe non-orienté et v un sommet de G.
On peut construire le parcours en profondeur de G partant de v ,

Cet algorithme revient à parcourir le graphe en avançant tant que possible et en remontant lorsque ce n'est plus possible, c'est-à-dire au père du sommet courant si ce dernier n'a pas de voisin non visités. Il s'arrête lorsque le sommet courant n'a plus de voisin non visité et que c'est la racine.

Source https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm