Université de Caen Normandie
UFR des Sciences

M1-INFO
Graphes et Recherche Arborescente
2017-2018

Devoir Maison à faire par binôme (date de remise : Mercredi 08/11/2017)

Soit $G = (V, E, w)$ un graphe complet valué par w une fonction de coût de E dans R^+.

- Le Problème du Voyageur de Commerce (PVC) consiste à déterminer un cycle hamiltonien de coût minimal. Ce problème est souvent formulé comme celui d'un voyageur de commerce qui doit visiter un ensemble de clients situés dans des villes différentes. Le graphe a alors pour sommets l'ensemble des villes des clients et le domicile du représentant. Le poids d'une arête $w(i, j)$ représente le coût pour se déplacer de la ville i à la ville j. Ce coût peut être lié à la distance entre les 2 villes, au temps de trajet nécessaire et/ou à d'autres grandeurs. Le voyageur doit partir de son domicile, puis visiter une fois et une seule chaque client, et revenir à son domicile. Evidemment, on cherche à minimiser le coût d'un tel parcours.
Le PVC permet de modéliser de nombreux problèmes dans des domaines aussi différents que la constitution de tournées (bus scolaires, collecte du lait, distribution de colis …), l'ordonnancement de production, le câblage de circuits, la synthèse de circuits logiques, … En savoir plus :

https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_voyageur_de_commerce
https://en.wikipedia.org/wiki/Travelling_salesman_problem
http://www.math.uwaterloo.ca/tsp/

- Un algorithme d'approximation permet de calculer une solution approchée d'un problème d'optimisation en garantissant que le rapport entre la qualité de la solution obtenue et la qualité de la solution optimale soit inférieur[1] à une constante $(k > 1)$, ceci pour toutes les instances possibles du problème.
Ainsi, H est un k-algorithme d'approximation pour le PVC ssi pour tout graphe (V, E, w) la solution s (obtenue par H) est de coût $w(s) \leq k.w(s^*)$, où s* est un cycle hamiltonien de longueur minimale de (V, E, w).

Dans ce devoir, on considère les instances (V, E, w) du PVC où la ***fonction de coût w vérifie l'inégalité triangulaire.*** **Le but de ce devoir est d'étudier deux algorithmes d'approximation pour le PVC**. Ces deux algorithmes utilisent la notion d'Arbre Recouvrant de Poids Minimal (ARPM). Le 1er algorithme constitue une 2-approximation alors que le 2nd (algorithme de CHRISTOFIDES) constitue une 3/2 approximation (ce qui en fait une approximation de meilleure qualité).
Ces deux algorithmes commencent par calculer un ARPM du graphe. Puis, chacun d'eux, à sa façon, complète cet ARPM pour obtenir un cycle hamiltonien, tout en garantissant la qualité de la solution obtenue.

- Le devoir s'organise en trois parties :

1. Mise en œuvre de l'algorithme de KRUSKAL en utilisant le type abstrait de données UNION-FIND
2. Etude et mise en œuvre de l'algorithme de 2-approximation
3. Etude de l'algorithme de CHRISTOFIDES (3/2-approximation)

On souhaite implanter l'algorithme de 2-approximation en utilisant l'algorithme de KRUSKAL. ***Sont à rendre :***

- le code source commenté avec quelques exemples d'exécution des Parties 1 et 2
- un dossier :
  - décrivant et justifiant vos choix d'implantation
  - et répondant aux questions posées dans la suite de cet énoncé

[1] Dans le cas de la minimisation

Partie 1 : Mise en œuvre de l'algorithme de KRUSKAL

1. On s'intéresse au **type abstrait de données (ADT) « partition d'un ensemble »** (UNION-FIND)[2] ou relation d'équivalence définie sur un ensemble. Il s'agit de trouver une structure de données appropriée pour gérer trois opérations :

1. Créer une relation d'équivalence « triviale » où chaque élément est seul dans sa classe.
2. Tester si deux éléments sont dans la même classe d'équivalence.
3. Faire l'union de deux classes d'équivalence pour n'en faire qu'une seule.

On notera **Sx** l'ensemble auquel appartient l'élément **x** (= la classe d'équivalence de **x**).

Ce type abstrait possède **trois primitives :**

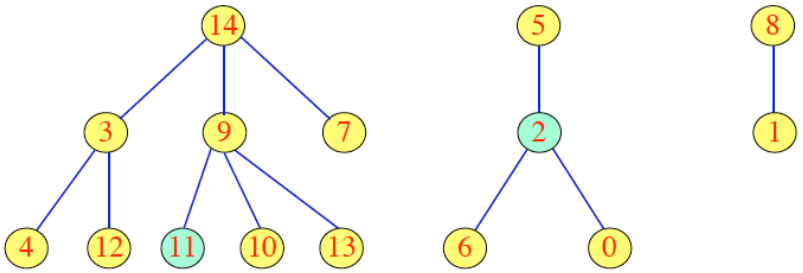
- CRÉER-ENSEMBLE(x : Elément) crée un nouvel ensemble Sx dont le seul élément (et donc le représentant) est l'élément x.

- UNION(x, y : Elément) effectue l'union des ensembles disjoints Sx et Sy. Le représentant de (Sx U Sy) peut être un élément quelconque de cet ensemble. Bien souvent, on choisit, comme représentant de cette union, soit le représentant de Sx, soit le représentant de Sy.

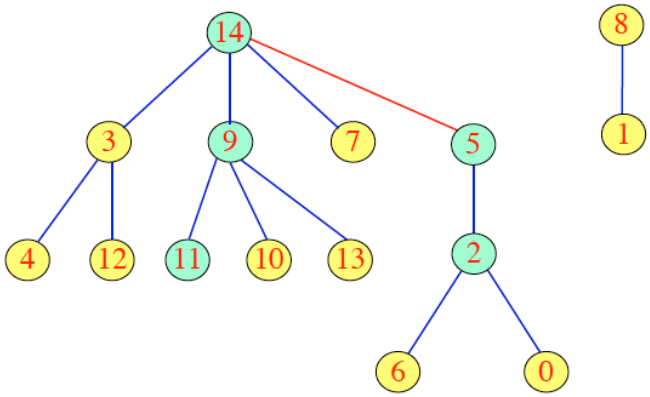
- TROUVER-ENSEMBLE(x : Elément) qui retourne le représentant de l'ensemble Sx contenant l'élément x.

2. Concernant l'algorithme de KRUSKAL, les ensembles sont des composantes connexes représentées par des arbres n-aires (non nécessairement binaires). Le représentant d'un arbre sera sa racine.

Exemple : soit n=15 sommets (numérotés de 0 à 14) et une forêt de trois arbres :



Le représentant du sommet #11 est le sommet #14 (racine de l'arbre contenant le sommet #11). Le représentant du sommet #2 est le sommet #5 (racine de l'arbre contenant le sommet #2).



Pour réaliser UNION(11, 2), on détermine le représentant du sommet #11, puis le représentant du sommet #2 et enfin, on joint les 2 racines (arête en rouge).

[2] *Chap. 21, Introduction à l'algorithmique », T. Cormen, C. Leiserson, R. Rivest et C. Stein, Dunod, 2004.*

3. Soit un graphe *G = (V, E, w)*. A l'aide de cet ADT, l'algorithme de KRUSKAL se formule :

```
fonction KRUSKAL (G : Graphe) --> ARPM ;
        début
1.      E := {} ;
2.      pour chaque sommet v de V faire CRÉER-ENSEMBLE(v) ;
3.      Trier les arêtes de E par ordre croissant selon w ;
4.      pour chaque arête (u, v) de E (par ordre croissant selon w) faire
5.              si TROUVER-ENSEMBLE(u) ≠ TROUVER-ENSEMBLE(v)
6.                      alors   E := E U {(u, v)} ;
7.                              Union(u, v) ;
8.      fin pour ;
9.      retourner E
        fin
```

Question #1.1

a) Commenter cet algorithme, en particulier les lignes de #4 à #8 que l'on justifiera.

b) Cet algorithme envisage toutes les arêtes (ligne #4). Est-ce toujours indispensable ? Si pas, proposer une autre condition d'arrêt.

4. Mise en œuvre. Pour représenter un arbre n-aire, on fait les choix suivants :

- Tout fils désigne son père.
- La racine se désigne elle-même.

Question #1.2 Donner successivement le pseudo-code de chacune des primitives :

a) procédure CRÉER-ENSEMBLE(x : Elément)

b) fonction TROUVER-ENSEMBLE(x : Elément) -> Elément

c) procédure UNION(x, y : Elément) en indiquant quel arbre devient le fils de l'autre

Indication : on pourra utiliser un tableau T tel que T[i] désigne le sommet père du sommet i.

5. Complexité temporelle

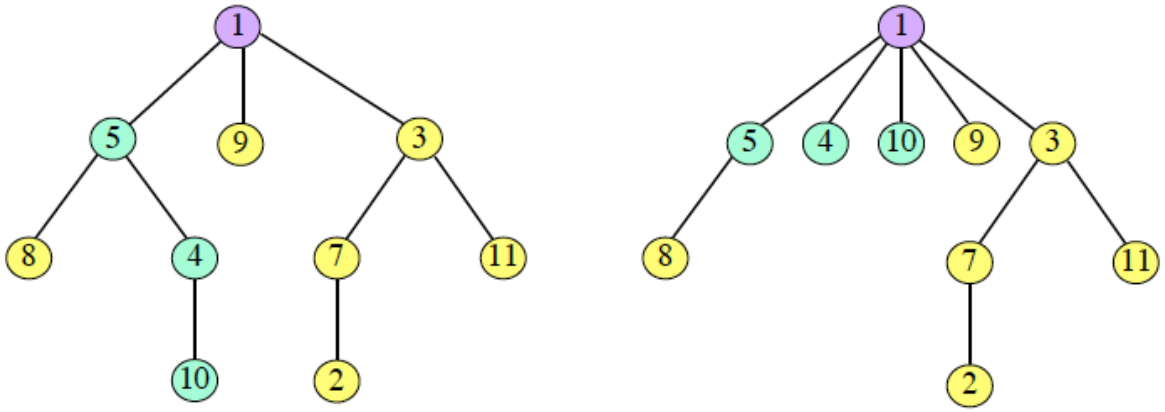
Question #1.3 Si l'on ne prend aucune précaution particulière lors de la procédure UNION(x, y : Elément), quelle sera la hauteur maximale de l'arbre obtenu dans le pire cas ?

Question #1.4 (Equilibrage) Lors de chaque union, l'arbre de plus petite hauteur devient un fils de l'arbre de plus grande hauteur. Montrer que si l'on fait ce choix, la procédure UNION(x, y) maintient toujours la condition suivante : pour chaque arbre obtenu, on a l'inégalité $h \leq log\_2(k)$ où h représente la hauteur de l'arbre et k son nombre de nœuds.

Indication: faire une preuve par induction sur le nombre d'opérations UNION effectuées à partir de la situation initiale où l'on n'a que des arbres à un seul nœud, avec donc k=1 et h=0=log_2(k)

Question #1.5 En supposant que l'étape de tri des $|E|=m$ arêtes selon leurs poids w s'effectue en temps $O(m.log\_2(m))$, quelle est la complexité temporelle (pire cas) de l'algorithme de KRUSKAL. Justifier.

6. Compression de chemin. Dans la fonction TROUVER-ENSEMBLE(x), lorsque l'on remonte l'arbre pour trouver le représentant associé à x, on peut en profiter pour que chaque nœud parcouru désigne directement la racine. **Exemple :**



On souhaite déterminer le représentant de l'arbre contenant le nœud #10. A l'aide de la fonction TROUVER-ENSEMBLE, on remonte vers la racine (nœud #1) en visitant les sommets en vert sur l'arbre de gauche. On en profite alors pour que chaque nœud ainsi parcouru désigne son représentant (racine).

Question #1.6 Qu'apporte la compression de chemin ? Mais, a-t-elle un impact sur la complexité pire cas ?

Question #1.7 Donner le pseudo-code final, (i.e. celui qui comporte équilibrage et compression) de chacune des 3 primitives : CRÉER-ENSEMBLE(x), TROUVER-ENSEMBLE(x) et UNION(x, y).

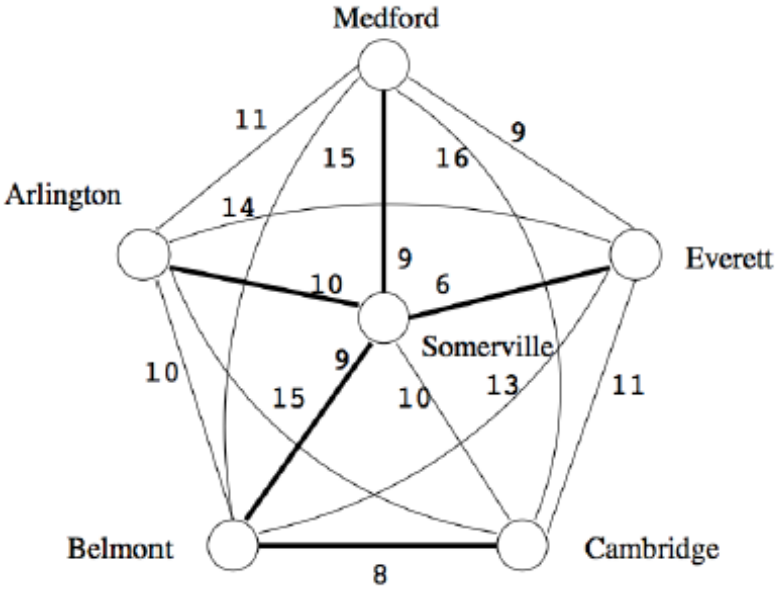
Partie 2 : Etude et mise en œuvre d'un algorithme de 2-approximation

Soit un graphe $G = (V, E, w)$. L'algorithme procède en 3 étapes :

1. Construire un ARPM M du graphe G (cf Partie #1)
2. A partir de M, construire un cycle C
3. Faire un parcours du cycle C en supprimant les sommets visités plus d'une fois

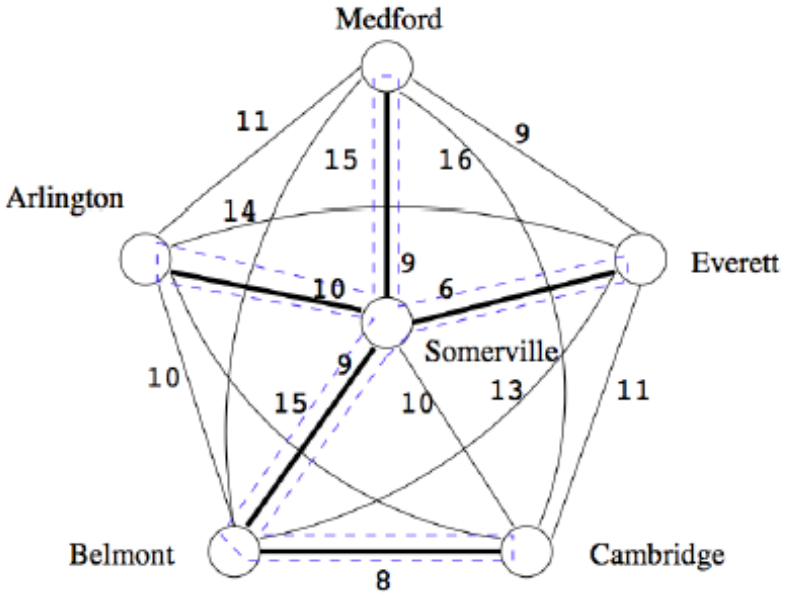
Illustrons chacune des étapes à l'aide de l'exemple ci-dessous (un 2nd exemple est présenté page #6)

- 1ère étape : le graphe ***G*** et un ARMP ***M*** (en gras)



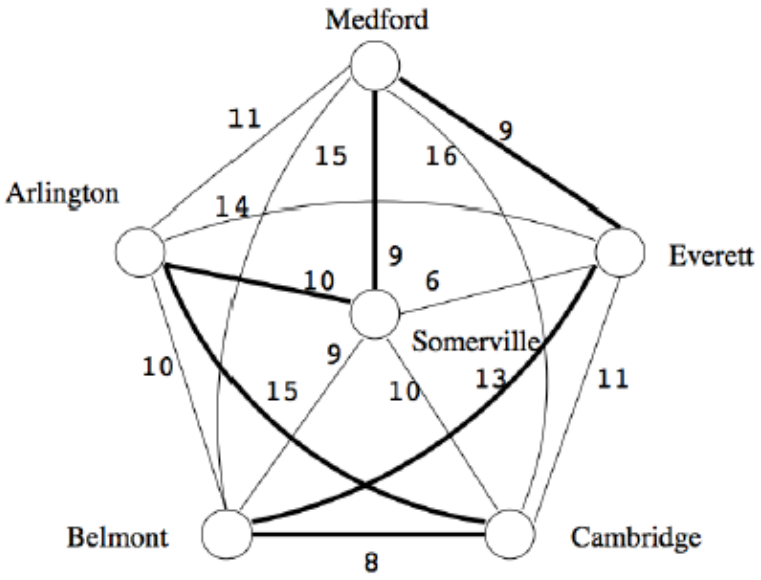
En désignant chaque ville par son initiale, on a **M** = { (S, M), (S, E), (S, B), (B, C), (S, A) }

- **2ème étape :** construire un cycle ***C*** en « doublant les arêtes de ***M*** » (en pointillé, Figure ci-dessous).



En effectuant un parcours en profondeur de ***M***,
on obtient alors le cycle ***C*** = S, M, S, E, S, B, C, B, S, A (et retour en S)

- **3ème étape :** effectuer un parcours du cycle ***C*** en supprimant les sommets visités plus d'une fois.



On obtient alors un cycle hamiltonien ***C'*** = S, M, ~~S~~, E, ~~S~~, B, C, ~~B~~, ~~S~~, A (et retour en S)

Soit un graphe ***G = (V, E, w)***, et soit ***C**** un cycle de longueur minimale **z***.

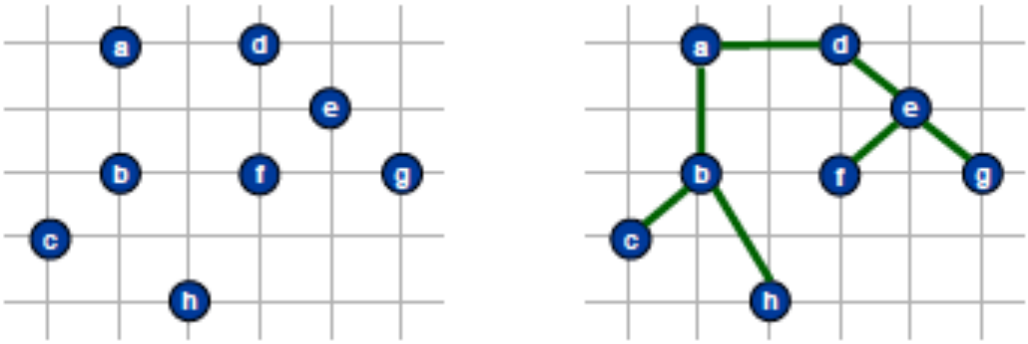
Question #2.1 Soit ***M*** un ARPM de ***G***, et soit ***C*** le cycle obtenu à l'issue de la 2-ième étape.
On veut montrer que : ***w(C) < 2.z****
Pour cela, montrer tout d'abord que ***w(M) < z****. Puis, conclure.

Question #2.2 Lors de la 3-ième étape, pourquoi peut-on supprimer les sommets de ***C*** visités plus d'une fois ? Soit ***C'*** le cycle ainsi obtenu. En déduire que ***w(C') < 2. z****

Question #2.3 Quelle est la complexité temporelle (pire cas) de l'algorithme de 2-approximation. Justifier.

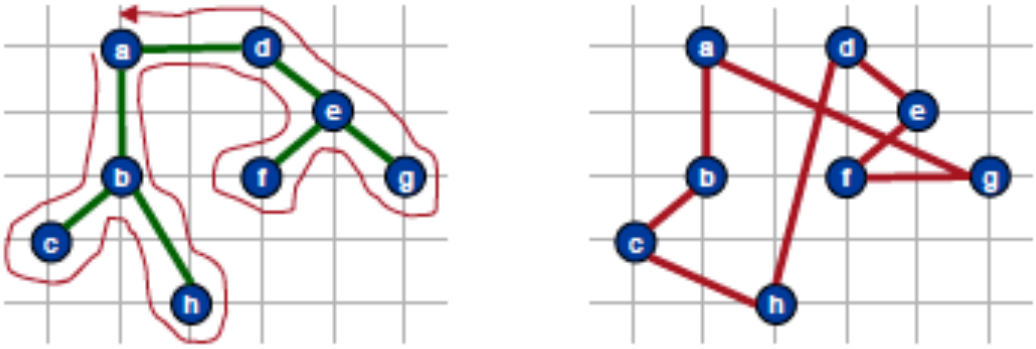
Autre exemple illustrant l'algorithme de *2*-approximation

Figure 1.



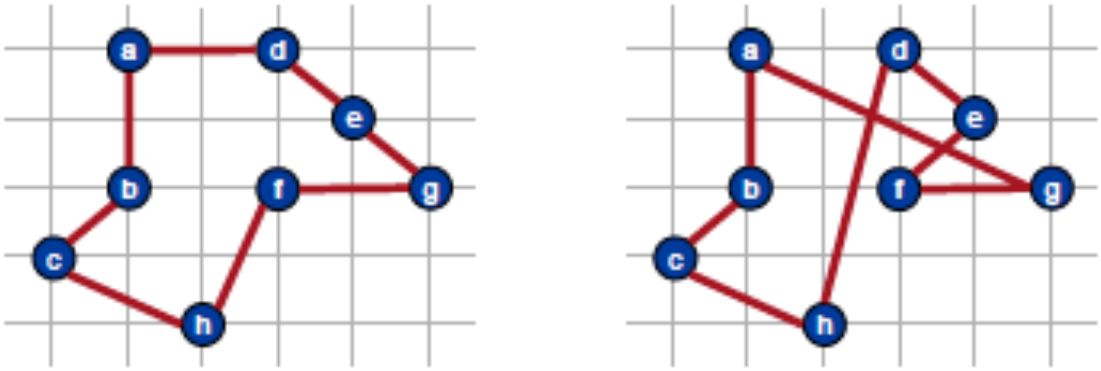
Partie gauche. Le graphe **G** de départ. Le graphe étant complet, on n'indique pas les arêtes.
La fonction de coût ***w*** utilisée est la distance euclidienne (elle vérifie donc l'inégalité triangulaire).
Partie droite, **M** qui est ARPM de **G**

Figure 2.



Partie gauche. Le cycle ***C*** **= a, b, c, b, h, b, a, d, e, f, e, g, e, d, a** obtenu en parcourant en profondeur **M**.
Partie droite. Le cycle hamiltonien ***C'*** **= a, b, c, h, d, e, f, g, a** obtenu en supprimant du cycle ***C*** les sommets visités plus d'une fois.

Figure 3.



Partie gauche. La solution optimale **s*** de coût **z*=14,715**
Partie droite. Rappel de ***C'*** obtenu par l'algorithme de 2-Approximation. Le coût de ***C'*** est **z=19,074**

Partie 3. Algorithme de CHRISTOFIDES (3/2-approximation)

Soit un graphe *G = (V, E, w)*. L'algorithme de CHRISTOFIDES procède en 3 étapes analogues à l'algorithme de 2-approximation.

1. Construire un ARPM *M* du graphe *G*
2. A partir de *M*, construire un cycle *C*
3. Faire un parcours du cycle *C* en supprimant les sommets visités plus d'une fois

La seule différence (avec la 2-approximation) concerne la 2-ième étape où la construction d'un cycle repose sur les notions de couplage parfait et de recherche d'un cycle Eulérien.

3.1 Préambule

Définition. Un multi-graphe est un graphe où deux sommets donnés peuvent être reliés par plusieurs arêtes.

Question #3.1 Montrer que tout multi-graphe possède un nombre pair de sommets de degré impair.

Définition d'un parcours Eulérien. Un multi-graphe connexe *G = (V, E)* possède un parcours Eulérien *ssi* on peut parcourir *E* en passant, une fois et une seule, par chaque arête.

Le théorème d'EULER donne une *CNS* d'existence d'un parcours Eulérien :

- si tous les sommets sont de degré pair, alors le parcours est un cycle
- si seuls 2 sommets (x et y) sont de degré impair, alors le parcours est une chaîne de x à y (ou de y à x)
- dans tous les autres cas, il n'existe pas de parcours Eulérien.

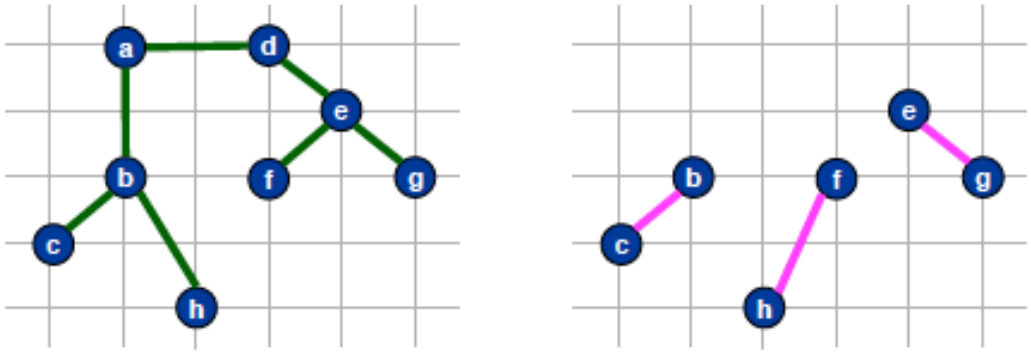
Définition. ***Un couplage parfait*** d'un graphe *G* est un couplage *M* du graphe tel que tout sommet du graphe *G* est incident à exactement une arête de *M*.

3.2 Construction d'un cycle *C* à partir de l'ARPM *M*

Soit ***M*** un ARPM de ***G***, et soit ***G'*** le sous-graphe de ***G*** restreint aux sommets de ***M*** de degré impair dans ***M***.

A) Dans un premier* temps,** on calcule ***M*, couplage parfait de ***G'*** de coût (somme des coûts des arêtes de M*) minimal. L'étude d'un tel algorithme dépasse le cadre de ce devoir[3]. On admettra que le calcul d'un couplage parfait de coût minimal d'un graphe complet valué ayant *2.n* sommets s'effectue en temps *O(n^3)*.

Question #3.2 Pourquoi est-on sûr qu'il existe (au moins) un couplage parfait de ***G'*** ?
Reprenons l'exemple page #6.



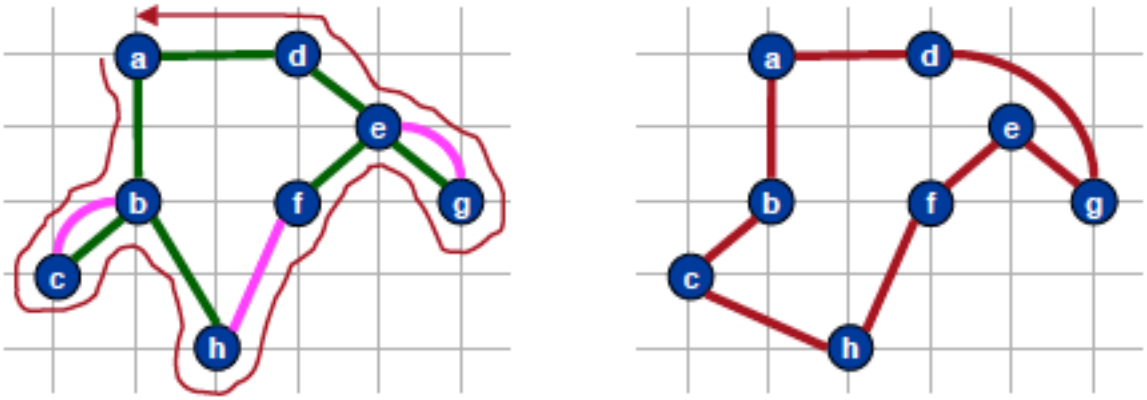
Partie gauche. Tous les sommets de l'ARPM ***M*** sont de degré impair, sauf les sommets ***a*** et ***d*** (de degré 2)
Partie droite. Un couplage parfait ***M**** de coût minimal du sous-graphe restreint aux sommets de degré impair de ***M*** : ***b, c, e, f, g*** et ***h***

[3] *Paths, Trees, and Flowers, D. EDMONDS, Canadian Journal of Math, vol 17, pp 449-467, 1965.*

B) Dans un 2nd temps, on considère le multi-graphe $H = M \cup M^*$ et on construit un cycle Eulérien C de H.

Question #3.3 Pourquoi est-on sûr que H possède un cycle Eulérien ?

Reprenons notre exemple :



Partie gauche. C cycle Eulérien de $H = M \cup M^*$

Partie droite. Le cycle hamiltonien $C' = a, b, c, h, f, e, g, d, a$ obtenu en supprimant les sommets visités plus d'une fois (cf Etape #3).

3.3 L'algorithme de CHRISTOFIDES constitue une 3/2 -approximation

Soient un graphe $G = (V, E, w)$, C^* un cycle de longueur minimale z^*, M un ARPM de G, et soient :

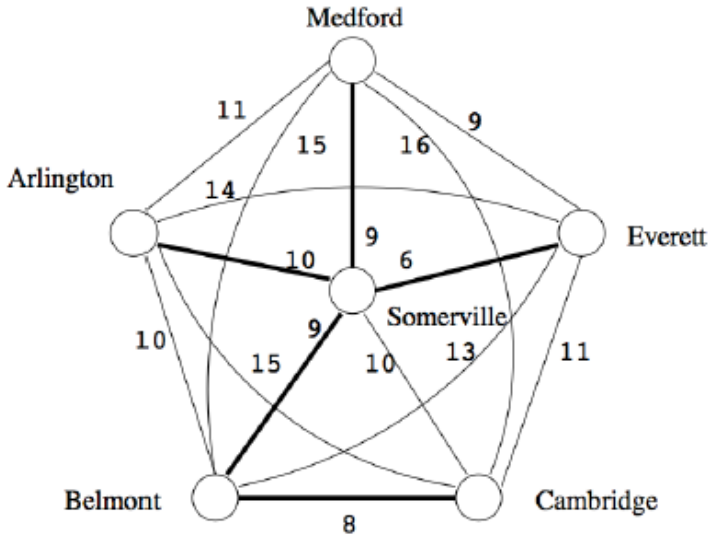
- G' le sous-graphe de G restreint aux sommets de M de degré impair
- M^* un couplage parfait de G' de coût minimal
- C un cycle eulérien de $H = M \cup M^*$
- C' un cycle hamiltonien obtenu en supprimant les sommets de C visités plus d'une fois

Question #3.4

a. Montrer que : $w(M^*) \leq 1/2\ z^*$

b. En utilisant la Question #2.1, en déduire que : $w(C') \leq 3/2\ z^*$

Question #3.5 Dérouler cet algorithme sur le 1er exemple en prenant comme ARPM M (page 4)



Question #3.6 Quelle est la complexité temporelle (pire cas) de l'algorithme de CHRISTOFIDES. Justifier.